Preface

Proceedings of the 28th International Teletraffic Congress

ITC 28

Tobias Hoßfeld, Brian Mark, Gary Chan, Andreas Timm-Giel
(ITC 28 TPC Co-Chairs)
Session 1.A: Clouds and Data Centers

Offering Resilient and Bandwidth Guaranteed Services in Multi-tenant Cloud Networks: Harnessing the Sharing Opportunities .............................................. Hyame Assem Alameddine, Sara Ayoubi, and Chadi Assi

Dynamic Virtual Network Traffic Engineering with Energy Efficiency in Multi-location Data Center Networks ................................................................. Mirza Mohd Shahriar Maswood, Chris Develder, Edmundo Madeira, and Deep Medhi

An Energy-Aware Embedding Algorithm for Virtual Data Centers ................................................ Tran Manh Nam, Nguyen Van Huynh, Le Quang Dai, and Nguyen Huu Thanh
Session 1.B: Traffic and Network Management

Disaster Avoidance Control against Tsunami ................................................................. 26
Phuong Nga Tran and Hiroshi Saito

Building a Low Latency Linux Software Router .......................................................... 35
Alexander Beifuß, Torsten M. Runge, Daniel Raumer, Paul Emmerich,
Bernd E. Wollinger, and Georg Carle

Traffic-Driven Implicit Buffer Management - Delay Differentiation
without Traffic Contracts .............................................................................................. 44
Martin Karsten, Daniel S. Berger, and Jens Schmitt

Session 2: Wireless

Self-Optimization of Software Defined Radios through Evolutionary Algorithms .................................................. 53
Zubair Shaik, André Puschmann, and Andreas Mitschele-Thiel

Opportunistic Channel Estimation for Implicit 802.11af MU-MIMO .................................................. 60
Ryan E. Guerra, Narendra Anand, Clayton Shepard, and Edward W. Knightly

DiVote: A Distributed Voting Protocol for Mobile Device-to-Device Communication .................................................. 69
Peter Danielis, Sylvia T. Kouyoumdjieva, and Gunnar Karlsson

Session 3.A: Cellular

Joint Optimization of User Association and User Satisfaction in Heterogeneous Cellular Networks .................................................. 78
Farah Moety, Mustapha Bouhtou, Taoufik En-Najjary, and Ridha Nasri

Joint Resource Allocation and User Association for Heterogeneous Cloud Radio Access Networks .................................................. 87
Ying Loong Lee, Li-Chun Wang, Teong Chee Chuah, and Jonathan Loo

Performance-Oriented Association in Large Cellular Networks with Technology Diversity .................................................. 94
Abishek Sankararaman, Jeong-Woo Cho, and François Baccelli

Session 3.B: Video Streaming

Bridging the Gap between QoE and User Engagement in HTTP Video Streaming .................................................. 103
Christian Moldovan and Florian Metzger

A Markov Model for Evaluating Resource Sharing Policies for DASH Assisting Network Elements .................................................. 112
Jan Willem Kleinrouweler, Sergio Cabrero, Rob Van Der Mei, and Pablo Cesar

Mobile Live Video Upstreaming .................................................. 121
Philip Lundrigan, Mojgan Khaledi, Makito Kano,
Naveen Dasa Subramanyam, and Sneha Kasera
PlanetIgnite: A Self-Assembling, Lightweight, Infrastructure-as-a-Service

Andy Bavier, Rick McGeer, and Glenn Ricart

Session 4.A: Caching Strategies

Stochastic Dynamic Cache Partitioning for Encrypted Content Delivery .............................................. 139
Andrea Araldo, György Dán, and Dario Rossi

Access-Time Aware Cache Algorithms .................................................................................................. 148
Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari

Asymptotically Exact TTL-Approximations of the Cache Replacement Algorithms LRU(m) and h-LRU ......................................................................................................................... 157
Nicolas Gast and Benny Van Houdt

Session 4.B: Performance Analysis

Meeting Soft Deadlines in Single- and Multi-server Systems ................................................................. 166
Esa Hyytiä, Rhonda Righter, and Jorma Virtamo

Performance Analysis of CoDel and PIE for Saturated TCP Sources .................................................... 175
Fabian Schwarzkopf, Sebastian Veith, and Michael Menth

Stochastic Upper and Lower Bounds for General Markov Fluids .......................................................... 184
Florin Ciucu, Felix Poloczek, and Jens Schmitt

Session 5: Demo Session

Demo: Resilient Integration of Distributed High-Performance Zones into the BelWue Network Using OpenFlow .............................................................................................................................. 193
Mark Schmidt, Robert Finze, Daniel Reutter, and Michael Menth

Demonstrating a Personalized Secure-by-Default Bring Your Own Device
Solution Based on Software Defined Networking ...................................................................................... 197
Steffen Gebert, Thomas Zinner, Nicholas Gray, Raphael Durner, Claas Lorenz, and Stanislav Lange

Demonstrating Context-Aware Services in the Mobility First Future Internet
Architecture ............................................................................................................................................... 201
Francesco Bronzino, Dipankar Raychaudhuri, and Ivan Seskar

jLISP: An Open, Modular, and Extensible Java-Based LISP Implementation ........................................ 205
Andreas Stockmayer, Mark Schmidt, and Michael Menth

Network as a Service - A Demo on 5G Network Slicing ......................................................................... 209
Rastin Pries, Hans-Jochen Morper, Nandor Galambosi, and Michael Jarschel

Automated Establishment of a Secured Network for Providing a Distributed Container Cluster .................. 212
Marcel Großmann and Andreas Eiermann
Session 6.A: Softwarization
Sector: TCAM Space Aware Routing on SDN

Sai Qian Zhang, Qi Zhang, Ali Tizghadam, Byungchul Park,
Hadi Bannazadeh, Raouf Boutaba, and Alberto Leon-Garcia

Port Based Capacity Extensions (PBCEs): Improving SDNs Flow Table Scalability

Robert Bauer and Martina Zitterbart

Performance Modeling of Softwarized Network Functions Using Discrete-Time Analysis

Steffen Gebert, Thomas Zinner, Stanislav Lange, Christian Schwartz,
and Phuoc Tran-Gia

Session 6.B: Information and Social Networks

Cache the Queues: Caching and Forwarding in ICN from a Congestion Control Perspective

Dinh Nguyen, Kohei Sugiyama, and Atsushi Tagami

Optimizing Time to Exhaustion in Service Providers Using Information-centric Networking

Ali Shariat, Ali Tizghadam, and Alberto Leon-Garcia

Binary Opinion Dynamics with Biased Agents and Agents with Different Degrees of Stubborness

Arpan Mukhopadhyay, Ravi R. Mazumdar, and Rahul Roy

LiveTalk: A Framework for Collaborative Browser-Based Replicated-Computation Applications

Matthew Hemmings, Daniel Ingalls, Robert Krahn, Rick McGeer,
Glenn Ricart, Marko Röder, and Ulrike Stege

Session 7.A: Measurements

IntegraTag: A Framework for High-Fidelity Web Client Measurement

Charles Thomas, Jeff Kline, and Paul Barford

CLUE: Clustering for Mining Web URLs

Andrea Morichetta, Enrico Bocchi, Hassan Metwalley, and Marco Mellia

Testing for Traffic Differentiation with ChkDiff: The Downstream Case

Riccardo Ravaiolli, Guillaume Urvoy-Keller, and Chadi Barakat

Session 7.B: Caching

ModelGraft: Accurate, Scalable, and Flexible Performance Evaluation of General Cache Networks

Michele Tortelli, Dario Rossi, and Emilio Leonardi

Distributed Algorithms for Content Caching in Mobile Backhaul Networks
Performance Evaluation for New Web Caching Strategies Combining LRU with Score Based Object Selection .................................................................322

Gerhard Hasslinger, Konstantinos Ntougias, Frank Hasslinger, and Oliver Hohlfeld

Session 8: Virtualization

A Power Efficient and Robust Virtual Network Functions Placement Problem ..................................................................................................331

Antonio Marotta and Andreas Kassler

Elastic Network Service Provisioning with VNF Auctioning .........................................................340

Mathis Obadia, Mathieu Bouet, Vania Conan, Luigi Iannone, and Jean-Louis Rougier

ITC 28 Author Index ..............................................................................................................................................349
Welcome Message from General Co-Chairs

On behalf of the Organizing Committee, we are delighted to welcome you to the 28th International Teletraffic Congress (ITC 28) to be held on September 12-16, 2016 in lovely Würzburg, Germany!

Since its inception in 1955, ITC has witnessed the evolution of communications and networking: the influence of computer science on telecommunications, the advent of the Internet and the massive deployment of mobile communications and optics, the emergence of peer-to-peer networking and social network services, the ever increasing speed and flexibility of new communication technologies, networks, devices, and applications, and the ever changing operational challenges arising from these developments. ITC has also documented this evolution with state-of-the-art measurement studies, performance analyses of new technologies, recommendations for provisioning and configuration, and greatly contributed to the advancement of methodologies for network design and analysis.

Its inherent roots in solid methodological foundations have allowed ITC to constantly adapt its technological focus without losing its original identity. ITC continues to serve as a broad and lively community for researchers and practitioners dedicated to advancing the limits of knowledge in networking. As such, ITC regularly organizes such events as congresses, specialist seminars and workshops for experts to gather and discuss the latest developments in design, modelling, and performance evaluation of communication systems, networks, and services.

This year’s ITC technical program is composed of 37 contributed full papers and 6 short demo papers to be presented in two parallel sessions, three keynote addresses and a demo session. We also sponsor three workshops dedicated to timely topics: Workshop on Programmability for Cloud Networks and Applications (PROCON), 2016 International Workshop on Quality of Experience Centric Management (QCMa), COST Action ACROSS Workshop on “Quality Engineering for a Reliable Internet of Services”.

We are especially grateful to our keynote speakers: Dr. Nikhil Jain (Qualcomm Technologies), who will talk on “Internet of Everything: Engineering Challenges and Opportunities”; Prof. Wolfgang Kellerer (Technische Universität München), who will talk on “Towards flexible networking in dynamically changing environments”; and Dr. Eitan Altman (INRIA Sophia Antipolis), who will talk on “Dynamic games for analyzing competition in the Internet”.

We also thank ITC’s International Advisory Committee (IAC) for their support of student travel grants and best paper awards. The IAC has graciously decided to offer a number of travel grants available to full-time students. ITC 28 has set up three prestigious awards. The Best Paper Award will be granted to the best contribution presented at ITC 28. The Best Student Paper Award will be conferred upon the best paper whose first author is a full-time student at the time of submission of the paper and is the presenter. The Best Demo Award will be granted to the best demo presented during the ITC 28 meeting. These awards will be selected based on scientific merit and oral presentation or demo presentation quality.

A successful conference requires dedication and engagement of many people. We would like to recognize the efforts of the TPC Co-Chairs, Professors Tobias Hößfeld, Brian Mark, Gary Chan and Andreas Timm-Giel, who put together this excellent technical program. We thank Mrs. Alison Wichmann for the local arrangements, Dr. Matthias Hirth and Dr. Florian Wamser, Local Organization Co-Chairs, as well as Mr. Christopher Metter and Dr. Florian Metz-
ger, Web & EDAS Co-Chairs. Dr. Prosper Chemouil as Award Chair reviewed the student travel grant applications and will lead the best paper award selection process. Our Publicity Co-Chairs, Prof. Florin Ciucu and Prof. Sheng Zhou, disseminated information about ITC 28 throughout the world. Our Publication Co-Chairs, Prof. Jörg Liebeherr and Prof. Michael Menth, organized the publications with the CPS publisher.

Our thanks also go to Dr. Florian Wamser, Dr. Roberto Bruschi and Dr. Anastasios Zafeiro-poulos for organizing the PROCON workshop; Dr. Thomas Zinner, Dr. Oliver Hohlfeld, Dr. Raimund Schatz, and Prof. Prasad Calyam for organizing the QCMan workshop; Prof. Hans van den Berg and Prof. Rob van der Mei for organizing the ACROSS workshop. We greatly appreciate everyone who submitted papers to the conference, particularly those who will be presenting their work at ITC 28. The IEEE, IEEE Communications Society, and the Information Technology Society within VDE (ITG VDE) kindly agreed to technically co-sponsor ITC 28, and ACM SIGCOMM helped us through their in-cooperation agreement. Last, but not least, we are grateful to our corporate patrons: Infosim, kubusIT, and Orange, who generously provided financial support to ITC 28, as well as the Julius Maximilian Universität Würzburg for their support in organizing and hosting the conference.

Phuoc Tran-Gia (University of Würzburg, Germany)
Hisashi Kobayashi (Princeton University, USA)

September 2016
Welcome Message from Technical Program Co-Chairs

Welcome to Würzburg and the 28th International Teletraffic Congress (ITC 28)!

The evolution of communication and networking is changing the world we are living in. The digital connected world is triggered by the advances on telecommunications, the penetration of the Internet, the massive deployment of mobile communications and optics, the adoption of collaborative networking and social networks, the ever-increasing speed and flexibility of new communication technologies, networks, user devices, and applications, and various operational challenges arising from this development.

ITC was established as the first international conference on networking science and practice. It gathers a wide and lively community of researchers and practitioners dedicated to pushing the envelope in the area of networking. As such, ITC has provided a forum for leading researchers from academia and industry to present and discuss the latest changes and developments in design, modelling, measurement, and performance evaluation of communication systems, networks, and services.

ITC 28 has continued this tradition, while employing some new approaches to attract high-quality papers and researchers. In particular, ITC 28 introduced the concept of areas and a demo session. ITC 28 is structured into eight different areas which address hot topics in networking. Each area is chaired by two internationally well recognized experts in that area. The area chairs organized a smaller TPC per area. The idea was that the area chairs invited experts for their areas from the ITC community as well as other well-known experts worldwide. On the one hand, the concept was aimed at expanding the ITC community and attracting high-quality submissions. On the other hand, the areas helped to improve the quality of the review process. The area chairs assigned the reviews to experts in their domain and evaluate all papers in their domain.

In addition, we introduced demo sessions for ITC 28 that cuts thematically across the areas. The demo session is distinguished from the regular sessions only in the presentation format. “Demo papers” are papers whose content is best understood by an audience if the material is demonstrated rather than presented in a lecture style slide presentation. With the demo session, we aimed to provide a different kind of interactions among the participants, so as to make ITC more attractive for other communities.

Accordingly, ITC 28 is structured into the following eight different areas and demo session with the listed chairs:

Area 1: Smart cities and IoT (Alberto Leon-Garcia, Yanmin Zhu)
Area 2: Cloud services and networking (Arup Acharya, Patrick Lee)
Area 3: Mobile, wireless and 5G (Kin Leung, Thomas Hou)
Area 4: Next generation and future Internet architectures (Michael Zink, Thomas Zinner)
Area 5: Network and traffic management (Florin Ciucu, Peter Reichl)
Area 6: Network design and optimization (Thomas Bauschert, Eric Wong)
Area 7: Network measurements and analysis (Marco Mellia, Mark Squillante)
Area 8: Networked applications (Zhu Li, Lea Skorin-Kapov)

Demonstration Session (Mark Berman, Michael Jarschel, Rick McGeer)
ITC 28 attracted 116 international paper submissions across all areas, while 157 papers were registered. The 116 papers were submitted by authors from 33 different countries, out of which 20% were from the USA and Canada, 68% from Europe/Middle East/Africa, 11% from Asia/Pacific and the remainder from Latin America.

Each submitted paper was reviewed by at least three experts assigned by the area chairs and TPC chairs. All papers are single-blind reviewed. In special cases, when the discussion of reviewers did not converge, additional expert reviews were requested to come to a solicited decision. In total, there were 420 completed reviews for the 116 submitted papers, i.e. an average of 3.6 reviews per paper. The area chairs and TPC members fostered discussions to converge the reviewers’ recommendations towards a decision. In total, 300 discussion posts were provided for papers with diverging review scores. The area chairs provided a ranked list of papers with suggestions for papers to be accepted and rejected.

A full-day TPC meeting was held at the University of Würzburg, Germany, from 9.00 – 19.30 on May 3, 2016. The meeting was structured according to the areas. The area chairs presented the papers submitted to their area and the list of ranked papers.

Based on the reviews and the recommendations from the area chairs, it was decided during the TPC meeting which papers were to be accepted or rejected per area. In addition, for each area a few reserve papers were identified. It should be noted that those papers were also good contributions. After the discussion of all areas, the reserve papers were discussed by the physically attending TPC members in Würzburg. The papers were evaluated and compared across different areas in order to identify the best papers from among the reserve papers. If an accepted paper was flagged as needing improvement, shepherding of such papers was initiated by the area chairs. Shepherding was led by the area chairs or a TPC member assigned to a particular area.

Finally, 37 full papers were accepted out of the 108 full paper submissions, yielding an acceptance rate of 34%. In addition, 6 short demo papers were accepted. The statistics per area are given below. From among the authors of accepted papers, 28% are from USA and Canada, 61% from Europe/Middle East/Africa, 8% from Asia/Pacific and the remaining are from Latin America.

<table>
<thead>
<tr>
<th>Area</th>
<th>Registered</th>
<th>Submitted</th>
<th>Accepted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Smart cities and IoT</td>
<td>8</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>2. Cloud services and networking</td>
<td>14</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>3. Mobile, wireless and 5G</td>
<td>19</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>4. Next generation and future Internet architectures</td>
<td>26</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>5. Network and traffic management</td>
<td>23</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>6. Network design and optimization</td>
<td>18</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>7. Network measurements and analysis</td>
<td>21</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>8. Networked applications</td>
<td>12</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>Demos</td>
<td>16</td>
<td>14</td>
<td>3</td>
</tr>
</tbody>
</table>
Given the accepted papers, we then group the papers according to their topics. On behalf of the Technical Program Committee (TPC), we proudly present to you an excellent technical program covering a wide range of topics which are manifested in 12 technical oral sessions and a demo session.

<table>
<thead>
<tr>
<th>Session 1.A Clouds and Data Center</th>
<th>Session 1.B Traffic and Network Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session 2 Wireless</td>
<td>Session 6.A Softwarization</td>
</tr>
<tr>
<td>Session 3.A Cellular</td>
<td>Session 6.B Information and Social Networks</td>
</tr>
<tr>
<td>Session 4.B Performance Analysis</td>
<td>Session 8 Virtualization</td>
</tr>
</tbody>
</table>

The technical program is presented in the form of double-track sessions spanning three days, from September 13 to 15, 2016. The demo session, the three keynote speeches, and two selected sessions are presented as plenary sessions. On the first day of the congress, September 12, 2016, a half-day workshop on Programmability for Cloud Networks and Applications (PROCON) takes place. On the final day of the congress, September 16, there are two full-day workshops: (1) 2016 International Workshop on Quality of Experience Centric Management (QCMAN) and (2) Workshop of COST Action ACROSS on “Quality Engineering for a Reliable Internet of Services”.

We are delighted to have three excellent keynote speakers in the main program. We thank them for agreeing to be keynote speakers and presenting their visions in spite of their busy schedules.

- Nikhil Jain (Vice President of Technology, Qualcomm Technologies, Inc.): Internet of Everything: Engineering Challenges and Opportunities
- Wolfgang Kellerer (Technical University of Munich (TUM), Germany): Towards Flexible Networking in Dynamically Changing Environments
- Eitan Altman (INRIA Sophia Antipolis, France): Dynamic Games for Analyzing Competition in the Internet

The TPC co-chairs wish to thank in particular, the area chairs who did a fantastic job and dedicated much effort to make ITC 28 a success. We thank the TPC members and experts that provided paper reviews, contributed to the discussions and attended the TPC meeting for the conference. Without their diligence and hard work the program could not have been put together. And, of course, we thank everyone who submitted a paper and those who are presenting their work at the conference.

Further we wish to give special thanks to the University of Würzburg for hosting the TPC meeting and we are particularly indebted to Thomas Zinner for his willingness to help in all aspects of organizing ITC 28. Special thanks for their efforts in the TPC meeting are dedicated to Benny Van Houdt, Florin Ciucu, and Michael Zink. We thank the members of the ITC steering committee, particularly Michela Meo for providing guidance. Last but not least we wish to thank the previous ITC organizers for passing on their thoughts and experiences: Dragos Illie, Peter Van Daele, Markus Fiedler, Michela Meo, Sabine Wittevrongel. We thank Harry Rudin for supporting us in setting up the Elsevier Computer Networks Special Issue on “Softwarization and Caching in NGN” related to ITC 28.
Special thanks go to the ITC 28 publications chairs, Michael Menth and Jörg Liebeherr, who took care of the publication process and made the technical co-sponsorship happen with IEEE Communications Society (IEEE ComSoc) as well as the cooperation with ACM SIGCOMM. We acknowledge the publicity chairs, Florin Ciucu and Sheng Zhou, for their extensive efforts to make ITC 28 visible and to attract submissions and attendees. We thank Prosper Chemouil, the awards chair, for taking care of the student travel grants and the best paper awards. We extend our sincere thanks to Florian Metzger for facilitating the paper submission and review process electronically in EDAS, Christopher Metter for taking care of the ITC 28 mailing lists and web site, as well as the local organizers Matthias Hirth, Florian Wamser and Alison Wichmann for implementing the ITC 28 registration process, all local arrangements and the social events to make ITC 28 happen.

Finally, we would like to express our appreciation the general chairs, Phuoc Tran-Gia and Hisashi Kobayashi, for all of their hard work in putting together an excellent overall program and a wonderful ITC 28 event.

Tobias Hoßfeld (University of Duisburg-Essen, Germany)
Brian L. Mark (George Mason University, US)
Gary Chan (The Hong Kong University of Science and Technology, China)
Andreas Timm-Giel (Hamburg University of Technology, Germany)

September 2016
Committees

Conference Co-Chairs

Phuoc Tran-Gia  University of Würzburg, Germany
Hisashi Kobayashi  Princeton University, US

Technical Program Committee Chairs

Tobias Hoßfeld  University of Duisburg-Essen, Germany
Brian Mark  George Mason University, US
Gary Chan  The Hong Kong University of Science and Technology, China
Andreas Timm-Giel  Hamburg University of Technology, Germany

Local Organization Co-Chairs

Matthias Hirth  University of Würzburg, Germany
Florian Wamser  University of Würzburg, Germany

Award Chair

Prosper Chemouil  Orange Labs Networks, France

Publicity Co-Chairs

Florin Ciucu  University of Warwick, UK
Sheng Zhou  Tsinghua University, China

Publication Co-Chairs

Jörg Liebeherr  University of Toronto, Canada
Michael Menth  University of Tübingen, Germany
Web & EDAS Co-Chairs

Christopher Metter  University of Würzburg, Germany
Florian Metzger   University of Duisburg-Essen, Germany

International Advisory Council

Chair: Michela Meo  Politecnico di Torino, Italy
Vice-Chair: Dario Rossi  Telecom ParisTech / Ecole Polytechnique, France
Markus Fiedler  BTH, Sweden
Fabrice Guillemain  Orange Labs, France
Tobias Hoßfeld  University of Duisburg-Essen, Germany
Benny Van Houdt  University of Antwerp, Belgium
Michael Menth  University of Tuebingen, Germany
Zhisheng Niu  Tsinghua University, China

Area Chairs

Alberto Leon-Garcia  University of Toronto, Canada
Yanmin Zhu  Shanghai Jiatong University, China
Arup Acharya  IBM Research, US
Patrick Lee  The Chinese University of Hong Kong, China
Kin Leung  Imperial College, UK
Thomas Hou  Virginia Tech, US
Thomas Zinner  University of Würzburg, Germany
Michael Zink  University of Massachusetts Amherst, US
Florin Ciucu  University of Warwick, UK
Peter Reichl  University of Vienna, Austria
Thomas Bauschert  TU Chemnitz, Germany
Eric Wong  City University of Hong Kong, China
Marco Mellia  Politecnico di Torino, Italy
Mark Squillante  IBM Research, US
Lea Skorin-Kapov  University of Zagreb, Croatia
Zhu Li  Samsung Research America, US
Mark Berman  GENI Project Office, US
Michael Jarschel  Nokia, Munich, Germany
Rick McGeer  SAP / University of Victoria, US
## Technical Program Committee Members

### Area 1: Smart Cities and IoT

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azzedine Boukerche</td>
<td>University of Ottawa, Canada</td>
</tr>
<tr>
<td>Zhichao Cao</td>
<td>Tsinghua University, China</td>
</tr>
<tr>
<td>Prosper Chemouil</td>
<td>Orange Labs, France</td>
</tr>
<tr>
<td>Ken Duffy</td>
<td>National University of Ireland Maynooth, Ireland</td>
</tr>
<tr>
<td>Omar Elloumi</td>
<td>Alcatel-Lucent, France</td>
</tr>
<tr>
<td>Yaser P. Fallah</td>
<td>West Virginia University, US</td>
</tr>
<tr>
<td>Yacine Ghamri-Doudane</td>
<td>University of La Rochelle, France</td>
</tr>
<tr>
<td>Marco Gribaudo</td>
<td>Politecnico of Milano, Italy</td>
</tr>
<tr>
<td>Hongyu Huang</td>
<td>Chongqing University, China</td>
</tr>
<tr>
<td>Yu Hua</td>
<td>Huazhong University of Science and Technology, China</td>
</tr>
<tr>
<td>Yuming Jiang</td>
<td>Norwegian University of Science and Technology, Norway</td>
</tr>
<tr>
<td>Linghe Kong</td>
<td>McGill University, Canada</td>
</tr>
<tr>
<td>Udo Krieger</td>
<td>Otto Friedrich University Bamberg, Germany</td>
</tr>
<tr>
<td>Vincenzo Mancuso</td>
<td>IMDEA Networks Institute, Spain</td>
</tr>
<tr>
<td>Rob van der Mei</td>
<td>Centrum voor Wiskunde en Informatica, the Netherlands</td>
</tr>
<tr>
<td>Hamed Mohsenian-Rad</td>
<td>University of California at Riverside, US</td>
</tr>
<tr>
<td>Zhengguo Sheng</td>
<td>University of Sussex, UK</td>
</tr>
<tr>
<td>JaeSeung Song</td>
<td>Sejong University, Korea</td>
</tr>
<tr>
<td>Yutaka Takahashi</td>
<td>Kyoto University, Japan</td>
</tr>
<tr>
<td>Ali Tizghadam</td>
<td>University of Toronto, Canada</td>
</tr>
<tr>
<td>Sabine Wittevrongel</td>
<td>Ghent University, Belgium</td>
</tr>
<tr>
<td>Weigang Wu</td>
<td>Sun Yat-sen University, China</td>
</tr>
<tr>
<td>Guanglin Zhang</td>
<td>Donghua University, China</td>
</tr>
<tr>
<td>Xiaolong Zheng</td>
<td>Tsinghua University, China</td>
</tr>
</tbody>
</table>

### Area 2: Cloud Services and Networking

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samuli Aalto</td>
<td>Aalto University, Finland</td>
</tr>
<tr>
<td>Fabrice Guillemin</td>
<td>Orange Labs, France</td>
</tr>
<tr>
<td>Xiaoming Fu</td>
<td>Georg-August-University of Goettingen, Germany</td>
</tr>
<tr>
<td>Carol Fung</td>
<td>Virginia Commonwealth University, US</td>
</tr>
<tr>
<td>Jagadeesh Harshan</td>
<td>Nanyang Technological University, Singapore</td>
</tr>
<tr>
<td>Yuchong Hu</td>
<td>Huazhong University of Science and Technology, China</td>
</tr>
<tr>
<td>Atsushi Iwata</td>
<td>NEC Corporation, US</td>
</tr>
<tr>
<td>Dan Li</td>
<td>Tsinghua University, China</td>
</tr>
<tr>
<td>Richard T. B. Ma</td>
<td>National University of Singapore, Singapore</td>
</tr>
<tr>
<td>Shachi Sharma</td>
<td>IBM Research, India</td>
</tr>
<tr>
<td>Dennis Shea</td>
<td>IBM Thomas J. Watson Research Center, US</td>
</tr>
<tr>
<td>Hong Xu</td>
<td>City University of Hong Kong, China</td>
</tr>
<tr>
<td>Farhana H. Zulkernine</td>
<td>Queen’s University, Canada</td>
</tr>
</tbody>
</table>
Area 3: Mobile, Wireless and 5G

Ozgur Akan Koc University, Turkey
Nirwan Ansari New Jersey Institute of Technology, US
Albert Banchs Universidad Carlos III de Madrid, Spain
Azzedine Boukerche University of Ottawa, Canada
Claudio Casetti Politecnico di Torino, Italy
Song Chong Korea Advanced Institute of Science and Technology, Korea
Baek-Young Choi University of Missouri - Kansas City, US
Chen-Nee Chuah University of California at Davis, US
Luis Correia IST – University of Lisbon, Portugal
Do Young Eun North Carolina State University, US
Nelson da Fonseca State University of Campinas, Brazil
Luigi Fratta Politecnico di Milano, Italy
Sergey Gorinsky IMDEA Networks Institute, Spain
Linke Guo Binghamton University, US
Song Guo The University of Aizu, Japan
Teruo Higashino Osaka University, Japan
Rose Qingyang Hu Utah State University, US
Jianwei Huang Chinese University of Hong Kong, China
Wonjun Lee Korea University, Korea
Douglas Leith Trinity College, Ireland
Victor Leung University of British Columbia, Canada
Renato Lo Cigno University of Trento, Italy
Marco Ajmone Marsan Politecnico di Torino, Italy
Sándor Molnár Budapest University of Technology and Economics, Hungary
Sumit Roy University of Washington, US
Zhengguo Sheng University of Sussex, UK
Sumei Sun Institute for Infocomm Research, Singapore
Li-Chun Wang National Chiao Tung University, Taiwan
Moshe Zukerman City University of Hong Kong, China

Area 4: Next Generation and Future Internet Architectures

Hans van den Berg TNO, Delft, the Netherlands
Roberto Bruschi University of Genoa, Italy
Pedro Casas AIT Vienna, Austria
Ignacio Castro Queen Mary, University of London, UK
Ciprian Dobre University Politehnica of Bucharest, Romania
Christian E. Rothenberg University of Campinas, Brazil
Serge Fdida Université Pierre et Marie Curie (UPMC), France
Oliver Hohlfeld RWTH Aachen, Germany
Hirotada Honda NTT Network Technology Laboratories, Japan
David Irwin University of Massachusetts Amherst, US
Wolfgang Kellerer TU München, Germany
### Area 5: Network and Traffic Management

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samuli Aalto</td>
<td>Aalto University, Finland</td>
</tr>
<tr>
<td>Paul Barford</td>
<td>University of Wisconsin, US</td>
</tr>
<tr>
<td>Giuseppe Bianchi</td>
<td>University of Rome “Tor Vergata”, Italy</td>
</tr>
<tr>
<td>Thomas Bonald</td>
<td>Telecom ParisTech, France</td>
</tr>
<tr>
<td>Sem Borst</td>
<td>Alcatel-Lucent, Bell Labs, US</td>
</tr>
<tr>
<td>Minghua Chen</td>
<td>Chinese University of Hong Kong, China</td>
</tr>
<tr>
<td>Jon Crowcroft</td>
<td>University of Cambridge, UK</td>
</tr>
<tr>
<td>Anja Feldmann</td>
<td>TU Berlin, Germany</td>
</tr>
<tr>
<td>Richard Gibbens</td>
<td>University of Cambridge, UK</td>
</tr>
<tr>
<td>Benny Van Houdt</td>
<td>University of Antwerp, Belgium</td>
</tr>
<tr>
<td>Ramin Khalili</td>
<td>HUAWEI, Germany</td>
</tr>
<tr>
<td>T.V. Lakshman</td>
<td>Bell Labs - Alcatel-Lucent, US</td>
</tr>
<tr>
<td>Patrick Loiseau</td>
<td>Eurocom, France</td>
</tr>
<tr>
<td>Ravi Mazumdar</td>
<td>University of Waterloo, Canada</td>
</tr>
<tr>
<td>Aiko Pras</td>
<td>University of Twente, Netherlands</td>
</tr>
<tr>
<td>James Roberts</td>
<td>IRT SystemX, France</td>
</tr>
<tr>
<td>Costin Raiciu</td>
<td>University Politehnica Bucharest, Romania</td>
</tr>
<tr>
<td>Hiroshi Saito</td>
<td>NTT, Japan</td>
</tr>
<tr>
<td>Miklos Telek</td>
<td>Technical University of Budapest, Hungary</td>
</tr>
<tr>
<td>Steve Uhlig</td>
<td>Queen Mary London, UK</td>
</tr>
<tr>
<td>Xinbing Wang</td>
<td>Shanghai Jiao Tong University, China</td>
</tr>
</tbody>
</table>

### Area 6: Network Planning and Optimization

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fabio D’Andreagiovanni</td>
<td>ZIB Berlin, Germany</td>
</tr>
<tr>
<td>Achim Autenrieth</td>
<td>ADVA Optical Networking, Germany</td>
</tr>
<tr>
<td>Andreas Bley</td>
<td>University of Kassel, Germany</td>
</tr>
<tr>
<td>Wolfgang Bziuk</td>
<td>University of Braunschweig, Germany</td>
</tr>
<tr>
<td>Sammy Chan</td>
<td>City University of Hong Kong, China</td>
</tr>
<tr>
<td>Joachim Charzinski</td>
<td>Hochschule der Medien, Stuttgart, Germany</td>
</tr>
<tr>
<td>Didier Colle</td>
<td>iMinds, Ghent University, Belgium</td>
</tr>
<tr>
<td>Matthias Ermel</td>
<td>Detecon, Dresden, Germany</td>
</tr>
</tbody>
</table>
Markus Fiedler  Blekinge Institute of Technology (BTH), Sweden
Gerhard Hasslinger  Deutsche Telekom, Darmstadt, Germany
Brigitte Jaumard  Concordia University, Canada
Ulrich Killat  TU Hamburg-Harburg, Germany
Arie M.C.A Koster  RWTH Aachen, Germany
Wolfram Lautenschläger  Alcatel Lucent Bell Labs, Stuttgart, Germany
Yiu-Wing Leung  Hong Kong Baptist University, China
Xi Li  NEC Eurolab, Heidelberg, Germany
Rongping Lin  University of Electronic Science and Technology of China, China
Ralf Lehnert  TU Dresden, Germany
Carmen Mas Machuca  TU München, Germany
Deep Medhi  University of Missouri, Kansas City, US
Michal Pioro  Warsaw University of Technology, Poland / Lund University, Sweden

Christian Raack  Atesio, Berlin, Germany
Jacek Rak  Gdansk University of Technology, Poland
Mathias Schweigel  Deteccon, Dresden, Germany
Gangxiang Shen  Soochow University, China
Phuong Nga Tran  TU Hamburg-Harburg, Germany
Anna Tzanakaki  University of Bristol, UK
Eugen Wallmeier  Nokia Networks, Ulm, Germany
Roland Wessäly  Atesio, Berlin, Germany

Area 7: Network Measurements and Analysis

Patrik Arlos  BTH, Sweden
Urtzi Ayesta  CNRS LAAS France
Sem Borst  Alcatel-Lucent, Bell Labs, US
Anna Brunstrom  Karlstad University, Sweden
Giovanna Carofiglio  Cisco Systems, France
Pedro Casas  AIT Vienna, Austria
Danilo Cicalese  Telecom ParisTech, France
Benoit Donnet  Université de Liège, Belgium
Douglas Down  McMaster University, Canada
Anja Feldmann  TU Berlin, Germany
Luigi Fratta  Politecnico di Milano, Italy
Marco Gribaudo  Politecnico di Milano, Italy
Yingdong Lu  IBM Research, US
John C.S. Lui  The Chinese University of Hong Kong, China
Siva Theja Maguluri  IBM Research
Ravi Mazumdar  University of Waterloo, Canada
Rob van der Mei  Centrum voor Wiskunde en Informatica, Netherlands
Michela Meo  Politecnico di Torino, Italy
Marco Milanesio  Eurecom, France
Luca Muscariello  Orange Labs - France Télécom, France
Area 8: Networked applications

Ake Arvidsson  Ericsson, Sweden
Imed Bouazizi  Samsung Research America, US
Sangtae Ha  University of Colorado, US
Poul Heegard  Norwegian University of Science and Technology, Norway
Cheolkon Jung  Xidian University, China
Luntian Mou  Beijing University of Technology, China
Symeon Papavassiliou  National Technical University of Athens, Greece
Peter Počta  University of Zilina, Slovakia
Zhan Ma  Nanjing University, China
Nikolaos Thomos  University of Essex, UK
Christian Timmerer  Alpen-Adria-Universität Klagenfurt, Austria
Martín Varela  Technical Research Centre of Finland, Finland
Dan Wang  Hong Kong Polytechnic University, China
Xin Wang  Huawei Media Lab, US
Liang Zhou  Nanjing University of Post & Communications, China
Rong Zheng  McMaster University, Canada

Demo Session

Andy Bavier  Princeton University, US
Justin Cappos  NYU Polytechnic, US
Chip Elliott  GPO/BBN, US
Deniz Gurkan  University of Houston, US
Marc Körner  
TU Berlin, Germany

Thanasis Korakis  
NYU Polytechnic, US

Robert Krahn  
Communications and Design Group, US

Joe Mambretti  
Northwestern University, US

Sebastian Meier  
University of Stuttgart, Germany

Hausi Muller  
University of Victoria, US

Akihiro Nakao  
University of Tokyo, Japan

Simon Oechsner  
NEC Laboratories Europe, Germany

Max Ott, NICTA  
Australia

Subharthi Paul  
Cisco, US

Rastin Pries  
Nokia Munich, Germany

Glenn Ricart  
US Ignite, US

Niky Riga  
Geni Project Office, US

Christian E. Rothenberg  
University of Campinas, Brazil

Charalampos Rotsos  
University of Lancaster, UK

Paul Ruth  
RENCI, US

Dennis Schwerdel  
University of Kaiserslautern, Germany

James Sterbenz  
University of Kansas, US
Tuesday 13th September, 2016

09:00 – 09:30  Opening by Phuoc Tran-Gia

09:30 – 10:30  Keynote by Nikhil Jain (Qualcomm Technologies)
                *Internet of Everything: Engineering Challenges and Opportunities*

10:30 – 11:00  Coffee break

11:00 – 12:20  Session 1.A: Clouds and Data Centers
                Offering Resilient and Bandwidth Guaranteed Services in Multi-Tenant
                Cloud Networks: Harnessing the Sharing Opportunities by
                Hyame Assem Alameddine; Sara Ayoubi; Chadi Assi
                Dynamic Virtual Network Traffic Engineering with Energy Efficiency in
                Multi-Location Data Center Networks by Mirza Mohd Shahriar
                Maswood; Chris Develder; Edmundo Madeira; Deep Medhi
                An Energy-Aware Embedding Algorithm for Virtual Data Centers by
                Tran Manh Nam; Nguyen Van Huynh; Le Quang Dai; Nguyen Huu
                Thanh

12:20 – 13:45  Lunch

13:45 – 15:15  Session 2: Wireless
                Full Demo 1: Self-Optimization of Software Defined Radios Through
                Evolutionary Algorithms by Zubair Shaik; André Puschmann;
                Andreas Mitschele-Thiel
                Opportunistic Channel Estimation for Implicit 802.11af MU-MIMO by
                Ryan E. Guerra; Narendra Anand; Clayton Shepard;
                Edward W. Knightly
                DiVote: A Distributed Voting Protocol for Mobile Device-to-Device
                Communication by Peter Danielis; Sylvia T. Kouyoumdjieva;
                Gunnar Karlsson

15:15 – 15:45  Coffee break
Tuesday 13th September, 2016 (cont.)

15:45 – 17:05  **Session 3.A: Cellular**  
Joint Optimization of User Association and User Satisfaction in Heterogeneous Cellular Networks by Farah Moety; Mustapha Bouhtou; Taoufik En-Najjary; Ridha Nasri  
Joint Resource Allocation and User Association for Heterogeneous Cloud Radio Access Networks by Ying Loong Lee; Li-Chun Wang; Teong Chee Chuah; Jonathan Loo  
Performance-Oriented Association in Large Cellular Networks with Technology Diversity by Abishek Sankararaman; Jeong-woo Cho; François Baccelli  

**Session 3.B: Video Streaming**  
Bridging the Gap Between QoE and User Engagement in HTTP Video Streaming by Christian Moldovan; Florian Metzger  
A Markov Model for Evaluating Resource Sharing Policies for DASH Assisting Network Elements by Jan Willem Kleinrouweler; Sergio Cabrero; Rob van der Mei; Pablo Cesar  
Mobile Live Video Upstreaming by Philip Lundrigan; Mojgan Khaledi; Makito Kano; Naveen Dasa Subramanyam; Sneha Kasera

18:30 – 20:00  Welcome Reception
Wednesday 14th September, 2016

09:00 – 10:00  Keynote by Wolfgang Kellerer (Technical University of Munich, Germany)
Towards Flexible Networking in Dynamically Changing Environments

10:00 – 10:30  Full Demo 2: PlanetIgnite: A Self-Assembling, Lightweight, Infrastructure-as-a-Service Edge Cloud by Andy Bavier; Rick McGeer; Glenn Ricart

10:30 – 11:00  Coffee break

11:00 – 12:20  Session 4.A: Caching Strategies
Stochastic Dynamic Cache Partitioning for Encrypted Content Delivery by Andrea Araldo; György Dán; Dario Rossi
Access-time Aware Cache Algorithms by Giovanni Neglia; Damiano Carra; Mingdong Feng; Vaishnav Janardhan; Pietro Michiardi; Dimitra Tsigkari
Asymptotically Exact TTL-Approximations of the Cache Replacement Algorithms LRU(m) and l-LRU by Nicolas Gast; Benny Van Houdt

Session 4.B: Performance Analysis
Meeting Soft Deadlines in Single- and Multi-Server Systems by Esa Hyytiä; Rhonda Righter; Jorma Virtamo
Performance Analysis of CoDel and PIE for Saturated TCP Sources by Fabian Schwarzkopf; Sebastian Veith; Michael Menth
Stochastic Upper and Lower Bounds for General Markov Fluids by Florin Ciucu; Felix Poloczek; Jens Schmitt

12:20 – 13:45  Lunch
Wednesday 14th September, 2016 (cont.)

13:45 – 15:15  **Session 5: Demo Session**

*Demo: Resilient Integration of Distributed High-Performance Zones into the BelWue Network Using OpenFlow* by Mark Schmidt; Robert Finze; Daniel Reutter; Michael Menth

*Demonstrating a Personalized Secure-By-Default Bring Your Own Device Solution Based on Software Defined Networking* by Steffen Gebert; Thomas Zinner; Nicholas Gray; Raphael Durner; Claas Lorenz; Stanislav Lange

*Demonstrating Context-Aware Services in the MobilityFirst Future Internet Architecture* by Francesco Bronzino; Dipankar Raychaudhuri; Ivan Seskar

*jLISP: An Open, Modular and Extensible Java-Based LISP Implementation* by Andreas Stockmayer; Mark Schmidt; Michael Menth

*Network as a Service – A Demo on 5G Network Slicing* by Rastin Pries; Hans-Jochen Morper; Nandor Galambosi; Michael Jarschel

*Security of Distributed Container Based Service Clustering with Hypriot Cluster Lab* by Marcel Großmann; Andreas Eiermann

15:15 – 15:45  Coffee break

15:45 – 17:05  **Session 6.A: Softwarization**

*Sector: TCAM Space Aware Routing on SDN* by Sai Qian Zhang; Qi Zhang; Ali Tizghadam; Byungchul Park; Hadi Bannazadeh; Alberto Leon-Garcia; Raouf Boutaba

*Port Based Capacity Extensions (PBCEs): Improving SDNs Flow Table Scalability* by Robert Bauer; Martina Zitterbart

*Performance Modeling of Softwarized Network Functions Using Discrete-Time Analysis* by Steffen Gebert; Thomas Zinner; Stanislav Lange; Christian Schwartz; Phuoc Tran-Gia

**Session 6.B: Information and Social Networks**

*Cache the Queues: Caching and Forwarding in ICN From a Congestion Control Perspective* by Dinh Nguyen; Kohei Sugiyama; Atsushi Tagami

*Optimizing Time to Exhaustion in Service Providers Using Information-Centric Networking* by Ali Shariat; Ali Tizghadam; Alberto Leon-Garcia

*Binary Opinion Dynamics with Biased Agents and Agents with Different Degrees of Stubbornness* by Arpan Mukhopadhyay; Ravi R. Mazumdar; Rahul Roy

19:00 – 23:00  Social Event
Thursday 15th September, 2016

09:00 – 10:00  **Keynote** by Eitan Altman (INRIA Sophia Antipolis, France)
*Dynamic Games for Analyzing Competition in the Internet*

10:00 – 10:30  **Full Demo 3: LiveTalk: A Framework for Collaborative Browser-Based Replicated-Computation Applications** by Matthew Hemmings; Daniel Ingalls; Robert Krahn; Rick McGeer; Glenn Ricart; Marko Röder; Ulrike Stege

10:30 – 11:00  Coffee break

11:00 – 12:20  **Session 7.A: Measurements**
*IntegraTag: a Framework for High-Fidelity Web Client Measurement* by Charles Thomas; Jeff Kline; Paul Barford
*CLUE: Clustering for Mining Web URLs* by Andrea Morichetta; Enrico Bocchi; Hassan Metwalley; Marco Mellia
*Testing for Traffic Differentiation with ChkDiff: The Downstream Case* by Riccardo Ravaioli; Guillaume Urvoy-Keller; Chadi Barakat

**Session 7.B: Caching**
*ModelGraft: Accurate, Scalable, and Flexible Performance Evaluation of General Cache Networks* by Michele Tortelli; Dario Rossi; Emilio Leonardi
*Distributed Algorithms for Content Caching in Mobile Backhaul Networks* by Valentino Pacifici; Sladjana Jošilo; György Dán
*Performance Evaluation for New Web Caching Strategies Combining LRU with Score Based Object Selection* by Gerhard Hasslinger; Kostas Ntougias; Frank Hasslinger; Oliver Hohlfeld

12:20 – 13:45  Lunch

13:45 – 15:15  **Session 8: Virtualization**
*A Power Efficient and Robust Virtual Network Functions Placement Problem* by Antonio Marotta; Andreas J. Kassler
*Elastic Network Service Provisioning with VNF Auctioning* by Mathis Obadia; Mathieu Bouet; Vania Conan; Luigi Iannone; Jean-Louis Rougier

15:15 – 15:45  Closing Session
ITC 28 Sponsors

The International Advisory Committee (IAC) of the ITC has decided to offer a number of travel grants that will be available to support full-time students for attending ITC 28. The IAC financially supports three prestigious awards for ITC 28: Best Paper Award, Best Student Paper Award, Best Demo Award.

Technical Sponsors

ITC 28 is technically co-sponsored by IEEE Communications Society (IEEE ComSoc) and the Information Technology Society within VDE (ITG VDE), and in-cooperation with ACM SIGCOMM.
Offering Resilient and Bandwidth Guaranteed Services in Multi-Tenant Cloud Networks: Harnessing the Sharing Opportunities

Hyame Assem Alameddine, Sara Ayoubi, Chadi Assi
{hy_alame, sa_ayou, assi} @encs.concordia.ca
Concordia University, Montreal, Canada

Abstract—The sharing of computing and networking resources in the cloud is challenged by several obstacles, such as providing bandwidth guarantees for a predictable performance of the hosted applications, as well as maintaining the availability of their services following outages. Therefore, the wide scale adoption of this emerging computing paradigm remains highly dependent on overcoming these challenges. In fact, a lack of bandwidth guarantees extends the completion time for jobs, thus increasing expenses for clients paying for their time of use. In addition, outages in data centers may result in severe revenue losses for both, the cloud operators and their clients alike. To overcome these challenges, cloud operators should be empowered with a strategic design plan that is able to guarantee resilient and predictable performance for hosted applications. Such a plan consists of provisioning additional backup resources (e.g., virtual machines, bandwidth) while ensuring efficient network bandwidth utilization. In this work, we study the design of various facets of such a plan. Namely, we exploit several capability sharing opportunities in multi-tenant cloud networks while offering resilient and bandwidth guaranteed services. In contrast to previous works which target cloud clients satisfaction, we focus on optimizing network bandwidth utilization in order to increase the cloud operators revenues while maintaining such bandwidth allocation transparent to the clients. Through several motivational examples, and numerical studies, we highlight the sharing opportunities and show that they are able to increase cloud operators revenues by an average of 21.4% while providing up to 50% of bandwidth gain in the network.

I. INTRODUCTION

Today, the cloud computing paradigm is gaining increasing interests, within academia and industry alike, for its capability of offering elastic and on-demand computing resources. Owing to its efficient pay-as-you-use charging model, this paradigm enabled cloud providers to lease their computing resources in the form of virtual machines (VMs) with isolated performance on CPU and memory. Thus, cloud computing will substantially reduce enterprises’ IT expenses, which may cost around 10$ million to 25$ million per year to build and manage their dedicated data centers (DCs) [1].

The essence of cloud computing is to enable the sharing of the underlying network resources among the hosted cloud applications, thereby increasing the revenues of cloud operators. Such sharing of resources may however come at the expense of decreasing the popularity of the cloud, since the reputation of cloud computing highly depends on its ability to provide bandwidth guarantees and high service availability to its clients’ applications [2], [3].

In fact, the bandwidth needed for the communication between the VMs of a cloud client (tenant) fluctuates significantly due to the best-effort sharing at the flow level of the Transmission Control Protocol (TCP) used in today’s DCs [2]. Given the bandwidth sharing in the network, competing applications may interfere with each others, resulting in unpredictable performance. Such variable applications performance may result in important revenue losses for tenants due to the uncertainty of their jobs execution times. Another performance attribute which is equally important, is to provide high service availability for the cloud clients, especially those running critical applications, such as banking, retail systems, etc [4]. A single element failure can cause severe revenue losses for those tenants. A recent survey [4] estimated the cost of one hour downtime of such applications to vary between 25,000$ and 150,000$. Thus, cloud providers seek to offer guaranteed performance and high service availability to incentivize enterprises to move their services to the cloud.

Many work in the literature discussed either one of both problems; bandwidth guarantee ([2], [5], [6], [7], [8], [9]) and service survivability ([4], [10], [11], [12], [13]), but only few tackled the trade-off that exists between them ([14], [3], [15], [16], [17]). In fact, it was observed that collocating VMs of a particular application reduces the bandwidth to guarantee for their communication, allowing cloud providers to admit more tenants in the network. However, such allocation degrades the application’s fault tolerance. Thus, cloud providers are interested in realizing the trade-off between providing high survivability while guaranteeing predictable applications performance and efficiently utilizing their DCs network resources. Previous research focused on guaranteeing bandwidth in the cloud along with high survivability to respond to clients requirements and satisfy them. In contrast, we focus in this work on the cloud providers requirements in increasing their revenues while respecting tenants’ demands. This can be accomplished through the exploration of several efficient network resources utilization opportunities. In this work, we consider a single element failure and we guarantee 100% service availability through the provisioning of redundant computing and network resources (backup VMs, backup bandwidth) in the form of a protection plan [14]. Given the provisioned

This work is supported by NSERC Discovery Grant.
We observe that: 1) the provisioned backup bandwidths remain idle until the occurrence of a failure; 2) considering a single element failure, some tenants may not fail at the same time while others may without contending for the same protection resources, upon a failure. Therefore, those tenants may be able to share their backup bandwidths. Such sharing remains transparent to the tenants but beneficial to the cloud providers. We show through illustrative examples that up to 50% bandwidth gain can be attained by exploiting the sharing opportunities. To the best of our knowledge, sharing bandwidth between tenants has not been proposed or discussed in the literature before. Thus, we present and formulate the **Tenants Bandwidth Share Design (TBS-Design)**, a novel approach to decide on the optimal sharing strategies that will achieve the highest bandwidth gain. We show that TBS-Design is an NP-complete problem and we propose an efficient heuristic for solving it. Our numerical results show that bandwidth sharing between tenants increases the tenants’ admission rate in a cloud DC and allows cloud providers to generate more revenues.

The remaining of this paper is organized as follows: In Section II, we provide a literature review of the bandwidth guarantee and high survivability problems in a cloud DC. Section III explains a bandwidth allocation approach in a cloud DC and depicts the protection plan design problem. Section IV presents efficient bandwidth utilization opportunities. Section V provides a definition and a formulation of the TBS-Design problem. Section VI explains the Tenants Bandwidth Share Design-A heuristic (TBSH-Design) that solves the TBS-Design problem. Our numerical evaluation is exposed in Section VII. We conclude in Section VIII.

II. RELATED WORK

Recently, there has been much effort for guaranteeing network performance and providing high survivability for cloud applications. In order to reduce the bandwidth usage in the core of the network, Oktopus [6] developed a VM embedding heuristic that collocates VMs of the same tenant under the smallest sub-tree while guaranteeing bandwidth based on the hose model [6], [5], [2]. The hose model is an abstraction model that allows tenants to express their resources requirements (VMs, bandwidth) to the cloud provider independently from the underlying infrastructure. However, Oktopus overlooked the fact that collocation decreases fault tolerance.

Alternatively, the CloudMirror team [15], [16] proposed the Tenant Application Graph (TAG) that reflects the structure of the tenant’s application, to guarantee bandwidth. Such structure is unknown by the tenant which makes the TAG model not practical. It also considered the Worst Case Survivability (WCS) requested by the client to provide fault tolerance. WCS being the smallest number of VMs that should remain functional during a failure of a single sub-tree, causes service degradation in case of failure. Bodik et al. [3] also employed the WCS as a measure of fault tolerance. They proposed the K-way cut algorithm to provide an initial embedding for the VMs while minimizing the bandwidth at the core of the network. They improved this initial allocation by realizing multiple moves of the embedded VMs in order to achieve fault tolerance. They assumed that a physical server can only host one VM of the same virtual data center (VDC) which extensively spreads the tenant’s VMs and leads to a higher bandwidth usage.

The work in [13] provided 1-redundant and k-redundant approaches to support the failure of critical nodes. In order to minimize the incurred bandwidth cost, they implement bandwidth sharing techniques known as cross-sharing and backup-sharing. To reduce the idle backup bandwidth, the study in [11] provided two heuristics; the first one solves the virtual node embedding and the link embedding problems separately. It chooses the virtual node embedding solution that minimizes the reserved backup bandwidth. The other heuristic solves both problems jointly by adopting a link packing approach. Both work [13], [11] guarantees VM to VM bandwidth by supposing that a dedicated link exists between each pair of communicating VMs. This assumption is unrealistic because links are shared among multiple VMs. In addition, VMs communication dependencies change over time. The pipe model [9], [16] approach refers to such assumption.

The work in [17] used the anycast routing principle to provide resilience against failure while reducing the backup footprint. Anycast routing is based on choosing, out of a set of candidate destinations, a destination for a given service request originating from a known source node. They considered backup path sharing upon a link failure but did not examine bandwidth sharing between tenants upon any node failure.

Our work differs from those in the literature as we do not focus on solving the bandwidth guarantee and the survivability problems. We rather use the solutions provided for those to further decrease the backup footprint through the sharing of backup bandwidth between tenants. Our work can be addressed with any of the above techniques.

III. PRELIMINARIES

A. Bandwidth Provisioning in cloud DCs

To provision bandwidth for tenants, a cloud provider requires the knowledge of their requirements (in terms of VMs and bandwidth required for their communication). In addition, tenants need a simple and intuitive interface to express these requirements independently of the underlying physical infrastructure of cloud DCs. Such interface is known as “Abstraction Model”. In fact, many abstraction models were discussed in the literature. The pipe model [9], [2] guarantees host-to-host connectivity; however, it is not practical because it assumes the knowledge of the communication matrix between VMs which is hard to determine or to estimate by the tenant. The hose model [9], [2], [6] guarantees the minimum bandwidth required by each VM. The tenant application graph (TAG) proposed by [15], [16] relies on the tenant’s knowledge of the application structure in order to determine the bandwidth to guarantee. Without loss of generality, we use in this work the hose model to guarantee bandwidth between VMs due to its simplicity in expressing the tenant’s requirements.

Consider a tenant request $<N, B>$ of $N$ primary VMs and $B$ bandwidth to be guaranteed for the communication between
those VMs. The hose model interconnects the $N$ VMs to a central switch of $N+B$ bandwidth (Fig.1(a)). This ensures $N+B$ bandwidth as a maximum communication rate between those VMs. Because multiple VMs are likely to communicate at the same time with a single destination that can only receive data at rate $B$, the hose model provisions the minimum bandwidth needed by each VM.

For instance, consider a tenant request $<6, B>$ of 6 primary VMs, and $B$, the bandwidth to be guaranteed for the communication between those VMs. Such request is embedded as shown in Fig.1(b). We observe that the link of interest (thick line in Fig 1(b)) divides the network into 2 component $C1$ of $m = 5$ VMs and $C2$ of $N-m = 6-5 = 1$ VM. Hence, the bandwidth to be guaranteed on this link, based on the hose model, is $\min(m, N-m) * B$. We refer to the hose interconnecting the primary VMs as the pre-failure hose.

### B. Protection Plan Design

Guaranteeing bandwidth for a tenant in a cloud DC, depends on the number and the placement of VMs provisioned for the tenant. In fact, today, cloud DCs are prone to several failures causing the disruption of one or more cloud clients. Thus, tenants demand a certain level of service availability. In order to respond to those availability requirements, a cloud provider should devise a protection plan for its clients through the provisioning of backup resources. Backup resources correspond to the backup VMs which become active only upon a failure, and the backup network bandwidth used by the post-failure hose. A post-failure hose is the interconnection between all VMs (primary and backup) of the tenant, that are operational following any failure affecting the tenant.

Designing a protection plan for a hosted tenant is a hard problem that involves the following steps: 1) identifying the required number of backup VMs to provision, 2) deciding on their placement, 3) determining the primary-to-backup VMs correspondence; that is, which backup VMs will protect which primary VMs upon a failure, 4) establishing the backup bandwidth to reserve. Depending on the approaches used to perform each of the previous steps, multiple protection plans can be designed.

In [14], we designed a protection plan that aimed at providing 100% service availability upon a single node failure. In order to ensure efficient utilization of resources (VMs, bandwidth) while providing 100% service continuity, we provisioned a number of backup VMs equal to the maximum number of primary VMs of a tenant hosted on the same physical server. Such number represents the minimum backup VMs footprint required to protect all the tenant’s primary VMs. Embedding those backup VMs while collocating them with the primary ones reduce the backup bandwidth footprint [6]. Such footprint is also affected by the primary-to-backup VMs correspondence plan. The best correspondence is the one that minimizes the backup bandwidth to reserve. It can be determined by the protection plan design (PPD) model discussed in [14]. Now, considering the knowledge of the primary and backup VMs embedding, in addition to the primary-to-backup VMs correspondence, determining the backup bandwidth to reserve on a link can be accomplished by evaluating the backup bandwidth to provision upon the sequential failure of each physical server hosting primary VMs of the specified tenant. Let $l$ denote a link in the network and $b_l$ be the backup bandwidth that must be provisioned on this link. $b_l = \max(b_l')$; where $b_l'$ is the backup bandwidth required on link $l$ to assume the failure of the VMs hosted on server $S_i$. $b_l'$ is determined according to the correspondence between primary VMs and backup VMs. The total bandwidth to reserve on link $l$ becomes: $b_l = b_l + b_l'$, where $b_l'$ refers to the bandwidth required on link $l$ for the pre-failure hose.

To illustrate the backup bandwidth provisioning process, we consider the primary and backup VMs embedding of a tenant $<6, B>$ as shown in Fig.2(a). This tenant requires 6 primary VMs, hosted on servers $S1, S2$ and $S3$. The total bandwidth to provision for the communication between those primary VMs is 8$B$ as depicted by the specified pre-failure hose on that same figure. We determine a protection plan for this tenant through the provisioning of 4 backup VMs (Fig.2(a)) and we define the primary-to-backup VMs correspondence as follows: the backup VMs hosted on server $S2$ and $S3$ protect the primary VMs embedded on $S1$, backup VMs hosted on $S1$ and $S3$ take care of the primary VMs hosted on $S2$. Similarly, the primary VMs hosted on $S3$ is protected by the backup VM embedded on $S1$. Hence, by considering a single node failure of the physical servers $S1$ (Fig.2(b)), $S2$ (Fig.2(c)) and $S3$ (Fig.2(d)) and determining the backup bandwidth needed upon each failure, we can determine the backup bandwidth to be reserved on each link as the maximum backup bandwidth of all those provisioned by all the defined post-failure hoses. It can be easily verified that this backup bandwidth depicted in Fig.2(e), is sufficient to ensure service continuity upon any single node failure.

### IV. EFFICIENT BANDWIDTH UTILIZATION OPPORTUNITIES

A good protection plan design entails an effective network utilization. In the following, we uncover several strategies to better make efficient use of the network resources.

#### A. Bandwidth reuse

Upon any failure of a physical server affecting a tenant, the primary bandwidth reserved for the communication of the primary VMs (hosted on the failed server) is released, and thus, it can be reused by the post-failure hose of the same tenant. Consequently, instead of reserving the sum of primary and backup bandwidths on each link ($b_l = b_l + b_l'$) (Fig.2(f)), one can provision the maximum of both as shown in Fig.2(g). By considering such bandwidth reuse, we can save
7B by reserving 11B (Fig. 2(g)) instead of 18B (Fig. 2(f)) (39% bandwidth saving) [14].

B. Bandwidth sharing between multiple tenants

Since backup bandwidth is only used following a failure, it can be shared between multiple tenants that will not require it simultaneously. In fact, upon considering a single node failure, we can identify two cases in which tenants can share their backup bandwidths:

1-Non concurrent failure of tenants

By considering a single node failure, tenants that do not have primary VMs hosted on the same physical servers will not be vulnerable to a simultaneous service disruption. Hence, they can share their protection bandwidth on the common links along their routes (on their protection plans).

In the example of Fig. 3, we depict the embedding of 2 tenants: tenant 1 <5, B1>, tenant 2 <3, B2>. Since the primary VMs of both tenants are hosted on different physical servers, any single node failure will result in the service disruption of only one of the tenants at a time. This suggests that the backup VMs of tenant 1 and tenant 2 will not be used at the same time. Thus, those tenants can share the backup bandwidth that is needed for their communication. Such bandwidth sharing requires the reservation of the maximum backup bandwidth needed by each of them on their shared links (thick lines in Fig. 3) (Fig. 3(b)). By comparing Fig. 3(a) and (b), one can notice the importance of bandwidth sharing between tenants, which results in the saving of 3B2 (when B1=B2=B, a saving of 12.5% is obtained). A key observation is that sharing between tenants is only possible on those links where no bandwidth reuse (Section IV-A) of the same tenant is considered between its primary and backup bandwidth. In fact, even though the dashed links in Fig. 3 are common for both tenants, no bandwidth sharing is possible on such links because the primary bandwidth of each tenant is reused by its post-failure hose on those links. Thus, we make the following observation:

Observation 1. Bandwidth sharing on a link l is allowed between tenants whose protection plans do not reuse their primary bandwidths on l.

2-Simultaneous failure of tenants

While in the previous example (Fig. 3) we have shown that services that do not fail simultaneously can share their backup bandwidths, we illustrate in Fig. 4 the cases where tenants who are vulnerable to a simultaneous failure may also share their backup bandwidths on the same links traversed by their corresponding post-failure hoses. We consider a network of two tenants: tenant 1 of <3, B1> and tenant 2 <2, B2> embedded with their backup VMs as presented in Fig. 4(a). Tenant 1 primary VMs hosted on S1 are protected by its backup VMs hosted on S2 and S4 while the primary VM hosted on S4 is protected by its backup VM embedded on S6. Both tenants have primary VMs hosted on server S4, thus they can fail simultaneously if S4 fails.

In order to determine the bandwidth which needs to be reserved for each of the two tenants, we consider the failure of each of the servers S1, S3 and S4 hosting the primary VMs of both tenants. When S1 fails (Fig. 4(b)), only tenant 1 fails, requiring bandwidth B1 on links l1, l2 and l3. When server S3...
fails (Fig. 4(c)), only tenant 2 fails, demanding bandwidth $B_2$ on links $l_1$, $l_2$ and $l_3$. However, if we consider the failure of S4 (Fig. 4(d)), both tenants fail since the two of them have primary VMs hosted on this server. Tenant 1 demands bandwidth $B_1$ to be reserved on links $l_1$ and $l_2$ for its service restoration, while tenant 2 requires bandwidth $B_2$ to be reserved on link $l_3$ (in addition to those already reserved on links $l_1$ and $l_2$). Since tenant 1 and tenant 2 require backup bandwidth to be reserved simultaneously on links $l_1$ and $l_2$, upon the failure of S4, they cannot share their backup bandwidth on those links. Thus, we make the following observation:

**Observation 2.** Two tenants having primary VMs hosted on the same server $S$ and their post-failure hoses go through the same link $l$, upon the failure of $S$, they cannot share their backup bandwidth on $l$.

Now, following any failure on any server hosting primary VMs of tenant 1 and tenant 2 ($S_1$, $S_3$ or $S_4$), one of the two tenants requires bandwidth on link $l_3$ at a time. Hence, both tenants can share bandwidth on this link. Thus, instead of reserving $B_1 + B_2$ on $l_3$, we can reserve $\max(B_1, B_2)$. In Fig. 4(e), we represent the total primary and backup bandwidth ($10B_1 + 7B_2$) needed for both tenants while considering bandwidth reuse on the dashed links. As mentioned previously, no bandwidth sharing is possible on links where bandwidth reuse is considered. Fig. 4(f) depicts that $10B_1 + 6B_2$ is to be reserved for the communication of tenant 1 and tenant 2 while considering bandwidth reuse and bandwidth sharing between them, saving one $B_2$ through sharing (when $B_1 = B_2 = B$, a saving of 6% is obtained).

**V. TENANTS BANDWIDTH SHARE DESIGN (TBS-DESIGN)**

Given a substrate network, a set of hosted and protected tenants, we seek an optimal use of the network capacity by reducing the amount of the overall reserved network bandwidth. This can be accomplished either through reusing the primary bandwidth as backup bandwidth upon a failure (Section IV-A) or by sharing the backup bandwidth between multiple tenants (Section IV-B). In this Section, we explore the bandwidth sharing problem. We start by providing a definition and a formulation of the optimal Tenants Bandwidth Share Design (TBS-DESIGN) problem.

**A. Problem definition**

Given that: 1) network links have limited capacity and are shared between multiple tenants; 2) tenants require a guaranteed and predictable performance through fixed dedicated network bandwidth; 3) cloud providers are interested in serving the largest number of clients to generate more revenue; we seek to explore the opportunities for sharing bandwidth between tenants while meeting their network requirements.

Based on the motivational examples presented in Section IV-B, we know that two tenants are able to share their backup bandwidth on a certain link if they do not use it simultaneously to route their backup traffic upon any single server failure. In this case, only the maximum backup bandwidth required by these two tenants can be reserved instead of their sum. Clearly, the best sharing approach that can be achieved on any given link $l$ is to reserve the maximum backup bandwidth required by any tenant whose backup traffic is routed through $l$ (all tenants using $l$ are able to share their backup bandwidth on $l$); that is: $b_l = \max\{b_1, b_2, b_3, ..., b_n\}$ where $b_1, b_2, b_3, ..., b_n$ are the backup bandwidths required by tenants $t_1, t_2, t_3, ..., t_n$ on $l$ respectively. In contrast, the worst case scenario is when all the tenants whose backup paths traverse $l$ fail at the same time, and thus, are not able to share their backup bandwidth on $l$ (Observation 2). Hence, the backup bandwidth to be reserved on $l$ becomes equal to the sum of the backup bandwidths needed by each of the tenants: $b_l = b_1 + b_2 + b_3 + ... + b_n$.

In fact, some of the tenants may be able to share their backup bandwidth on $l$, when others may not. Thus, we can group those tenants into several subsets which we define as independent sharing sets. An independent sharing set is a subset of one or more tenants $t \in T$ that may share their backup bandwidths on $l$ (an independent set is a subset of nodes of a graph $G$, such that no two of them are adjacent
We denote $T$ as the set of tenants requiring backup bandwidth on $l$ and not having any bandwidth reuse on this link (Observation 1). Note that each tenant $t \in T$ can only be an element of exactly one sharing set of $l$. Accordingly, we define for each link $l$ in a network the set $S_l$ of sharing sets $s_l$ such that $s_l \cap s_{l'} = \emptyset$. There exists multiple combinations of sharing sets for a single link $l$. Consider the example where a tenant $t_1$ can share its backup bandwidth with $t_2$ and $t_3$; however, $t_2$ and $t_3$ cannot share their bandwidth on $l$, consequently, they cannot be in the same sharing set. Thus, we can form 3 combinations of sharing sets:

1. $s_1 = \{t_1\}; s_2 = \{t_2\}; s_3 = \{t_3\}$
2. $s_1 = \{t_1, t_2\}; s_2 = \{t_3\}$
3. $s_1 = \{t_1, t_3\}; s_2 = \{t_2\}$

Since our objective is to save bandwidth on $l$, the combination of sets that yield the best sharing is the one that minimizes the total backup bandwidth $b_l$ to reserve on $l$, $b_l$ being equal to the sum of backup bandwidth $(b_l')$ to be reserved for each sharing set $s_l$:

$$b_l = \sum_{i=1}^{\infty} b_l'$$

where $b_l' = \max \{b_i\}$ ($b_l'$ being the backup bandwidth required by each tenant $t$ in $s_l$ on $l$).

Therefore, the TBS-Design problem consists of determining for each link in a network, the independent sharing sets that maximize its saved bandwidth. Thus, we provide the following definition for the problem:

**Definition 1.** Given a set of tenants $(t \in T)$, each requiring a backup bandwidth $b_l$ on a link $l$, find the independent sharing sets combination that minimize the bandwidth to reserve on $l$.

**Theorem 1.** The optimal TBS-Design problem is NP-complete.

**Proof.** We prove that the TBS-Design problem is NP-Complete by a reduction from the graph coloring problem, known as NP-Complete. For completeness, we present a formal definition of the graph coloring decision problem.

**Definition 2.** “Let $G = (V,E)$ be an undirected graph. Is there a $k$-coloring of $V$, such that no two adjacent vertices have the same color?” [19].

Given a substrate link $l$, and a set $T$ of tenants requiring backup bandwidth on $l$ and not incurring any bandwidth reuse on it (observation 1); we construct a conflict graph $G_p = (V_p, E_p)$, where each vertex $v_i \in V_p$ corresponds to a tenant $t_i \in T$ ($|V_p| = |T|$). $E_p$ is the set of edges in the conflict graph, where an edge $e$ is added between two vertices $v_i, v_j \in V_p$ whose corresponding tenants, $t_i, t_j \in T$, can not share their backup bandwidth $l$ (observation 2). Subsequently, we can reformulate the TBS-Design decision problem as follows:

“Given a link $l$, a set of tenants, and a conflict graph $G_p = (V_p, E_p)$ where every vertex $v_i \in V_p$ corresponds to a tenant $t_i \in T$, and every edge $e \in E_p$ denotes that a pair of tenants cannot share their bandwidth on $l$; Is there a partitioning of $V_p$ into $k$ independent sets?”

First, we show that the TBS-Design problem is in the NP-Class for the given graph $G_p(V_p, E_p)$. We consider a partitioning of $V_p$ into $k$ independent sets. One can verify, in polynomial time, that if $v_i \in V_p$; $v_i$ belongs to exactly one independent set $s_i \in S$, i.e., $s_i \in S^l, s_i \neq s_j$.

Next, we show that the graph coloring problem is polynomial-time reducible to the TBS-Design problem, which proves that the TBS-Design problem is NP-Hard. Consider an instance $(G = (V,E), k)$ of the graph coloring problem where $V$ is the set of vertices and $E$ is a set of edges. $k$ is the number of colors used to color the graph $G$. We transform $G$ into an instance of the TBS-Design problem $(l, T, G_p = (V_p, E_p), w)$, where $V_p = V$, $E_p = E$ and $k = w$. Further we restrict our TBS-Design problem by considering that all the tenants in $T$ require a uniform backup bandwidth on $l$. Now, we show that there exists a $k$-coloring of $V_p$ in $G_p$, if and only if there exists $w$ independent sets of $V_p$.

Suppose that there exists a $k$-coloring of $V_p$ in $G_p$ such that no two adjacent nodes in $V_p$ have the same color, then each color corresponds to an independent set $s$ of vertices in $G_p$, that are not connected by any edge in $E_p$. Thus, there exists $k$ independent sets of $V_p$. Conversely, if the TBS-Design problem $(l, T, G_p = (V_p, E_p), w)$ has a solution, which yield partitioning $V_p$ into $w$ independent sets, such that in each independent set, there exists no edge between a pair of vertices; it follows that each independent set corresponds to a color in $G$. Thus, we obtain a $w$-coloring of $V$ in $G$. Indeed, if any adjacent vertices in $G$ were associated with the same color, it means that this pair have an edge between them in $G_p$, which contradicts the fact that they belong to the same independent set in $G_p$. This completes the proof of the reduction. It follows that the restricted TBS-Design problem is NP-Complete. Further, the problem is trivially as hard when all the tenants in $T$ require heterogeneous backup bandwidth.

Note that, the graph coloring problem only solves the restricted version of the TBS-Design problem where all the tenants $t_1, t_2, ..., t_n \in T$ require a uniform backup bandwidth $b_l = B$ on $l$. In this case, defining the MINIMUM number $w$ of independent sharing sets will solve our TBS-Design problem. This is true, because $b_l = \sum_{i=1}^{w} b_l' = \sum_{i=1}^{w} \max \{b_i\} = \sum_{i=1}^{w} B = w \times B$ where $w$ is the number of independent sharing sets and $b_l, b_l', b_i$ are defined earlier.

**B. Problem formulation**

In this section, we provide a mathematical formulation of the TBS-Design problem. This work targets a single path tree topology, however our work can be easily extended to handle any other network topologies if the needed inputs are known and valid. Let $G(V,E)$ be the constructed auxiliary graph as described in section V-A. The TBS-Design problem can be formulated as follows:

**Parameters**

$q_{tt'} \in \{0,1\}$: indicates whether tenants $t$ and $t'$ can not share their backup bandwidth on the link $l$ (1) or can (0).

$b_i$: defines the backup bandwidth required for tenant $t$ on $l$.

**Decision Variables**

$y_i \in \{0,1\}$: specifies if tenant $t$ is part of the sharing set $i$ (1) or not (0).

$b_l$: the backup bandwidth that needs to be reserved for the sharing set $i$.

$M$: set of sharing sets, to be defined. In the worst case, non
of the tenant will be able to share its bandwidth, thus, \( m = |M| = |V| \) sharing sets.

**Model**

Minimize \( \sum_{i=1}^{m} \hat{b}_i \) \hspace{1cm} (1)

\[
y_i^t + y_i^r \leq 1 \quad \forall \in M; \forall (l',r) \in E: q_{l'r} = 1 \hspace{1cm} (2)
\]

\[
\sum_{i=1}^{m} y_i^r = 1 \quad \forall \in V \hspace{1cm} (3)
\]

\[
\hat{b}_i = \max \{ b_i y_i^t \} \quad \forall \in M; \forall \in V \hspace{1cm} (4)
\]

The objective function (Eq.1) consists of defining independent sharing sets that minimize the backup bandwidth to reserve on a given link \( l \). Constraint 2 specifies that two conflicting tenants that can not share their backup bandwidths, only one of them can be part of a specified set \( i \). Constraint 3 depicts that a tenant should be part of one and only one set. Constraint 4 determines the bandwidth which need to be reserved for the tenants in a set \( i \) which is equal to the maximum backup bandwidth of all the tenants in the set. This constraint can be converted to a linear programming format as follows:

\[
\hat{b}_i \geq b_i y_i^r \quad \forall \in M; \forall \in V \hspace{1cm} (5)
\]

Since the objective is to minimize \( \hat{b}_i \), the model will set \( \hat{b}_i \) to the maximum backup bandwidth required by all the tenants in the set (max \( \{ b_i y_i^t \} \)).

The TBS-Design model is a Mixed integer linear program which is complex to solve. Next, we present, the **Tenants Bandwidth Share Design - A Heuristic (TBSH-Design)** to solve it.

**VI. TENANTS BANDWIDTH SHARE DESIGN - A HEURISTIC (TBSH-DESIGN)**

The tenants bandwidth share design problem consists of determining the independent sharing sets of tenants for every link in the network. In addition, it specifies the bandwidth to be reserved on each link for the defined sets. To solve this problem we developed the **Tenants Bandwidth Share Design heuristic (TBSH-Design)** depicted in Algorithm 1.

Our methodology for solving the TBS-Design problem for each link \( l \) in the network consists of selecting the tenants that are eligible to share their backup bandwidths on \( l \). An eligible tenant is a tenant that is not reusing his primary bandwidth as backup bandwidth on \( l \) (Observation 1). Because the bandwidth to reserve for a sharing set is equal to the maximum backup bandwidth of all the tenants in the set (Section V-A), trying to place the eligible tenants who request the biggest amount of backup bandwidth on \( l \) in the same set, may reduce the total bandwidth to reserve on it. Motivated by this intuition, our approach sorts the eligible tenants in decreasing order of their backup bandwidth requirements on \( l \). It also stores them in an array, that we denote \( \text{sortedRequests} \).

Given the \( \text{sortedRequests} \) array, the TBSH-Design recursively builds the sharing sets of \( l \). The heuristic starts by creating a sharing set \( s \) for \( l \) (line 4) and adds to it the first tenant in the \( \text{sortedRequests} \) array (lines 5-6). This tenant will be the one with the highest backup bandwidth demands on \( l \). Thus, the algorithm sets the bandwidth to reserve for \( s \) equal to the backup bandwidth demands of this tenant (line 7). This latter is then removed from the \( \text{sortedRequests} \) array (line 8). Afterwards, the algorithm loops over the remaining requests in the array (line 9), and checks if each one of them is able to share its backup bandwidth with all the requests that belong to \( s \), through the call of \( \text{canShareBw}(s) \) function (line 11). The \( \text{canShareBw}(s) \) function verifies that the post-failure hoses of the request of interest do not go through \( l \) upon the service disruption of any of the requests in \( s \) (Observation 2). If the request of interest can share its backup bandwidth on \( l \) with all the requests in \( s \), it will be added to the set \( s \) (line 12) and removed from the \( \text{sortedRequests} \) array (line 13). After evaluating all the requests in the \( \text{sortedRequests} \) array and adding those who can share their bandwidth to the set \( s \), the heuristic will try to build a new sharing set with the remaining requests (the requests that are not part of \( s \)). This is performed by calling the TBSH algorithm again and passing to it the updated \( \text{sortedRequests} \) array (line 17). The code will keep on calling the TBSH heuristic until all the eligible requests become part of a sharing set of \( l \). The bandwidth reserved on \( l \) will be updated to consider the bandwidth to reserve for each defined sharing set. If the \( \text{sortedRequests} \) array was of size \( N \), looping over the \( N \) requests will take \( O(n^2) \). In addition, if none of the requests was able to share its bandwidth, the TBSH-Design will be called recursively \( N \) times. Hence, the worst case complexity of the TBSH-Design algorithm is \( O(n^2) \).

**Algorithm 1 TBSH (Array sortedRequests)***

1: Given:
2: \( l \): link on which we are solving the TBS-Design problem
3: 4: \( \text{SharingSet} s = \text{newSharingSet}(l) \);
5: \( s \).requests.add(0)
6: 7: \( s \).backupBandwidth = \( s \).backupBandwidth(0)
8: \( \text{sortedRequests} \).remove(0)
9: 10: for \( i = 0; i < \text{sortedRequests}.size(); i++ \) do
11: \( r = \text{sortedRequests}.get(i) \);
12: if \( r \).canShareBw(0) then
13: \( s \).requests.add(0)
14: end if
15: end for
16: if \( \text{sortedRequests}.size() > 0 \) then
17: \( \text{TBSH} (\text{sortedRequests}) \);
18: end if

**VII. NUMERICAL RESULTS**

We carry out an extensive empirical study to evaluate the performance of our TBSH-Design against our TBS-Design model and a no bandwidth share approach. The no bandwidth share method consists of embedding the requests and protecting them without performing any bandwidth share between the admitted tenants. The three methods previously mentioned use the same primary embedding algorithm which consists of collocating the VMs of a request in the smallest sub-tree
to reduce the bandwidth use in the network ([6], [15], [16]). They also provide 100% reliability for the admitted requests by designing their protection plan based on the approach specified in Section III-B.

We simulate a three-level fat tree topology with no path diversity. We assume that all VMs are of homogeneous CPU and memory capacity. Additionally, we consider that each physical server has a capacity of \( \theta \) VM slots. We perform our simulations over a network of 128 physical servers with \( \theta = 6 \). We set the capacity of the links interconnecting the switches to 10 Gbps. We randomly generate sets of 100 requests each, of varying VMs ([5-25] VMs) and network ([100-500] Mbps) requirements. All our numerical evaluations are conducted using Cplex version 12.4 to solve the optimization problem on an Intel core i7-4790 CPU at 3.60 GHZ with 16 GB RAM. We use two different approaches in our tests:

**A. Offline Approach**

In order to evaluate the performance and the scalability of the TBSH-Design vs the TBS-Design, we run our tests using an offline approach where tenants are known a priori and do not leave the network once embedded. This increases the sharing probability, since a bigger number of requests are occupying the network.

1-Execution time and optimality gap: We consider a single link and vary the number of requests using it. We randomly generate the backup bandwidth to reserve for each request on this link to be between [100-500] Mbps. In addition, we build a 2-dimensions array which randomly specifies if each pair of the generated requests can/can not share their backup bandwidth on this link. We use the TBSH-Design and the TBS-Design to share bandwidth between those requests. The results depicted in Table I clearly prove that the TBSH-Design is much more scalable than the TBS-Design model. TBSH-Design is able to share bandwidth between 5000 requests in only 65 ms, while the TBS-Design runs for approximately 3 hours to decide on the sharing sets for 20 requests only. The optimality gap between the TBSH-Design and the TBS-Design is 1.8% for 10 requests and 13.6% for 20 requests. It is clear that, this optimality gap increases with the increase of the number of requests. Further investigation is needed for improving the performance of the TBSH-Design.

<table>
<thead>
<tr>
<th>Nb. of requests</th>
<th>TBSH-Design</th>
<th>TBS-Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
<td>501</td>
</tr>
<tr>
<td>15</td>
<td>94</td>
<td>949</td>
</tr>
<tr>
<td>20</td>
<td>147</td>
<td>1479</td>
</tr>
<tr>
<td>40</td>
<td>400</td>
<td>4000</td>
</tr>
</tbody>
</table>

**Fig. 5: Average bandwidth gain per link type.**

**B. Online Approach**

We conduct our tests using an online approach. We consider a Poisson traffic arrival of requests. We alter the load by varying the arrival rate (\( \lambda \)) while fixing the average service time (\( \mu \)) of the requests (\( load = \lambda / \mu \)). Our numerical results are depicted in Fig. 6.

1- Bandwidth Gain Over Time: Sharing bandwidth between the admitted tenants increases the amount of available network resources and can provide up to 50% of bandwidth gain as depicted by Fig.6(a). This figure presents the bandwidth gain (Eq.(6)) over time obtained by the TBSH-Design and the TBS-Design for a single run over a \( load = 6 \).

One can clearly notice that at a certain point in time \( t \), the TBSH-Design may provide more bandwidth gain than the TBS-Design, and vice versa. This can be explained by the fact that the sharing sets built by each of those two methods at a time \( t' < t \) are different. Thus, the bandwidth gain provided by each of them may be spread over different links. This affects the embedding of every new request which arrives and gets admitted at time \( t \). Thus, the bandwidth gain provided at time \( t \) by each of the two methods is different.

2- Rejection Rate: The rejection rate is an important metric to look at, especially that cloud providers are interested in admitting more tenants in their DCs. Here, we compare the rejection rate with the no bandwidth share method against the TBSH-Design and the TBS-Design. Our results presented
Fig. 6: Comparative analysis between the TBSH-Design, TBS-Design and no bandwidth share method.

in Fig.6(b) are averaged over 5 runs of 100 requests each, for every load and presented with 95% confidence interval. The rejection rate is calculated as the ratio of the number of rejected requests ($\text{RejectedNb}$) over the total number of requests ($\text{TotalNb}$) (Eq.(7)).

$$\text{RejectionRate} = \frac{\text{RejectedNb}}{\text{TotalNb}} \times 100 \quad (7)$$

Since sharing bandwidth between tenants increases the available bandwidth in the network, one can directly guess that the rejection rate should decrease. This is true, given that the requests that were rejected because of lack of bandwidth in the network using the no bandwidth share approach, are more probable to get admitted using the TBSH-Design and the TBS-Design methods. This is clearly depicted in Fig.6(b), which shows that the TBS-Design can decrease the rejection rate by an average of 30.5% over the load, while the rejection rate is decreased by 21.6% using the TBSH-Design. This decrease is calculated in comparison with the no bandwidth share approach results. Alternatively, the average rejection rate gap between TBSH-Design and the TBS-Design is 11%.

3- Revenue Over Time: Admitting more tenants in the network yield an important factor to increase cloud providers’ revenue. Fig.6(c) presents the revenue over time obtained by a single run of 100 requests for a load = 6. The revenue is calculated as shown in Eq.(8) where $m$ is the number of requests ($<N_i,B_i>$) admitted in the network, $c_{vm}$ and $c_{bw}$ are the costs of leasing one unit of VM and one unit of bandwidth respectively. We consider that $c_{vm} > c_{bw}$ in our tests.

$$\text{Revenue} = \sum_{i=1}^{m} N_i c_{vm} + \sum_{i=1}^{m} B_i c_{bw} \quad (8)$$

Fig.6(c) shows that the TBSH-Design can increase the cloud providers revenue by an average of 21.4% over time, while the TBSH-Design gives a similar average increase of returns approximated to 18.97% in comparison with the no bandwidth share approach.

VIII. Conclusion

This paper exploits several bandwidth sharing techniques between tenants, making efficient use of cloud DC networks. Given an embedding and a protection plan design for each tenant in the DC, we formulate the TBS-Design model to solve to optimality the tenants bandwidth share problem. We proved that the optimal TBS-Design problem is NP-complete. Thus, we developed the TBSH-Design, a heuristic, shown to be much more scalable than the TBS-Design model. Through extensive simulations, we confirmed that our bandwidth sharing techniques are able to increase cloud operators’ revenue by an average of 21.4% over time while reducing the rejection rate by an average of 30.5%. Our sharing techniques increase the bandwidth gain in the network up to 50% and can be applied to any network topology. However, we do believe that studying the advantages of these sharing techniques using different type of embedding and protection plan designs is indeed a relevant and interesting problem that we leave for future work.

REFERENCES

Dynamic Virtual Network Traffic Engineering with Energy Efficiency in Multi-Locational Data Center Networks

Mirza Mohd Shahrriar Maswood1++, Chris Develder2, Edmund Madeira3§, Deep Medhi1+§
1University of Missouri–Kansas City, USA, 2Ghent University – iMinds, Belgium,
3University of Campinas, Brazil
Email: mmnt7@mail.umkc.edu, chris.develder@intec.ugent.be, edmundo@ic.unicamp.br, dmedhi@umkc.edu

Abstract—For cloud enterprise customers that require services on demand, data centers must allocate and partition data center resources in a dynamic fashion. We consider the problem in which a request from an enterprise customer is mapped to a virtual network (VN) that is allocated requiring both bandwidth and compute resources by connecting it from an entry point of a data center to one or more servers, such that data center be selected from multiple geographically distributed data centers. We present a dynamic traffic engineering framework, for which we develop an optimization model based on a mixed-integer linear programming (MILP) formulation that a data center operator can use at each review point to optimally assign VN customers. Through a series of studies, we then present results on how different VN customers are treated in terms of request acceptance when each VN class has a different resource requirement. We found that a VN class with a low resource requirement has a low blocking even in heavy traffic, while the VN class with a high resource requirement faces a high service denial. On the other hand, cost for the VN with the highest resource requirement is not always the highest in the heavy traffic because of the significantly high service denial faced by this VN class.

Index Terms—Data Center Networks, Resource Optimization and allocation on-demand, Denial of Service, Energy Efficiency, Virtual Network

I. INTRODUCTION

With the increasing dependency on various services such as e-commerce, electronic libraries, video streaming, and audio-video conferencing, the need for both compute and storage has significantly increased. To cater to these needs, cloud data centers have become a popular platform in recent years. Today, companies such as Amazon, Google, Facebook, and Yahoo! routinely use data centers for storage, web services, and large-scale computations [1], [2], [3]. Because of this increase in the use of data centers, a cost-effective system design for storage and processing data has become a challenging problem. This increasing need for equipment such as routers, switches, and server racks in data centers also incurs significant power consumption that contributes to the operational cost of data centers.

Acknowledgement: * Partially supported by a School of Graduate Studies Research Grant at the University of Missouri–Kansas City. ** Partially supported by National Science Foundation Grant # 1526299. § Partially supported by CNPq (National Counsel of Technological and Scientific Development), Brazil.

There has been significant work so far to improve the capability of data centers by increasing the utilization of the servers and reducing operational cost. However, little research has been conducted on dynamic traffic engineering for handling requests for different customers and how both network resources in the data center and compute resources at the hosts are allocated for such customers. We also observe that most work related to traffic engineering of intra DC networks considers east-west traffic, i.e., the intra-data center traffic between hosts. In our work, we focus instead on enterprise customers’ requests that result in north-south traffic in data centers. We focus on serving different customer groups using virtual networks (VNs) at data centers through dynamic traffic engineering by allocating both network bandwidth and processing resources efficiently, while factoring in energy consumption. We present a dynamic traffic engineering framework where virtual network customers are served at review points. At each review point, we propose to solve a traffic engineering problem for arriving requests from virtual network customers. The requests that are admitted by this process then use resources for a certain duration. An important contribution in this work is that we consider a request to consist of a two-tuple demand, one for data center network bandwidth and the other for the processing demand at the end hosts.

Thus, our work is different from existing work in two distinct ways. First, we consider the north-south traffic environment where each request consists of a two-tuple demand model. Furthermore, we consider this for virtual network customers by taking issues such as power consumption at the hosts into consideration. We present a novel mixed-integer linear programming (MILP) formulation to solve at each review point that minimizes a composite objective from a traffic engineering point of view to satisfy the virtual network customers by using minimum resources from data centers. Our formulation allows for the flexibility that requests arriving at a review point may be allocated to any of the available data centers; for the selected data center, it may use any of the entry points for the north-south traffic at the north end, and any of the hosts available at the south-end.

The second contribution of this work is to present an insight on how different VN customers are affected in terms of resource allocations with north-south traffic in data centers. That is, there are a number of questions to which we seek
to find answers. How are the cost and blocking affected as the request arrival rate from VN customers increases? When the bandwidth demand and the resources per request vary uniformly from an average value, how are the cost and blocking affected compared to when the bandwidth demand and resources for each request were kept fixed? Furthermore, how are different VN classes affected in terms of cost and blocking when each class has a different bandwidth and CPU resource demand? Does the system favor one VN class over another? If so, by how much? Finally, we wish to know how much the power consumption is reduced by our optimization model. By considering a number of cases in a systematic manner, we were able to answer to these questions.

The rest of the paper is organized as follows. In Section II, we present the optimization formulation of the traffic engineering problem to be solved at each review point. In Section III, we present the simulation setup and results of our analysis. The related work is discussed in Section IV. Finally, in Section V, we summarize our concluding remarks and discuss potential future work.

II. MODEL FORMULATION

Our dynamic traffic engineering approach considers new request arrivals at random from customers, for which the resource allocation (both data center network bandwidth and host resources) is done at review point $t \in T$, where $T$ is a discrete temporal window for dynamic traffic engineering consisting of review points. The duration of a new VN request that uses the data center is assumed to be random. Note that since the data center is set up to serve VN customers, at any time instant, there are existing VN tunnels and host resources allocated for prior requests. Thus, any (micro)-workload that needs immediate access to resources, that is, workload that cannot wait until the next review point, is assumed to be served by existing VN channels and host resources assigned to the customers that were set up at earlier review points. Since such immediate workloads are served through existing resources, they are not modeled in our case. In other words, the scope of our work is to consider new requests at review points that are major requests requiring allocation of new bandwidths, virtual network tunnels and new resources.

We optimally solve the resource allocation problem for traffic engineering at each review point $t$. For this, we first present a mixed-integer linear programming (MILP) formulation in which we attempt to accommodate as many requests as possible while minimizing the resources requirement towards satisfying those requests in order to reduce the overall cost. To illustrate our approach, consider the single data center network topology shown in Fig. 1, which depicts just one site of the multi-location data center that our model considers. The entry point in a data center is then the north-end and the serving host is the south-end of the north-south traffic. Our approach assumes that there is a central controller that is responsible for solving the proposed optimization model and setting up the allocations. For instance, this can be accomplished by using a software-defined network (SDN) based approach.

In our model, each request consists of 2-tuple $\langle h, r \rangle$ where $h$ is the bandwidth demand of the request and $r$ is the processing resources required from a serving host. Thus, at a particular review point $t$, if a VN customer $v \in V$ has a request, the request tuple is further represented by $\langle h^v(t), r^v(t) \rangle$, which is to be served by data center $d \in D$. While the bandwidth demand needs to be satisfied by the capacity of the links within the data center $l \in L_d$ from the entry point $i \in I_d$ to a server $j \in J_d$, the processing resources must be satisfied by the servers’ available resources. We assume that there is a given set of paths $P_{ij}^d(t)$ from the entry point $i$ to server $j$, which could be potentially different at each review point $t$.

For energy consumption, we consider that every server can run at a given set of CPU frequencies $f \in F$. At each particular frequency, a server works at a particular processing capacity $a^f_{ij}$. A specific amount of power $b^f_{ij}$ is required to run the server at that frequency. If we run the server at the highest frequency, it offers the highest processing capacity, but consumes the highest amount of power. All notations used in our model are summarized in Table I.

We now present the constraints in our formulation. First, one DC out of the $N$ DCs ($D = \{DC_1, ..., DC_N\}$) is at most selected to meet the request for a VN $v$ at review point $t$:

$$\sum_{d \in D} u^v_d(t) \leq 1, \quad v \in V$$ (1)

The total link bandwidth demand must then be served by the chosen data centers:

$$\sum_{d \in D} s^v_d(t) = h^v(t), \quad v \in V$$ (2)

Once a data center is responsible to fulfill the link bandwidth demand from a VN, then this data center must be the one from which the capacity is allocated:

$$s^v_d(t) \leq h^v(t)u^v_d(t), \quad v \in V, d \in D$$ (3)

The total amount of the link bandwidth demand from a particular VN $v$ that will be served by a particular data center $d$ is the summation of the bandwidth that is allocated from all chosen entry points $i$ to all chosen servers $j$ of data center $d$ at review point $t$:

$$\sum_{i \in (I_d \cup J)} \sum_{j \in J_d} y^v_{ij}(t) = s^v_d(t), \quad v \in V, d \in D$$ (4)
TABLE I
NOTATIONS USED IN FORMULATION

<table>
<thead>
<tr>
<th>Constants/Parameters:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>Set of data centers, $N = #(D)$</td>
</tr>
<tr>
<td>$I_d$</td>
<td>Set of servers in one data center</td>
</tr>
<tr>
<td>$I_p$</td>
<td>Set of entry points in one data center</td>
</tr>
<tr>
<td>$V$</td>
<td>Set of virtual networks</td>
</tr>
<tr>
<td>$F$</td>
<td>Set of frequencies in which a particular server can run</td>
</tr>
<tr>
<td>$L_d$</td>
<td>Set of links in one data center</td>
</tr>
<tr>
<td>$P_{t, ij}^d$</td>
<td>Set of paths from an entry point $i$ to a server $j$ in a data center $d$ for a VN at time $t$</td>
</tr>
<tr>
<td>$M$</td>
<td>A large positive number</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>A very small positive number</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Bandwidth demand for a VN at time $t$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Binary decision variable to satisfy the requests of a VN at time $t$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Binary decision variable to choose a data center $d$ at time $t$</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Binary decision variable to choose the optimum frequency $f$ at time $t$</td>
</tr>
<tr>
<td>$\beta^d$</td>
<td>Normalized cost of a data center $d$ at review point $t$</td>
</tr>
</tbody>
</table>

**Variables:**
- $w_{t, ij}^d$: Binary decision variable to choose a data center $d$ to satisfy a request from a virtual network $v$ at review point $t$.
- $s_{t, ij}^d$: Bandwidth allocation going to data center $d$ for virtual network $v$ at time $t$.
- $y_{t, ij}^d$: Bandwidth allocation for a request from VN $v$ at an entry point $i$ to a server $j$ of data center $d$ at review point $t$.
- $g_{t, ij}^d$: Binary decision variable to select a request to be satisfied from a VN $v$ to an entry point $i$ and served by a server $j$ of data center $d$ at review point $t$.
- $x_{t, ij}^d$: Bandwidth allocation in path $p$, if a request comes to an entry point $i$ of that data center at time $t$ to be served, 0 otherwise.
- $\beta^d$: Weight parameters related to 3 optimization objectives.

Next, we introduce a binary shadow variable $\tilde{y}_{t, ij}^d$ corresponding to $y_{t, ij}^d(t)$ to track one-to-one mapping from a particular entry point $i$ to a particular server $j$ at any review point $t$ by using a large positive number $M$ and a small positive number $\varepsilon$:

$$g_{t, ij}^d(t) \leq M\tilde{y}_{t, ij}^d(t), \quad j \in J_d, i \in I_d, v \in V, d \in D$$  

(5)

$$g_{t, ij}^d(t) \geq \varepsilon\tilde{y}_{t, ij}^d(t), \quad j \in J_d, i \in I_d, v \in V, d \in D$$  

(6)

Here, (5) and (6) together addresses the requirement that $\tilde{y}$ is 1 when the corresponding variable $y$ has a positive flow; otherwise, $\tilde{y}$ as 0 when $y$ is 0.

The bandwidth that is allocated to a particular path from entry point $i$ to server $j$ of a particular data center $d$ is given by using the path flow variables $x_{t, ij}^d$:

$$\sum_{p \in P_{t, ij}^d} x_{t, ij}^d(t) = y_{t, ij}^d(t), \quad j \in J_d, i \in I_d, v \in V, d \in D$$  

(7)

If any bandwidth is allocated on a particular path $p$ to satisfy a portion of the request of bandwidth demand $h^v$ from any VN $v$, then all the links associated with that path $p$ have to carry that portion of demand $h^v$. Therefore, we can determine the link flow on $l$ for tuple $(v, d)$:

$$\sum_{i \in I} \sum_{j \in J} \sum_{p \in P_{t, ij}^d} e_{t, ijlp} x_{t, ij}^d(t) = z_{t, vd}^d(t)$$  

$$d \in D, l \in L_d, v \in V$$  

(8)

while the total amount of bandwidth required in one link $l$ of a data center $d$ to satisfy the requests of all VNs must not exceed the capacity of that link of this data center:

$$\sum_{v \in V} z_{t, vd}^d(t) \leq e_{t, l}^d(t), l \in L_d, d \in D$$  

(9)

Next we address resource allocation of $r_{t}(v)$ to the appropriate tuple $(d, i, j)$, ensuring this in accordance with shadow variable $\tilde{g}$:

$$\sum_{d \in D} \sum_{i \in I_d} \sum_{j \in J_d} g_{t, ij}^d(v) = r_{t}(v), \quad v \in V$$  

(10)

$$g_{t, ij}^d(v) \leq M\tilde{g}_{t, ij}^d(v), \quad j \in J_d, i \in I_d, v \in V, d \in D$$  

(11)

$$g_{t, ij}^d(v) \geq \varepsilon\tilde{g}_{t, ij}^d(v), \quad j \in J_d, i \in I_d, v \in V, d \in D$$  

(12)

$$\sum_{v \in V} \sum_{i \in I_d} g_{t, ij}^d(v) = e_{t, d}^d(v), j \in J_d, d \in D$$  

(13)

In (13), $e_{t, d}^d(v)$ represents the total amount of resources required from a particular server $j$ to satisfy the requests of all VNs that use the server coming through all entry points of a particular data center. The total resources required by a particular server must be less than or equal to the available resources of that particular server of a data center:

$$e_{t, d}^d(v) \leq \sum_{f \in F} a_{t, df}^d w_{t, df}^d(v), j \in J_d, d \in D$$  

(14)

Finally, a particular server $j$ running at a particular frequency $f$ can produce a particular capacity $a_{t, df}^d$. However, a server cannot run at more than one frequency at a time:

$$\sum_{f \in F} w_{t, df}^d(v) \leq 1, j \in J_d, d \in D$$  

(15)

For the goal of the optimization problem, we considered three cost components in the objective function: the network bandwidth cost, the server resource cost, and the data center location cost. These three sources of costs are assigned different weight parameters, $\alpha, \mu, \gamma$, to understand the influence of each term on the overall decision. Briefly, the goal is to accommodate as many requests as possible and this can be accomplished by minimizing the amount of resources used. Furthermore, since resources are of different types, we take an utility function-based approach by assigning weights to
different components that form the objective function. That is, the objective function can be written as:

\[
\begin{align*}
\min_{\alpha} & \sum_{d \in D} \sum_{e \in V} \sum_{v \in L_{d}} z_{e}^{v}(t) + \mu \sum_{d \in D} \sum_{f \in L_{d}} w_{f}^{d} u_{f}^{d}(t) \\
& + \gamma \sum_{d \in D} \beta_{d}^{e} u_{d}^{e}(t)
\end{align*}
\]  

(16)

To summarize, our unified formulation addresses decision choices at three different levels: data center, entry point, and then the destination server. Secondly, we take power consumption into account in determining the right frequency for operating a server. Finally, we consider three cost components in the composite objectives.

### III. Simulation Study Setup and Result Analysis

To conduct our study, we chose the data center topology shown in Fig. 1. We set a maximum of two data centers \((N = 2)\) to be selected. Each data center is considered to be identical in this study; each consisted of \(I_{d} = 4\) entry points and \(J_{d} = 16\) servers and all links inside the data center are set with the same capacity. We set \(P_{r}^{v} = 4\) paths from an entry point to a server among which only one path will be used for a specific request for the duration of this request. Parameter values used for the DCs are summarized in Table II.

We consider \(V = 3\) virtual network customers that generate the requests. Recall that a request is represented by the tuple \((h, r)\). We vary \((h, r)\) for different simulation cases, while the arrival is generated randomly. Specifically, we assume that the request arrivals follow a Poisson process. We varied the arrival rate from 0.2 to 1.0 in increments of 0.2 for each VN customer.

The service duration for the request arrivals is assumed to follow the negative exponential distribution with an average value of 5 time units measured in terms of the number of discrete review points.

To solve the optimization model at each review point \(t\), we use an AMPL/CPLEX (v 12.6.0.0) tool environment. For the experiments we conducted, solving the MILP model using CPLEX took 1 second on an average at each review point. In most cases, the MILP problem was solved optimally. For the instances when it was not solved optimally, the highest optimality gap was found to be 2.73\%.

Through initial experimentation, we first determined the warm-up time for the simulation and then collected the data for a steady-state region after the warm-up time. For each arrival rate, we used 10 seeds and report the results on the average value. We also computed the confidence interval and found the 90\% confidence interval to be approximately 5\% in cost variation for low arrival rates to 2.5\% for high arrival rates. Since our optimization model considers the power consumption factor, we use the power consumption and processing capacity of a particular server that runs at a specific frequency, as shown in Table III.

In Table IV, we summarize the four cases we studied. These studies reflect a number of systematic changes to understand the impact. First, we started with the case of all demands being homogeneous for VN customers, i.e., we set \((h, r) = (10, 1.65)\) (Case-H). In the next case, we assigned the demand to be uniformly chosen at random from the discrete values in the range given by \((h, r) = ([8, 12], [0.55, 2.75])\). For each arrival rate, we used 10 seeds and reported the results on the average value. We also computed the confidence interval and found the 90\% confidence interval to be approximately 5\% in cost variation for low arrival rates to 2.5\% for high arrival rates. Since our optimization model considers the power consumption factor, we use the power consumption and processing capacity of a particular server that runs at a specific frequency, as shown in Table III.

### A. Cost and Blocking

Case-H is the baseline case where all services are homogeneous. The cost and blocking are shown in Fig. 2 and Fig. 3. Not surprisingly, as the arrival rate increases, the cost of the network increases while the blocking also increases.

If we consider the first variation Case-R from Case-H, where the average demand and resource requirements are the same as Case-H except that the value taken by each request is chosen uniformly from a range, we can see that the cost increase is similar between Case-H and Case-R, while Case-R has a higher cost at the lower arrival rates that changes at higher arrival rates. On the other hand, blocking for case-R is noticeably higher than that for Case-H for all arrival rates.

<table>
<thead>
<tr>
<th>TABLE II DC RELATED PARAMETERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of links in each DC</td>
</tr>
<tr>
<td>Capacity of each link</td>
</tr>
<tr>
<td>Number of nodes in each DC</td>
</tr>
<tr>
<td>Number of entry points</td>
</tr>
<tr>
<td>Number of Servers</td>
</tr>
<tr>
<td>Normalized cost of using each DC, (\beta_{d}^{e})</td>
</tr>
</tbody>
</table>
TABLE III
CPU FREQUENCIES, CAPACITIES AND OPERATIONAL COST [5]

<table>
<thead>
<tr>
<th>Frequency Option</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (GHz)</td>
<td>1.4</td>
<td>1.57</td>
<td>1.74</td>
<td>1.91</td>
<td>2.08</td>
<td>2.25</td>
<td>2.42</td>
<td>2.6</td>
</tr>
<tr>
<td>Normalized Capacity</td>
<td>5.385</td>
<td>6.038</td>
<td>6.692</td>
<td>7.346</td>
<td>8</td>
<td>8.645</td>
<td>9.308</td>
<td>1</td>
</tr>
<tr>
<td>Power Consumption (watts)</td>
<td>60</td>
<td>63</td>
<td>66.8</td>
<td>71.3</td>
<td>76.8</td>
<td>83.2</td>
<td>90.7</td>
<td>100</td>
</tr>
</tbody>
</table>

TABLE IV
VALUES OF THE GENERAL PARAMETERS USED FOR THIS RESEARCH IN DIFFERENT CASES.

<table>
<thead>
<tr>
<th>Cases</th>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case-H: Homogenous Bandwidth and CPU Processing Capacity for each request from all 3 VNs</td>
<td>Bandwidth Demand from VN-1, VN-2 and VN-3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>CPU Processing Capacity Demand from VN-1, VN-2 and VN-3</td>
<td>1.65</td>
</tr>
<tr>
<td>Case-R: (Bandwidth and CPU processing capacity demand is from a same range of value for each request for all VNs)</td>
<td>Bandwidth Demand</td>
<td>unif{8, 9, 10, 11, 12}</td>
</tr>
<tr>
<td></td>
<td>CPU Processing Capacity Demand</td>
<td>unif{0.55, 1.0, 1.65, 2.20, 2.75}</td>
</tr>
<tr>
<td>Case-VH: Different Bandwidth and CPU Processing Capacity demand for different VNs while the demand is fixed within each VN</td>
<td>Bandwidth Demand-VN-1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Bandwidth Demand-VN-2</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Bandwidth Demand-VN-3</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>CPU Processing Capacity Demand-VN-1</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>CPU Processing Capacity Demand-VN-2</td>
<td>1.65</td>
</tr>
<tr>
<td></td>
<td>CPU Processing Capacity Demand-VN-3</td>
<td>2.75</td>
</tr>
<tr>
<td>Case-VR: Different Bandwidth and CPU Processing Capacity demand for different VNs while with random within a fixed range for each request from a particular VN</td>
<td>Bandwidth Demand-VN-1</td>
<td>unif{7, 8, 9}</td>
</tr>
<tr>
<td></td>
<td>Bandwidth Demand-VN-2</td>
<td>unif{8, 9, 10, 11}</td>
</tr>
<tr>
<td></td>
<td>Bandwidth Demand-VN-3</td>
<td>unif{11, 12, 13}</td>
</tr>
<tr>
<td></td>
<td>CPU Processing Capacity Demand-VN-1</td>
<td>unif{0.35, 0.55, 0.75}</td>
</tr>
<tr>
<td></td>
<td>CPU Processing Capacity Demand-VN-2</td>
<td>unif{1.45, 1.65, 1.85}</td>
</tr>
<tr>
<td></td>
<td>CPU Processing Capacity Demand-VN-3</td>
<td>unif{2.55, 2.75, 2.95}</td>
</tr>
</tbody>
</table>

Re-visiting Postulate-1, we can see that our result confirms Postulate-1 for blocking. On the other hand, in regard to cost, Postulate-1 does not hold for cost in a highly loaded environment when the blocking for Case-R is so high that the average number of requests admitted to the system is much less than that for Case-H, which in turn means that the cost incurred is lower. Certainly, this raises the question on why the cost of Case-R is higher compared to Case-H at a low arrival rate. This is because at a low arrival rate, the blocking is low: this allows requests from higher end of resources to be admitted, thus incurring higher cost. In terms of distribution of the cost components, the comparison between Case-H and Case-R is shown in Fig. 4; the pattern of the different cost components has a similar behavior like the total cost.

Next we compare Case-VH vs. Case-VR. Note that in both these cases, the demand requirement for each VN class is different. The second difference is that for Case-VH, the demand and processing requirement for each VN class is kept the same, they are uniformly varied within the VN class in Case-VR. First, we discuss blocking. From Fig. 5, we can see that the average blocking for Case-VH is lower than that for Case-VR as the arrival rate increases. This is in line with what we observed comparing Case-R against Case-H. More notably, it is important to see how the blocking behavior changes from one VN class to another VN class. Recall that VN-1 requires the least resources per request while VN-3 requires the most.
resources. This is reflected when we observe the blocking for each VN class. For VN-1, the blocking is less than 2% at even the highest arrival rate (1.0), while for VN-2, the blocking is around 12%, and it is significantly high at 32% for VN-3. In other words, in a congested situation, the network favors admitting requests that require less resources. We notice this difference starting from a low arrival rate of 0.4.

Now consider the variation from the resource requirement being fixed within each VN class against the same being uniformly random (“H” vs. “R”). We found that there is little difference in blocking for VN-1 between Case-VH and Case-VR. On the other hand, this difference is noticeable for VN-2, and quite prominent for VN-3. In other words, when the request is randomly distributed within a range with the VN class, this behavior is similar to what we noticed when we compared Case-R against Case-H. The main difference is that the observation is much more pronounced for VN-3 as this class requires significantly more resources.

Next, consider the cost of providing connectivity to each VN customer (Fig. 6). The cost of provisioning VN-1 is always the lowest regardless of the arrival rate. However, with VN-2 and VN-3, we notice that the provisioning cost is higher for VN-3 for lower arrival rates, but at a higher arrival rate,
this is not so. This difference in cost can be explained by the observation that the blocking for VN-3 is significantly higher than that for VN-2 at a higher arrival rate to the point that the network is denying many VN-3 requests and in turn, the cost has also dropped. Revisiting Postulate-2, we find that this holds for blocking; however, for cost, Postulate-2 hold for a lower arrival rate, but not at the higher arrival rate. The basic reason is the same as the one explained with Postulate-1 for Case-H and Case-R. We also plot just the bandwidth cost for each VN when comparing these two cases in Fig. 7. We found the behavior to be similar except that for VN-3, the bandwidth cost is slightly higher for Case-VH then that for Case-VR at the highest arrival rate (heavy traffic).

B. Energy Consumption

We next focus on energy consumption. As we stated earlier, our model takes the energy issue into consideration. We first solved the optimization model using energy as the only cost in the objective and compared it again if the hosts were to continually run at the higher power consumption level. This is shown for all four cases in Fig. 8. We observe that our approach reduces the energy consumption to about one-sixth of the maximum energy cost at low arrival rate to two-thirds at the highest arrival rate. We also note that when the model is simply optimized for energy cost, the energy cost is not much different between the four cases.

Next we consider difference in the cost of energy consumption among VN classes by considering Case-VH and Case-VR (Fig. 9) when the entire objective function is optimized. We note that the randomness in resource requests around the average does not have much impact on VN-1 compared to if the resource request were fixed. On the other hand, for the highest VN class, VN-3, this variation in request makes a noticeably larger impact on the energy consumption cost. It may be noted that the energy cost drops off near the highest arrival rate. This is aligned with the cost phenomenon discussed with regard to Fig. 6.

IV. RELATED WORK

Early research on data center networks investigated architectural construction, operation and scalability of DCs [6], [7], [8], [9], [10], [11]. Joint VM placement and routing for data center traffic engineering was addressed by [12]. Similar to [12], we also consider our problem from a traffic engineering point of view but we do not focus on VM placement; rather, we keep routing flexible in such a way that no dedicated server is required to satisfy demand from a particular VN. Any idle server is able to handle the request from any VN tenant. To satisfy a particular request, a server is chosen based on the resource demand and available resources of the server. Unlike their work, we take bandwidth guarantee into consideration. The issue of multiple service classes with heterogeneous requirements have been addressed for access control [13], [14]; however, they do not consider two-tuple demands nor the implication of network routing.

In [4], the authors presented a formulation to optimize the link cost in one data center, while we consider connecting multiple data centers. Unlike [4], we also take two issues into account, which are energy consumption by the servers, and the DC VN mapping cost. [5] discussed the servers’ operational cost optimization without taking data center architecture into consideration, and they did not consider the on-demand model either. Furthermore, in our case, we combine three cost components (reducing link costs, power cost, and the DC VN mapping cost) together and impose weight parameters on each of these components to reflect their relative importance. Another novel contribution beyond the state-of-the-art of this research is the dynamic nature of our model to provide on-demand service considering north-south traffic and finding the optimal resource requirement to contain service blocking within a tolerable range. Moreover, we can also identify which servers are not used to serve the VN requests at a particular time, which can give us the opportunity to keep those servers in a lower power consumption mode.
V. CONCLUSION AND FUTURE WORK

In this work, we presented a dynamic traffic engineering framework for resource allocation due to north-south traffic in a multi-location data center environment. We presented a novel MILP formulation that is solved in this framework at each review point. Our approach is geared for enterprise customers that require resource guarantees from data centers.

We then conducted a systematic study to understand the cost and blocking relation in normal traffic to overload traffic conditions by considering a number of cases. This sequence of considered cases allowed us to answer a number of questions when resource requirements may vary for each request as well as may differ between different customers. In general, we observed that VN customers with the lowest resource requirements face the lowest blocking as the traffic is increased in the system. For VN customers with high resource requirement, blocking is significantly higher for heavy traffic to the point where the cost incurred to serve this customer classes’ accepted requests can be less than other customer classes.

There are several future directions we wish to address. In our current model, we do not factor in that a blocked request could incur a penalty cost due to loss in revenue. Secondly, we do not allow partial fulfillment of a request if there is not sufficient resources to fully consider a request. We also plan to consider large-scale network cases to mimic real world data center networks by considering a large number of virtual networks and with a large number of servers for data centers to test the scalability of our model and present the performance of the model with respect to optimality and solution time. Furthermore, we plan to add performance evaluation on the loads to a data center based on its geographical distance from different VNs. In addition, we intend to propose a heuristic and show the comparison between the solution achieved by using the heuristic and CPLEX form the point of view of computational complexity and performance such as different types of costs and blocking probability. These aspects will be addressed in a future work.
An Energy-Aware Embedding Algorithm for Virtual Data Centers

Tran Manh Nam, Nguyen Van Huynh, Le Quang Dai, Nguyen Huu Thanh
School of Electronics and Telecommunications
Hanoi University of Science and Technology
Hanoi, Vietnam

Abstract—Cloud computing has emerged in the recent years as a promising paradigm that facilitates such new service models as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). As the number of cloud service providers increases, there exists a demand to dynamically provision virtual data centers (VDC) on top of the infrastructure provider’s physical data centers. This research addresses problems related to embedding virtual data centers inside physical data centers. VDC embedding is challenging as it is an NP-hard problem that should meet multiple objectives. In the paper, we propose a new heuristic VDC embedding algorithm that takes into account energy consumption as well as physical resources of data centers. We also realize this virtualization paradigm based on a Software-Defined Network architecture.

I. INTRODUCTION

Cloud computing is becoming increasingly important nowadays as it supports new business models such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). One important component of cloud computing are data centers, which are used by cloud service providers to house cloud-based resources and services. In cloud computing paradigms, cloud service providers can typically build their own data centers to offer cloud services or alternatively make use of data centers provided by third-party Infrastructure Providers (InP). In either former and later case, data center virtualization comes into play, which is a concept of network virtualization (NV) that allows creating multiple, separated virtual data centers (VDC) on top of physical data centers. Creating VDCs on top of the physical cloud substrates has the following advantages:

• Cost saving: data center virtualization allows reducing capital expenditure (CAPEX) and operational costs (OPEX) as several cloud service providers can share the same physical data center of a third-party infrastructure provider.

• Energy saving: Recent surveys have shown that the energy consumption in a data center considerably contributes to its operation costs. A remarkable part of the large energy volume consumed in data centers today is due to the over-provisioning of such network resources as switches, links, and servers to meet the stringent requirements on reliability. By dynamically scale up and down the VDC instead of maintaining a fixed number of physical servers, the under-utilization of servers and network can be avoided that leads to more energy-efficient usage of the data center.

• Flexibility: the VDC can be dynamically provisioned on demand based on service requirements, scalability, time duration and required resources.

A major challenge of network virtualization in data centers is the virtual data center embedding (VDCE) problem that consists of two sub-problems: (1) embedding virtual machines (VM) on physical servers in the physical data center based on the VDC embedding request (see III-A); and (2) creating a virtual network consisting of virtual switches and links interconnecting these virtual machines.

Solving the VDC embedding problem is NP-hard. For that reason current research mostly follows heuristic and meta-heuristic approaches. In this research, we focus on energy-efficient virtual data center embedding approaches with the following contributions:

• A novel VDC embedding algorithm with the following objectives: (1) Resource efficiency that deals with efficient mapping of virtual resources on substrate resources in terms of CPU, memory and network bandwidth; (2) Energy efficiency that deals with minimizing energy consumption of the virtual data center while meeting mapping demands; and (3) Flexibility that deals with how the VDC can be dynamically provisioned, changed and removed according to actual needs. Evaluation results show that our algorithm performs better than some existing ones in terms of acceptance ratio and energy consumption.

• A data center virtualization architecture based on Software-Defined Networking technology that has full control of network and server virtualization in the physical data center and allows performing VDC mapping dynamically and flexibly.

The rest of the paper is organized as follows. Section II discuss some related work on virtualization in data centers. Section III formulates the VDC embedding problems and power profiling and modeling. In Section IV, we propose a new energy-aware VDC embedding algorithm with the above objectives. Section V shows some evaluation results. The last section concludes the work.
II. RELATED WORK

A. Virtual data center embedding

As already addressed above, one of the key challenges in building virtual data centers is the VDC embedding problem, which requires to map VDC components such as virtual machines (VM) and network devices (switches and links) onto physical servers, nodes and links. There is only few research work that has addressed the VDC embedding problem. For instance, VDC Planner [6] and Venice [9] were proposed as VDC embedding methods based on migration-aware model to maximize the revenue of InPs. Gue et al. [11] proposed a data center network architecture called SecondNet that incorporate a greedy algorithm for resource allocation to VDC. SecondNet focuses on providing bandwidth guarantees among multiple VMs in a multi-tenant virtualized data center. Amokrane et al. [18] introduced GreenHead - a holistic resource management framework for embedding VDCs across geographically distributed data centers connected through a backbone network. For energy efficiency, Han et al. [10] proposed SAVE - an SDN Assisted VDC Embedding system. However, those methods did not consider the life time of VDCs as well as arriving and leaving time of their requests. Besides, recently there is some work focusing on energy efficiency of servers by using servers consolidation and placement algorithms for VMs that can be mapped onto physical servers [1], [2], [3]. Nevertheless, these approaches only focus on a single group of VMs requests and not on the embedding of virtual data centers that include multiple groups of VM requests at a time. To the best of our knowledge, energy-efficient VDC approaches that take into account the dynamic embedding of dynamic VDC requests and address energy consumption of both physical servers and network devices are still lacking.

B. DC network – Fat-tree

DC network topology: In this work, we choose the Fat-tree topology [16] as the network topology of data center. One of the most advantageous of this topology is the reduction of the oversubscription ratio that allows removing bottleneck point of the hierarchical architecture. A k Fat-tree is a network architecture of DC that uses the same k-port switches with three layer: edge, aggregation and core. Like [18] we divide the traffic pattern into three scenarios: near, middle and far. In the near traffic scenario, the source and destination of the flow are connected to the same edge switch, so that the exchanged traffic traverses over only one switch. On the other hand, a flow in the middle traffic scenario has the source/destination pair residing in the same POD but does not connect to the same edge switch, so that the traffic traverses over three switches (two edge and one aggregation switch). Finally, in far traffic scenario, the source/destination pair of flow stay on different PODs (Performance Optimized Data Centers), so that core, aggregation and edge switches are involved in the communication. Based on these scenarios, in this work, we propose three groups of servers for the embedding algorithm (discuss later in section IV).

C. Related technologies for virtualization

Despite the advances in virtualizing computing and storage elements, the network is still mostly statically configured in a box-by-box manner. One might think that long standing virtualization primitives such as VLANs (virtualized L2 domain), NAT (virtualized IP address space), and MPLS (virtualized path) are enough to provide full and automated network virtualization. However, there is no single unifying abstraction that can be leveraged to configure (or reconfigure) the network in a global manner. As a consequence, current network provisioning can take long, while computing provisioning takes only minutes [20]. Optimizing and realizing Network Virtualization has attracted much attention of research communities and important role in data center virtualization at this time.

There are many approaches of network virtualization that have already been under research and used for the future of the Internet testbeds [4], [5]. Network virtualization provides an abstraction of coexistence of multiple virtual networks on the same physical substrate network. In data center contexts, NV should cooperate with virtual machine consolidation. The introduction of Software-Defined Networking, an emerging networking paradigm that gives hope to change the limitations of current network infrastructures, has made network virtualization more easily. The network virtualization concept in the context of Software-Defined Networking was first mentioned by Sherwood et al. [21]. The authors propose a framework called FlowVisor which allows the same hardware forwarding plane to be shared among multiple logical networks, each with distinct forwarding logic. In [22] we extend FlowVisor to multilayer NV with resource reservation. Software-Defined Networking [20] is an emerging architecture of the future Internet that is programmable, flexible and centralized manageable, making it ideal for the high-bandwidth, dynamic nature of today’s Internet services. In this paper, we propose a data center virtualization architecture based on Software-Defined Networking technology that allows performing VDC mapping flexibly.

III. PROBLEM FORMULATION

A. Data center modeling

In this section, we model the virtual data center embedding as an optimization problem focusing on minimizing total power consumption of both servers and network devices spent on embedding VDC requests.

Physical DC infrastructure: We model a physical DC infrastructure as a weighted graph \( G^P = (S^P, N^P, L^P) \) where \( S^P \) denotes a set of physical servers, \( N^P \) denotes a set of network devices (switches) and \( L^P \) denotes a set of physical links. For physical servers, the attributes generally include memory and CPU capacity, \( M_{cap}(S^P) \) and \( C_{cap}(S^P) \) denote available (or leftover) memory and CPU of \( S^P \), respectively. For physical links, the attribute is bandwidth so that \( B_{cap}(L^P) \) denotes available bandwidth of a link \( L^P \). Note that the modeling, analysis and algorithm in this paper can be easily extend to incorporate other attributes.
TABLE I
TERMINOLOGY USED THROUGHOUT THIS WORK

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP(Sp, Np, Lp)</td>
<td>Physical DC infrastructure</td>
</tr>
<tr>
<td>Sp, Np, Lp</td>
<td>Set of physical machines, switches and links, respectively</td>
</tr>
<tr>
<td>Be(lg)</td>
<td>Available bandwidth of physical link Lg</td>
</tr>
<tr>
<td>Ccap(Sp)</td>
<td>Available CPU of physical server Sp</td>
</tr>
<tr>
<td>Mcap(Sp)</td>
<td>Available memory of physical server Sp</td>
</tr>
<tr>
<td>Rv(Sp, Lp, d, t)</td>
<td>VDC request with set of virtual machines, matrix of links demand, arrival time and duration</td>
</tr>
<tr>
<td>p</td>
<td>Requested bandwidth from source vm to destination vm</td>
</tr>
<tr>
<td>Md, Cd, Bd</td>
<td>Memory, CPU demand of server and bandwidth of link, respectively</td>
</tr>
<tr>
<td>cap: Sp ∪ Lp → GP</td>
<td>Function assigns a available capacity to an element of GP (either servers or links)</td>
</tr>
<tr>
<td>deml: VMi ∪ Lni → GP</td>
<td>Function assigns a demand to an element of VDC request Rv (either VM or bandwidth)</td>
</tr>
<tr>
<td>fi: VMi → Sp</td>
<td>Function that maps virtual machine to physical machine (VmM)</td>
</tr>
<tr>
<td>ki: Lni → (Np, Lp)</td>
<td>Function that maps matrix of BW demands onto a part of physical DC (VLM)</td>
</tr>
<tr>
<td>Pp, Pn</td>
<td>Power consumption of servers and network devices</td>
</tr>
</tbody>
</table>

VDC request: A sequence of virtual data center requests joins and leaves over time. Similarly to GP, we model ith VDC as a weighted graph Rvi = (VMi, Lni, t, di), in which ti and di denote the arrival and departure time of VDC, respectively. VMi denotes the a set of virtual machines with the corresponding computing resource Ci = (VMi) and memory Mi = (VMi) demands. Lni denotes the matrix of bandwidth demand including lsd ∈ Lni that is the bandwidth demand from the source vm to the destination vm.

VDC embedding: The main challenge in creating virtual data centers is the VDC embedding problem, which maps VDCs onto the physical data center, which includes physical servers and links. Given a VDC request Rvi and a physical data center GP, embedding Rvi onto GP means to find a subset of Sp, Np, Lp at time ti that satisfies the requirement of VMi and Lni. Solving this embedding problem by Integer Linear Programming is NP-hard [13]. In this work we divide it into two subproblems: (1) virtual VM mapping (VmM) that maps the VMs of VDC request onto the physical servers; and (2) virtual link mapping (VLM) that maps matrix of link demands onto the substrate links. Let cap: Sp ∪ Lp → GP be a function that returns an available capacity of physical DC, either servers or network devices. Besides, for each VDC request ith, let deml: VMi ∪ Lni → GP be a function that assigns demand to an element of this VDC. Then, a VDC embedding consists of two functions VmM (Eq.1) and VLM (Eq.2).

\[ f_i: VM_i \rightarrow Sp \]  
\[ k_i: L^{ni} \rightarrow (N^p, L^p) \]  

Such that these two mapping functions form an embedding for VDCi. Computational resources (CPU and memory) required by a vm must be lower than those of physical server hosting and the required bandwidth of a virtual link must be lower than the available bandwidth of all physical links on the path of the DC that the virtual link lsd is mapped.

\[ \forall vvm_i \in VM_i : deml(vvm_i) \leq cap(f_i(vvm_i)) \]  
\[ \forall l_{sd}^v \in L^{vi} \cup l_{sd}^s \in k_i(l_{sd}^v) : deml(l_{sd}^v) \leq cap(l_{sd}^v) \]  

Let x^i_k be a binary function indicating whether the VM i is allocated in server k. Let statei: Sp ∪ Np ∪ Lp → Gp denote the function returning a state at time t of an element of the DC by binary values, which return 1 when turning on (ON State) and 0 otherwise (OFF State). Thus,

- state(s, t) - The working state of the physical server s ∈ Sp at time t.
- state(n, t) - The working state of the physical network device n ∈ Np at time t.
- state(L, t) - The working state of the physical links Lp ∈ k_i(l_{sd}^v) at time t

Then we have the constraints of the functions fi and ki as below.

One virtual machine is mapped on only one physical server (Eq.5) if successful or none if unsuccessful (Eq.6).

\[ \sum_{i \in Sp} x^i_k = 1, \forall k \in VM \]  
\[ x^i_k = 0 \]  

All physical elements that a VDCi is allocated on must be turned on (Eq.7 and Eq.8).

\[ \forall S_{pi}^p \in f_i(VM_i) : state(S_{pi}^p, t) = 1 \]  
\[ \forall L_{pd}^v \in L^{vi} \cup L_{pd}^s \in k_i(l_{sd}^v) : state(L_{pd}^v, t) = 1 \]  

B. Energy modeling

The working state of physical machines, switches and links are formulate as follows:

1) Physical machines: Energy consumption model of physical machines is defined as Eq.(9) where stsi is binary indicator whether ith server turned on (1) or turned off (0) and psi is energy consumption.

\[ P_s(t) = \sum_{s_i \in Sp} state(s_i, t). p_s \]  

2) Network devices: The working state of switches i at time t is defined by binary indicator stni(t) = 0 (switch is turned off) or 1 (otherwise). The energy consumption of the network part of DC (switches and links), Pn(t), at time t is denoted as summing of all switches with static power (baseline power), P_stat, and power consumption of interfaces under their operating speeds (see Eq.(10)).
\[
P_N(t) = \sum_{\forall n \in N} state(n,t) \cdot [P_{\text{static}} + \sum_{j=1}^{k} (m_j \cdot P_j)]
\]

(10)

In this work, for power measurement of network devices, we use power profile of an energy-aware commercial 24-port switches HP [14]. The switch is able to change the clock frequencies of its network interfaces for different power states. Thus \(P_{\text{static}}\) and power consumption of ports under separate operating speeds 10Mbps, 100Mbps, 1000Mbps \((P_j)\) is summarized in Table II. Besides, the average typical power consumption for a server IBM 2U rackmount x86 is 400W as described in [15].

<table>
<thead>
<tr>
<th>Operating speed</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_{\text{static}})</td>
<td>39,000</td>
</tr>
<tr>
<td>(P_{100}) - 100Mbps per port</td>
<td>420</td>
</tr>
<tr>
<td>(P_{1000}) - 1000Mbps per port</td>
<td>480</td>
</tr>
<tr>
<td>(P_{1000}) - 1Gbps per port</td>
<td>900</td>
</tr>
</tbody>
</table>

Table II: Power summary for a HP Enterprise Switch

C. Energy-saving problem formulation

Basically, the main objective of this work is to reduce the total energy consumption of the physical DC. Let \(P_{\text{DC}}\) represent the total energy consumption of all servers and network devices of the DC. The energy-aware VDC embedding algorithm focuses on decreasing the number of ON state servers and network devices. This objective is defined as Eq.(11):

\[
\min \left( \sum_{\forall n \in N} \text{state}(n,t) \cdot P_n + \sum_{\forall n \in N} \text{state}(n,t) \cdot [P_{\text{static}} + \sum_{j=1}^{k} (m_j \cdot P_j)] \right)
\]

(11)

The constraints of this VDC embedding are from Eq. (3) – Eq. (8).

D. VDC request modeling

Collecting traffic load measurements in real data centers is challenging. Especially, the measurements and modeling of VDC requests on top of a physical data center can hardly been found in the recent literature as data traffic of network and cloud operators is often inaccessible to the outside world. However, according to recent research work, there is a consensus that the arrival rate of VDC requests follows the Poisson distribution; the lifetime of a VDC is exponentially distributed [6], [7], [8], [9]. In this work we also follow these configurations for VDC requests. Section V-A will discuss more about our test scenarios.

IV. ENERGY-AWARE VIRTUAL DATA CENTER

A. Energy-aware VDC architecture

In this work, we construct an energy-aware virtual data center architecture EA-VDC that allows deploying any energy-aware embedding algorithm. As shown in Fig. 1, the architecture consists of two main blocks: Management and DC Hypervisor. The Management block consists of three subblocks: (1) Centralized Monitoring that monitors the states and resources of network devices and servers; (2) VDC Embedding that realizes various mapping algorithms; and (3) DC Configuring that controls the physical DC based on SDN technology by interacting with both the Network Hypervisor and VM Hypervisor subblocks. There are already several platforms for server hypervisor such as HyperV, OpenStack, KVM, XEN, VMware [26], [27]. We find that one of the most challenges of the VDC architecture is network hypervisor. In [22] we develop the ReSerVNet platform as an extension of FlowVisor [21] that deploys a new resource management and allocation concept for network virtualization. By combining server and network hypervisor based on SDN technology, the proposed architecture could be a promising approach for energy efficiency of DC virtualization.

B. Virtual Embedding Algorithm

This section describes our proposed Heuristic Energy-Aware VDC Embedding (HEA-E) algorithm for virtual data center embedding. The main objective of HEA-E is to reduce power consumption of the physical data center while improving the embedding acceptance rate. To the best our knowledge, there are two existing articles, SecondNet [11] and GreenHead [8], that focus on a sequence the VDC requests joins and leaves over time. Thus in this work, we compare our proposed
algorithm with the aforementioned approaches. There are some drawbacks of existing embedding algorithms. First, when embedding a VM request, the VM embedding algorithm in SecondNet [11] chooses a group of physical servers with the number of servers greater than or equal to the number of VMs. However, SecondNet does not take into account the availability of these servers as well as their available capacity in terms of memory and CPU. This directly affects the acceptance rate and energy saving level of the DC. Furthermore, the virtual link embedding algorithm performed afterwards uses the Breadth First Search (BFS) algorithm [25] to find the shortest path for link mapping. In some cases, the shortest path may require to turn on more immediate physical switches [28] instead of using ON State switches that leads to more power consumption. On the other hand, GreenHead [8] makes use of VLM with BFS. Based on the BFS algorithm, GreenHead does focus on the available bandwidth of the links on shortest paths. If the available resources of the shortest path cannot satisfy the bandwidth demand, the VDC request is rejected.

In contrast to GreenHead and SecondNet, HEA-E is an energy-efficient embedding algorithm that is based on two Power Scaling and Idle Logic approaches that have been attracted much attention from research communities [12]. Power scaling reduces power consumption of devices by adaptively changing operating rates of processing engines or link’s speeds while Idle logic saves energy by quickly turning the devices off when there is no traffic load and rapidly waking these up if traffic is available. On the other hand, energy saving approaches should also maintain QoS and network reliability. That is, in case of no traffic the DC network maintains a Minimum Spanning Tree (MST) for minimum connectivity between servers. As traffic load increases, more servers and links in the Fat-Tree are turned on on demand [18]. In HEA-E, at initial phase when there is no traffic demand among servers, then: (1) all servers are turned off; (2) only one first left core switch Sw_{core} runs in lowest operating speed; (3) one first left aggregation switch Sw_{agg} and all access switches Sw_{acc} run in lowest operating speed. In this case there are only $\frac{k^2}{2} + k + 1$ switches that are in OFF State (green switches in Fig.2). As a result, $\frac{k^2}{2} + k + 1$ are in OFF State.

The proposed embedding consists of VLM and VLM that are described in Algorithm 1 and 2, respectively. When a VDC request arrives, HEA-E first uses VLM to map virtual machines onto physical servers. Subsequently, based on

Algorithm 1 Virtual Machines Mapping algorithm

1. input: $G^p(t), R^i_j$
2. //Get list group in near, middle, far order
3. $GR \leftarrow$ getListGroup()
4. isSuccess = False
5. for all $gr_i \in GR$ do
6. count = 0
7. for all $S_j \in gr_i$ do
8. if $M_{cap}(S_j) > 0$ and $C_{cap}(S_j) > 0$ then
9. count = count + 1
10. end if
11. end for
12. if count $\geq |M^i|$ then
13. $listPossibleGroup \leftarrow gr_i$
14. end if
15. end for
16. //Sort listPossibleGroup in increasing order by number of server need to turn on
17. listSatisfiedGroup = tryMapAndSort(listPossibleGroup)
18. selectedGroup = getFirstGroup(listSatisfiedGroup)
19. if selectedGroup $\neq \emptyset$ then
20. isSuccess $\equiv$ True
21. vmResults $\leftarrow$ mapVm(VM$_i$, selectedGroup)
22. end if
23. output: isSuccess, vmResults, $G^p$

VLM results, VLM creates virtual links interconnecting newly mapped VMs on top of the physical network substrate. In order to improve the acceptance rate, in HEA-E we construct a re-mapping mechanism that allows re-mapping VmM if VLM does not perform successfully. The flowchart of HEA-E is described in Fig. 3.

1) VLM: In this paper, we define three groups of servers, namely near group, middle group and far group that correspond to near traffic, middle traffic and far traffic, respectively (Line 2 of Alg. 1 – see also Sec. II-B). When a VDC request arrives, the following actions are taken:

- Step 1: Finding all possible mapping groups that have the number of available servers greater or equal to the number of VMs in the request. To improve reliability, a source and destination VM pair will not be mapped on the same physical server.
- Step 2: For all selected server groups, choose candidate groups with the least servers so that the power consumption of servers can be saved.
- Step 3: For all candidate groups with the least number of servers in Step 2, choose a group that has as many servers
that are near to each other as possible. That is, the candidate groups are prioritized in the near → middle → far order. By doing this, it is likely that the virtual links interconnecting these VMs should traverse over less immediate hops, thus reducing the power consumption of network devices.

Algorithm 2: Virtual Link Mapping algorithm

1: input: \( G^p(t), R^m, \text{vmResults} \)
2: isSuccess = False
3: \( L_j^* = \text{sort}(L_j^*, \text{key} = \text{bw}\_\text{Request}, \text{order} = \text{desc}) \)
4: for all \( v\_\text{Link} \in L_j^* \) do
5: \( \text{linkType} = \text{getType}(\text{vmResults}, v\_\text{Link}) \)
6: if \( \text{linkType} \equiv \text{Near} \) then
7: \( s\_\text{Link} = [s\_\text{Phy}, s\_\text{Edge}, d\_\text{Phy}] \)
8: if \( s\_\text{Link} \equiv \text{satisfied} \) then
9: \( v\_\text{LinkResults} \leftarrow \text{map(link, s\_\text{Link})} \)
10: end if
11: else
12: \( \text{listSwitch} = \text{getListSwitch}(\text{vmResults}, v\_\text{Link}) \)
13: //Construct link from listSwitch
14: \( \text{listLink} = \text{constructLink(listSwitch)} \)
15: \( \text{listLink} = \text{sort(listSwitch, key} = \text{cap, order} = \text{asc}) \)
16: for all \( s\_\text{Link} \in \text{listLink} \) do
17: if \( s\_\text{Link} \equiv \text{satisfied} \) then
18: \( v\_\text{LinkResults} \leftarrow \text{map(link, s\_\text{Link})} \)
19: end if
20: end for
21: end if
22: end for
23: if \( |v\_\text{LinkResults}| = |L_j^*| \) then
24: isSuccess = True
25: else
26: Return to VM Mapping with next group
27: end if
28: output: isSuccess, v\_LinkResults, G^p

2) VLiM: The proposed virtual link mapping algorithm is inspired by Heller et al. [17] approach. Heller proposed the Elastic Tree concept in order to reduce consumed energy of DC network by maintaining a minimal logical topology on top of the Fat-Tree based on actual traffic demands. In HEA-E, VLiM (Algorithm 2) firstly arranges the matrix of virtual link requests of a VDC in non-increasing order of bandwidth demands. Then with each request corresponding to a source virtual machine \( vm_s \) and destination machine \( vm_d \), the VLiM algorithm identifies the type of traffic scenario (i.e., near, middle or far) according to their relative positions that are already decided by the VmM algorithm. It then performs the correlative actions:

- **Near traffic** scenario: If the VMs are in the physical machines that are connected to the same edge switch \( sw_{edge} \), the request is mapped onto the physical links connecting the VMs through that switch.

- **Middle or far traffic** scenarios: Otherwise, find all possible paths with necessary set of active switches. Select the path satisfied the traffic demand that has least available bandwidth and consumes the least additional energy.

If the virtual link mapping does not succeed, HEA-E performs virtual node re-mapping as described in Fig.3.

V. Performance Evaluation

To investigate the performance of the proposed algorithm HEA-E and the feasibility of the EA-VDC architecture, simulations have been carried on. The performance of HEA-E is compared with two existing VDC embedding algorithms, namely GreenHead [8] and SecondNet [11]. Three metrics have been used for the performance evaluation, which are embedding acceptance ratio, power consumption, and complexity in terms of running time.

A. Simulation environment

We developed a Java-based simulator for data center that makes use of \( k \) fat-tree topology. In this research \( k = 4 \) corresponds to a data center with 16 servers. The topology consists of the same \( \frac{5k^2}{4} \)-port switches and \( k^2 \) servers. The power profile of switches and devices are taken from [14] and [15] (see Sec. III-B). Each server has 8 CPUs and 64GB of memory.

In the tests, similarly to previous work published in [6], [7], [8], [9], VDC requests are generated randomly following a Poisson distribution with arrival rate of \( \lambda = 8 \) VDCs per hour. The duration of a VDC is exponentially distributed with 2 hours average lifetime. In order to test the performance of embedding algorithms under different load scenarios, the number of VMs per VDC request varies from 4 to 16. All traffic demands between VM pairs are randomly distributed between 10Mbps and 90Mbps, so that the data center utilization varies from 10% to 90%. The VMs within a VDC are interconnected with a random topology generated by Waxman algorithm [19]. Eq. 12 represents the probability that there exists a link connecting two arbitrary nodes \( u \) and \( v \) in the Waxman algorithm:

\[
P(u, v) = \alpha e^{-d(u,v)/(\beta L)}
\] (12)

Parameters \( \alpha, \beta \) in Eq.12 are in range of \( (0, 1) \). \( d \) is the distance in Cartesian coordinates among VM \( u \) and \( v \), \( L \) is the maximum distance between any two nodes in the graph. A rise in the parameter \( \alpha \) increases the probability of existing links between any nodes in the graph, and an increase in \( \beta \) yields a larger ratio of long links to short links. Similar to some previous work [23], [24], in the simulations we set \( \alpha = 0.5 \) for average connectivity and link distance.

B. Experimental Results

1) Acceptance rate: As can be seen in Fig.4, HEA-E outperforms GreenHead and SecondNet in terms of acceptance ratio. Especially in highly loaded scenarios, the probability of successful VDC mapping in HEA-E is remarkably higher than the other two. The reason is, while taking into account the available resources of the physical substrate, HEA-E also has a flexible re-mapping mechanism.
2) Resource-Efficiency Ratio: We define the Resource Efficiency Ratio (ReR) as the ratio of the number of accepted VDC requests over total available virtual machines in the physical DC. It is expected that with the same arrival rates of VDC requests $\lambda$, the physical DC resource usage dedicated to accepted VDCs be as high as possible. That is, with the same physical resources and arrival rate $\lambda$, the more ReR is, the more VDC requests can be accommodated.

In our experiments, a fat-tree with $k = 8$ or 128 servers is deployed. We assume that each physical server can accommodate 4VMs so that there are maximum 512VMs in a DC. The arrival rates of VDC requests vary from 10 to 80 requests per hour, each VDC request consists of 8VMs. As can be seen in Fig. 5, the Resource Efficiency Ratio of HEA-E is higher than those of SecondNet and GreenHead.

3) Power consumption: HEA-E prioritizes ON_State devices, including both servers and switches, it also places the VMs as near to each other as possible thus reducing the number of active network devices. Simulation results in Fig. 6 show that the power consumption of the new algorithm is better than SecondNet and as good as GreenHead.

We also measure the power consumption of each successful VDC as described in Eq. 13 where $\text{Num}_{VDC}(t)$ is the number of mapped VDC. As can be seen in Fig. 7, when the arrival rates vary from 30 to 90 requests per hour, (1) the consumed power per each VDC is decreasing; (2) the consumed power of proposed algorithm HEA-E is less than both existing algorithms; and (3) as the requests rate increase, the power consumption of a VDC in case of HEA-E and GreenHead decreases very slowly, which implies that power consumption of the physical DC stays nearly linear to the number of embedded VDCs. We call this as the power proportional property.

$$\text{Power}_{VDC}(t) = \frac{P_{V}(t) + P_{S}(t)}{\text{Num}_{VDC}(t)}$$  \hspace{1cm} (13)

In other scenario, arrival rate is 64 VDC requests per hour and each VDC request consists 8VMs. A $k = 8$ fat-tree and 128 servers is deployed (each server can accommodate 4VMs). The duration of VDC request is 2 hours. After 24h, we measure the average consumed power of each VDC under three algorithms HEA-E, GreenHead and SecondNet. The experimental results are shown in Table III.

<table>
<thead>
<tr>
<th></th>
<th>HEA-E</th>
<th>GreenHead</th>
<th>SecondNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumed Power (W)</td>
<td>929.36</td>
<td>1080.30</td>
<td>1146.44</td>
</tr>
</tbody>
</table>

![Fig. 4. Acceptance Rate](image)

![Fig. 5. Resource-efficiency Ratio](image)

![Fig. 6. Power Consumption](image)

![Fig. 7. Consumed power per VDC](image)
4) Time complexity: In order to evaluate the complexity of the embedding algorithms, we run different embedding requests on a 4GB Intel Core 2 Duo 2.67GHz and measure the execution time. As shown in Fig.8, the execution time increases with the rise of VM requests. The execution time of GreenHead algorithm is the smallest. That is because after $V_{mM}$, GreenHead uses BFS for $VLM$, and if this shortest path does not satisfy the capability demand then this algorithm will drop the VDC request without any ‘re-mapping’ process. Contrary to GreenHead, SecondNet and HEA-E have re-mapping technique so that it needs to take more time for calculation.

From the results there are some remarks as the follows: (1) Three energy-aware embedding algorithms are able to adapt the power consumption of the DC proportionally to the number of servers and the corresponding load. Among energy-aware embedding algorithms, HEA-E and GreenHead are the two best ones in terms of energy efficiency. Both algorithms can save up to 80% of the energy consumption of a 16-server fat-tree DC (i.e., 500W over total 3000W); (2) However, the payoff of GreenHead is that its acceptance ratio is the worst. In contrast, HEA-E is the best algorithm in terms of acceptance ratio. That is, HEA-E can improve the acceptance ratio while maintaining its power consumption the lowest.

VI. CONCLUSION AND FUTURE WORK

In this work we propose a virtual data center embedding algorithm as well as a new SDN-based virtualization architecture that allows deploying both network and server virtualization on the physical DC infrastructure. The newly proposed HEA-E embedding algorithm can reduce up to 80% of the data center energy consumption, while it can remarkably improve the acceptance ratio in comparison to some existing algorithms.

REFERENCES

[27] https://www.openstack.org/ Accessed October 2015
Disaster Avoidance Control against Tsunami

Phuong Nga Tran
NTT Network Technology Laboratory
Tokyo, Japan
Email: p.tran@lab.ntt.co.jp

Hiroshi Saito
NTT Network Technology Laboratory
Tokyo, Japan
Email: saito.hiroshi@lab.ntt.co.jp

Abstract—We investigated challenges in network disaster management against tsunamis. A tsunami is a natural disaster of which the arrival time and devastating effect can be predicted. Based on this prediction, network operators can carry out different disaster management actions to avoid or reduce the damaging effect of tsunami on the network. We developed and evaluated heuristic algorithms to efficiently migrate service (virtual) networks away from a disaster affected area to minimize traffic loss when a tsunami arrives. The problem was first formulated as an integer linear programming problem, which can be solved using optimization solvers for small/medium-sized networks. For large networks, our heuristic algorithms ensure a good solution within a reasonable time. Extensive simulations were carried out to evaluate the performance of our proposed algorithms. On the basis of the tsunami predicted information and network status, network operators can select a suitable algorithm for their disaster management action.

Index Terms—Disaster management, virtual network reconfiguration, tsunami

I. INTRODUCTION

Due to global warming, the world has been impacted by an increasing number of furious natural disasters that do not only take thousands of human lives, but also cause widespread destruction of social infrastructures including communication networks [1]. In spite of the increasing reliability and resiliency of modern communication networks against physical damage, the risk associated with network failures remains serious due to society’s growing dependence on communication facilities.

To reduce damage caused by natural disasters, network operators mainly follow two approaches: protection (provisioning) and recovery (restoration) [1]–[3]. The former is to provide spare resources when designing and planning communication networks for backup purpose. The latter includes actions during and after disasters, such as looking for backup resources, to restore networks from failures. These two approaches complement each other and play a central role in disaster management, although they are fundamental parts of fault and recovery management. After a disaster, network operators may also use temporary systems. A transportable terrestrial station of a satellite communication system is such an example.

Saito recently proposed the concept of Disaster-Free Network (DFN) [4] that includes robust physical network design [5] and disaster avoidance control against spatial disasters [6]. The concept of DFN is to avoid or reduce encountering disasters and is different from technologies based on fault and recovery management. In particular, disaster avoidance control is a rather new concept, which aims to avoid disasters. It includes mechanisms/algorithms for migrating network functions and service (virtual) networks away from forecasted disaster areas to avoid or reduce the damaging effect of physical network failures. This idea is well supported by software defined networking (SDN) – an emerging network paradigm [7] - which provides an adequate level of flexibility and programmability in the control plane; hence, it can facilitate the live migration of virtual networks (VNs) and devices.

The idea of disaster avoidance control is also supported by the fact that predicted information on natural disasters is increasingly accurate thanks to advancements in meteorology. Furthermore, historical disaster data and hazard maps describing high-risk areas for each type of disaster is also open to the public. On the basis of this information, network operators can implement different disaster avoidance control algorithms to enhance the robustness of communication networks against certain predictable natural disasters, such as tsunamis, hurricanes and heavy rains. As Mukherjee et al. suggested [2], progress in such technical areas is key to a new direction of disaster management.

In this paper, we propose and investigate a disaster avoidance control algorithms against tsunamis. A tsunami is caused by an earthquake occurring in the ocean that can push up powerful waves. The occurrence of tsunamis can be forecasted because earthquakes causing the tsunami can be observed. Based on earthquake information, such as seismic center, magnitude and observed seismic intensity, the arrival time and height of a tsunami can be predicted. It often approaches land within a few minutes for an earthquake occurring near the coast and nearly a day for an earthquake occurring at the opposite side of the ocean. When a tsunami is forecasted to hit an area and potentially break a section of the network, network operators can execute a disaster avoidance control algorithm to relocate service network functions to a disaster-free area to avoid or reduce service network disruption. Since most large cities in Japan are near the sea, the physical network cables are therefore mainly installed along the coast line. The probability of a tsunami reaching and destroying a network cable is hence considerable. This motivated us to carry out this research.

In this study, we concretely investigated the relocation of virtual links in virtual private networks (VPNs) as a use case. Based on the evaluated disconnection probability between two given nodes, virtual links are rerouted to minimize the potential traffic loss when tsunamis arrive. In this research, we
use tsunami as a use-case. However, our proposed relocation method can be applied to other disasters of which the arrival time can be predicted in advance.

The rest of the paper is organized as follows. A review of related work is presented in Section II, a problem formulation is presented in Section III, our proposed heuristic algorithms are introduced in Section IV, a performance evaluation is discussed in Section V and a conclusion is given in Section VI.

II. RELATED WORK

Disaster management in communication networks has become increasing important because the number of natural disasters that may severely damage networks is growing, and at the same time human dependence on communication network is strengthening.

Two popular approaches to disaster management are protection (also called provisioning) and restoration (also called recovery). A thorough survey of these disaster management approaches was conducted [3]. However, these approaches imply that we cannot escape from natural disasters. Therefore, spare resources should be provided to maintain service networks and fast recovery mechanisms are required to minimize the discontinuity of services.

Network applications including grid- and cloud-computing services can be implemented by embedding VNs over physical infrastructure. The protection and restoration problems have evolved to a survivable VN mapping (SVNM) problem [8]–[18], which attempts to protect and recover service networks from failures. Yu et al. proposed four heuristic algorithms for VN mapping, which guarantee 100% recovery against an arbitrary regional failure [8]. Liu et al. proposed two mapping algorithms, minimum link risk prior selection and asymmetric parallel flow allocation, to embed VNs against multiple regional failures [9]. The advantage of their study is to investigate multiple regional failures, which is more adequate regarding huge natural disasters.

Meixner et al. developed disaster-resilient and post-disaster-survivable VN mapping algorithms using a probabilistic disaster model to reduce the expected VN disconnections and capacity loss [10]–[13]. With the post-disaster mapping algorithm, they take into account cascading failures, which may be consequences of a massive disaster. Rahman and Boutaba proposed a mapping algorithm using protection and restoration policies that take into account service level agreement (SLA) to increase long-term business profit of infrastructure provider [14]. Guo et al. [15] investigated a failure dependent protection scheme, in which each primary facility node would have a different backup facility node, as opposed to the failure independent protection. They introduced a so-called Enhanced-VN, which has one node more than requested to protect the original VN. The embedding algorithm is applied to the Enhanced-VN instead of the requested VN. A similar idea was proposed by Yu et al. [16]. However, they investigated a K-redundant scheme rather than the 1-redundant scheme. Guo et al. proposed two survivable mapping schemes, shared on-demand and shared pre-allocation backup, which make better use of substrate resources than the dedicated backup scheme without sharing, [17]. Similarly, Yeow et al. proposed a survivable mapping algorithm based on shared-backup resources to reduce the physical footprint of virtual backups while guaranteeing certain reliability [18]. In [19], Eriksson et al. proposed a framework that uses historical data of natural disasters to evaluate risk routes and to inform network operators with respect to backup paths, route changes and provisioning recommendations.

Disaster avoidance control [4], [6], which aims to migrate network functions and services away from forecasted disaster areas, is a new approach to disaster management. Unfortunately, research on this area is still under explored. In this study, we developed algorithms for disaster avoidance control, which attempts to relocate service networks to avoid disaster areas. In particular, this is the first step of disaster avoidance control against tsunamis and is also the first attempt that takes into account concrete disasters.

III. DISASTER AVOIDANCE CONTROL - MODEL AND PROBLEM FORMULATION

A. Background Information

In this study, we considered regional/nation-wide networks, in which network cables are installed in an underground duct and network nodes are located in a large network building with its own power generator. It is unlikely that the network buildings, which are constructed to withstand the largest earthquake, would be destroyed by earthquakes and tsunamis. Therefore, we considered only link failures.

In Japan, a duct is usually sealed against water. When an earthquake occurs, the duct with cables can be in one of three states: disconnected, damaged (unsealed), or normal. The “disconnected” state means the disconnection of cables in the duct. Thus, service networks will be disrupted. In this case, a recovery mechanism must be carried out to restore the networks. If the duct is just damaged (unsealed), the cables in the duct and the network using them still work. However, if a tsunami hits the cracked point of a damaged duct, the cables in the duct will be disconnected due to water leakage. The corresponding network link will consequently fail. Fortunately, a tsunami arrives within several minutes or hours after an earthquake occurs. The approximated arrival time and height of a tsunami are predictable, although its actual height can be different depending on the local terrain. Therefore, network operators can take appropriate actions to migrate VPN routes away from the tsunami disaster area.

To migrate VPN routes away from a disaster area, the failure probability of each network cable when a tsunami arrives has to be estimated. This is explained in the next sections.

B. Estimation of Damage

After a large earthquake occurs, by temporary monitoring and measuring, network operators can detect the disconnection of a cable and damage to ducts. However, the exact place where the duct is damaged and the number of damaged points...
are unknown. If damaged parts are in a tsunami-free area, the damage will not result in a disconnection. Otherwise, the network link will be disconnected when tsunamis hit the duct. Therefore, we need to evaluate the damage along the duct/cable to estimate the failure probability of the link. (In the remainder of this paper, we focus on the damaged or normal parts of the network by removing the disconnected links in advance. Therefore, failure means the failure of the cable in the damaged duct.)

After an earthquake, the Japan Meteorological Agency makes an announcement of the earthquake intensity of each city/town. Since we maintain geographical route information of the cable network, we know which part of the network cable will be affected by the earthquake of a particular intensity. (In practice, we can use the geomorphological surface structure data for each square area of 250m × 250m to adjust the earthquake intensity of the square in a city/town.)

The probability that a cable is damaged at location \( x \) depends on the earthquake intensity, terrain conditions at \( x \) such as a river (along a bridge crossing a river), construction methods, such as aerial or underground installation, and network component, such as the type of ducts/cables and years of use. NTT has collected data on the failure probability of network components under various earthquake intensities over several years. Hence, the failure/damage probability of each type of network component under certain conditions including earthquake intensity can be estimated. Consequently, the damage probability \( \beta(x) \) of a link segment at \([x, x + dx]\) due to an earthquake can be obtained using such field data.

As a result, the link segment at \([x, x + dx]\) will be damaged and disconnected by a tsunami with probability \( \beta(x)1(x \subset \Omega) \), where \( \Omega \) is the area covered by the forecast tsunami and \( 1(x) \) is the indicator function, which is 1 if \( x \) is true and 0 otherwise. We assume that the forecasted tsunami area has no forecast error, but a numerical example discussed later will demonstrate the effect of such error.

C. Estimation of physical link failure probability

Let \( D \) be the event in which the link is damaged during an earthquake and \( F \) be that in which the link will fail due to tsunamis. A link might be damaged by an earthquake but not necessary failed when a tsunami comes. Vice-versa, a failed link due to tsunami is surely damaged by the earthquake. According to Bayes’s theorem, the conditional probability that a link fails given that it is damaged is:

\[
Pr(F|D) = \frac{Pr(D|F) \cdot Pr(F)}{Pr(D)}
\]

Because the cable will definitely be damaged if it fails due to tsunamis, \( Pr(D|F) = 1 \). Therefore,

\[
Pr(F|D) = \frac{Pr(F)}{Pr(D)}
\]

Assume that damage along a cable independently occurs with probability \( \beta(x) \) \((\forall x \in \mathcal{L})\). Thus, a failure independently occurs along a cable with probability \( \beta(x)1(x \subset \Omega) \forall x \in \mathcal{L} \) where \( \mathcal{L} \) is the geographical route of the network cable. Geographical dependence of failures are modeled by the geographical dependence of \( \beta(x)1(x \subset \Omega) \), although failures are assumed to independently occur.

We have:

\[
Pr(F) = 1 - \prod_{x \in \mathcal{L}} (1 - \beta(x)1(x \subset \Omega))
\]

\[
Pr(D) = 1 - \prod_{x \in \mathcal{L}} (1 - \beta(x))
\]

Hence,

\[
Pr(F|D) = \frac{\prod_{x \in \mathcal{L}} (1 - \beta(x)1(x \subset \Omega))}{1 - \prod_{x \in \mathcal{L}} (1 - \beta(x))}
\]

(1)

According to these link failure probabilities, we have an overview in which part of the network is under risk of failure and another part is risk-free. Thus, a disaster avoidance control algorithm will be executed correspondingly.

D. Problem Formulation

Given a physical network with a set of nodes \( \mathcal{V} \) and links \( \mathcal{E} \). Each physical link \( e \in \mathcal{E} \) has a capacity of \( C_e \). The failure probability of each physical link, \( \beta(e) \) \((e \in \mathcal{E}) \), is given by Eq. (1) where \( \mathcal{L} \) is a route of \( e \in \mathcal{E} \).

Assume that a set of virtual links \( S = \{s\} \) is embedded on the given physical network. Each virtual link \( s \in \mathcal{S} \) has a required bandwidth of \( b_s \) and a set of possible physical routes \( \mathcal{R} = \{r\} \). If \( s \in \mathcal{S} \) is embedded on a physical route \( r \in \mathcal{R} \), its failure probability \( f_s^r \) under the assumption of independent failure is calculated as:

\[
f_s^r = 1 - \prod_{e \in r} \beta(e)
\]

(2)
where $e \in r$ indicates that $e \in \mathcal{E}$ belongs to $r \in \mathcal{R}^s$.

If two physical links (e.g. $e_1, e_2$) partly share the same duct, their failures will no longer be independent; hence, Eq. (2) cannot be directly applied. Assume that the shared part is $e^*$, we first need to evaluate the failure probability of three independently failed physical parts, which are $(e_1 - e^*), (e_2 - e^*)$, and $e^*$ (using Eq. (1)), where $(e_1 - e^*)$ and $(e_2 - e^*)$ are the non-shared parts of $e_1$ and $e_2$, respectively. Equation (2) is then applied to these three independently failed parts.

From the given physical network topology and set of virtual links, we compute a parameter $\delta^r_{e,c}$. $\delta^r_{e,c} = 1$ if $e \in \mathcal{E}$ belongs to $r \in \mathcal{R}^s$, and $\delta^r_{e,c} = 0$ otherwise.

Let $x^r_e$ be a binary variable that takes the value of 1 if virtual link $s \in S$ is embedded on $e \in \mathcal{R}^s$ and 0 otherwise.

$$\min \sum_{s \in S, e \in \mathcal{R}^s} x^r_e \cdot f^r_e \cdot b^s$$

$$\text{subj. to } \sum_{e \in \mathcal{R}^s} x^r_e = 1 \quad \forall s \in S$$

$$\sum_{s \in S, e \in \mathcal{R}^s} x^r_e \cdot \delta^r_{e,c} \cdot b^s \leq C_e \quad \forall e \in \mathcal{E}$$

The objective is to minimize the expected traffic loss when a tsunami arrives. The first constraint (Eq. 4) is to guarantee that all virtual links are successfully embedded on the physical network. The second constraint (Eq. 5) is a capacity constraint that ensures that the total required bandwidth of virtual links embedded on a physical link does not exceed the capacity of the physical link.

With the above formulation, it is assumed that all virtual links can be reconfigured. This will result in the best solution. However, the number of virtual links is often very large and network operators may want to reconfigure only virtual links that are under risk of disconnection when a tsunami arrives. In this case, the set of reconfigured virtual links becomes $S^∗(\subset S)$, which contains only virtual links in tsunami-affected areas, and the capacity constraint Eq. 5 is rewritten as:

$$\sum_{s \in S^∗, e \in \mathcal{R}^s} x^r_e \cdot \delta^r_{e,c} \cdot b^s \leq C_e^s \quad \forall e \in \mathcal{E}$$

where $C_e^s$ is the residual link capacity (the bandwidth allocated to risk-free virtual links is subtracted from link capacity $C_e$).

IV. HEURISTIC ALGORITHMS

While optimization solvers, such as Cplex [20] and Glpk [21], can obtain the optimal solution for virtual link reconfigurations, the integer linear programming (ILP) formulation presented in Section III-D exhibits a high computation time because the problem is NP-hard. Therefore, we propose the following heuristic algorithms to provide approximate solutions in a reasonable time, which is more applicable for large networks.

A. Iterative Reconfiguration

In this section, we present our two heuristic algorithms based on an iterative approach, namely, step-by-step searching for a new physical route of each virtual link in a certain order. The first algorithm is called Relocation Before Release (RBR) and the second is Reconfiguration After Release (RAR).

Algorithm 1: Reconfiguration Before Release (RBR)

Data: Physical network topology $\mathcal{G} = \{V, \mathcal{E}\}$ and link capacity $C_e (e \in \mathcal{E})$, failure probability of each physical link $\beta(e) (e \in \mathcal{E})$, virtual links $S = \{s\}$ with current physical route $r_s \in \mathcal{R}^s$ and required bandwidth $b^s (s \in S)$

Result: New embedding solution for $S$

Select virtual links in disaster-prone areas $S^∗ = \{s^∗\}$

Compute expected traffic loss of each virtual link $T^r_e = f^r_e b^s (s^∗ \in S^∗)$

Sort virtual links by $T^r_e$ in descending order

for $s^∗ \in S^∗$ do

Find all physical routes $\mathcal{R}^s = \{r\}$ for $s^∗$

Sort physical routes by their number of hop-counts in ascending order

for $r \in \mathcal{R}^s$ do

if $r$ has enough bandwidth for $s^∗$ then

Compute expected traffic loss $T^r_e$ of $s^∗$ if embedded on $r$

Select $r (r \in \mathcal{R}^s)$ that has lowest expected traffic loss as the final embedding solution for $s^∗$

Update residual capacities of physical links

end if

end for

end for

The main difference of the two algorithms is that with RBR, we keep the current mapping and step by step reconfigure each virtual link if a better solution can be found, while with RAR, we assume that no virtual links in disaster areas are embedded and then step-by-step search is conducted for the best embedding solution for each virtual link. With RBR, we can guarantee that all virtual links will be embedded because if we do not find a better route for a virtual link, it can stay with the old embedding solution. With RAR, this does not apply. Since we release all virtual links under risk, links that are embedded at the last step may not find an available route. However, we expect that if all virtual links can be embedded, RAR will result in a better solution because there are more free link capacities during embedding, which lead to more possibilities to find a good solution. This effect is investigated in more detail in Section V.

B. Grouping Reconfiguration

With RBR and RAR, we relocate each virtual link separately. In fact, the number of virtual links can be quite large. However, the number of source-destination pairs in a realistic physical network is not that large. Let us consider a realistic regional network in Japan, which has a ladder topology of
Algorithm 2: Reconfiguration After Release (RAR)

**Data:** Physical network topology \( G = \{ V, E \} \) and link capacity \( C_e (e \in E) \), failure probability of each physical link \( \beta(e) (e \in E) \), virtual links \( S = \{ s \} \) with current physical route \( r_s \in R^* \) and required bandwidth \( b^s (s \in S) \)

**Result:** New embedding solution for \( S \)

Select virtual links in disaster-prone areas \( S^* = \{ s^* \} \)

Sort virtual links by \( b^s \) in descending order

Release \( s^* \in S^* \) from the physical network and update residual capacities of physical links

for \( s^* \in S^* \) do

Find all physical routes \( R^* = \{ r \} \) for \( s^* \)

Sort physical routes by their number of hop-counts in ascending order

for \( r \in R^* \) do

if \( r \) has enough bandwidth for \( s^* \) then

Compute expected traffic loss \( T_e^* \) of \( s^* \) if embedded on \( r \)

Select \( r (r \in R^*) \) that has lowest expected traffic loss as the final embedding solution for \( s^* \)

Update residual capacities of physical links

end

end

end

Algorithm 3: Grouping Reconfiguration Algorithm

**Data:** Physical network topology \( G = \{ V, E \} \) and link capacity \( C_e (e \in E) \), failure probability of each physical link \( \beta(e) (e \in E) \), virtual links \( S = \{ s \} \) with current physical route \( r_s \in R^* \) and required bandwidth \( b^s (s \in S) \), grouping factor \( \nu \)

**Result:** New embedding solution for \( S \)

Select virtual links in disaster-prone areas \( S^* = \{ s^* \} \)

Group virtual links by the given grouping factor \( \nu \)

Reconfigure equivalent virtual links using either RBR, RAR, or Optimization solvers

For this algorithm, we introduce a parameter called grouping factor \( \nu \). This indicates the number of virtual links that are grouped together to form an equivalent virtual link. Higher \( \nu \) results in a smaller number of equivalent virtual links, which leads to a lower computation time. In certain cases, we can use an LP solver to solve the reconfiguration problem if the number of equivalent virtual links is not so large. However, we also expect that the performance will decrease with an increasing grouping factor.

The complexity of all three heuristic algorithms is \( O(|S||R^*|) \).

V. PERFORMANCE EVALUATION

A. Simulation scenarios

In this paper, we focus on the regional network of 12 nodes and 14 links in Fig. 2.

We assume all physical links have the same capacity of 100 units.

12 nodes and 14 links. In this network, there are 66 source-destination pairs. Each pair has an average of 5.4 (a maximum of 8) different physical routes. Therefore, the number of distinct virtual links in terms of source, destination and physical route in this example is just approx. 355, which seems much smaller than a typical number of virtual links. Based on this observation, we propose another heuristic algorithm called “grouping reconfiguration”. The main idea is to group virtual links that originate and terminate at the same nodes and follow the same physical route to form an equivalent virtual link. The reconfiguration process will be done on the equivalent virtual links, instead of each actual virtual link.

![Fig. 2. Regional network for numerical studies](image)

We consider three earthquake types that will occur with probability of at least 1% according to J-SHIS [23]. These earthquakes and their associated tsunamis will occur at different areas with different intensities and can damage various physical links. The devastating effect of tsunamis can be found at [22].

In reality, after an earthquake occurs, network operators will monitor and measure network status to determine which link is disconnected or damaged. They can use their field data, together with the earthquake and tsunami hazard maps, and geographical route information to estimate the failure probabilities of physical network links. In this study, we just used hazard maps and route information to determine the links in disaster-affected areas. Due to the lack of concrete field data and for the sake of simplicity, we assume that \( \beta(e) \) is 0.5 for all \( e \).

Virtual links are randomly generated as follows: 1) source and destination nodes are evenly distributed in the network; 2) the required bandwidth of a virtual link is uniformly distributed in \([0.01 - 0.1]\); 3) virtual links are embedded on the shortest available physical routes. We generated from 1000 to 9000 virtual links, which leads to different traffic load in the...
network. Traffic load is defined as the ratio of total allocated bandwidth to the total link capacity. Each result is obtained by taking the average results from 1000 same experiments.

B. Reduction in expected traffic loss

We first evaluated the performance of our proposed algorithms in terms of expected traffic loss under three earthquakes with their associated tsunamis. The effect of the three earthquakes on the physical network is as follows:

[Scenario 1] Local earthquake 1 occurs to the east of the network. No disconnected links but three links (1-12), (12-11) and (11-10) are damaged in the forecasted tsunami area.

[Scenario 2] Local earthquake 2 occurs to the east of the network. No disconnected links but two links (12-11) and (11-10) are damaged in the forecasted tsunami area.

[Scenario 3] Huge earthquake occurs to the south of the network. No disconnected links but six links (1-12), (12-11),(11-10), (1-2), (2-3) and (3-4) are damaged in the forecasted tsunami area.

In this experiment, we applied RBR. The result are illustrated in Fig. 3. The y-axis shows the ratio of the expected traffic loss after reconfiguration compared to that when no reconfiguration was carried out. Obviously, when the network load is low, it is easy to find alternative routes that go through disaster-free areas. Consequently, the expected traffic loss after reconfiguration is also low. In contrast, when the load is high, the reduction in expected traffic loss decreases because there are less free resources for relocating virtual links. From the network load of approx. 70% or more, it is almost impossible to migrate virtual links away from the disaster areas. This implies that to make disaster avoidance control effective, network operators should provision enough redundant resources for migration.

In this scenario, even when the network load is very low (only 10%), it is still impossible to migrate all virtual links away from the disaster areas. This is because several virtual links originate or terminate at node 11 (or node 2 & 3 in case of huge earthquake), which is connected by two links that are both under risk of failure. Therefore, it is impossible to make virtual links originate and terminate at node 11 risk-free.

The results in this simulation reveal that for the investigated regional network, it is impossible to migrate all virtual links to disaster-free areas. Nevertheless, this gives some hints to network operators on how to further improve network robustness. A solution can be installing an additional link connecting to node 11 that the tsunami does not reach.

C. Comparison of heuristic algorithms

The difference in the solution with our proposed heuristic algorithms and the optimal solution obtained using Glpk solver are presented in Fig. 4. In this experiment, we considered Local earthquake 1 as a use case. Since there are thousands of virtual links, it is impossible to directly use Glpk solver to solve the problem. We therefore applied grouping reconfiguration to reduce the number of virtual links so that the problem can be solved by using LP solver. We used a very high grouping factor, which groups all virtual links of the same source-destination and physical paths into an equivalent virtual link. This also applies for the results of grouping reconfiguration shown in Fig. 4. In this experiment, the grouping approach used RBR to do the reconfiguration.

Figure 4 shows that at low network load (≤ 30%), all algorithms performed similarly. This is easy to understand, because there are plenty of free resources for the migration, so it is possible to find the best alternative routes for all virtual links. Hence, the obtained solution is also the optimal solution. When the load increased, we observed a clear difference in the solution with the heuristic algorithms and the optimal solution. The difference was approx. 10% at a load of approx. 65% and above.

At middle load (40-50%), RAR performed better than RBR. The difference increased with network load. This is because RAR releases resources of all virtual links before re-embedding them, which leads to higher flexibility during
the reconfiguration. However, when the network load was very high, RAR resulted in infeasible solutions, namely some virtual links could not be embedded on physical networks. As a result, RAR should be chosen when the network is middle loaded and too large to be solved with an LP solver. However, the choice of algorithm also depends on the requirements of VNs. If "connect before break" is required to maintain service continuity, we have to use RBR. The optimal solution with Glpk also cannot be used in this case, since this approach implies that all virtual links should be released before being re-embedded.

It is surprising that grouping reconfiguration performed very similarly to RBR. This can be explained as follows. Let us consider a network at a load of approx. 70%. We generated 7000 virtual links, among which approx. 3000 virtual links needed to be reconfigured. In this regional network, there were roughly 300 distinct virtual links. Using the grouping approach, an equivalent link will contain approx. 10 virtual links on average and its bandwidth will be approx. 0.5 (since each virtual link has a bandwidth uniformly distributed in \([0.01-0.1]\)). At a network load of 70%, there were approx. 30 units of free capacity on each link on average. This is much larger than the required bandwidth of an equivalent virtual link. Therefore, we observed almost no difference between the two algorithms. Furthermore, the greedy behavior of RBR may even favor the grouping approach. This explains why at a high load (higher than 70%), grouping reconfiguration performed slightly better than RBR without grouping.

D. Grouping reconfiguration with different grouping factors

In the previous sections, we mentioned that grouping reconfiguration performed very similarly to RBR. We therefore expect no notable difference in performance when applying different grouping factors. However, if we use an LP solver to solve the problem, it is expected that the higher the grouping factor, the worse the performance.

Figure 5 presents the performance of grouping reconfiguration with three different grouping factors (non-grouping (factor 0), group 2 virtual links into one (factor 2), and group all identical virtual links) at high load. In this experiment, we generated 700, 800, and 900 virtual links with bandwidth in the range \([0.1-1]\). We reduced the number of links so that we could obtain the solution with Glpk solver, but we increased the virtual link bandwidth to achieve a high network load. Links were grouped randomly; therefore, equivalent links had the bandwidth in \([0.2-2]\).

As expected, there was a difference in performance of the algorithm with different grouping factors. The difference is relatively small (approx. 2.5% for grouping and non-grouping). The reason is due to the uniform distribution of source-destination pairs and requested bandwidth of virtual links, as discussed in Section V-C.

In fact, the difference in the performance among different grouping factors depends strongly on the distribution of source-destination pairs of virtual links. If some source-destination pairs have a significantly higher number of requested virtual links, the performance gap between high and low grouping factors is expected to be clearly larger. To demonstrate this, we carried out an experiment in which the number of virtual links between nodes (1) and (10) takes 50% of the total requested virtual links, and the remaining virtual links are uniformly distributed among other pairs. The results are shown in Fig. 6. As expected, the performance gap was larger than 10% in the range of middle traffic load. In this case, it makes sense to choose different grouping factors according to the available computation time.

E. Effect of Estimation Error of Link Failure Probability on Performance

The failure probability for a damaged link is estimated on the basis of collected field data and predicted natural hazard information. These data are never 100% accurate; hence, the estimated link failure probability will have errors. In this section, we investigate the effect of estimation error on the
performance of RBR. The effect on the performance of RAR and grouping reconfiguration is expected to be similar.

We considered Scenario 1 (as described in Section V-B). For a given link failure probability, we added an estimation error, which is uniformly distributed in $[-\epsilon, \epsilon]$. We also generated two cases in which a link under failure risk is considered as risk-free and a risk-free link is considered to fail with high probability. The reconfiguration was done on the basis of failure probabilities with errors. We then generated failed links according to the given probabilities (without error) and calculated the traffic loss for each reconfiguration.

![Fig. 7. Performance of RBR with different estimation errors of failure probability](image1)

The results are shown in Fig. 7. They revealed that the accuracy of failure probability had little effect on the performance of the proposed algorithms in terms of traffic loss. The important factor is to accurately detect links under failure risk and their relative failure probabilities. This is because the algorithms always attempt to relocate virtual links to disaster-free areas regardless of whether the failure probability of physical links is small or large, or if not possible, to areas where physical links fail with lower failure probabilities. Therefore, when a link may fail but is considered as risk-free link, virtual links embedded on it will not be migrated and even worse, other virtual links may be relocated to this link, which leads to further loss. Similarly, if a link does not fail but is estimated to fail with a high probability, virtual links on it will be relocated to lower risk links, if it is not possible to relocate them all to risk-free links. This explains why traffic loss in these cases is much higher.

We expect that detecting a potentially broken cable due to a tsunami and its degree of failure (failed with high probability or low probability) is much simpler than estimating its accurate failure probability. This proves that our proposal is practically efficient.

**F. Computation time**

In this section, we investigate the computation time of our proposed algorithms. We carried out an experiment on two networks, one regional network of 12 nodes and 14 links, and one nation-wide network of 18 nodes and 21 links under the assumption that three links are under risk with a failure probability of 0.5. Obviously, the computation time depends on the number of virtual links that need to be relocated and the number of alternative physical routes of the virtual links, which strongly depends on network topology.

![Fig. 8. Computation time for different networks and number of virtual links](image2)

The results are shown in Fig. 8. Only the computation time of RBR is shown in the graph but that of RAR is expected to be the same. The computation time of our heuristic algorithms is predictable and increases with the number of virtual links and physical paths. With grouping reconfiguration (in which all identical virtual links are grouped into one equivalent link), the computation time of the heuristic algorithms reduced significantly and remained almost stable because the number of equivalent virtual links stayed almost constant and did not depend on the number of actual virtual links.

Without grouping reconfiguration, Glpk solver running on a quad-core 2.2GHz processor computer hardly obtained a solution within an hour in case of thousands of virtual links. With grouping reconfiguration, the number of variables decreased significantly and a solution was obtained within minutes. However, the computation time increased drastically with the number of virtual links, even though we grouped all identical virtual links, which resulted in an almost constant number of equivalent virtual links. When the number of actual virtual links increased, the traffic load also increased. At high network load, there might be many similar embedding solutions than that at low network load, which would result in a long searching time for the branch-and-bound algorithm (algorithm implemented in Glpk).
G. Which algorithm should be used?

There are several factors that affect the decision of choosing an appropriate algorithm. We now introduce a framework, which takes the following factors as inputs for the decision making process:

- Requirements of the reconfiguration process
- Network load
- Available computation time
- Network topology and number of reconfigured virtual links

If it is required to always set up a new virtual link before tearing down the old one, RBR must be used even though it yields the lowest reduction in expected traffic loss at a high network load.

If the network load is low (< 30%), we can apply RBR (or RAR) with grouping reconfiguration, since the performance is the same as the optimal one but the computation time is small. If the network load is above 40%, the computation time should be taken into account to achieve the best possible performance.

The available computation time is the predicted arrival time of tsunamis subtracted by the time required to finish the reconfigurations. The computation time strongly depends on the network topologies and the number of relocated virtual links as illustrated in Fig. 8. If the computation time is critical, i.e., a tsunami is forecasted to arrive within minutes, grouping reconfiguration combined with either RAR or RBR should be used. However, if a tsunami is predicted to arrive in a day, the grouping approach combined with an optimization solver can be used in order to obtain a better solution than with the heuristic algorithms. Thanks to the development of cloud computing, which can provide very high computational power on demand, the problem is expected be solved within a reasonable time. In addition, a better LP solver, such as Cplex, can also solve the problem faster than Glpk.

VI. Conclusion

We proposed disaster avoidance control algorithms against tsunamis, which contribute to the realization of the DFN concept. Our focus was to develop efficient algorithms for migrating virtual links away from tsunami-affected areas. The objective of the algorithm was to minimize the expected traffic loss, given the information on the devastating effect of a tsunami. We formulated the problem as an ILP problem and proposed three heuristic algorithms (RAR, RBR, and grouping reconfiguration) to solve it in case of large networks. The simulation results revealed that when the network is lightly loaded, all algorithms perform similarly, and when the network is heavily loaded, the algorithm resulting in a lower expected traffic loss often takes longer time. On the basis of predicted arrival time of tsunamis and network status, network operators can choose an appropriate algorithm for reconfiguration, which performs well but also fast enough to avoid tsunamis.

This study showed that RAR (and optimization solver) performs better than RBR but does not guarantee no disrupted service during the reconfiguration. However, we expect that if the reconfiguration of virtual links is done in an appropriate order, we can obtain a good solution while still ensuring the "connect before break" requirement. Therefore, a future research direction can be to guarantee no service disruption during the reconfiguration process while minimizing expected traffic loss at the same time.

REFERENCES

[16] H. Yu, et al., Cost efficient design of survivable virtual infrastructure to recover from facility node failures, in IEEE International Conference on Communications (ICC), Kyoto, Japan, June 2011.
Building a Low Latency Linux Software Router

Alexander Beifuß\(^1\), Torsten M. Runge\(^1\), Daniel Raumer\(^2\), Paul Emmerich\(^2\), Bernd E. Wolfinger\(^1\), Georg Carle\(^2\)

\(^1\)Universität Hamburg, Department of Computer Science, Telecommunications and Computer Networks
{beifuß|runge|wolfinger}@informatik.uni-hamburg.de
\(^2\)Technische Universität München, Department of Computer Science, Network Architectures and Services
{raumer|emmerich|carle}@in.tum.de

Abstract—Packet processing (e.g. routing, switching, firewall) with commodity hardware is a cost-efficient and flexible alternative to specialized networking hardware. On commodity hardware the CPU typically becomes the bottleneck in packet processing. However, in well-known QoS mechanisms (e.g. DiffServ), the outgoing link is assumed to be the bottleneck. This limitation is unfavorable, in particular for latency-sensitive applications (e.g. VoIP, video conferencing, online gaming). Thus, we propose and implement a QoS concept for a Linux software router to prioritize latency-sensitive traffic at the incoming network interface. Our testbed measurements show that our prototype implementation improves the packet processing w.r.t. the latency of latency-sensitive traffic even under high traffic loads.

Keywords—commodity hardware; packet processing; quality of service; low latency; NIC driver

I. INTRODUCTION AND RELATED WORK

Since decades, packet processing with specialized networking hardware like hardware routers has been the state of the art. Nowadays, cost-efficient commodity hardware has benefited from many optimizations (e.g. multi-core CPUs, multi-queue NICs, DCA, DMA, PCIe) to exploit parallelism in the packet processing with software [1], [2]. By this way, the packet processing software like the network interface card (NIC) drivers and the operating systems (OS) also received several enhancements like interrupt moderation for saving CPU cycles in high-load situations. Thus, so-called software routers became a potential replacement for specialized networking hardware in many cases (e.g. campus networks). In contrast to hardware routers, software routers are more cost-efficient and more easy to extend, which allows for fast adaptation and introduction of new features.

Many research projects show that the CPU constitutes the bottleneck in the packet processing with commodity hardware [1]–[4]. However, there is only limited support for differentiated ingress traffic treatment regarding the packet processing by the CPU bottleneck. For instance, Linux only supports packet filtering but no class-based traffic differentiation at the ingress. Unfortunately, the well-known QoS approaches like DiffServ [5] and IntServ [6] are only applicable as queuing disciplines (Qdisc) at the egress because the outgoing link is assumed to be the bottleneck. Consequently, the absence of ingress traffic classification and prioritization might have a negative impact on applications which have specific quality of service (QoS) requirements (e.g. high bandwidth, low latency, low jitter). Additionally, this problem is strengthened by the increase of real-time traffic such as voice over IP (VoIP), video conferencing, video on demand (VoD) or online gaming.

We argue that QoS-sensitive traffic should also be prioritized at the ingress network interface to achieve QoS-aware packet processing with commodity hardware. In this manner, we presented and evaluated a new QoS concept in our previous work [7], [8], in which we simulated the ingress QoS packet processing and showed improvements for QoS-sensitive traffic. In our QoS concept, the ingress traffic is classified by the NIC into dedicated receive queues (Rx rings). By this way, the traffic classification is offloaded from the CPU to the NIC. Then, QoS-sensitive traffic can be prioritized according to a configurable scheduling strategy. This QoS concept is also applicable for other packet processing systems (e.g. switches, load balancers, firewalls, end systems).

Furthermore, optimized networking frameworks like netmap [9] or DPDK [10] were proposed to bypass the Linux networking stack. These approaches achieve high throughputs with batching of multiple packets and process them more efficiently in the user space (e.g. preallocation, zero-copy). However, these approaches usually show the drawback of insufficient packet latency. To support low latency packet processing, Ueda et al. [11] introduce dedicated interrupt requests (IRQ) for the reception of real-time traffic. But this additional IRQ overhead leads to a strong decrease in the overall system performance (e.g. maximum throughput). Furthermore, Cummings and Tamir [12] propose a busy poll based concept for network interfaces which can be set by the application. Their approach is designed for end systems but it causes high CPU utilization and prevents the CPU from energy saving, even at low traffic loads.

In this paper, we present and evaluate a prototype implementation of our QoS concept for a Linux software router. As a proof of concept, the tested measurements show that with our prototype the packet latency of latency-sensitive traffic remains very low, even under high traffic loads. This is accomplished without significant performance degradation, e.g. in terms of the achievable maximum throughput.

The remainder of the paper is structured as follows. First, the Linux packet processing is described in Section II. We present our QoS concept in Section III. In Section IV, we explain important aspects of our prototype implementation. In Section V, we evaluate our prototype based on testbed measurements. Finally, we summarize the paper in Section VI.
II. LINUX-BASED PACKET PROCESSING

Before version 2.6, the Linux kernel followed an interrupt-driven approach for receiving network data, so that each received packet causes an IRQ. However, the throughput collapses with high offered loads due to the high IRQ effort. This is known as the receive livelock state [13], in which the CPU is only utilized with IRQ handling and has no CPU cycles left for the actual packet processing or other processes. Therefore, Salim et al. [14] presented a new packet reception approach, the so-called NAPI (New API) which was introduced with Linux kernel version 2.6. Today, the NAPI is still a fundamental part of the Linux kernel network subsystem. The NAPI is a hybrid mechanism which combines the advantages of interrupt-driven and poll-driven approaches. In case of a low offered load, the system behaves like an IRQ-driven system where each packet causes an IRQ. Thus, the waiting-time of a packet is rather low which implies low packet latency. At high offered loads, the system rather behaves like a poll-driven system where IRQs are disabled and multiple packets are served in batches. Therefore, CPU cycles are mainly spent for the packet processing at high offered loads which maximizes the achievable throughput. This behavior of the NAPI is achieved as described in the following.

Each network device (aka. QVector, cf. Sec. IV) uses a dedicated IRQ line per CPU core. By default, there is a one-to-one relationship between a device and a packet reception queue (Rx ring). An IRQ which is generated by a device will cause the execution of the interrupt service routine (ISR). A NAPI-compliant driver performs the following tasks:

- The IRQ line of the device is disabled in order to ensure that no further IRQs are generated if packets arrive in the corresponding Rx ring. Nevertheless, received packets are still transferred into main memory via DMA.
- The corresponding device is enqueued in a so-called poll list and further packet processing is scheduled for later execution (by generating a so-called Soft-IRQ), so that the ISR quickly returns. Later, if the Soft-IRQ is handled, the poll list is served by the NAPI in FIFO manner. This means that packets are polled from the enqueued device and are passed to the IP stack for the actual packet processing (e.g. routing).

The polling of a device terminates due to one of the following reasons:

- All backlogged packets of the device have been processed. Then, the device is removed from the poll list and the IRQ line that corresponds to this device is re-enabled.
- A maximum number of packets that corresponds to the device budget (aka. poll size) has been processed but there are still further packets backlogged in the Rx ring of the device, which are waiting for being processed. Then, the IRQ line of this device remains disabled and the device entry is moved to the tail of the poll list again.

III. CONCEPTION OF LOW LATENCY SUPPORT

The NIC drivers and also the Linux NAPI [14], [15] do not support traffic classes. However, this is important for the differentiated treatment of QoS-sensitive traffic. Therefore, in our QoS concept the received packets are classified into traffic classes and accordingly directed into the dedicated waiting queues (Rx rings) of an device. Finally, the packet processing of these Rx rings with QoS-sensitive traffic can be prioritized by the corresponding CPU core. An example of such a low latency (LL) software router with the two traffic classes Real-Time (RT) and Best Effort (BE) is depicted in Fig. 1. In the following, our QoS concept is described in detail.

A. Traffic Classification

Modern network cards have various features to offload specific packet processing tasks from the CPU to the NIC. For example, the Intel Ethernet NIC controller X540 [16] supports multi-queuing (MQ) and receive-side scaling (RSS) to efficiently distribute incoming packets among the available CPU cores. Especially, the Flow Director is a specific NIC hardware filter which allows traffic classification based on MAC or IP header fields, TCP/UDP ports, VLAN tags and even flexible 2 Byte tuples in the first 64 Byte of a packet. Based on this information the NIC is able to classify received packets and sort them into multiple dedicated Rx rings which in turn can be assigned to specific CPU cores via the RSS feature. If a received packet does not match any of the NIC filter rules, then the NIC will direct it into the Rx ring of the traffic class with the lowest priority (e.g. BE). Therefore, the packets are transferred by Direct Memory Access (DMA) into the main memory without any involvement of the CPU. As a consequence, an IRQ to the appropriate CPU core is signaled if the corresponding IRQ line of the device is enabled.

In our QoS concept, these NIC features are exploited by a packet processing system for traffic classification (Classifier). Each device provides a dedicated Rx ring per traffic class. A device uses only one IRQ line. Therefore, this IRQ line is shared between multiple Rx rings to save IRQ overhead and thus CPU cycles.

Deri et al. [17] also used NIC hardware filters to accelerate a traffic analysis framework by reducing the number of packets to the relevant ones only. Their case study showed that although the configuration may increase the complexity of a system, it can improve performance of CPU-based sampling approaches. Tanyingyong et al. described an OpenFlow switch where some matches were offloaded to the NIC [18]. In 2012, they also proposed a fast processing path for a router, where the routing decision for a limited number of flows (typically those with high packet rates) is offloaded to the NIC by using the Flow Director [19].

B. Traffic Prioritization

Well-known QoS mechanisms for differentiated packet treatment (e.g. DiffServ, IntServ) are only supported as a Qdisc on the outgoing network interface. On the incoming network interface, multiple Rx rings (which are associated with the
same CPU core) are simply served in a round-robin manner. So, traffic prioritization on the ingoing network interface is not supported. However, the Rx rings with QoS-sensitive traffic of the ingoing network interface should be preferred to support differentiated packet treatment. Thus, our QoS concept introduces Ingress QoS. By this way, a scheduling strategy (cf. Sec. III-C) is applied for the ingress traffic to prefer RT Rx rings w.r.t. the actual packet processing by the CPU core. Additionally, a Qdisc can be applied on the egress network interface to prioritize the RT traffic (Egress QoS). The extensions are highlighted in Fig. 1.

In the following our QoS concept, as illustrated in Fig. 2, is described in detail. Firstly, the Linux NAPI checks the poll list whether devices were added for packet processing. Multiple devices in the poll list are served in a round-robin manner corresponding to the device budget. The device budget (usually 64 packets for GbE adapters) limits the number of packets which can be processed in a row by a device.

As mentioned in Section II, there is a one-to-one relationship between a network device and an Rx ring. Instead of a single Rx ring per device and per CPU core, our concept provides a dedicated Rx ring per traffic class. Consequently, a specific device uses a dedicated Rx ring per traffic class. The packet processing of such a device is controlled by the device budget and dedicated Rx rings. While the device budget still defines the number of packets that may be consecutively polled from the device, the ring budget specifies the number of packets per Rx ring that can be processed at maximum before switching to another Rx ring of the same device. In our concept, the ring budget is dependent on the scheduling strategy (cf. Section III-C) and can be adapted according to the priority of the associated traffic class.

Finally, it is checked whether all Rx rings are empty. If this is true, then the NAPI removes this device entry from the poll list and the IRQ line is re-enabled. Otherwise, if any of the Rx rings still contains packets, it is checked whether the device budget was exhausted. If the device budget is not exhausted, all Rx rings are served again. Otherwise, if the device still has packets to be processed then the NAPI moves the device to the tail of the poll list for later processing.

Fig. 1: Ingress and egress QoS support of a multi-core software router with dedicated Rx rings for RT and BE traffic

In principle, our QoS concept supports an arbitrary number of traffic classes. Through the differentiated packet treatment at the ingoing network interface, we expect that the latency of latency-sensitive applications is significantly reduced. In Section V, the effects of our QoS concept with two traffic classes (RT, BE) are evaluated based on real testbed measurements.

C. Scheduling Strategies

In this paper, we consider the following scheduling strategies for our ingress QoS concept (cf. Table I).

- **Single Queue (SQ)**: The incoming packets are not classified but directed to one Rx ring per device. Thus, all packets are served in a FIFO manner which represents the state of the art without traffic classification and prioritization.
- **Round-Robin (RR)**: The incoming packets are classified by the NIC and directed into the Rx ring of a dedicated device. This represents the state of the art with NIC-based traffic classification but without traffic prioritization.
- **Low Latency Round Robin (LL-RR)**: The incoming packets are classified by the NIC into dedicated Rx rings per device. Each device has a dedicated Rx ring per traffic class. All Rx rings have the same ring budget, thus, this represents our QoS concept without prioritization.
- **Low Latency Weighted Fair Queuing (LL-WFQ)**: Similar to LL-RR, but the budget of an Rx ring corresponds to the traffic class priority.

<table>
<thead>
<tr>
<th>Traffic classification</th>
<th>SQ</th>
<th>RR</th>
<th>LL-RR</th>
<th>LL-WFQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared IRQ</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Traffic prioritization</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

TABLE I: Scheduling strategies
D. Theoretical Considerations

In this section we estimate the worst case latency of our QoS concept by means of an elementary queuing model. The model consists of a server (CPU core) which processes two queues in an alternating manner (namely, the RT ring and the BE ring) as illustrated in Fig. 3. A specific ring budget exists for serving each of both queues. After reaching this ring budget the server passes over to the other queue, if the current queue has not been emptied already before reaching the ring budget. Anyway, we assume that the ring budget is only relevant for the BE ring because the ring budget to serve the RT ring is chosen sufficiently large to make sure that the RT ring is always empty when the server switches to serving of the BE ring. So, the ring budget of the RT ring is never exhausted. In the following, we use the variables depicted in Table II.

![Fig. 3: Queueing model of ingress QoS](image)

Now, we can determine a (slightly conservative) upper bound for the latency of RT packets. As we assume the RT ring to be empty when the CPU core switches to BE ring, the latency of a packet in the RT ring is bounded by the sum of the following times:

- The waiting time for the CPU core to return after serving the BE ring within one service cycle using its complete ring budget $b$ corresponds to $bx$.
- The time needed to serve the backlog of packets in the RT ring, which accumulated during the time the CPU core spent at for serving BE ring; this first backlog consists of at $bx\lambda_{RT}$ packets in mean, leading to a mean service time requirement of a first backlog of $bx\lambda_{RT}x$.
- The time needed to serve the second backlog of packets in the RT ring, which accumulated during the time the CPU core spent at for serving the first backlog; the service time requirement of a second backlog of $bx(\lambda_{RT}x)^2$.

So we can sum all single portions of delays to get the desired bound $T_{\text{max}}$ for the packet latency in RT ring:

$$T_{\text{max}} = bx + \sum_{i=1}^{\infty} bx(\lambda_{RT}x)^i = bx\sum_{i=0}^{\infty} \rho_{RT}^i = \frac{bx}{1 - \rho_{RT}}$$

(1)

If we assume the system is not overloaded by RT traffic, then this geometric series converges because $\rho_{RT} < 1$. Evidently, the first term of the sum in the middle of Eq. (1), i.e. $bx$, represents the maximum waiting time for the CPU core to return to serving the RT ring after a complete service cycle of the BE ring. The second term of the sum, namely the geometric series, represents the maximum duration of a complete service cycle of the RT ring, cf. similarity of this result to the expected length of the busy period in M/M/1 queuing systems [20].

It should be noted that the important assumption used to derive Eq. (1), namely that, in an interval of length $T$, we can expect that $T\lambda_{RT}$ packets will arrive at the RT ring, may not be fulfilled for small values of $T$. Nevertheless, if the traffic arriving at the RT ring is rather smooth (e.g., no large bursts exist) our assumption should be sufficiently valid. Now, Eq. (1) also allows us to determine an appropriate value for the budget $b$ if, e.g., we want to bound the maximum latency in the RT ring by a value $T^*$ which is a multiple $m$ of $x$, i.e. $T^* = mx, m \in \mathbb{N}$. This requirement is fulfilled if the following equation holds.

$$\frac{bx}{1 - \rho_{RT}} \leq T^* = mx \quad \Leftrightarrow \quad b \leq m(1 - \rho_{RT})$$

(2)

As a specific example, let us determine an acceptable value for $b$ if $m = 6$ and $\rho_{RT} \leq 0.3$. In this situation, $b \leq 4.2$ holds and this means that we could choose $b = 4$, because we want to have $b$ as large as possible to minimize the switching overhead between the RT ring and the BE ring. Analoously, for $\rho_{RT} \leq 0.5$ (and still $m = 6$) we could choose $b = 3$. Determining the value of the ring budget to serve the RT ring is trivial, because it is sufficient to take a value which is large enough so that the budget is nearly never reached in order to completely serve the RT ring in an RT service cycle.

IV. IMPLEMENTATION OF LOW LATENCY SUPPORT

In this section we give an overview of the modifications that we applied to the Linux driver module of a 10 GbE adapter in order to realize our prototype implementation. Since our testbed is equipped with 10 GbE NICs from Intel, we implemented the proof of concept prototype for the corresponding ixgbe driver (version 3.22.3).

Within the ixgbe driver, Intel provides a data structure which is wrapped into the NAPI device structure to be compatible with the NAPI. This data structure is called QVector (short for Queue Vector) and it is responsible to store information about Rx rings. Fig. 4(a) shows a schematic view of how such a QVector is structured. Each QVector holds two distinct containers. One container for Tx rings and another one for Rx rings. The main purpose for this data structure is to save IRQs, as all referenced rings of a QVector share the same IRQ line. For instance, this is exploited by a technique called virtual machine device queues (VMDq) where the NIC sorts packets into specific receive queues (Rx rings) which are then grouped by a QVector and thus share an IRQ line. Therefore, a single IRQ causes the virtual machine monitor (VMM) to handle...
multiple Rx rings (for different VMs) in batch and provide the VMs with bulks of packets to improve I/O performance. A standard RSS setup without VMs also benefits from IRQ mitigation due to QVectors. For each NIC and for each CPU core a dedicated QVector mitigates Rx and Tx IRQs while it also avoids locking in case of parallel processing. Compared to the general VMDq case with multiple Rx and Tx rings per ring container (i.e. as many as VMs are hosted), each of both ring containers now references exactly one ring (cf. Fig. 4(b)).

The purpose of our modifications is to extend the ixgbe driver to provide QVectors within RSS setups that has two Rx rings, as illustrated in Fig 4(c), since our QoS concept (cf. Sec. III) requires one Rx ring for RT traffic and another one for BE traffic. In the following, we describe the extensions that we applied to the standard ixgbe driver in order to realize our QoS concept prototype implementation (cf. Section III).

A. Grouping of Multiple Rx Rings

The first part of our driver modification refers to group two Rx rings per QVector (cf. Fig. 4(c)). We extended the driver to provide a new module parameter which we named per ring buffer (PRB). PRB accepts an arbitrary long array of integers (cf. Listing 1), whereby each integer causes the driver to add an Rx ring to each created QVector. Thus, an arbitrary number of traffic classes is supported by the QVectors. The PRB values refer to the ring budgets (cf. Sec. III-B).

B. Polling with Ring Budgets

Our second driver modification refers to the poll function ixgbe_poll which is called by the NAPI and which is responsible to fetch the queued packets from the Rx rings and pass them to the IP stack, one by one. For this purpose, the NAPI provides ixgbe_poll with a reference device from the poll list. Based on this device, ixgbe_poll determines the associated QVector and starts to free packets buffers of already sent packets that reside in the Tx rings within Tx ring container. Afterwards, the poll function will process the packets from the Rx rings. For VMDq, the standard implementation of ixgbe_poll will equally distribute the device budget between all Rx rings that remain in the Rx ring container. For example, if the device budget is 64 (default) and if there are two Rx rings, then, up to 32 packets from the first Rx ring are processed before up to 32 packets from the second Rx ring are processed. Afterwards, the driver returns to the NAPI which switches to the next device. However, this behavior is not suitable for QoS-sensitive packet processing because an RT poll might be interrupted for up to $32 \cdot 0.6 \mu s \approx 19 \mu s$ if we assume a packet service $x = \frac{1}{\text{transmission time}} = 0.6 \mu s$ according to the maximum throughput (cf. Sec. V-B1). Therefore, we propose to apply smaller per ring budgets for the BE Rx ring in order to improve the latency of RT traffic (cf. Sec. III-D). Additionally, we also want to exploit the device budget of 64 due to efficient usage of the CPU resource. For this purpose, we extended ixgbe_poll by an additional loop that allows both Rx rings to be polled alternately with small ring budgets until the poll size is exhausted (cf. Fig. 2).

C. Usage of Modified Driver

Listing 1 shows an example for loading our modified driver\(^1\). The PRB parameter defines two ring budgets, thus, the QVector groups two Rx rings. The order of the PRB values from left to right refers to the Rx rings with increasing traffic class priority. Therefore, the ring budget of the BE Rx ring refers to 4, whereas the RT Rx ring gets a ring budget of 60. Furthermore, the driver arranges one QVector for our ingress NIC (RSS=1) and the interrupt throttling is disabled (ITR=0).

```
Listing 1: Loading of ixgbe driver with the PRB parameter
```

V. EVALUATION

Our goal is to investigate whether our QoS concept has positive effects on the latency of real-time traffic and whether the throughput, which is potentially decreased due to the overhead of our implementation, is still acceptable for practical usage. Thus, we measure and evaluate the performance of our ingress QoS driver (LL-RR, LL-WFQ) and compare it to the performance of the state-of-the-art driver (SQ, RR). The measurements were conducted in our testbed wherein we already performed various other performance tests [21], [22].

A. Measurement Setup

1) Hardware Configuration: The device under test (DuT) which serves as the software router is equipped with a SuperMicro X9SCL/X9SCM motherboard, a 3.20GHz Intel Xeon CPU E31230, 16 GB RAM, and an Intel X540-T2 NIC.\(^1\)

\(^1\)publicly available at: http://www.informatik.uni-hamburg.de/memphis
All measurements were performed with Linux kernel version 3.16.7 and ixgbe version 3.22.3. We deactivated features like Intel Turbo Boost and Hyperthreading as they introduce unpredictable behavior. Additionally, we configured the CPU to run at a fix rate of 3.20 GHz. Furthermore, we deactivated the interrupt throttling (ITR) which would increase the packet latency. Ethernet flow control was disabled to conduct meaningful measurements even in overload situations.

2) Software Configuration: For comparison, our set of measurements was performed on four differently configured DuTs (cf. Section III-C). For all configurations the Rx ring size is 512 (default) and the device budget is 64 (default).

In our previous work, we showed that the maximum throughput of the packet processing scales nearly linearly with the number of CPU cores [23]. Thus, we simplified the measurements and configured the DuT to utilize only one CPU core for Linux IP packet processing.

The first two scenarios (SQ, RR) represent the state-of-the-art case and are conducted with the original ixgbe driver. With the SQ scheduling strategy, all traffic (RT, BE) is sorted into the same Rx ring. With RR, RT and BE traffic is distributed to the corresponding Rx rings of two different devices. These devices are served according to the NAPI, which is some kind of round robin polling (cf. Section III).

The other scenarios (LL-RR, LL-WFQ) are performed with our modified ixgbe driver. For LL-RR we configured the ring budget to be 4 (as deduced in Section III-D) for both Rx rings. Thus, the CPU is equally shared between the traffic classes if both rings are sufficiently utilized. We assume that the offered load to the RT ring is at most 30% of the maximum throughput. Therefore, we expect in case of 4 arrived BE packets $4 \cdot 0.3/(1 - 0.3) \approx 1.7$ RT packets in mean. Thus, an RT ring budget of 4 is slightly oversized which has the advantage that backlogs, due to small bursts (e.g. if the short-period real-time percentage is above 30%), decrease faster.

With LL-WFQ the RT ring budget is 60 while the BE ring budget remains 4. Thus, the RT ring budget is sufficiently large (as assumed in Section III-D) but not that large that the poll size of 64 is exceeded. Otherwise the BE traffic potentially might starve in overload situations.

3) Methodology: Although, we are preliminary interested in throughput and latency, we also recorded DuT internal meters like CPU utilization and IRQ rates, that help to explain several performance related effects. The IRQ counts and the CPU utilization of the DuT are obtained via the process file-system (/proc/interrupts) and perf (a performance evaluation infrastructure/tool for Linux), respectively. Throughput and latency is sampled by our packet generator MoonGen [21], which is packet source and sink at the same time. The DuT and the packet generator are directly connected.

4) Network Load: We measured the throughput for one CPU core of the DuT at different packet rates ranging from 0.05 Mpps to 2.0 Mpps in steps of 0.05 Mpps and different real-time percentages of the overall traffic ranging from 0% to 100% in steps of 5%. Each measurement took 60 s during which test traffic based on a Poisson traffic was applied.

In our previous work [23], we showed that the CPU constitutes the bottleneck in a software router for small packet sizes. Hence, we prevent that Ethernet links become the bottleneck by limiting the packet size to 128 Byte (10 GbE links are theoretically able to cope with approx. 8.45 Mpps of this size).

B. Measurement Results

1) Throughput: Firstly, we analyzed the maximum throughput achieved by one CPU core of the DuT for all scheduling strategies, in order to compare whether the throughput of the RR and LL strategies deviate too much from SQ which would be unsuitable for the practical usage. Therefore, for each strategy and for each RT ratio, we determined the maximum offered load that is achieved. This is done by stepwise increasing the offered load until packet loss is encountered. Since we are not interested in a very fine-grained rendering of the maximum throughput, we chose a step size of 0.05 Mpps in order to reduce the efforts for measurements.

Fig. 5(a) illustrates the maximum throughput of the different strategies at RT ratios from 0% to 100%. SQ is our baseline and achieves nearly a constant maximum throughput of about 1.75 Mpps, as the RT and BE traffic share the same Rx ring, whereby the throughput is independent from the RT ratio. Additionally, it is to mention that SQ reaches the highest maximum throughput we observe for RT ratios from 0% to 85%. From 90% to 100% it might be suggested that LL-WFQ reaches a much higher throughput than SQ. However, this is an artifact that results from the coarse step size. In contrast to that, RR has the worst maximum throughput which is between 1.6 Mpps and 1.65 Mpps. LL-RR and LL-WFQ show a slightly worse behavior than SQ but are still better than RR and also reach a high maximum throughput of at least 1.65 Mpps.

In summary, we observe that all strategies cope with an offered load of 1.5 Mpps. Hence, a software router, which is equipped with 10 CPU cores, will be able to satisfy the line rate of a 10 GbE adapter (14.8 Mpps for 64 Byte packets).

In order to be able to equitably compare all strategies, we further focus on two specific measurements. (1) We evaluate measurements at an offered load of 1.5 Mpps in experiments where the offered load is fixed. (2) We investigate measurements with a fixed RT ratio of 30%, since we assume that todays RT traffic in the Internet is 30% at most.

2) CPU Utilization: Fig. 5(b) illustrates the CPU utilization of the DuT for offered loads, ranging from 0.05 Mpps to 2.0 Mpps at a fixed RT ratio of 30% in steps of 0.05 Mpps.

We observe that LL-RR, and LL-WFQ basically exhibit the same behavior as SQ. Between offered loads of 0.05 Mpps and 0.2 Mpps we see a steep increase in the CPU utilization, which is caused by IRQ handling. Then, from 0.2 Mpps to 0.7 Mpps the CPU utilization is nearly constant, as the IRQ rate gets throttled by the NAPI. Afterwards, the CPU utilization starts to increase linearly with growing offered load until 100% is reached at approx. 1.7 Mpps.

Compared to all other strategies RR behaves differently, since RR reveals an increased CPU utilization at offered loads above 0.7 Mpps. Interestingly, RR reaches 100% CPU
Fig. 5: Maximum throughput and CPU utilization for different scheduling strategies

Fig. 6: Interrupt rate for different scheduling strategies

utilization earlier than the other strategies (i.e. at 1.55 Mpps), which explains the decreased throughput we observed before. Hence, for an offered load of 1.5 Mpps, which we previously chose for experiments with fixed offered loads, we observe that all strategies lead to nearly full CPU utilization but not to overload, which is a reasonable and fair operating point to compare the QoS characteristics of all strategies.

3) IRQ Rate: Fig. 6(a) illustrates the IRQ rate of the DuT for different RT ratios, at a fixed offered load of 1.5 Mpps. The IRQ rate is averaged over the 60 s interval.

For SQ we observe a nearly constant IRQ rate for all RT ratios, which is approx. 32k IRQ/s. Compared to that, we see a behavior of RR that is dependent from the RT ratio. The IRQ rate of RR starts with approx. 20k IRQ/s and increases to approx. 58k IRQ/s until a RT ratio of approx. 50% is reached. Afterwards, the IRQ rate decreases. This effect is a consequence of the two separate devices that are used by this strategy (cf. Section III-C), whereby both devices generate IRQs independently but according to the offered load. Thus, the RT device generates as much IRQs at an RT ratio of 30% as the BE ring generates at an RT ratio of 70%. As the IRQ rates of both devices sum up, we observe a symmetric IRQ rate which is in worst case (at 50% RT ratio) approx. two times higher than the IRQ rate of SQ. For LL-RR and LL-WFQ, we observe roughly the same behavior of the IRQ rate, which is nearly constant at approx. 20k IRQ/s. Interestingly, the IRQ rate of both LL strategies is 10k IRQ/s lower than the IRQ rate of SQ, which corresponds to a decrease of approx. 30%.

Fig. 6(b) illustrates the IRQ rate for offered loads, ranging from 0.05 Mpps to 2.0 Mpps, at a fixed RT ratio of 30%.

For SQ we see an interesting effect at offered load from 0.8 Mpps to 1.0 Mpps. First, the IRQ rate drops to 25k IRQ/s before it increases to 32k IRQ/s where it remains until an offered load of approx. 1.6 Mpps is reached. At an offered load of 1.7 Mpps the IRQ rate drops to zero, as the the NAPI prevents from the reactivation of the IRQ line.

For RR we observe that the IRQ rate is generally higher than for SQ at all offered loads larger than 0.15 Mpps, which is again due to the two devices that are used. The more mentionable effect is, that RR still generates approx. 17,000 IRQ/s in case of overload (1.7 Mpps to 2.0 Mpps). These IRQs are generated by the RT device, as RR allows to spend
As latency optimization is the major objective. Mpps which is above the actual RT load and Mpps. Hence, it is very likely that the IRQ line of the RT device is often re-enabled since the device budget is not exhausted.

For both LL strategies we observe the same behavior of the IRQ rate with an exception at an offered load of 1.0 Mpps where LL-RR has a higher IRQ rate than LL-WFQ. LL-RR and LL-WFQ only differs in the budget of their RT Rx rings (cf. Section III-C). Thus, we conclude that in case of LL-RR the low per ring budget of both rings ensures a fair distribution of the device budget, which helps to avoid backlogs in any of both rings in case of bursts, as caused by the Poisson traffic pattern. LL-WFQ has a much higher RT Rx ring budget and in turn the probability for BE backlogs is high if RT bursts arrive. Thus, it is plausible that the IRQ line is less often re-enabled with LL-WFQ. As SQ does not differentiate between RT and BE, the IRQ re-enabling is less complicated and SQ reaches higher IRQ rates at offered loads larger than 1.05 Mpps.

4) Latency: As latency optimization is the major objective of our QoS concept we discuss our latency related measurement results in-depth. In Fig. 7(a) the average packet latency is plotted against the RT ratio at a fixed offered load of 1.5 Mpps.

For SQ, we see that the latency is constant at approx. 11.7 µs and therefore independent of the RT ratio, since a traffic classification is missing. Thus, each packet experiences the same mean waiting time in the Rx ring before it is processed.

Obviously, the RR strategy has poor real-time properties, as all measured latencies for both traffic classes are above that of SQ, which is again due to the numerous IRQs (cf. Fig. 6(a)). Also, we observe that the RT latency is above the BE latency at RT ratios below 50%, as the RT Rx ring will suffer from the long poll phase as caused by the highly utilized BE Rx rings.

Compared to RR, the LL-RR implementation behaves similar regarding the effect that RT and BE latencies cut across at an RT ratio of 50%. However, as the per ring budget is relatively low (i.e. 4 for both traffic classes), even highly utilized BE Rx rings will not interrupt RT Rx rings for longer than the processing of 4 packets. Thus, we observe a low RT latency of approx. 10.4 µs, which is approx. 11% better than the latency of SQ (11.7 µs). However, this improvement comes on cost of the BE latency, which is notably higher than for SQ at RT ratios below 50%.

For LL-WFQ we observe nearly the same low latency of RT traffic as for LL-RR (10.4 µs) but LL-WFQ is able to provide lower RT latencies than SQ for RT ratios between 50% and 80%. This is due to of the high per ring budget of 60 for the RT ring compared to the BE ring budget of 4. Another effect is that both LL strategies suffer from higher latencies (compared to SQ) in the extreme cases where we load the DuT with 0% or 100% RT traffic, respectively. We refer this increase of latency to the overhead that is introduced by our modifications.

Fig. 7(b) illustrates the packet latency at different offered loads, ranging from 0.05 Mpps to 2.0 Mpps, at the specific RT ratio of 30%. Regarding SQ we see two interesting effects. The first one is, that the latency increases significantly between 0.75 Mpps and 1.0 Mpps. We refer this effect to the NAPI, as we observe IRQ throttling within this range of offered load (cf. Fig. 6(b)). Special attention should be drawn to the second effect, which is that the latency of SQ will abruptly increase at an offered load of 1.65 Mpps as bursts cause backlogs in the Rx ring and thus increase the waiting time of packets.

RR shows higher RT than BE latency at all offered loads, as RT packets are not prioritized and therefore suffer from long waiting times. Thus, we conclude that RR is not acceptable for latency-sensitive applications.

Regarding LL-RR and LL-WFQ we observe that RT latencies are quite similar at all offered loads. Thus, we conclude that a BE and RT ring budget of 4 is sufficient for RT ratios of 30% and less. For LL-RR and LL-WFQ we also see that the RT latency increases between 0.05 Mpps and 0.75 Mpps while it decreases between 0.75 Mpps and 1.7 Mpps. This behavior
might not be obvious on the first sight, but due to a low and stable IRQ rate (cf. Fig. 6(b)) we conclude that the NAPI is mostly in polling mode. Thus, the packet latency is not that much impaired by the time it takes to handle an IRQ. Instead, the packet processing basically behaves as explained in Section III-D, where we stated that new RT packets might arrive while others are processed. Such packets will have small waiting times, thus, the mean latency is low at high offered loads and even in overload situations.

In contrast, we observe that the BE latency for LL-RR and LL-WFQ is notably higher than for RT and like for SQ it also increases significantly in case of overload. However, for LL the RT latency remains low, even in case of overload. This improvement of the RT latency comes on the cost of the BE latency, but we argue that BE traffic has no real-time requirements and high delays are acceptable.

In conclusion, we found that RR has no benefits compared to SQ. More IQRs burden the CPU and as a consequence less packets are processed whereby the throughput decreases (1.6 Mpps). Additionally, RR provides poor RT latency, as highly utilized BE Rx rings in combination with large ring budgets cause long waiting times for RT packets. In contrast, our LL driver exploits separate ring budgets to provide low RT latencies (approx. 11.4 μs), even in case of overload. Since the design of our LL driver minimizes additional IQR and packet reception overhead, it achieves almost the same throughput as SQ (1.7 Mpps), which is the upper limit that is given by the CPU speed and the networking module of Linux itself.

VI. SUMMARY

Over the last years we face the trend of growing real-time traffic which has to share the limited resources of the Internet with other traffic classes. Thus, we see the demand for mechanisms that efficiently allocate these resources to these traffic classes in order to provide QoS. While QoS is not a new topic, all previous concepts assume the outgoing link to be the bottleneck. However, seeing the trend towards flexible CPU based data plane devices where general purpose hardware in combination with software serves arbitrary needs, we argue that more routers will be CPU bounded in future.

As traffic prioritization behind the bottleneck (i.e. CPU) introduces avoidable latency, we proposed a new QoS concept for software routers that prioritizes traffic before being served by the CPU (ingress QoS). Based on our concept we extend the driver code of a common 10GBe NIC. Afterwards, we conducted extensive real-world measurements to compare the performance of our new QoS concept with the state-of-the-art. An in-depth analysis regarding throughput and latency showed that our QoS concept improves the latency of real-time traffic while the throughput is nearly unaffected. These satisfying and meaningful results demonstrate that software routers are able to cope with real-time traffic, even at high offered loads.

ACKNOWLEDGMENT

This research has been supported by the German Research Foundation (DFG) as part of the MEMPHIS project. We thank our colleagues Florian Wohlfart and Sebastian Gallenmüller for their valuable feedback.

REFERENCES

Traffic-Driven Implicit Buffer Management—Delay Differentiation Without Traffic Contracts

Martin Karsten  
David R. Cheriton School of Computer Science  
University of Waterloo  
Email: mkarsten@uwaterloo.ca

Daniel S. Berger and Jens Schmitt  
Department of Computer Science  
University of Kaiserslautern  
Email: {berger,jschmitt}@cs.uni-kl.de

Abstract—Different network applications have different service preferences regarding packet delay and buffering. Delay management requires scheduling support at routers, which traditionally also requires some form of traffic specification and admission control. In contrast, this paper studies the problem of guaranteeing queueing delay bounds for multiple service classes without traffic contracts and without affecting the throughput rate for each class. A solution to this problem is given by decoupling throughput and delay management via traffic-driven implicit buffer management. Using this concept, the Delay Segment FIFO (DSF) packet scheduler guarantees differentiated delay targets in the presence of unregulated throughput rates. This decoupling represents a modular approach and DSF embodies a small and self-contained feature set. Furthermore, DSF’s service model satisfies even a strict interpretation of network neutrality, while effectively guaranteeing delay targets for multiple service classes. DSF’s design and service characteristics are analyzed mathematically and validated through simulations.

I. INTRODUCTION

The Internet serves a multitude of diverse applications that have different service requirements. Traditional bulk data transfers seek to optimize their overall throughput. On the other hand, interactive multi-player games or live conferencing systems have stringent end-to-end delay requirements on the order of 50–100ms. A number of applications fall in between, e.g., multimedia streaming requires moderate throughput and has somewhat relaxed end-to-end delay requirements. As an illustrative example, current mobile networks define 13 service classes with end-to-end delay targets of 50ms-300ms [1], i.e., within one order of magnitude. However, the controllable portion of the end-to-end delay is queueing delay. When subtracting speed-of-light propagation latency, the actual queueing delay diversity covers two orders of magnitude.

Traditional approaches to limit and differentiate worst-case queueing delay for different types of applications combine offline or signalled traffic contracts with differentiated scheduling, such that forwarding capacity is allocated to dedicated service classes. As long as traffic arrivals conform to the negotiated traffic specification, the queueing delay is guaranteed by ensuring a particular minimum throughput service. Packet schedulers used in this context include processor sharing and its variants [2], as well as priority scheduling. Thus, delay management is tied to throughput management. However, the exclusive nature of these agreements requires a typically complex policy framework to determine which traffic is eligible for certain service classes. This traffic discrimination might violate a strict interpretation of network neutrality.

An alternative approach is controlling queueing delays without explicit throughput management - by implicitly limiting the effective queue, i.e., buffer, size available for each service class: packets that (will) miss their delay target are discarded. To avoid any throughput bias, service rates must be proportional to arrival rates, while the effective queue size for each class must be automatically adjusted to enforce the respective delay target. This traffic-driven approach provides less direct control, but has a number of practical advantages:

• Without explicit throughput management, sources can pick any service class. This removes the need for a complex policy system determining service class eligibility.

• With unrestricted access to service classes, the network service clearly satisfies the so-called no paid prioritization rule, which is one of the three basic rules in the recent FCC Open Internet Order [9].

• Because throughput and delay management are decoupled, the resulting architecture is modular.

The first contribution of this paper is the definition of a service metric throughput interference index ($TT^2$) to quantitatively capture the above objectives for a packet scheduler. The main challenge is devising an efficient packet scheduler that allocates service rates in proportion to arrival rates to achieve a good $TT^2$ while enforcing per-class queueing delays by dynamically and automatically adjusting the effective queue size for each class. This paper presents the first proper solution to this problem and makes the following contributions:

• The non-technical descriptions of network neutrality are reframed using the more restrictive notion of $TT^2$.

• Our approach – delay segmentation – is shown to achieve a low $TT^2$, while providing effective delay guarantees.

• A novel multi-class scheduler using delay segmentation is proposed and evaluated. It guarantees delay targets with minimal throughput interference.

The rest of the paper is organized as follows. Section II provides background and the definition of $TT^2$. Section III discusses related work. Section IV presents the DSF algorithm. The principles behind DSF are analyzed in Section V and it is evaluated using simulations in Section VI. The paper is wrapped up with a brief conclusion in Section VII.
II. BACKGROUND AND MOTIVATION

A. Network Neutrality

The academic discussion of network neutrality spans almost two decades in the legal literature [9], [21], [28], [30]. Yet, from a technical perspective, this discussion has been criticized as being largely oversimplified [7]. In particular, the question which specific packet schedulers and/or service models conform to network neutrality has received little attention in the legal literature. Only FIFO scheduling has been discussed widely and, in fact, many claim that FIFO scheduling is necessary to attain network neutrality [6], [31], [32]. The current network neutrality rules in the USA [9] do not require specific schedulers but instead establish three principles: 1) no blocking, 2) no throttling, 3) no paid prioritization. Further clarification seems necessary [28] to apply these principles. The literature about detecting network neutrality violations typically uses a broader definition. In recent work [26], for example, network neutrality violations are detected by comparing each flow’s loss rate to a baseline loss rate, e.g., of some traffic aggregate. The network is neutral, if the difference between each flow and the baseline is sufficiently small. The same principle can be used to derive a metric for quantitatively assessing specific packet schedulers or service models.

B. Throughput Interference Index

We propose a new metric to assess schedulers and service models without the need for a baseline: The throughput interference index $TI^2$ quantifies the degree by which a scheduler changes the throughput of multiple flows.

Definition 1: The throughput interference index ($TI^2$) of a rate transformation for $n$ flows, with $\lambda_i^{in}$ and $\lambda_i^{out}$ being the input and output rates for flow $i$, is measured as

$$TI^2 = 1 - \left( \frac{\sum_{i=1}^{n} \lambda_i^{out}}{\sum_{i=1}^{n} \lambda_i^{in}} \right)^2.$$ 

Essentially, the $TI^2$ definition applies Jain’s fairness index [15] to relative throughput rates. $TI^2$ ranges from 0 to $\frac{n}{n-1}$, with values closer to 0 indicating lower scheduler interference with throughput rates. In contrast, higher relative throughput differences result in a larger $TI^2$. This definition is backwards compatible with previous definitions. Firstly, FIFO indeed has a low $TI^2$ as shown in the associated technical report [17]. Secondly, this definition incorporates the no blocking and no throttling rules [9] because it ensures that relative service rates correspond to relative arrival rates. Thirdly, this definition incorporates the loss-based metric from recent work [26], because the ratio $\frac{\lambda^{out}}{\lambda^{in}}$ corresponds to $1 - loss$ for each flow. The analysis of the DSF scheduler proposed in this paper is based on the $TI^2$ metric.

C. Incentive Compatibility

If all service classes of a multi-class service model are accessible to all traffic without restriction, this trivially satisfies the no paid prioritization rule [9]. However, if a service class is strictly superior to another one in terms of service quality, the resulting race to the top renders the lower class useless. Thus, no service class should be dominated by another one in terms of service quality. The Incentive-Compatible Differentiated Scheduling (ICDS) proposal [18] for multi-class delay differentiation satisfies this requirement. As evident by its name, ICDS identifies the incentive-compatible nature of such a scheduler and presents a simple proof of this property. The same proof can be applied to some of the systems discussed in the next section, as well as the DSF algorithm presented in Section IV. Thus, DSF is also incentive-compatible. Furthermore, ICDS can be expected to have near-zero $TI^2$, because it is designed to allocate service rates in proportion to arrival rates. However, no practical algorithm is given. In contrast, this paper introduces a specific algorithm, DSF, that also has a very low $TI^2$.

III. RELATED WORK

The Alternative Best Effort (ABE) scheduler [14] offers a bounded-delay service class (“Green”) and a throughput-oriented service class (“Blue”). The Blue class is guaranteed to achieve the same throughput as in the equivalent FIFO system, while the Green class receives priority service whenever possible without violating the throughput guarantee for Blue. However, this priority-based concept is fundamentally limited to two classes. A potential third class cannot receive priority service (in comparison to Blue) and throughput guarantees (in comparison to Green) at the same time. In contrast, DSF supports a general service model with multiple service classes. Virtually Isolated FIFO Queueing (VIFQ) [16] is a previous attempt to generalize ABE to multiple classes and to implement ICDS [18]. Unfortunately, VIFQ’s delay control approach is inherently flawed (cf. Sections V-A and VI-A).

Various schedulers provide non-dominant service differentiation with trade-offs between throughput and delay. The most recent incarnations of this approach are the RD proposal [25] and QJump [12]. RD provides two service classes with a fixed throughput ratio between individual application flows in each class. Appropriate buffer sizing ensures a maximum queuing delay for the delay-oriented class. The design resembles weighted processor sharing with a specific throughput ratio, which would entrench a particular service policy in routers and does not satisfy the objective of a low $TI^2$. Similarly, QJump couples priority levels (for delay guarantees) with strict limits on the maximum throughput to create trade-offs. Its delay guarantees depend on network parameters, while priorities and throughput limits must be enforced at end hosts. These are unrealistic assumptions for the general Internet. In contrast, DSF is incentive-compatible and makes no such assumptions.

ABE, RD, and QJump all propose penalties for low-delay traffic. This is often justified by the observation that long-term TCP goodput is inversely proportional to the average round-trip delay, which includes the average queuing delay [23]. For example, ABE and RD contain complex rate control mechanisms to prevent TCP-like traffic from gaining an advantage by using the low-delay class. In ABE, the low-delay service class
is penalized using a probabilistic parameter $\alpha$, which can be computed precisely, only if the round-trip propagation delay of all competing TCP flows is identical and known. In RD, a fixed ratio $k > 1$ is configured as the ratio of the average per-flow rate of the throughput class in relation to the average per-flow rate of the delay class. RD maintains the relative throughput of both classes according to $k$ and the number of flows in each class, which must be estimated. In contrast, DSF is based on a much simpler scheduling algorithm that does not require parameter estimation.

Mathis [22] has questioned this traditional, narrow focus on TCP-friendly rate control. For example, recent alternative TCP rate control algorithms [8] seek to overcome the dependency of TCP’s rate on the round-trip delay. In fact, these rate control algorithms demonstrate how a particular service policy in routers would unnecessarily contribute to the ossification of the Internet architecture. Therefore, TCP’s current properties should not curtail the search space for packet schedulers without further investigation. DSF only gives delay guarantees (by offering limited queueing space) without prioritizing service. Without prioritization, there is no need to actively penalize low-delay traffic, which simply seeks less queueing space.

Several differentiated service proposals use differentiated buffer management [5], [13]. This segregation of buffers is different from DSF, which manages a single shared buffer. Unlike differentiated services, DSF is incentive-compatible and does not require traffic contracts.

Another approach has recently (re)gained research interest: controlling queueing delay through active queue management (AQM). A prominent example is the CoDel algorithm [24]. CoDel counteracts prolonged periods of high queueing delay by increasing the packet drop rate until the delay reaches a configured value, but tolerates transient phases of higher delay to absorb short-term traffic bursts. This approach is expected to consolidate low delay and high goodput, and indeed, a recent evaluation shows that CoDel achieves the best goodput among several other AQMs [20]. Nevertheless, CoDel does not solve the fundamental trade-off between delay and goodput. In some scenarios, CoDel can increase the file completion time by 42% [20]. In summary, there is no conclusive evidence that AQM can support all desirable delay targets under all circumstances without negative side effects [17].

IV. DELAY SEGMENT FIFO (DSF)

Delay Segment FIFO (DSF) is a multi-class packet scheduler that closely approximates the per-flow throughput of FIFO and thus attains a low $T/F^2$. Each service class has a fixed delay target and packets are assumed to carry a service class identifier in their header, for example, in the DiffServ code point [4]. The DSF algorithm applies two key principles, Delay Discard and Queue Segmentation, to achieve its objectives.

Delay Discard decouples buffer admission control from per-class delay management. This is facilitated by using two separate queue types – one for slots that represent a right to service and another one for packets. Each arriving (and accepted) packet creates a corresponding slot that is appended to the global slot queue, while the packet is stamped with its per-class delay deadline and added to a per-class packet queue. For each slot in the slot queue, the service routine processes the corresponding packet queue and discards packets that have missed their deadline. The next packet from this class that satisfies the delay target, is sent. By enforcing the target delay in this way, Delay Discard implicitly adapts the effective buffer size for each service class to its arrival rate. This design is motivated by viewing the buffer demands of a traffic flow as the dual of its delay objective. A traffic flow with average rate $r$ and delay target $d$ can reasonably expect at most a buffer capacity of $dr$. The earlier VIFQ proposal [16] is an attempt at a practical implementation of this conceptual design. However, Section V-A presents a Markov chain analysis that shows a fundamental flaw of relying on Delay Discard only. In particular, when the slot of a low-delay class reaches the front of the slot queue for service, the class might not have a packet available that satisfies the delay target, if its traffic rate is relatively low. This results in a high $T/F^2$ in those cases.

Therefore, DSF introduces Queue Segmentation in the buffer organization: each delay target is represented by a corresponding queue segment. The intuition behind this approach is making the system appear as a short virtual FIFO queue for low-delay traffic, while offering a bigger FIFO queue for higher-delay traffic. The key algorithmic change introduced with Queue Segmentation is the last-in-first-out (LIFO) service order between queue segments, which addresses the high throughput interference of Delay Discard. Assume that $n$ delay classes are ordered according to their delay targets. The total buffer space is split into $n$ queue segments. The size of segment $i = 1, \ldots, n$ corresponds to the incremental delay target of class $i$. In other words, if the link rate is $R$ and the delay targets are $D_1, \ldots, D_n$, then the capacity of each segment $i$ is $S_i = (D_i - D_{i-1}) \cdot R$ (with $D_0 = 0$). Thus, the aggregate capacity of all segments is equivalent to the highest delay target. For each service class a packet queue holds the packets in arrival order, ensuring FIFO service order per class.

A. Algorithm

DSF is illustrated in Figure 1 and specified in Algorithms 1 and 2. Data structures and variables are described in Table 1.

Upon arrival, the first non-null segment is determined (Lines 3-5). If successful, a slot is appended to the segment queue

![Fig. 1: Delay Segment FIFO](image-url)
TABLE I: DSF Variables and Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Index</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>squeue</td>
<td>segment</td>
<td>slot queue: {class, size}</td>
</tr>
<tr>
<td>pqueue</td>
<td>class</td>
<td>packet queue: {packet, deadline}</td>
</tr>
<tr>
<td>buffer</td>
<td>class</td>
<td>buffer size (sum of waiting slots)</td>
</tr>
<tr>
<td>service</td>
<td>class</td>
<td>service credit/credit</td>
</tr>
<tr>
<td>D[t]</td>
<td>class</td>
<td>delay target</td>
</tr>
<tr>
<td>now()</td>
<td>N/A</td>
<td>current time</td>
</tr>
</tbody>
</table>

(Line 6) and the per-class buffer size is increased (Line 7). Lines 8 and 9 implement a Drop Front mechanism per class, which is discussed in the next paragraph. Last, the arrived packet is appended to the per-class packet queue (Line 10).

At service time, if a non-empty segment is found (Lines 1,2), the first slot is removed (Lines 3,4) and the corresponding service class is given the appropriate service credit (Line 5). The essence of the Delay Discard function is found in Lines 6-10 of the service routine, which trivially guarantees that delay targets are met. As long as there is a positive service credit and available packets, a packet from the per-class packet queue is retrieved (Lines 6,7). If it satisfies the delay target (Line 8), it is transmitted (Line 9,10). Variable packet sizes are handled through the service credit counter. Because each packet is sent in entirety, a class might temporarily receive more service than it is entitled to. However, the service error is limited to one packet per class and corrected over time. Also, the delay test of DSF operates on packet serialization start times to avoid favouring small packets. This introduces a delay error in the amount of at most one maximum packet size per class.

Using only Delay Discard and Queue Segmentation would not limit the total amount of packet buffer needed and would increase the amount of processing in the service routine to discard packets that never had a chance to meet their delay target. Therefore, DSF uses an additional Drop Front mechanism in the arrival routine. There is no point in storing more packets in the packet queue than the total amount of sending rights for a traffic class given by the sum of its slot sizes. Lines 8,9 of the arrival routine thus limit the number of packets in the packet queue and also ensure that only the most recently arrived packets are kept, because those packets have the best chance of meeting their delay target.

Algorithm 1 DSF Arrival Routine (per packet)

1: $p \leftarrow$ received packet
2: $c \leftarrow$ class($p$)
3: $i \leftarrow 1$
4: while $i \leq c$ and squeue[i].full() do $i \leftarrow i + 1$
5: if $i \leq c$ then
6:   squeue[i].push($c$, size($p$))
7:   buffer[$c$] $\leftarrow$ buffer[$c$] + size($p$)
8: while pqueue[$c$].size() + size($p$) $> \text{buffer}[c]$ do
9:   pqueue[$c$].pop()
10: pqueue[$c$].push($p$, now() $+ D[c]$)

B. Remarks

The DSF algorithm is traffic-driven, because the effective buffer size for each service class is determined by its arrival process in combination with the fixed delay target. Buffer management is implicit, because the buffer size is not computed explicitly before packet admission control, but is derived indirectly from admitted slots.

DSF is inherently a single-node technique. Therefore, it can be deployed incrementally, for example at critical bottlenecks only. The flip side is that a flow traversing multiple bottlenecks might be exposed to multiples of the delay target. However, the number of bottlenecks is usually small. DSF can support very aggressive delay targets, as shown in Section VI. In a system without Delay Discard, late packets are forwarded and increase downstream load, despite being ultimately useless.

Algorithm 2 DSF Service Routine (loop)

1: $i \leftarrow 1$
2: while squeue[i].empty() do $i \leftarrow i + 1$
3: $\{c, s\} \leftarrow$ squeue[i].pop()
4: buffer[$c$] $\leftarrow$ buffer[$c$] - $s$
5: service[$c$] $\leftarrow$ service[$c$] + $s$
6: while service[$c$] $\geq 0$ and pqueue[$c$].nonempty() do
7:   $\{p, d\} \leftarrow$ pqueue[$c$].pop()
8:   if now() $< d$ then
9:   service[$c$] $\leftarrow$ service[$c$] - size($p$)
10: send($p$)

C. Overhead and Complexity

The computational overhead of the DSF algorithm is small and limited. The only data structures that are used for DSF are plain FIFO queues storing either slots or packets. The only numerical operations are addition and subtraction. There is no sorting or other rearrangement of information that would impose computational complexity. The segment search in the arrival routine (Lines 3-5) and the service routine (Lines 1,2) can be implemented using bitstrings, along with the constant-time find-first-set operation that is available on modern hardware. The overhead of the arrival routine is proportional to the size of the arriving packet divided by the minimum packet size. Similarly, the total overhead of the service routine is proportional to the amount of sending rights stored in the slot segments divided by the minimum packet size. Therefore, DSF has amortized constant complexity per packet. On the other hand, the worst-case overhead between two consecutive service invocations might be increased by a series of packets that cannot meet their delay target in the loop in Line 6 of the service routine. However, the Drop Front function in the arrival routine limits the size of the overall packet buffer and furthermore, should keep the number of packets that miss their deadline small. Therefore, it is expected that the length of such service drop episodes is limited in practice. For example, over all the experiments reported in Section VI, the average length of service drop episodes is 1.3 packets with a standard
deviation of 1.18. Additional improvements are possible by exploiting parallel hardware, which is abundantly available.

V. ANALYSIS

This section presents analytical models for the two key principles of DSF: Delay Discard and Queue Segmentation. To keep the models tractable, they are based on fixed-size packets and a discrete time model with each slot corresponding to the service time of one packet. Sources are fully described by their arrival rate (i.e., no correlations are assumed); no single source has more than one packet arriving in a single time slot, which essentially means geographically distributed packet inter-arrival times. Only two delay classes are modelled with Class 1 having a delay target of \( N_1 \) time slots. Consequently, the total buffer capacity is \( N = N_2 \) slots. The arrival rates for Class 1 and 2 are denoted as \( r_1 \) and \( r_2 \).

A. Delay Discard Only

The goal of this section is to find a closed-form expression for the \( T1^2 \) of Delay Discard (DD). Under the above assumptions, a global Markov chain could be created that closely tracks the dynamics of the overall system. The state of the system would be captured by a vector (of length \( N \)) whose entries could take three values: (1) assigned slot (reserved for an in-time packet of Class 1 or for a Class 2 packet), (2) uncertain slot (reserved for Class 1, but packet that created it is late) (3), empty slot. The Markov chain could be used to determine the probability of an uncertain slot arriving at the head of the queue without a suitable Class 1 packet available to use it. Such a slot is called expired. We have experimented with this model, but deem it intractable, because of its complex structure, as well as a dramatic state space explosion, which also prohibits a numerical solution.

1) Decoupling Approach: Since the global Markov chain is not a feasible choice for the analysis of DD, an approximation is used that assumes a decoupling of the arrival and slot generation processes. The system is modelled as two Markov chains that interact with each other in a simple way. In particular, one Markov chain is used to model the slot generation process by tracking the total queue length to derive the drop rate due to a full slot queue. In addition, it is used to determine the overload probability, i.e., the probability that the number of service slots in the slot queue exceeds \( N_1 \), the delay target of Class 1. The drop rate computed from this Markov chain can be used to drive another Markov chain, which keeps track of the position of the first uncertain slot in the relevant buffer spaces ranging from 1, \ldots, \( N_1 \). This Markov chain is conditioned on the system being in overload, which provides another link between the two Markov chains.

Based on the two Markov chains, the steady-state probability for an expired slot of Class 1 is obtained as

\[
pe_i = \lim_{t \to \infty} P(E_i(t))
= \lim_{t \to \infty} P(E_i(t) | O_i(t - N_1)) P(O_i(t - N_1))
= pe_i | O_i \cdot po_i,
\]

where \( E_i(t) \) denotes the event that a service slot of Class 1 expires at time \( t \), and \( O_i(t - N_1) \) denotes the event that the system was in overload at time \( t - N_1 \). Note that the law of total probability is correctly applied in the second line as \( O_i(t - N_1) \) is necessary for \( E_i(t) \), i.e., \( E_i(t) \subset O_i(t - N_1) \). The steady-state probabilities \( pe_i | O_i = \lim_{t \to \infty} P(E_i(t) | O_i(t - N_1)) \) and \( po_i = \lim_{t \to \infty} P(O_i(t - N_1)) \) are calculated from the Markov chains for the uncertain slot position and for the total queue length, respectively.

2) Markov Chain for Total Queue Length: This first Markov chain models the total queue length oblivious to traffic classes. Under the given assumptions, this is very similar to a Geol/D/1/N queue [11] with the slight difference that the arrival process is a superposition of two different flows with independent geometric inter-arrival times. Therefore, it is no longer a simple Bernoulli process (for example there can be more than one arrival per time slot). Still, the Markov chain is time-homogeneous, irreducible, finite, and aperiodic and has a simple birth-death structure. The state variable \( i \) counts the number of slots in the queue and its steady-state probability distribution can be determined by using the detailed balance equations [19] between neighbouring states:

\[
\pi_0 = \frac{1 - q}{1 - q^N}, \quad \forall i \leq N : \pi_i = q^i \pi_0,
\]

where \( q = \frac{r_1}{1 - \frac{r_1}{1 - r_2}} \). With the probability of a packet drop at time \( t \) due to a full buffer denoted as \( D(t) \), the steady-state drop probability can be calculated as

\[
p_D = \lim_{t \to \infty} P(D(t)) = \pi_N.
\]

The steady-state probability of overload is calculated as

\[
p_o = \sum_{i=0}^{N_1} \frac{N - i}{1 - q^{N_1}}.
\]

3) Markov Chain for Uncertain Slot Dynamics: The second Markov chain models the position of the first uncertain slot reserved by Class 1 in the buffer spaces numbered 1 to \( N_1 \). If an uncertain slot reaches the head of the buffer and no packet of Class 1 is in the packet queue, this slot expires; this is accounted for by State 0 and thus the interest is in the steady-state probability of State 0, denoted as \( \pi_0 \). Another peculiarity of this Markov chain is the State \( G \), which accounts for the situation that there is no uncertain slot in the buffer spaces from 1 to \( N_1 \). Remember that the Markov chain is conditioned on the system being in overload. Therefore, from State \( G \) new uncertain slots are generated with rate \( s \). This is where the two Markov chains interact and the slot generation rate is set to \( s = r_1 (1 - p_D) \), with \( p_D \) calculated from the Markov chain for the total queue length (see Eq. 1). For ease of presentation, \( s \) is used to denote the slot generation rate and \( r = r_1 \) denotes the packet arrival rate of Class 1.

The Markov chain state diagram is shown in Figure 2. While this Markov chain is also time-homogeneous, irreducible, finite, and aperiodic, it is no longer reversible and thus not amenable to detailed balance equations. Instead, the
global balance equations have to be solved. The following proposition provides the steady-state probability distribution of the uncertain slot Markov chain (see [17] for the proof).

**Proposition 1:** The steady-state probability distribution for the uncertain slot Markov chain is given as

\[
\pi_0 = \frac{1}{1 + \sum_{i=1}^{N_1} \frac{\pi_0}{\pi_0} + \frac{E_{2i}}{\pi_0}} = \frac{1}{\frac{r_1}{r_1} + \frac{r_1}{1 - r_1} - \frac{r_1}{1} + \frac{r_1}{1 - r_1} - r_1}. \\
\pi_i = \frac{(1 - s)^{i-1}}{(1 - r)} \left( 1 - s (1 - r)^i \right) \pi_o \quad i = 1, \ldots, N_1, \\
\pi_N = \frac{1}{1 - r} \left( 1 - s \right)^{N_1} \left( 1 - s \left(1 - r\right)^{N_1+1} \right) \pi_o.
\]

4) **Connecting the Decoupled Chains:** Both Markov chains are connected again and thus, the throughput interference under DD can be assessed. Let \( r_1 = r \) again and \( s = (1 - p_D) r_1 \), as well as \( q = \frac{1}{1 - 2j(1 - r_2)} \), then the (steady-state) probability for an expired slot of Class 1 is

\[
p_{E_1} = p_{E_1|Q_1}, \quad p_{O_1} = \frac{q^{N_1+1} - q^{N_1} + 1}{\frac{r_1}{r_1} + \frac{r_1}{1 - r_1} - r_1 (1 - s)}. \\
T_{DD} \approx 1 - \frac{\left( 1 - p_{E_1} - p_{D_1} \right)^2}{2 \left( \frac{1 - p_{E_1} - p_{D_1}}{r_1} + \frac{p_{D_1}}{r_1} \right)^2}.
\]

This predicts a very high \( T_{DD} \) for DD under low-rate traffic with low delay requirements, as shown in Figure 3a for a total load of 1, i.e., \( r_1 + r_2 = 1 \). In an extreme case, the \( T_{DD} \) of DD reaches 0.5, which is the maximum \( T_{DD} \) possible for two flows. Hence, DD on its own, while guaranteeing delay properties of ICDS [18].
In order to solve each horizontal queue from Figure 4, observe that each row $j$ has the structure of a simple queue when considering only the transitions between $(Q_j,0),\ldots,(Q_j,S_1)$ (ignoring transitions between rows). Using states $(i)$ with $i \in 1,\ldots,S_1$ to denote the length of this simple queue, the local solution is obtained, for $0 < i \leq S_1$:

$$\tilde{\pi}_i = q^i \tilde{\pi}_0, \quad \tilde{\pi}_0 = \frac{1-q}{1-qS_1-T_1}. \tag{3}$$

In order to bring both local solutions together, each row $j$ is normalized to match $\pi_Q$, i.e., $\sum_{i=0}^{S_1} \tilde{\pi}_i = \pi_Q$. This leads to an approximation for the steady-state distribution of the overall Markov chain for each state $(j,i)$: $\hat{\pi}_{j,i} = \tilde{\pi}_i/\pi_Q$. This is finally used to obtain the approximate $T1^2$ for DD+QS.

**Proposition 2:** A DD+QS scheduler has

$$T1^2 \approx 1 - \frac{\left(1 - \frac{\lambda_2^n}{2} \sum_{j=0}^{S_2} \hat{\pi}_{j,S_1} + \frac{\lambda_1^n}{2} \hat{\pi}_{S_2,S_1}\right)^2}{\left(1 - \frac{\lambda_2^n}{2} \sum_{j=0}^{S_2} \hat{\pi}_{j,S_1}\right)^2 + \left(1 - \frac{\lambda_1^n}{2} \hat{\pi}_{S_2,S_1}\right)^2}$$

**Proof:** The steady-state loss rate of each class is derived by combining the local solutions Eq. (2) and Eq. (3). Loss for Class 1 occurs, if there are two arrivals in one of the right-most states in Figure 4. Therefore, the output rate of Class 1 is $\lambda_1^n = r_1 - \frac{\lambda_1^n}{2} \sum_{j=0}^{S_2} \hat{\pi}_{j,S_1}$. Loss for Class 2 occurs, if there are two arrivals in state $(S_2,S_1)$. Accordingly, $\lambda_2^n = r_2 - \frac{\lambda_2^n}{2} \hat{\pi}_{S_2,S_1}$. The final equation then follows according to Definition 1 by letting $\lambda_1^n = r_1$ and $\lambda_2^n = r_2$.

The $T1^2$ of DD+QS is calculated using Proposition 2 for different delay targets and relative arrival rates and the results are shown in Figure 3b for a total load of 1. Consistently, the $T1^2$ stays below 0.02, and is clearly much better than for DD only. Furthermore, it is only slightly higher than FIFO’s $T1^2$ (not shown) under the same circumstances, so it can be considered a fairly low value. Based on these results we conclude that using QS and applying the LIFO service principle between queue segments is key to achieving low throughput interference.

**VI. Evaluation**

All simulations reported in this section are run in the ns-3 environment and are available online for reproducibility.\(^1\) The simulation scenario is a single-bottleneck dumbbell topology with synthetic workloads. This choice is deliberate. While a simple structured topology and synthetic workloads are not as “realistic” as other setups, they are better suited to develop a systematic understanding of this novel scheduler. The workload keeps the average load around 100%, because this is the interesting operating regime for the scheduler. The default configuration is 100 Mbit/s bottleneck rate, 1ms link propagation delay, 60s simulated time, and 20 repetitions.

**A. Validation**

The first experiment is designed to validate the $T1^2$ and delay characteristics of DSF in comparison with FIFO and static priority scheduling (PRIO). The workload is comprised of 4 traffic classes that each send at 25% of the bottleneck capacity on average. Each traffic class uses 32 on-off traffic sources with on and off periods drawn from a Pareto distribution with shape 1.4. The FIFO and PRIO schedulers are configured with a static buffer equivalent to 100ms queuing delay. With PRIO scheduling, each of the traffic classes uses a separate priority level. The DSF configuration provides 4 delay classes of 10ms, 50ms, 100ms, and 200ms. Each of the traffic classes uses one of the delay classes. The observed values are the average queuing delay and throughput over 10ms time intervals. The $T1^2$ is computed in 10ms steps using a sliding window of length 1s. The experiment is run once for 600s simulation time, because the observation is longitudinal over time. This longer time interval is sufficient to capture typical random effects. X-axes are shown in logarithmic scale to ensure visibility of all data points.

Figure 5a shows the empirical cumulative distribution function (CDF) of $T1^2$ values for the different schedulers. It can be seen that PRIO scheduling has worse $T1^2$ values overall, which corresponds to the understanding that its service rates do not match arrival rates as closely as FIFO or DSF. Note

\(^1\)https://cs.uwaterloo.ca/~emkarsten/nidd/
that the maximum $TT^2$ of PRIO is recorded as approximately 0.031, which corresponds to a throughput swing of 36%. The equivalent maximum numbers are 2% swing for FIFO and 4% swing for DSF. This observation is complemented by the empirical CDFs for queueing delay shown for each scheduler in Figures 5b-5d. For FIFO’s shared queue, there is only one delay curve and average delays go up to the maximum queue length of 100ms. With PRIO, given the load applied in this experiment, the three higher-priority classes show extremely low average delay, which comes at the expense of a high delay of up to 500ms for the lowest-priority class. DSF differentiates the delay for each service class. While these graphs show the average delays over small periods of time, the observed worst-case delays for DSF are in fact bounded by the given targets.

A second experiment revisits the $TT^2$ comparison between DSF and a system performing only Delay Discard, such as VIFQ. The different $TT^2$ values predicted by the analytical model in Section V become manifest in comparable packet-level simulations. The simulations are configured to mimic the analytical setup and run for 600s. The results presented in Figure 6 show the long-term $TT^2$ average over the whole duration of the experiment and thus confirm the structural advantage of adding Queue Segmentation to the scheduler.

B. Traffic Scenarios

The following experiment verifies that DSF is effective and can provide several delay classes – without deviating much from FIFO service in per-class throughput. The DSF scheduler at the bottleneck link is configured with 10 delay classes at 10ms, 20ms, ..., 100ms. Correspondingly, there are 10 traffic classes with an average respective load of 10.1% of the bottleneck capacity; each uses one delay class. In the first part, each traffic class is comprised of 8 Poisson sources.

In a second part, each traffic class is comprised of 32 Pareto sources with a shape value of 1.4. The DSF configuration is compared to a FIFO system with 100ms buffer.

Figure 7 shows the distribution of average throughput for each traffic/service class. The box-and-whisker plots show the average throughput, as well as the 10%, 25%, 75%, and 90% quantiles. The results also confirm the conjectured properties of DSF. The average throughput per class almost perfectly matches FIFO throughput. Traffic sources loose a slight but increasing fraction of packets when choosing smaller delay targets – evident when comparing the mean values for FIFO and DSF in Figure 7 across service classes. The experiment has been repeated with various numbers around 100% total average load without any significant change in outcome.

C. TCP

DSF provides delay guarantees for any number of service classes with very low throughput interference. However, TCP has two effects that relate buffering to goodput: 1) Higher levels of flow multiplexing reduce the overall amount of buffering needed at the bottleneck to achieve good link utilization [3, 29]. 2) Long-term TCP goodput is inversely proportional to the RTT [23]. Thus, DSF potentially has indirect side effects on TCP goodput through both delay control and buffer management. This is investigated by testing a comprehensive traffic mix with FIFO configurations for 5ms, 20ms, 100ms, and 200ms, as well as two DSF scenarios with 4 service classes for the four delay targets. The regular (DSFR) scenario assumes that all TCP traffic uses the default 200ms delay class, regardless of propagation delay. TCP traffic is typically throughput-oriented and at 200ms, long-term TCP throughput is not affected by bufferbloat [10] effects. In contrast, the impatient (DSFI) scenario investigates the effect of TCP traffic with smaller RTTs using a smaller delay class. Based on the classical notion of RTT unfairness and the concerns stated in the ABE [14] and RD [25] proposals, this could be expected to result in a highly imbalanced bottleneck capacity allocation.

A synthetic traffic mix, specified in Table II, is loosely modelled after Sandvine’s Global Internet Phenomena Report [27].
which reports traffic comprised of real-time entertainment (40-60%), web browsing and social networking (15-30%), and communication services (5-10%). Data traffic uses TCP NewReno while Voice, Video, and Other traffic is modelled as UDP. Each Data traffic flow performs renewable file transfers with sizes drawn from a Pareto distribution with shape 1.2. The traffic mix is scaled to various bottleneck link rates by proportionally scaling the number of flows in each traffic class.

Figure 8a shows the link-level throughput measurements for each service class at different bottleneck link rates. It confirms that DSF does not unduly influence throughput of service classes. DSFI shows a higher throughput for the 20ms classes, because the TCP flows from TC3 enter this class. To properly investigate TCP-level effects, Figure 8b shows the aggregate TCP goodput for each of the TCP traffic classes. The general problem with a small-buffer configuration is visible for F5 and F20, albeit not as strong as prior work has shown in certain scenarios (cf. Section III in [17]). The nature of TCP’s RTT unfairness is visible throughout all configurations, because TC3 generally obtains a significantly higher share of the bottleneck capacity than TC4. When taking F100 or F200 as the benchmark, DSFR performs equally well in terms of goodput and RTT unfairness, and much better than F5 or F20. This demonstrates the essential benefits of DSF. The results for DSFI are somewhat inconclusive. In the 100 Mbit/s experiment, RTT unfairness is reduced, while overall utilization suffers. In the 200 Mbit/s configuration, DSFI outperforms all other scheduler configurations. At 400 Mbit/s link speed, RTT unfairness is slightly higher, but DSFI is still better than F5 and F20. Thus, even in scenarios where the classical TCP models predict RTT unfairness, DSF appears competitive.

VII. CONCLUSION

The goal of this work is providing multiple delay classes while adhering to strict network neutrality requirements. The DSF scheduler is presented as an effective solution. DSF can provide multiple service classes with arbitrary delay targets. It is minimal and modular by focusing on a single feature (delay differentiation) with very little side effects. It is shown analytically and with simulations that delays are enforced without significant throughput interference. A non-intuitive outcome is that DSF does not necessarily cause additional RTT unfairness for TCP traffic. This poses an interesting challenge to possibly refine existing TCP models.
Self-Optimization of Software Defined Radios Through Evolutionary Algorithms

Zubair Shaik, André Puschmann and Andreas Mitschele-Thiel
Integrated Communication Systems Group, Technische Universität Ilmenau, Ilmenau, Germany
Email: [zubair.shaik, andre.puschmann, mitsch]@tu-ilmenau.de

Abstract—This paper presents a framework for building software-defined radios that are able to self-optimize their parameters using evolutionary algorithms. The framework has been implemented using the DEAP library for Python, which is based on the Genetic Algorithms (GAs). The paper discusses the overall system architecture and presents a system prototype that has been employed to optimize radio transmission parameters in an unknown radio environment in order to maximize the achievable throughput. Although GAs have been used before for optimizing the radio parameters of Software Defined Radios (SDRs), they have been limited to the number of parameters given as an input to the GA. The proposed algorithm is much more generic and comprehensive to utilize the advantages of genetic algorithms, by providing the flexibility to include any of the parameters of the configuration of the SDR, which needs to be optimized through the GA. Moreover, the entire project is based on open-source solutions. The current prototype targets Iris-based SDRs. However, as the entire software employs standard components for interfacing the SDR, it can easily be ported to GNU Radio or other SDR frameworks. We will also present preliminary results that have been obtained through over-the-air experiments in which we optimized different power parameters, modulation, coding schemes, etc., in an unknown radio environment.

Keywords—Software Defined Radios (SDRs), Distributed Evolutionary Algorithms in Python (DEAP), Iris, Genetic Algorithms (GAs), Multi-Objective Genetic Algorithms (MOGA)

I. INTRODUCTION

Wireless communication systems are built in a layered fashion comprising a multitude of protocols and services. The wireless signals are too volatile to its surroundings and to figure out the optimal working conditions is a challenging task. Software defined radios (SDR) provide the flexibility to implement these different services on a software platform, but the physical layer modules of modulation and coding or the MAC layer protocols. The configuration space of an SDR is too huge, and it becomes difficult for a regular user to figure out a good working set of these settings. We utilize the flexibility of an SDR to create a self-optimizing radio that intelligently adapts to its surroundings and works in an optimal scenario.

Generally, too many parameters guide the working of a wireless radio. The solution space includes many possible combinations of the parameters that govern the wireless radio, like frequency, bandwidth, transmit/receive power, modulation and coding schemes, gain values etc. These parameters could exist from the application layer to the MAC and physical layer. The number of combinations of these parameter values would be in millions and an exhaustive search would take years to figure out the best possible combination. The objective of the cognitive radio is to find tractable solutions for optimizing the radio link but in a desired time applicable to the user. Various algorithms have been proposed over the years to optimize the parameter configuration. Doerr et al. [1] classified these algorithms into four main categories: genetic algorithms, game theory, rule-based reasoning and neural networks. We choose genetic algorithm as it is good for large solution space and gradually evolves its solution set based on the experiences during its evolution.

This paper focuses on the self-optimizing radio which employs genetic algorithm to adjust its radio parameters. The radio engine developed is based on the following open-source solutions: (1) DEAP [2], an evolutionary computation framework for Python, (2) Iris [3], an application framework for building reconfigurable SDRs, and (3) OSPECORR [4], a middleware used to connect the individual software components together. An early prototype of the platform has been presented as a talk at FOSDEM 2016 [5].

The remainder of the paper is sectioned as follows: In Section II, we briefly describe genetic algorithms and give an overview about multi-objective decision making. Section III describes the application of GAs to software defined radios. Section IV-A presents the design and the component structure of the proposed system, while describing the employed genetic algorithm using DEAP. Section V contains the experimental results of the algorithm. Finally, we summarize our findings and conclude the paper in Section VI.

II. GENETIC ALGORITHMS

Genetic algorithms (GAs) are heuristic search algorithms inspired by biological observations. They adapt based on the evolutionary ideas of natural selection and genetics. As such, they belong to the class of evolutionary algorithms. The techniques used in GAs are based on the natural evolution, such as selection, mutation, crossover, and inheritance. These heuristics are frequently being used for optimization problems, as they exploit the information during their evolution to direct the search into a solution space that generally gets better each time. In particular, where the state space is large, multi-modal or a n-dimensional surface, GAs provide significant optimization solutions when compared to typical search techniques such as depth-first, breadth-first or praxis [6].

GAs are based on two components. The first component is the genetic representation of the possible solution space, the so-called chromosomes. This physical representation of a particular solution of the algorithm is modeled through bit strings of variable length that contain the value of each parameter, i.e., the gene. The second component is the fitness
function that evaluates each solution set. It can be a simple formula or a statistic from a complex simulation.

The algorithm starts by randomly generating an initial population, which is a pool of individual potential solutions. It evolves through three basic operators of selection, crossover and mutation. Selection involves giving preference to individuals, which allows their genes to be passed on to their future offspring. Each individual is associated with a fitness value that can be determined by the fitness function. This ensures that the individuals with a good fitness value quickly dominate the population. Mutation and crossover are variation operators used to create different children, but still related to their parents. Mutation is done to one individual at a time, and involves bit flipping of their bit strings, where as crossover operator takes two or more parents to make a child which is a combination of them. The probability values of crossover and mutation are given as inputs to decide the intensity of variation of the offspring. Survivor selection mechanisms based on the fitness and age of the parents and offspring determine which individuals are carried on for the next generation.

The GA, using the operators of selection, crossover, and mutation gradually evolves through generations to converge towards a global optimum solution [7]. A termination condition has to be applied to the genetic algorithm, to prevent it from an infinite or an exhaustive search. The termination condition can be based on the number of generations or an fitness threshold. This could be either an absolute value or when there is minimal change in the fitness values of the population. Figure 1 illustrate the basic process flow of a GA.

![GA process flow](image)

GAs have been mainly applied to problems with a single objective. However, most of the real-world problems are multi objectives. For example in the case of wireless communication systems, it’s not always the maximum throughput alone that’s an objective, but along with it, the transmit power, frame error rate, etc., also need to be considered. In Multi-objective Genetic Algorithms (MOGA) [7], weights can be attached to the individual objective functions, directing the results to achieve the maximum/minimum values as required.

To give an example, consider two objective functions of a GA, \( f_1(\cdot) \) and \( f_2(\cdot) \). Further assume that \( f_1(\cdot) \) is to be maximized and \( f_2(\cdot) \) to be minimized. In order to optimize for both objectives, MOGA uses a weighted sum of multi-objective functions to form one scalar fitness function. With the search evolving in different directions, a set of Pareto-optimal solutions are achieved, with individuals of high fitness values selected from each set to form the elite of the next generation. Pareto optimality is a state of allocation of resources where any change beneficial to one individual is detrimental to one or more others. When no further pareto improvements can be made, an allocation is called Pareto-optimal [9]. Figure 2 illustrates the direction of search of a MOGA with four non-dominating solutions, all laying on the Pareto front.

![Directions of the search in MOGA](image)

**III. APPLICATION OF GENETIC ALGORITHMS TO SDRs**

For a successful wireless transmission, the radio must be configured according to the channel conditions. The channel conditions are very diverse in nature and the impact of them on the radio transmission is very unpredictable. Numerous solutions have been proposed over time to stabilize the radio transmission, which includes better modulation and coding techniques, the use of right power and gain values, frequency selection, symbol rates etc. Looking at the huge solution space, it turns out to be a very daunting task to figure out which configuration works best. The idea is to let the GA decide and figure out the optimum solution.

By providing the range of feasible values for all the parameters that effect the radio transmission, even in an unknown radio environment, the algorithm takes a comprehensive control and starts the optimization process through gradual evolution. The time required to arrive at a good optimum solution is directly proportional to the solution space.

The parameters of both the transmitter and the receiver that are to be optimized form the genotype of an individual, as shown in Figure 3. Note that the genes of the chromosome may have different lengths, depending on how many feasible values exist per parameter.

The initial set of individuals are randomly generated by assigning random values to each of the parameter from their corresponding solution sets. Traffic conditions of the radio transmission govern the fitness evaluation function of the GA. The termination condition is controlled by the evolved number of generations. The parameters of the GA, like number of individuals, number of generations, the crossover and mutation probability are configurable, so as to let the user control the GA as required.

Probably among the first to apply GA to SDRs were Rieser and Rondeau. Rieser et al. [10] provide a cross-layer mechanism to deliver the requested Quality of Service (QoS) through Cognitive System Monitor (CSM), by controlling...
parameters like power, frequency, modulation type, FEC and TDMA timeslot ratios. Building upon that work, Rondeau et al. [11] point in the direction of controlling more parameters by providing dynamic fitness selection and evaluation. Their proposed Wireless System Genetic Algorithm (WSGA) is a MOGA based algorithm to realize cross-layer optimization of a radio. The parameters of PHY and MAC layer form the genes of a chromosome, and their analysis is done through fitness functions defined by evaluation of the radio channel. These fitness functions are dynamically linked from the database, so as to add and weigh them in the fitness function for the evaluation of the wireless link.

The algorithm proposed in this paper is inspired by Rondeau et al. [11]. It extends the dynamic functionality by including not just the fitness function but the configuration space as well. Any configurable parameter of the SDR, be it from the application, MAC, or PHY layer, can be dynamically included to form the chromosome of the individual. Furthermore, because it’s not just the radio properties of the transmitter that affect the wireless link, the receiver functionality too has to be optimized. The current algorithm includes the configuration of the transmitter as well as the receiver in the chromosome and optimizes both of them simultaneously to achieve the best results.

Very recently, Kozel [8] has employed GAs to optimize digital modulation schemes, i.e., to find a constellation that affects the wireless link, the receiver functionality too has to be included as well as the specific components used for the prototype implementation. However, this work is only based on simulation and does not consider any over-the-air experiments.

![Figure 3. Representation of a chromosome containing genes with a variable length (reproduced from [12]).](image)

**IV. SELF-OPTIMIZATION FRAMEWORK**

This section presents the proposed framework for enhancing existing SDRs in order to allow them to self-optimize their communication parameters in real-time. We will first explain the high-level system design and then provide details about our prototype implementation based on Iris.

**A. System Design**

The system architecture consists of two main components: the optimizer client and the optimizer controller, as shown in Figure 4. The client and the controller represent the transmitter and the receiver of the radio link respectively. The optimizer controller is the basic component that governs the complete process of optimization. The client is a passive module following the protocol set by the controller, executing its commands, and sending back the requested statistics. The controller has been developed in a way that it can be run independently from the transmitter and the receiver, e.g., on a different host machine.

The optimizer controller and client are connected over separated control and data interfaces. This has two main advantages. First, signaling messages sent over the control interface do not negatively effect measurements on the data interface. Second, and more importantly, a separated control interface allows to also configure radio parameters that result in a non-working communication link - something that may happen anytime during the optimization - without having to worry about how to detect and repair such a situation during the optimization procedure.

To evaluate the fitness level of each configuration of the individual, the results of the data transmission are to be computed. nuttcp tool which is used to generate the data traffic provides the results of the data transmission in terms of throughput. OSPECORR provides the means to collect the physical or MAC layer properties like EVM, RSSI, etc., which are accumulated through the time of data transmission. These computed results are sent to the optimizer controller through the control interface.

Figure 4 illustrates the core building blocks of the system as well as the specific components used for the prototype implementation, which is described below.

![Figure 4. System Design](image)

**B. Principle of Operation**

The message sequence chart of the controller protocol is shown in Figure 5. The optimization starts by opening a TCP connection between the optimizer controller and the optimizer client, over the control interface. The optimizer controller configures the client with the required settings for the data transmission of nuttcp and initiates the GA to continue its evolutionary process. During the GA, the controller is tasked with configuring the settings of each individual on the client, to signal the data transfer and then to collect the results of the data transmission. When the termination strategy decides the end of optimization, the controller configures the best selected configuration and terminates the optimization.

The GA itself is realized by leveraging DEAP, an "evolutionary computation framework for rapid prototyping and testing of ideas" [2]. The algorithm initially registers the modules required for the GA, like individual, population, evaluate,
their results prove that NSGA2 is able to search better solutions towards the optimum pareto front. The algorithm starts its run by creating a set of individuals by randomly selecting the parameter values in their genotypes and calculating the fitness of each individual. After the initialization, the evolution procedure begins, going through the mechanism of crossover and mutation of the offsprings and computing their fitness. The evolution continues till the terminate function decides to stop the evolution. In every evolution, a set of offsprings are formed by copying the individual chromosomes from the current generation. The selection procedure from the parents is based on NSGA2, as described before. In this algorithm, an equal number of offsprings are generated as that of parents. These offsprings now go through variation operators of mutation and crossover, to create individuals with different properties, but which are still related to their parents. In crossover, individuals are created by a combination of chromosome bit-strings of two parent individuals, by calling the mate operator. The offsprings then go through the mutation operator, which flips the bit-strings of their chromosomes to make new individuals. Crossover and mutation probability parameters decide the intensity of variation.

Fitness values of these newly created offsprings are calculated and they go through the selection phase of NSGA2 again. The population for the next generation is created by selecting the individuals among the parents and the offsprings, which have the best fitness values. The next step is the termination test, which decides if the evolutionary process has to be stopped or continued. In our algorithm, we stop the process after a fixed number of generations as selected by the user. But the logic can be extended to include conditions like, minimal change in the fitness values of the best individual in each generation, or until an absolute fitness threshold is reached. If the termination test is not met, the process continues to create new generations.

### C. Prototype Implementation

The prototype system consists of a uni-directional communication link between a SDR transmitter and receiver. We employ Iris as the underlying SDR framework and utilize the reconfigurability to allow the GA to reconfigure the radio parameters during run time. The optimizer controller is an extension of the Python-based *pySysMoCo* module of *OSPECCORR* [4]. *pySysMoCo* is a graphical application to monitor and control various parameters of the SDR. In particular, this allows to select the parameters and settings for the optimization algorithm through a graphical user interface. Furthermore, it allows to display the fitness of the optimization while it is running.

![Controller GUI](image)

Figure 6. Configuration of the algorithm through the controller GUI.
client interface as well the core parameters of the GA, e.g. mutation rate.

The fitness of the system evaluated by generating constant bit rate UDP traffic using nuttcp. Throughput and data loss form the two objective functions of fitness evaluation, with the former to be maximized and the latter to be minimized. In addition to that, we also employ error vector magnitude (EVM), a physical layer metric to quantify the performance of a digital communication system, as an additional objective function.

For the fitness value computation, the optimizer controller sends a frame to the optimizer client over the control interface, to execute nuttcp with the specified configuration. After the nuttcp has finished, the optimizer client sends the results to the controller. The controller then computes the fitness value of each individual in the population.

In our prototype, the control interface is a realized over a TCP connection over WLAN. The data interface represents the wireless connection that needs to be optimized, i.e., the SDR link.

V. EXPERIMENTAL EVALUATION

In order to evaluate the effectiveness of the GA, we have carried out a large number of experiments in our lab. For all experiments discussed in this paper, we employ a USRP B210 on the transmitter side and a RTL-SDR dongle on the receiver. The RTL-SDR is based on the Realtek RTL2832U chip, an inexpensive USB dongle which has a highest theoretical sample rate of 3.2 MS/s.

Table II shows the adaptable parameters selected on the transmitter and receiver configuration to run the experiment. The first column represents if the configuration is of the transmitter or the receiver, whose parameter configuration is handled accordingly by the optimizer controller.

Table II. ADAPTABLE PARAMETERS OF SDR

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Tx/Rx</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modulation</td>
<td>bpsk, ook, dpsk, apsk16, apsk32, qpsk, ...</td>
<td>Tx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td>none, repl, repl5, b74, ...</td>
<td>Tx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>software gain</td>
<td>-20 to 0 in steps of 2</td>
<td>Tx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>usrp tx gain</td>
<td>50 to 90</td>
<td>RX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rtl rx gain</td>
<td>0 to 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table II shows the adaptable parameters selected on the transmitter and receiver configuration to run the experiment. The first column represents if the configuration is of the transmitter or the receiver, whose parameter configuration is handled accordingly by the optimizer controller.

In general, the GA is able to include any of the exposed radio parameters in the optimization procedure. In our experiments, however, we limited the number of parameters to those mentioned in Table II. Some of those parameters, if configured on transmitter, the same value has to be configured on the receiver too. The optimizer algorithm checks for such parameters and configures them accordingly.

The list of modulations and coding is much more than what is mentioned in the table. As has been mentioned above, in order to evaluate the fitness we consider the data throughput acquired by running nuttcp as well as the EVM obtained from the physical layer. Of those two functions, throughput represents the function to be maximized and the absolute value of EVM is to be minimized. For the multi objective selection, throughput is given the higher weight than that of EVM. To calculate the baseline to be used later for comparison, we manually configure the radio to use 16-QAM as modulation

scheme and known values of power and gain. With this setup, we were able to achieve a maximum throughput of 1.7 Mbps.

The parameters of GA used in the experiments are mentioned in Table III. They are those used in [11]. But we run the experiments by increasing the mutation probability in each run. This allows for the increase in the variation of the offsprings from their parent chromosome, to discover better solutions. Initially, we run the experiment based on the parameter values from the column of set 1 and 3. In this experiment, the algorithm focuses on maximizing the fitness function of just the throughput.

Figure 7 shows the throughput values of the individuals produced in each generation over time. It can be observed that with a lesser probability of mutation, the throughput values of the individuals are less scattered, and stay close to that of their parents. In the initial stages, the throughput of the individual with the best fitness value, increases by a good amount, but starts saturating with the increase in time, leading us through the termination condition.

Although the test stops after 30 generations, we can observe that the high throughput has been attained in the halfway of the process. But with an increased mutation probability of 50% from the set 3, the algorithm tries with more variation in the radio parameter values while getting different throughput results than that of the parents. But each generation when formed consists of the elite individuals from the set of parents and the offsprings. Such a case, with a higher rate of mutation, allows variation in the parameter values, figuring out the results of some radio parameter combinations that could have been missed. Although in this experiment, both runs produce
the same maximum throughput of 3 Mbps. The experiment takes beyond 5000 s to run for 30 generations, and that is because of nuttcp’s timeout whenever the algorithm does not produce an useful configuration, i.e., produces 100% packet loss. The transmitter waits for a relatively long amount of time before trying out the next solution. This does not affect the optimization as such but can be changed by either modifying the timeout value or by using another application for traffic generation.

In the next experiment, EVM was included along with throughput in the fitness functions. This experiment was run based on all three parameter sets of the GA from Table III. Figure 8 shows the throughput values of the best individual in each generation. To compute the theoretical maximum throughput for the setup used in the experiment seems difficult. Comparison with other systems/approaches is difficult too because most radios use very different system parameters, i.e. bandwidth, frequency, available modulation and coding schemes, etc. Using the lower sample rate of RTL-SDR of 3.2 MS/s among the two radios and with a spectral efficiency of 2 bits/Hz, the optimistic maximum throughput can be derived to be 6.4 Mbps. In the first run, with a mutation probability of 5%, a maximum throughput of 3.2 Mbps was achieved, which had 16-QAM as the selected modulation scheme in its solution space. By increasing the probability to 25% from the set 2, the algorithm acquired a maximum throughput of 4.3 Mbps. In this case, V29 was selected as the modulation scheme to achieve such high throughput. Even in the previous run of 5% mutation probability, V29 was tried among its individuals, but the right combination of gain values was probably not selected.

The last run had a 50% mutation probability, which resulted in 4.67 Mbps of throughput. Even this run selected V29 as its modulation scheme solution, but by attempting different values of gain, it was able to achieve such higher results. Increasing the mutation probability beyond 50%, did not result in an increase in throughput anymore. We also observe that in all three cases, a very good throughput result was attained in less than 3 generations. After which the best throughput of the each generation increases slowly. In the first case, the maximum throughput was attained in 7th generation, after which it saturated. For the remaining two cases, it was attained midway, after around 18th generation. Once the algorithm figures out the best solution, it configures the selected radio parameters of the SDR. By initiating a data transfer, through nuttcp, we consistently get the best throughputs achieved by the algorithm. Both the experiments were run for 30 generations each, which is certainly not enough to guarantee the global optimum. It usually only achieves a local optimum. But genetic algorithms are well known to converge to a global optima eventually [11] [7].

VI. Conclusion

In this paper, we presented a multi-objective genetic algorithm based optimization approach to configure SDRs. By giving a complete unknown radio environment with a wide range of input parameters, the algorithm optimizes the radio configuration of the SDR to result in an optimum wireless transmission, while achieving the objectives of the evaluation function. In our experimental system, we observed a 180% increase in throughput while using the proposed algorithm when compared to a known manual configuration of the SDR. Although the time taken to achieve the optimized solutions is not always feasible for the bootstrapping time of a radio, the algorithm proves efficient for radios that are non-mobile and where the wireless conditions do not change too much. Understandably, if the surroundings change to worsen the selected configuration, the algorithm needs to be restarted again.

The proposed algorithm improves the performance of the wireless link between any two SDRs by optimizing the radio parameters of frequency, bandwidth, modulation, coding, or any configurable parameter on which the wireless link depends on. It is certainly not possible for the user to manually try the huge solution space of radio configuration or even to understand how the combination of these parameters influence the wireless link. Moreover, the algorithm optimizes the radio properties of both the transmitter and the receiver at the same time. We also presented ideas to improve the efficiency of the optimization results and to increase the convergence speed to determine an optimal parameter set.

The current algorithm works, but can still be made better to fit any user requirement specifications, by optimizing the GA parameters, the selection of fitness functions and their weights, the population selection strategy, and the termination criteria. Like EVM from the physical layer, which is considered as one of the fitness functions in our experiments, other parameters from the MAC/PHY layer can be added to the algorithm to achieve the desired results. The change in mutation probability is done manually in each experiment to attain the desired results. In the future, we plan to change the mutation/crossover rate from generation to generation through the algorithm, starting from a higher rate and decreasing it over the evolution. This would allow the algorithm to try varied solutions before converging towards a local optimum. We further plan to incorporate the optimization framework into GNU Radio [14].

The source code of the entire prototype [15], which requires a modified Iris module [16] is available under an open-source license for further extensions and improvements.
REFERENCES


Opportunistic Channel Estimation for Implicit 802.11af MU-MIMO

Ryan E. Guerra  
Rice University  
ryan@guerra.rocks

Narendra Anand  
Cisco Systems, Inc.  
nareanan@cisco.com

Clayton Shepard  
Rice University  
cws@rice.edu

Edward W. Knightly  
Rice University  
knightly@rice.edu

Abstract—Multi-User MIMO (MU-MIMO) linear channel coding can greatly increase wireless system capacity when Stations (STAs) have fewer antennas than the Access Point (AP), but it comes at the cost of significant Channel State Information (CSI) estimation overhead. Previous work has suggested that 802.11af MU-MIMO systems might benefit from long channel coherence time, extending the useful duration of CSI. In this paper, we propose and analyze an opportunistic channel sounding policy that avoids sounding overhead in wireless channels by gathering implicit CSI opportunistically. This policy not only avoids CSI overhead, but also has the potential to enable efficient interoperability of multi-user APs with legacy single-stream STAs. To investigate the performance of this new policy, we implement a new mobile channel sounding framework on a custom 802.11af Software-Defined Radio (SDR) system designed for UHF-band experimentation and evaluate channel sounding performance in indoor and outdoor environments under various mobility modes. Additional protocol analysis shows that in UHF channels with sufficient channel coherence time, an opportunistic channel sounding policy offers significant protocol optimization while improving the scalability of next-generation MU-MIMO systems.

Index Terms—MIMO, MU-MIMO, Beamforming, 802.11af

I. INTRODUCTION

MU-MIMO is a wireless channel coding technique that enables an AP equipped with multiple antennas to transmit simultaneous data streams to separate STAs, leveraging spatial diversity to scale data rates with the number of transmit antennas. For an AP to use this technique, it must first estimate the Channel State Information at the Transmitter (CSIT) between each of its transmit antennas and each receiving antenna through a method termed channel sounding. The estimated CSIT is then used to compute precoding weights for the multi-stream transmitter. CSIT can also be used for resource allocation, such as user grouping [1] and inter-cell interference mitigation [2].

IEEE 802.11af is a standard amendment for Wi-Fi to operate in unused UHF Television-band White Space (TVWS) channels [3]. The standard can also employ MU-MIMO features of IEEE 802.11ac [4]; here, explicit CSIT is obtained at the AP by first transmitting a sounding packet from the AP to the STA, then having each STA transmit the measured CSI to the AP as a control frame [5]. Unfortunately, the transmission overhead required for CSIT estimation increases with the number of transmit antennas at the AP, \( M \), and the number of aggregate STA antennas, \( K \), and recent results have shown that this overhead can severely decrease the achievable throughput gains [6], [7].

In this paper, we explore elimination of explicit channel sounding altogether via purely opportunistic channel sounding in which CSIT is implicitly estimated from each received uplink transmission, whether a data or control frame. Since each uplink frame already contains a training sequence in its preamble (e.g., the TVHTLTF in 802.11af [5]), we use every uplink reception from the STA to the AP, encompassing data, ACKs, and management frames in order for the AP to estimate downlink CSIT. This approach exploits a key property of UHF bands: they can be highly stable on the order of 100 ms while maintaining high multi-user diversity [8]. Thus, the opportunistic policy eliminates CSIT sounding overhead if the channel remains sufficiently unchanged between uplink transmissions.

We show that opportunistic sounding is beneficial in four operating regimes in which: \( (i) \) channel conditions are sufficiently stable such that beamforming error due to obtaining CSIT from a prior uplink transmission is negligible; \( (ii) \) legacy 802.11 STAs cannot respond to beamforming requests and otherwise could not leverage full spatial diversity; \( (iii) \) the number of spatial streams grows such that even implicit channel estimation generates significant overhead; and \( (iv) \) the Modulation and Coding Scheme (MCS) is sufficiently high that any wasted airtime due to channel sounding overhead imposes a high relative cost. Scenario \( (ii) \) is of particular interest because it enables new 802.11 APs with multi-user capabilities to operate in spectral-efficient multi-user modes with legacy 802.11 equipment that does not otherwise support multi-user modes.

To explore the key performance factors of opportunistic sounding, we design and manufacture a custom MIMO SDR front-end for the WARPv3 SDR platform [9]. This platform enables the first characterization of mobile multi-user UHF channels, enabling evaluation of opportunistic sounding even in the presence of STA or environmental mobility. The design simplifies high-power UHF-band 8x8 MIMO experiments and improves synchronous clocking over previous SDR testbeds. In addition to implementation of custom SDR hardware, we modify a novel SDR channel sounding framework designed for high-speed mobile implicit multi-user channel measurements [10] and port the framework to operate on our UHF equipment.

Finally, we obtain experimental radio licenses WH2XJV and WJ9XFF to operate our experimental equipment on UHF.
channels in Houston, TX and perform a series of indoor and outdoor measurement campaigns in various mobility scenarios to analyze MU-MIMO beamforming capacity with respect to CSIT overhead.

We find that fixed wireless nodes utilizing UHF spectrum exhibit long-term stable CSI under environmental and static mobility scenarios. Consequently, we find that with a low number of spatial streams, performance of both active and opportunistic implicit sounding policies significantly exceeds that of the current 802.11af protocol due to the reduced overhead of collecting CSI, even when taking into account the measured beamforming inefficiency of using delayed CSIT. We further extend our analysis to show that opportunistic implicit sounding with more spatial streams yields increasing benefits, enabling future systems with many more antennas than the current maximum of eight in commodity APs.

II. CSIT COLLECTION METHODS

In this section, we present the background necessary for understanding MU-MIMO beamforming, discuss the various approaches to acquiring CSIT and their tradeoffs, and propose a new method for acquiring CSIT opportunistically via implicit channel estimation that will be explored in this paper.

A. Application of CSIT to MU-MIMO

Downlink MU-MIMO is a transmission technique that enables a multi-antenna AP to simultaneously transmit separate data streams to a collection of STAs. This technique is enabled with linear precoding of individual data streams by a collection of complex steering weights. These weights create phase and amplitude-modulated copies of each data stream and simultaneously transmit them from each AP antenna.

Steering weights are represented as a complex weight matrix \( \mathbf{W} \in \mathbb{C}^{K \times M} \), the calculation of which requires knowledge of CSI, the complex magnitude and phase offsets between the transmitter and the receiver antennas represented as the complex channel matrix \( \mathbf{H} \in \mathbb{C}^{M \times K} \), at the transmitter.

A practical method for calculating \( \mathbf{W} \) from \( \mathbf{H} \) that approaches optimal performance is Zero-forcing Beamforming (ZFBF)\(^\dagger\). Zero-forcing drives interference between spatial streams to zero, and can be inefficient when users’ CSI is not sufficiently orthogonal \(^\dagger\). ZFBF requires calculation of the \( \mathbf{H} \) matrix’s pseudo-inverse:

\[
\mathbf{W} = \mathbf{H}^{\dagger} = (\mathbf{H}^\dagger \mathbf{H})^{-1} \mathbf{H}^\dagger,
\]

where \( (\cdot)^\dagger \) represents the matrix conjugate transpose and \( (\cdot)^\dagger \) is the Moore-Penrose pseudo-inverse. When the transmitter precodes with perfect ZFBF weights, \( \mathbf{W} \), signals ideally cancel the effects of the wireless channel at the receiver, allowing each user to receive their own, independent streams.

B. Implicit vs. Explicit Channel Sounding

Implicit beamforming relies on the assumption that the physical channel between the transmitter and receiver is reciprocal in nature so that estimating CSI in the downlink direction is equivalent to estimating CSI in the uplink direction and vice versa. Accurate array reciprocity calibration has been demonstrated \(^\dagger\) and \(^\dagger\) has experimentally demonstrated equivalent mutual information between downlink and uplink channel estimates utilizing transceiver hardware similar to our own. Therefore, we assume that uplink channel estimation is sufficient to estimate the downlink channel for our purposes, and we assume that all new APs will have the capability to perform reciprocity calibration and provide implicit channel estimation.

The benefits of implicit channel sounding vary based on node/environment mobility as well as the protocol and radio configuration utilized. For example, if the wireless channel varies rapidly due to high mobility, frequent channel sounding, whether implicit or explicit, will be required to obtain accurate CSIT. Per-packet channel sounding mechanisms that incur protocol overhead, such as the multi-user implicit sounding mechanism analyzed in \(^\dagger\) may be required to ensure that channel estimates are accurate in such environments.

However, in the case where the wireless channel remains coherent for long periods of time, for example, due to limited or lack of mobility, then it becomes possible to rely on previously collected CSI for current MU-MIMO transmissions \(^\dagger\), \(^\dagger\). Practically, such environments exist in wireless networks utilizing sub-GHz carrier frequencies, for instance TVWS networks \(^\dagger\), as well as certain fixed Wi-Fi networks.

C. Opportunistic CSIT Collection

In this section, we propose a new approach to collecting CSIT in 802.11af networks that avoids the overhead of MU-MIMO channel sounding altogether by relying on the opportunistic reception of implicit CSI for regular network traffic.

Fig. 1a diagrams explicit channel sounding, where first the downlink channel is estimated and then the channel estimates are fed back as data packets before each multi-user downlink transmission. This method, with additional polling and channel reservation overhead, is the version currently used in 802.11af \(^\dagger\). Explicit beamforming overhead scales as \( O(M \cdot K) \).

A proposed implicit sounding method \(^\dagger\) that transmits staggered Null Data Packets (NDPs) in the uplink direction allowing implicit downlink channel estimation following a multi-user trigger from the AP. Its timeline is similar to that of Fig. 1a, but instead the uplink packets are short NDPs rather than Compressed Beam-Forming Report (CBFR) packets, and polling is avoided. Implicit channel estimation overhead is significantly reduced from the explicit case since no CBFR polling or uplink payload is required before the downlink transmission. Implicit beamforming overhead scales as \( O(K) \) since all AP antennas are sounded simultaneously and is key for scaling \( M \), the number of AP antennas.

Fig. 1b displays our proposed opportunistic implicit sounding method that estimates the downlink channel implicitly from uplink data transmissions and utilizes that channel estimate for Multi-User Beamforming (MUBF) so long as it remains “fresh.” Acknowledgment (ACK) packets from a
successful downlink transmission can also refresh CSIT implicitly. Opportunistic implicit beamforming overhead scales as \(O(1)\) since no sounding overhead is injected to sound active STAs.

Given that the UHF channels for 802.11af networks can remain stable for relatively long periods of time, we target to avoid channel sounding altogether and rely on standard PLCP preambles [5] in overhead uplink transmissions to estimate the downlink channel since that estimate will remain valid over multiple packet timescales.

The key strategy for opportunistic sounding is as follows: when historical implicit CSI is available and “fresh,” the AP forms user groups and calculates precoding weights for the optimal multi-user transmission group determined by the MAC scheduler. We utilize two methods when implicit CSI is unavailable or stale for a particular STA: 1) a single downlink frame for the stale STA is de-queued and transmitted by the AP using MISO omni-directional transmission; the subsequent ACK will then provide an update of implicit CSI for that STA; or 2) alternately, the AP could fall back to legacy implicit sounding methods, e.g., [15], if no traffic is available.

In order to determine the feasibility of such an opportunistic sounding policy in 802.11af systems and explore the possible throughput gains, we measure a series of indoor and outdoor multi-user channel traces and perform protocol analysis to understand policy tradeoffs for opportunistic CSIT.

III. EXPERIMENTAL PLATFORM

Although MU-MIMO capabilities will be made available on “wave-two” 802.11ac ASIC chipsets in 2016, the research community is limited by the protocol modifications that can be made to commodity hardware. In addition, no 802.11af ASICs have been announced. For that reason, we have developed the hardware and software stack of a custom SDR platform that allows us to arbitrarily generate, intercept, and modify MU-MIMO transmissions.

Our approach to solving the processing delay in SDR equipment is to take advantage of the higher coherence time of low-frequency channels and perform over-the-air experiments on selected UHF (470-698 MHz) channels, where the processing latency becomes much less significant [8].

A. Hardware Platform Design

We extend the Wideband UHF Radio Card (WURC) UHF test equipment that we developed in previous work for rapid physical-layer prototyping in UHF bands [8], [16]. WURC was designed to enable high-power transmission up to 1 W and reception of wide-band radio signals in frequencies between 470-698 MHz [16] and each module provides one complete analog radio chain for use with a single WARPv3 SDR baseband [9]. Multiple WARPv3 boards can be clock synchronized with a daisy-chained reference clock and shared sampling trigger.

However, the equipment and daisy-chain topology of [8] suffers from a clocking topology that introduces additional transmission and reception phase errors and aperture jitter as the reference clock signal is forwarded [17]. In addition, it requires the sharing of a transmission/reception trigger over General Purpose Input/Output (GPIO) connectors that also suffers from phase-altering delays and signal bi-stability caused by clock-domain crossing of the trigger signals.

In order to address these issues, we design, layout, and manufacture a clock-synchronized 4-radio adapter board that
can connect up to 4 WURC radio front-ends to a single WARPv3 baseboard (Fig. 2: $4 \times 1$ WURC Adapter). With this architecture, baseband sampling and RF reference clocks are now buffered and distributed in a tree topology, while inter-radio triggering is no longer needed since all data streams come from the same FPGA and same clock domain. This has the added advantage of reducing the cost and hardware footprint of a 4-radio UHF AP.

**System Configuration.** Our over-the-air experiments utilize omni-directional 3 dBi August DTA240 portable UHF antennas for the AP, mobile STAs, and indoor STAs while the static outdoor STAs use Comtelco Y42400WB 7 dBi log-periodic antennas. During experiments, the AP antennas are configured in a linear array separated by a minimum of $\lambda/2$ distance to ensure sufficient spatial diversity.

**B. Measurement System: Rapid Implicit Sounding**

Channel sounding of mobile devices has always presented a challenge for SDR systems, which require extensive computational resources, power sources, and synchronization to operate as a stand-alone mobile device. For that reason, previous multi-user UHF research focused solely on fixed devices [8] and avoided investigations of channels with nodal mobility.

In order to sound the multi-user environment rapidly between a MU-MIMO AP and multiple mobile STAs, we port the recently published Argos MU-MIMO control channel [10] to inter-operate with our new MIMO WURC array by integrating custom HDL and embedded C libraries for the WARPv3 platform. The robust wireless synchronization scheme utilizing long correlatable signal sequences in Argos allows us to operate mobile STAs remotely, using a low-rate wireless side channel (the wireless bridge in Fig. 2) for STA initialization and control messages, while sample-level synchronization and implicit channel sounding occurs over the UHF channel. More details about the design of Argos are available in [10]; we configure the system to allow us to sound the uplink channel of a set of mobile/static STAs every 2.5 to 5 ms.

In order to increase the number of antennas available at the AP, we share reference clocks between two $4 \times 4$ Argos-WURC APs to create a single $8 \times 8$ AP. This significantly shortened clocking topology introduces no measurable decrease in signal or triggering error and has been validated over hours of operation.

**IV. EXPERIMENTAL EVALUATION**

In this section, we use the data obtained from our indoor and outdoor implicit channel measurements to emulate various implicit resounding policies in several environments.

**A. Sounding-Transmission (S-T) Interval**

In the following analysis of various alternative resounding policies for 802.11af systems, we focus our analysis on the effect of the time interval between when a channel is sounded and when the final beamformed transmission takes place. We call this time the “Sounding-Transmission Interval,” or S-T interval. Differences in the sounded CSIT compared to the actual physical channel at the time of zero-forcing transmission result in inter-stream interference between STAs as well as reduction in their desired signal strength. In mobile environments, it is highly likely that a larger S-T interval will yield higher inter-stream interference due to increased CSIT error and therefore lower Signal-to-Interference-and-Noise Ratio (SINR).

The S-T interval is important for understanding the performance of opportunistic implicit sounding since an opportunistic AP may have cached, or “stale” CSIT obtained from previous uplink transmissions made at different times. In order to use this CSIT, it will need to make a decision about future beamformed transmissions utilizing that stale CSIT. On the other hand, an implicit or explicit AP refreshes all CSIT simultaneously at the beginning of a multi-user packet, yielding an S-T interval of nearly zero.

Depending on the length of the S-T interval, an opportunistic system could exhibit high inefficiency due to unnecessary sounding overhead, or poor performance due to stale CSIT. In order to emulate opportunistic collection of CSIT, we need to
characterize how drift in the CSI of a single STA will affect the performance of a future beamformed transmission including multiple other STAs.

### B. Multi-user Achievable Rate with Increasing S-T Interval

In this section, we investigate the downlink zero-forcing throughput degradation as a function of the S-T interval in indoor and outdoor environments with both nodal and environmental mobility.

Our evaluation methodology relies on the assumption of channel reciprocity. We first record a series of uplink channel traces of an 8x4 MU-MIMO system with 4 single-radio STAs using the Argos-WURC system described in §III. This system is used to record multi-user CSI over the course of a minute at regular sampling intervals of 2.5 or 5 ms.

We then assume that the variation in our channel traces is only caused by changes in the physical MIMO channel rather than the radio hardware and use the empirical capacity of the uplink channel in place of the downlink. In [14], the authors demonstrated channel reciprocity using the same transceivers, and in [13] we demonstrated MIMO reciprocity calibration that we have repeated with our hardware from §III-A yet omit here due to space. When accurate reciprocity calibration is performed and interference is identical, the channel capacity in one direction is the same as the other direction.

Each of six different trials was performed either in a: (i) indoor office building environment with non-line-of-sight propagation less than 50 m distance through a wall and a hallway; or (ii) the outdoor heavily forested environment shown in Fig. 4 with non-line-of-sight propagation up to 200 m directly through multiple trees and underbrush. The tested environments were static, with no intentional mobility, environmental motion, with pedestrians walking around the fixed STAs, or mobile, with one (indoor) or two (outdoor) STAs being physically carried by a pedestrian.

**ZFBB Rate Calculation.** Let \( P_{jk} \) represent the signal power of spatial stream \( j \) received at STA \( k \). If we let \( w_{km} \in \mathbf{W} \) be the transmission precoding weight coefficients from AP antenna \( m \) to STA \( k \), and \( h_{mk} \in \mathbf{H} \) be the corresponding instantaneous MIMO channel coefficients at the moment of transmission, we can calculate the empirical transmission SINR at STA \( k \) as the following:

\[
\text{SINR}_k = \frac{P_{kk}}{N_k + \sum_{j,j\neq k} P_{jk}}
\]

\[
= \frac{|\sum_{m=1}^{M} h_{km} w_{mk}|^2}{N_k + \sum_{j,j\neq k} |\sum_{m=1}^{M} h_{km} w_{mk}|^2}
\]

Using the well-known Shannon-Hartley theorem, we calculate the empirical achievable rate of the beamformed channel as \( R_k = \log_2(1 + \text{SINR}_k) \).

We compare the loss of achievable per-user throughput in Figs. 5 and 6 as a function of the S-T interval. The zero-forcing achievable rate is the percent difference between the rate with fresh CSIT and the estimated rate using delayed CSIT.

**Effect of Mobility on Achievable Rate.** We first compare the \( 8 \times 4 \) zero-forcing results for the various STAs in Fig. 5.

When STAs are static, in Fig. 5a and Fig. 5b, we observe that there is minimal loss of beamforming performance as the S-T interval grows. While we would expect that little to no change in CSIT would occur in the largely static environment in Fig. 5a, an unexpected finding is that environmental mobility, even in the non-line-of-sight environment with pedestrians walking within the same hallway, Fig. 5b, had no significant effect on the averaged beamforming rate. Inspecting channel traces, we observe dips in beamforming performance as pedestrians walked by STAs, but such disruptions were small, momentary and had little effect on the average rate, returning to high rate after the pedestrian had passed. Even at 1 second S-T intervals, the system resounds rapidly enough that minimal disruption to the STAs average capacity is observed.

On the other hand, when the STA itself becomes mobile, in Fig. 5c, achievable capacity for the mobile STA dropped quickly after an S-T interval of approximately 20 ms. This still represents a timescale of tens of packets for a mobile STA, indicating that when sufficient uplink traffic is available, an opportunistic sounding AP would provide per-user beamforming performance within 15% of ideal to mobile nodes even with S-T interval on the order of a 20 ms. Even under environmental mobility, 15% of ideal beamforming performance would be achieved with a S-T interval on the order of a second.

**Effect of Environment on Achievable Rate.** We now repeat the same measurements with the same equipment in the outdoor forest environment in Fig. 6. We chose to perform channel sounding experiments in a heavily forested environment since one of the potential applications of 802.11af MU-MIMO networks is to provide last-mile connectivity for
residential networks in locations where line-of-sight channels are not available for 802.11ac equipment, which also has severe problems propagating through trees [18]. Parallel 2.4 GHz MU-MIMO measurements were attempted at this forested location, yet the signal could barely propagate more than 15 m in the environment and the results were abandoned.

Our results for the forested environment are similar to the indoor environment: as the S-T interval increases, the average supported capacity of the outdoor ZFBF system decreases slowly for the static STAs and much more rapidly for the mobile STAs. A noticeable difference is that the S-T interval breakpoint for the outdoor mobile nodes appears at approximately 50 ms while in the indoor tests it appears around 20 ms. This would be consistent with the outdoor environment that, while also non-line-of-sight, has fewer multi-path reflectors and thus exhibits less channel variation as the STAs move.

We find that based on measured beamforming capacity, up to 1 second of S-T interval is allowable to achieve within 15% of ideal per-user beamforming capacity to fixed STAs, or 20 ms of S-T interval to achieve within 20% of ideal beamforming capacity with mobile STAs in an 8 × 4 zero-forcing system.

Sum-Rate Results. We sum the individual results obtained in Figs. 5 and 6 in Fig. 7, to report that the sum-rate rate loss with increasing S-T interval is somewhat eased when considering the sum network throughput. This will be used to simulate opportunistic sounding in §IV-C.

Limits of a fixed S-T interval. We evaluate the effectiveness of using a fixed S-T interval to achieve a particular performance level. Vendors of fixed wireless 802.11 equipment are increasingly replacing the 802.11 DCF MAC with Time-Division Multiple Access (TDMA) alternatives for increased long-range efficiency and QoS [19] and could guarantee that opportunistic CSIT is available with a given S-T interval.

Fig. 8 depicts two seconds of the empirical achievable rate of the indoor 8x4 ZFBF system in order to demonstrate the problem of using fixed resounding intervals. The achievable rate of three STAs are shown in different colors; the solid line is the oracle ZFBF rate and the dotted line is the achievable ZFBF rate assuming a fixed 100 ms re-sounding interval. As expected, the mobile STA 1 in blue, which is carried at pedestrian speed within the hallway, demonstrates rapidly changing CSI that cannot be tracked accurately by this large fixed sounding interval. At each re-sounding point, the periodic system matches the oracle capacity, and then rapidly degrades to approximately 20% of optimal. As the mobile STA 1 physically moves by a static STA 2 (red, 27 seconds), it perturbs its relatively static wireless channel resulting in severe capacity loss.

Such an event is difficult to predict and could result in outages or large capacity loss unless identified and corrected. Based on our observations, a fixed S-T interval would either result in either unnecessary sounding or excessive capacity loss due to stale CSIT since channels can change mobility state rapidly.

Thus, we find that an opportunistic sounding policy should have an adaptive component that adjusts the maximum tolerable S-T interval based on current channel conditions and the mobility state of the STA.

C. Performance of Proposed 802.11af Sounding Alternatives

In this section, we investigate the protocol gains available from an opportunistic CSIT system with regard to various MAC-layer parameters. We also compare performance against an implicit sounding policy adapted from 802.11n standard proposals for multi-user operation [15].

The 802.11af standard attempts to amortize explicit sounding overhead by transmitting aggregated data frames, however the efficiency of this approach depends on the number of frames actually available to aggregate. We analyze the protocol performance of a MU-MIMO system with various channel sounding policies and with varying packet aggregation values in order to emulate both best and worst case scenarios.

We set the single frame size to 1500 bytes, the largest regular Ethernet frame size and the best case for CSIT overhead amortization before aggregation. We compare three different channel sounding policies:

Explicit 802.11af. This is the current standard operation of 802.11af MU-MIMO. CSIT overhead in this case is caused by the NDP Announcement, the sounding NDP, and the sequence of polls and CBFR responses from all 802.11af STAs before each downlink transmission [5]. The upper and lower bounds on explicit performance are calculated with minimum and maximum feedback compression of the CBFR payload, a highly vendor-specific implementation parameter. We assume no impairment on performance from feedback compression,
and plot the median performance while indicating the bounds with a shaded red region.

Although the 802.11af standard only supports up to 8 concurrent spatial streams, we assume that timing and protocol performance scales with the number of streams in order to provide a point of reference for scaling to large numbers of antennas. We label this policy “Explicit 802.11af” in the following plots.

**Implicit Proposal for 802.11af.** In [15], the authors proposed an alternative multi-user CSI sounding protocol that avoids the lengthy CBFR by estimating the channel implicitly with short NDPS. CSIT overhead in this case comes from the NDP Announcement and a staggered sequence of uplink NDPS that are used for implicit channel estimation before each multi-user transmission as proposed in [15]. Since the channel is estimated implicitly, there are no levels of feedback compression to display. We label this policy “Implicit” in the following plots.

**Opportunistic Proposal for 802.11af.** In this case, there is no CSIT overhead to multi-user transmissions. We explore three regions of operation for an opportunistic AP:

1) **“Opportunistic.”** The best-case performance assuming all CSIT is available opportunistically and there is no beamforming penalty for using stale CSIT.

2) **“Opportunistic with Bootstrap.”** An alternative fallback mode where at most one STA has stale CSIT and the AP sends a single packet to that STA before each multi-user transmission in order to implicitly refresh its CSIT. This can be viewed as a way of quickly bootstrapping opportunistic CSIT to a STA that previously was inactive.

3) **“Opportunistic with Stale CSIT.”** A trace-driven lower bound on opportunistic performance based on our environmental measurement traces. We assume that CSIT is refreshed opportunistically every second. According to our empirical results in Fig. 7, this would result in less than 10% reduction in achievable sum-rate in an environment with static STAs. Thus, we reduce the throughput of the best-case opportunistic scenario by the requisite amount, presenting a more fair approximation of how an implemented opportunistic system might perform.

All ACKs are staggered as per the 802.11af specification. For tractability, transmissions are assumed to be successful, requiring no retransmissions, and only downlink data flows are considered.

1) **Sounding Policy Performance: 4x4:** In Fig. 9, we vary the multi-user frame aggregation number from 1 to 64 for the lowest (top) and highest (bottom) 802.11af MCS in a 4 x 4 system where all STAs have only a single antenna.

**Effect of Frame Aggregation.** Frame aggregation allows the cost of channel sounding to be amortized over large payloads. While we expect that increased aggregation will generally decrease the efficiency of channel sounding reduction protocols, it also determines crossover points in terms of protocol performance.

At the lowest MCS in the top plot of Fig. 9 with no frame aggregation, there is a moderate performance gap between implicit channel sounding methods (opportunistic, implicit) and the current explicit 802.11af policy. An opportunistic sounding policy would increase throughput at best by 31%, while an implicit sounding policy would increase throughput by 21% over explicit 802.11af. However, as the aggregation rate increases, these alternatives rapidly converge.

**Effect of MCS.** In essence, the lower the MCS rate, the lower the relative overhead of sounding; thus the sounding mechanism matters much less at low rates than high rates. At base rate, shown in Fig. 9 (top), there is little advantage in opportunistic sounding, and our proposed bootstrapping method under-performs even explicit sounding. However, as the MCS of the system increases, the relative cost of sounding overhead also increases since airtime becomes more valuable, potentially compensating for the stale CSIT penalty.

In Fig. 9 (bottom), we show the same results for the maximum supported MCS. 802.11af sounding overhead is much more costly when the system could otherwise be operating at high MCS, since CBFRs, polling packets, and ACKs are all sent at base rate for robustness. The large range in explicit 802.11af sounding performance (red region) stems from the fact that uncompressed CBFR packets take a significant amount of airtime, resulting in very high overhead. At high MCS, opportunistic sounding can improve throughput by 186% and implicit sounding can improve by 94% without frame aggregation.

While opportunistic sounding with stale CSIT is strictly better than explicit 802.11af up to 35 aggregated frames, it
barely out-performs implicit sounding at low aggregation with fewer than 10 frames and then performs significantly worse with higher frame aggregation.

Therefore, we conclude that for a low number of spatial streams, opportunistic channel sounding has approximately equivalent performance compared to implicit channel sounding and potentially worse performance when considering beamforming error from stale CSIT. However, both opportunistic and implicit channel sounding offer significant throughput gains over the current explicit 802.11af standard.

The best usage scenario for opportunistic sounding in this regime is when implicit STA cooperation is not possible, such as with current 802.11 devices. A system design that leverages this observation would utilize opportunistic CSIT when per-user downlink traffic queues are below 3-52 MB, depending on the current MCS, and then revert to explicit sounding when queues exceed that size and sounding overhead can be sufficiently amortized. For legacy 802.11a/b/g/n devices that do not report any CSIT, only opportunistic CSIT would be available and the decision is made between multi-user and single-user transmission modes only.

2) Scaling to 32x16: At all MCS, the challenge of efficiently using narrow bands of UHF radio spectrum is clear; system throughput is no more than 50 Mbps even with full 4x4 spatial diversity at the maximum MCS (Fig. 9). For this reason, we explore the possibility of leveraging additional spatial streams for UHF-band communications as a means of increasing spectral efficiency.

Given the potential for large-scale 802.11af system installations to establish long-range point-to-multi-point networks and the need to support high throughput over narrow UHF channels, we extend our beamforming protocol analysis to a 32 x 16 system in Fig. 10. Previous work on many-antenna MU-MIMO systems has proposed implicit channel sounding as a means to avoid protocol collapse as the number of antennas at the AP grows [13]. Our results in § IV-B indicate that the CSI of stationary STAs in both indoor and outdoor environments remain constant for long periods of time, which supports the possibility of using opportunistic sounding policies to increase system throughput even further.

In all cases with a large number of spatial streams, explicit channel sounding suffers severely from protocol congestion due to the high number of spatial streams and amount of explicit data that is transmitted to the AP to report CSI.

In Fig. 10 (top), we see that for low MCS rates and frame aggregation below 18 frames, opportunistic sounding with stale CSIT out-performs even implicit sounding, given the number of STAs involved in each transmission.

In Fig. 10 (bottom) at the maximum supported MCS, strict relationships emerge between the sounding policies, since CSIT overhead dominates any other effects at this scale. When channel sounding becomes extremely expensive, the use of opportunistic CSIT is able to offer significant throughput gains over implicit sounding, ranging from 112% with no frame aggregation, to 18% at maximum aggregation, even when considering the penalty from stale CSIT. Explicit sounding should be avoided altogether.

V. RELATED WORK

Managing protocol overhead is crucial for achieving multi-plexing gains with MU-MIMO transmissions and thus numerous works propose techniques that seek to lessen the impact of obtaining CSIT.

Overhead amortization. One category of work seeks to reduce CSI overhead through the use of techniques such as frame aggregation that amortize overhead across multiple data frames. For example, the authors of [20] develop custom frame aggregation techniques for amortizing explicit overhead in 802.11ac systems. Additionally, works such as [21] concede that the overhead of CSIT acquisition is so detrimental, that they suggest avoiding MU-MIMO transmissions altogether.

Our protocol simulations considering both aggregation and compression in Fig. 9 and 10 demonstrate that implicit sounding offers significant benefit while opportunistic sounding can improve even further in certain cases.

Channel sounding suppression. The second category of techniques seek to reduce CSI overhead by avoiding channel sounding when possible. For example, MUTE [7] reduces explicit sounding overhead by opportunistically sounding users when the wireless channel is free and by tracking channel variation to avoid sounding the channel unnecessarily. In our work, we focus on more stable 802.11af channels and higher-order MIMO systems where it becomes feasible to avoid sounding altogether and maintain the same throughput performance as with full explicit channel sounding, while scaling well. Our approach further enables ZFBF to STAs without 802.11ac/af CSI reporting enhancements.
AFC [6] proposes a protocol that allows STAs to determine their own downlink CSI variation through the use of a “Compression Noise” (CNo) metric which tracks the difference in CSI measurements over time and only requests sounding when needed. We propose and analyze an alternative opportunistic approach that avoids explicit sounding altogether.

**Implicit channel sounding.** Precoding schemes rely on CSIT provided by the radio physical layer. Previous work shows that under many conditions, the additional explicit protocol overhead [22], [7] and CSI feedback compression error [15], [6] in explicit channel sounding can severely degrade the performance of the 802.11ac MU-MIMO protocol. A case where implicit CSI is not only necessary but necessary is when the number of antenna on a given wireless device grows large, such as in “massive” or many-antenna MIMO, where explicit sounding cannot scale efficiently [13].

While not the first to propose implicit channel sounding, we are the first to measure the beamforming and protocol cost associated with various channel sounding techniques and to propose a completely sounding-free approach for fixed wireless systems with long coherence time.

**CSIT Prediction.** Other work has focused on attempting to predict CSIT from historical measurements for adaptive modulation systems [23]. When deciding when to use opportunistic CSIT or when to fallback to other sounding modes, knowledge of the expected cost of stale CSIT is crucial. Our results in Fig. 6 demonstrate that beamforming degradation can vary from STA to STA based on their mobility state, therefore future work might focus on extending CSIT prediction algorithms for beamforming and sounding mode selection.

### VI. Conclusion

In order to scale the capacity of MU-MIMO beamforming systems, it is important to address the problem of CSIT overhead, particularly in 802.11af systems with potentially limited system bandwidth.

In this work, we developed a new SDR system specifically for UHF-band MU-MIMO that allowed us to gather the first mobile multi-user channel traces in the UHF band, which can be found in [24]. Based on our analysis of beamforming capacity with stale CSIT, we showed large S-T intervals can be tolerated in UHF frequency bands, enabling the gathering of CSIT purely opportunistically and enabling multi-user transmissions with legacy 802.11 equipment that can not provide CBFR reports.

We compared three different channel sounding policies and showed that for a small number of spatial streams, significant throughput gains are available with either of the implicit sounding policies, though the penalty of using stale CSIT would encourage the use of implicit sounding rather than opportunistic sounding, if available. However, as the number of spatial streams increases, the overhead of even implicit beamforming begins to become a bottleneck on 802.11af performance and opportunistic channel sounding becomes much more beneficial.

**Acknowledgements**

This research was supported by Cisco and by NSF grants CNS-1514285, CNS-1444056 and CNS-1126478.

**References**


DiVote: A Distributed Voting Protocol for Mobile Device-to-Device Communication

Peter Danielis, Sylvia T. Kouyoumdjieva, and Gunnar Karlsson
ACCESS Linnaeus Center, School of Electrical Engineering
KTH Royal Institute of Technology, Stockholm, Sweden
Email: {pdanieli, stkou, gk}@kth.se

Abstract—Distributed aggregation algorithms have traditionally been applied to environments with no or rather low rates of node churn. The proliferation of mobile devices in recent years introduces high mobility and node churn to these environments, thus imposing a new dimension on the problem of distributed aggregation in terms of scalability and convergence speed. To address this, we present DiVote, a distributed voting protocol for mobile device-to-device communication. We investigate a particular use case, in which pedestrians equipped with mobile phones roam around in an urban area and participate in a distributed yes/no poll, which has both spatial and temporal relevance to the community. Each node casts a vote and collects votes from other participants in the system whenever in communication range; votes are immediately integrated into a local estimate. The objective of DiVote is to produce a precise mapping of the local estimate to the anticipated global voting result while preserving node privacy. Since mobile devices may have limited resources allocated for mobile sensing activities, DiVote utilizes D-GAP compression. We evaluate the proposed protocol via extensive trace-driven simulations of realistic pedestrian behavior, and demonstrate that it scales well with the number of nodes in the system. Furthermore, in densely populated areas the local estimate of participants does not deviate by more than 3% from the global result. Finally, in certain scenarios the achievable compression rate of DiVote is at least 19% for realistic vote distributions.

I. INTRODUCTION

Distributed tasks and computations, e.g., to estimate the average or sum of a set of values, are often conducted based on inputs supplied by collaborative users. Such aggregation functions are of high importance in large-scale distributed systems where there is a need to compute global system properties [1].

In this paper, we focus on a specific class of distributed tasks, namely distributed voting in the context of urban polling. Potential urban polling applications collect and process information on locally-relevant questions and provide users in a community with answers to them [2]. Such questions can relate to urban planning optimization ("Is the switching behavior of this traffic light fast enough?") or to safety in a given region ("Do you feel safe in this area?").

In general, the information obtained during a poll can be processed either in a centralized or in a decentralized manner. Centralized processing requires nodes to submit their votes to a central entity. However, this approach lacks scalability and poses privacy concerns as users might in general not want their votes to be seen by a central entity [3]. In particular, this may be of high importance in countries where people do not trust authorities. Contrary, decentralized (or distributed) processing requires nodes to compute local estimates of the result based on partial system knowledge. As opposed to conventional distributed processing scenarios where nodes are considered to be static or semi-static [4], in this work we examine scenarios, in which nodes exhibit high mobility. We consider a node to be a pedestrian carrying some device equipped with a wireless communication interface such as a mobile phone. We rely on device-to-device communication for disseminating votes among participants in the poll. Mobile nodes opportunistically exchange data whenever they come in direct communication range [5]. For the purpose of urban polling, this data comprises voting information conveyed by means of broadcast messages and nodes immediately update their local estimate upon reception of a message.

In the context of distributed voting in urban environments, a distributed voting protocol needs to comply with the following requirements: (1) be scalable, (2) have fast convergence and high accuracy, and (3) preserve node privacy. Thus, in this work we present DiVote, a distributed voting protocol for mobile device-to-device communication, which provides all of the above characteristics. The main contributions are:

- We show that DiVote is suitable for operation in dynamic environments with high node mobility. DiVote makes use of the benefits of D-GAP compression for scalability of the protocol [6]. Furthermore, DiVote preserves node privacy by applying a cryptographic hash function to user identities.
- We perform extensive trace-driven simulations using realistic pedestrian mobility. We show that DiVote scales well with the number of nodes in the system. Furthermore, DiVote demonstrates both fast convergence and high accuracy, with local estimates deviating at most by 3 % from the global value in dense scenarios.
- DiVote is able to achieve at least 19% compression rate for realistic vote distributions, which makes it appropriate for execution on mobile devices with limited storage capabilities or with restrictions on the memory to be used.
- DiVote exhibits low processing load at the application layer as it requires only a fraction of the received broadcast messages (34 % in dense scenarios and even less in sparser scenarios) to be processed to achieve accurate...
local estimates.

The remainder of this paper is organized as follows: In Section II, we provide an overview of previous work in the field of distributed aggregation, and reason why current solutions are not suitable for operation in mobile environments. Section III introduces DiVote, a distributed voting protocol for mobile device-to-device communication. Section IV outlines the evaluation scenarios, and Section V presents results from realistic pedestrian mobility scenarios. Finally, we conclude the study in Section VI, and present directions for future work.

\section*{II. Related Work}

Distributed voting belongs to the class of distributed aggregation problems. Distributed aggregation in general comprises computations such as sum, average, minimum, or maximum over unreliable networks, in which no central entity is accessible or required. There are two main paradigms to address this problem, namely gossip-based and tree-based aggregation. Tree-based aggregation protocols have been shown to perform poorly in dynamic environments with high levels of churn [4], therefore for the rest of this section we focus on discussing the applicability of state-of-the-art gossip-based protocols to our scenario.

Gossip-based aggregation protocols that react to environment changes can be broadly classified in restarted and bookkeeping protocols.

We first discuss restarted protocols, many of which are based on the the push-sum algorithm presented in [7]. The basic idea of the algorithm is that nodes periodically exchange stored values with their neighbors and are thus able to compute the sum or average of all values. However, the main assumption in [7] is that values stay unchanged over time. In [8], the authors propose executing the push-sum algorithm in epochs to reflect changes in the network; the protocol is restarted after each epoch. The distributed random grouping algorithm (DRG) is proposed in [9]. In this algorithm, some nodes can periodically become group leaders and then determine group members by exchanging messages in a handshake manner. The establishment of several roles as well as the information exchange by means of a handshake makes this algorithm too slow to react on changes induced by moving nodes in our envisaged scenario. Another gossip-based distribution estimation approach is suggested in [10]. This algorithm exchanges and merges lists consisting of pairs with value and respective counter between nodes. However, duplicates may occur when applying this approach, which distorts the computed estimate. Most of the restarted gossip algorithms show this shortcoming and do therefore not achieve a high accuracy. In [11], the authors tackle the data duplication problem by simultaneously executing multiple instances of the proposed protocol however the solution exhibits low accuracy [12].

Bookkeeping gossip-based protocols are able to revert changes in the nodes’ states. In [13], a node saves the states on its neighbors and recovery is triggered when a node crashes or disappears. Here, a tradeoff between accuracy and protocol overhead has to be chosen so either scalability in terms of memory consumption or accuracy are decreased. In [14], the authors propose LiMoSense, an algorithm for live monitoring in dynamic sensor networks, which takes into account node churn and link failures at runtime. LiMoSense is not appropriate for dynamically changing environments, since it assumes a known set of neighbors, from which it randomly chooses a single neighbor at a time for message exchange. In [15], the authors present Flow Updating, a bookkeeping algorithm, which iteratively averages values towards the global network mean. Every node computes a flow value for each of its neighbors and stores the value in a matrix. The algorithm tries to enforce skew symmetry of this matrix. This however significantly decreases the convergence speed. To summarize, bookkeeping protocols usually require up to thousands of rounds to converge, thus they are not applicable in scenarios with high dynamics.

In Section I, we outlined the main characteristics of a protocol suitable for distributed aggregation in dynamic environments. In Table I, we compare tree-based, restarted, and bookkeeping gossip-based protocols with respect to these characteristics. We also show how our protocol DiVote compares to others in the literature. In the next section, we present DiVote in detail.

\section*{III. DiVote: A Distributed Voting Protocol}

In this section, we present DiVote, a distributed voting protocol for mobile device-to-device communication. We begin by introducing some of the building blocks of the protocol, i.e., the vector compression scheme and the fundamental vector operations that are introduced by the protocol. We then present the details of the algorithm behind DiVote.

\subsection*{A. The need for D-GAP compression}

In the context of distributed voting, each node can cast a binary vote (0 or 1) to a poll. (We note that the assumption of binary votes does not limit the reasoning to follow, and can easily be extended to any number of votes. In this paper, we only consider binary votes for the sake of brevity.) For nodes to be able to calculate the anticipated global vote, they need to keep track of the votes of other peers in their vicinity throughout their lifetime. However, keeping track of a simple moving average may result in votes being counted multiple times if a node and a peer come in communication range more
than once. Furthermore, keeping track of votes in the form of a binary vector may be consuming a lot of resources, especially if the vector contains long sequences of 0s or 1s. To address these problems, in DiVote we leverage the concept of D-GAP compression [6]. D-GAP compression provides a compressed representation of bit vectors in the form of integer vectors (later referred to as D-GAP vectors) and can be treated as a specialized variant of run length encoding. Each integer in a D-GAP vector represents the number of consecutive 0s or 1s that are present in the bit vector at a given position. Whether the integer corresponds to a sequence of 0s or 1s is determined by the leading bit of the D-GAP vector. A leading bit of 0 shows that the first integer corresponds to a number of consecutive 0s, followed by a number of consecutive 1s and so on; a leading bit of 1 indicates the opposite behavior.

An example of converting a 12-bit vector into a 9-bit D-GAP vector is illustrated in Figure 1. We calculate the total number of bits required for representing the D-GAP vector, $N(DGAP)$, as follows:

$$N(DGAP) = \sum_{i=1}^{n} (\lfloor \log_2 d_i \rfloor + 1) \tag{1}$$

where $d_i$ is the integer representation at position $i$ of the D-GAP vector, and $n$ is the size of the vector.

For decoding purposes, each D-GAP vector needs to have a corresponding D-GAP mask vector. In essence, a D-GAP mask vector is a bit vector of consecutive sequences of 0s and 1s, and each sequence indicates the boundaries of an integer in the D-GAP vector. Let us assume the following D-GAP vector: $\{0\} 2 \{1\} 2 \{0\} 1 \{0\} 0 \{1\}$ as an illustration of the problem. For the vector to be stored in memory it will be converted to $\{0\} 10111$. However, this representation alone is not enough for decoding, i.e., it is impossible to tell whether the original D-GAP vector was $\{0\} 2 \{1\} \text{ or } \{0\} 5$. To decode the D-GAP vector, we need to apply a mask vector $\{0\} 001$. The mask vector shows that the initial two bits correspond to the first integer while the third bit corresponds to the second integer. For longer D-GAP vectors, the D-GAP vector mask will iterate between sequences of consecutive 0s and sequences of consecutive 1s.

### B. Operations on D-GAP vectors

A D-GAP vector is solely a data structure. Hence, we define the following three operations for DiVote that can be performed on two D-GAP vectors of arbitrary lengths: merge, consolidate, and append. For brevity, let us consider vectors of different lengths, DGAP$_{min}$ and DGAP$_{max}$, denoting the shorter and the longer vector, respectively. Note that here vector length corresponds to the expanded bit vector length and is defined as $N(BVEC) = L(DGAP) = \sum_{i=1}^{n} d_i$.

- The **merge** operation combines DGAP$_{min}$ and DGAP$_{max}$ into a resulting vector DGAP$_{res}$. The merge operation is performed in an iterative manner until it reaches the end of DGAP$_{max}$.
- The **consolidate** operation combines the last position of DGAP$_{res}$ with the next position of DGAP$_{max}$ after the merge operation is performed. The consolidate operation is only performed if the integers at these two positions correspond to the same bit value.
- The **append** operation simply adds the remainder of DGAP$_{max}$ to DGAP$_{res}$.

Observe that if the two input vectors are of equal lengths, the only operation that will be performed is merge.

Figure 2 illustrates an example of all three operations that can be performed with D-GAP vectors.

### C. Functional principles of DiVote

With DiVote, each node locally keeps track of nodes it has obtained knowledge of, either directly or via other peers, as well as of their votes. This information is presented in the form of two correlated D-GAP vectors. Whenever a node first casts a vote, it adds itself to the shared-nodes vector, and it adds its vote to the votes vector. Observe that due to privacy preservation reasons, the position, in which information is stored in each of the D-GAP vectors, is determined by a cryptographic hash function such as MD5, which is calculated over a unique identifier, e.g., the node’s MAC address. For instance, if for a node, which casts a vote for 0, the cryptographic hash function returns a position of 25, both its shared-nodes vector and its votes vector would be initialized with $\{0\} 24 \{1\}$. If the same node were to cast a vote for 0, the votes vector would instead be initialized with $\{0\} 25$. If a node is compromised, it could potentially associate the vote to the respective node during such initialization. However, in this work we assume all nodes to be trustworthy.

Note that hash functions can compute the same hash value for different identifiers, i.e., collisions can occur. This would

---

Fig. 1. An example of D-GAP compression. A bit vector of 12 bits is converted into an integer D-GAP vector of 9 bits. The leading bit in the D-GAP vector indicates if the vector starts with 0s or 1s.

Fig. 2. Merging, consolidating, and appending D-GAP vectors. Bold integers denote positions under consideration, crossed integers are not considered, and the resulting vector is highlighted in red.
lead to positions that some nodes would share, i.e., some nodes would replace their information when shared alternately, which could falsify their local estimate. However, the collision probability of MD5, which computes 128-bits hash values, is as low as 2.7·10⁻¹⁷ when, e.g., calculating hash values from 2³² values (assuming the birthday paradox [16]) and is therefore neglected.

Each node periodically broadcasts a beacon containing its shared-nodes vector and its votes vector to peers in its vicinity. An example of a DiVote beacon message is presented in Listing 1. Whenever a node receives information from another peer, it immediately updates both its shared-nodes vector and its votes vector following the DiVote protocol outlined in Algorithm 1. We note that the procedure in Algorithm 1 is performed simultaneously with respect to both vectors. However, here we only show how new votes are incorporated into the votes vector for the sake of brevity. Whenever a node receives a beacon from another peer in proximity, it first extracts the received information and checks in a local database whether the received vector from peer i, DGAPᵢ, has changed (line 4). This check is currently done by looking up the node-id in the local database first. If the node-id is found, there has been prior communication with this peer and the advertised updated field value is compared to the previously registered updated field value. If they do not differ, the local vector DGAPₖₑₙ stays unchanged. Otherwise, DiVote consecutively executes the operations merge (lines 7-11), consolidate (lines 12-16) and append (line 17) in order to update the local estimate (line 18). Thus, the shared-nodes vector and the votes vector contain cumulative information of all nodes that have been shared over time, and their corresponding votes, even if these nodes have left the system. Furthermore, DiVote allows nodes to disclose peers that they have not encountered physically by propagating the knowledge accumulated by other participants in the system. This allows DiVote to achieve fast convergence and high accuracy in a distributed manner in scenarios with high levels of mobility, such as in urban environments. Finally, correlating votes and nodes is impossible by third party entities such as central collection points or compromised participants in the poll.

Algorithm 1 The DiVote Protocol

1: DGAPᵢVEC ← received DGAP vector from node i
2: DGAPᵢVEC ← local DGAP vector
3: DGAPᵢVEC ← resulting DGAP vector
4: if DGAPᵢVEC changed since last beacon from node i then
5: DGAPᵢmin = min(DGAPᵢVEC, DGAPᵢVEC)
6: DGAPᵢmax = max(DGAPᵢVEC, DGAPᵢVEC)
7: while ! DGAPᵢmin.end() do
8: DGAPᵢVEC = MERGE(DGAPᵢVEC, DGAPᵢVEC)
9: rpos ← end position in DGAPᵢVEC
10: mpos ← current position in DGAPᵢVEC
11: end while
12: stateᵢ = (rpos mod 2) xor DGAPᵢVEC[0]
13: stateᵢ = (mpos mod 2) xor DGAPᵢVEC[0]
14: if stateᵢ = stateᵢ then
15: DGAPᵢVEC = CONSOLIDATE(DGAPᵢVEC[rpos], DGAPᵢVEC[mpos])
16: end if
17: DGAPᵢVEC = APPEND(DGAPᵢVEC, DGAPᵢVEC[mpos : end])
18: DGAPᵢVEC ← DGAPᵢVEC
19: end if

IV. EVALUATION SCENARIO

In this section, we introduce the mobility scenario as well as the simulation setup and investigated performance metrics.

A. Mobility scenario

In order to realistically recreate pedestrian mobility, we use the Walkers traces [17] captured in Legion Studio [18], a commercial simulator initially developed for designing and dimensioning large-scale spaces via simulation of pedestrian behaviors. Its multi-agent pedestrian model is based on advanced analytical and empirical models which have been calibrated by measurement studies. Each simulation run results in a trace file, containing a snapshot of the positions of all nodes in the system every 0.6 s.

Fig. 3(a) and 3(b) present the scenarios considered in our evaluation: an outdoor urban scenario, modeling the Östermalm area of central Stockholm, and an indoor scenario, recreating a two-level subway station. We note that it is not possible to capture all states of human mobility with a single setup, however the scenarios are representative of typical daytime pedestrian mobility.

The Östermalm scenario consists of a grid of interconnected streets. Fourteen passages connect the observed area to the outside world. The active area, i.e., the total surface of the streets, is 5872 m². The nodes are constantly moving, hence the scenario can be characterized as a high mobility scenario.

The Subway station has train platforms connected via escalators to the entry-level. Nodes arrive on foot from any of five entries, or when a train arrives at the platform. The train arrivals create burstiness in the node arrivals and departures. Nodes congregate while waiting for a train at one of the platforms, or while taking a break in the store or the coffee shop at the entry level. The active area is 1921 m².

If not stated otherwise, the input parameters of the Östermalm and the Subway scenario result in approximately
the same mean node density of 0.1 nodes/m². (More information can be found in [19].)

B. Simulation setup

In our evaluation scenarios, we assume that all nodes carry devices and all are participating in the distributed poll in the area. Each node casts a binary vote \( v = \{0, 1\} \) upon entry in the simulation, and votes are distributed according to a distribution \( f(x) \) with a mean \( E(x) \).

For the evaluation, we use an implementation of an opportunistic content distribution system in the OMNeT++ simulator [20]. Each simulation run is executed in synchronous rounds of 0.6 s which corresponds to the granularity of the mobility traces we use. Nodes broadcast their shared-nodes vector and their votes vector at the beginning of each round. To avoid collisions on the wireless medium, the broadcast transmission of each node in each round is distributed uniformly at random \( \mathcal{U}(0, 0.5) \) s. The transmission range is set to 10 m.

C. Performance metrics

We focus on evaluating the following performance metrics.

- **Deviation** \( \Delta \): The deviation is a measure of the accuracy of the DiVote protocol, i.e., it shows how close the local estimate of a node is to the anticipated global result. The deviation is calculated as:

\[
\Delta = \left| \bar{x} - x \right|
\]

where \( \bar{x} = E(DGAP_{loc}) \) is the local estimate, and \( x \) is the anticipated global result depending on the nodes currently in the system.

- **Compression ratio** (CR): The compression ratio is a measure of the efficiency and scalability of the DiVote protocol in terms of resource management, i.e., how much storage space does the protocol require for performing distributed voting computations in a mobile environment. The compression ratio is calculated as:

\[
CR = 1 - \frac{N(DGAP)}{N(BVEC)}
\]

where \( N(DGAP) \) is calculated as per Eq. 1, and \( N(BVEC) \) is the number of bits required if the data were represented in the form of a bit vector.

- **Information overhead** (IO): The information overhead is a measure of the processing load reduction for a node in the system and therefore indicates scalability as well. It shows how many of the received broadcasts do not need to be processed. The information overhead is calculated at the application layer as:

\[
IO = 1 - \frac{n(BRC)}{N(BRC)}
\]

where \( N(BRC) \) is the total number of broadcast messages received by a node throughout its lifetime in the system, and \( n(BRC) \subseteq N(BRC) \) is the number of broadcast messages that were used for updating the local estimate of the node.

V. Simulation Results

In this section, we investigate simulation results for:

- different arrival rates \( \lambda \) while fixing the scenario and the distribution of nodes voting for one;
- two different scenarios while fixing the arrival rate \( \lambda \) and the distribution of nodes voting for one;
- two different distributions of nodes voting for one while fixing the arrival rate \( \lambda \).

A. Effect of arrival rate

First, we present results for the Östermalm scenario for the arrival rates \( \lambda = \{0.0025, 0.005, 0.01, 0.07, 0.15, 0.30\} \) nodes/s. We assume that votes are deterministically distributed, with a mean \( E(x) = 0.75 \), i.e., 75 % of all nodes vote for one and 25 % vote for zero. (We release this assumption in Section V-C.) In this case, the first node entering the system votes for zero whereas the following three nodes vote for one. After that, this distribution continues for all further nodes. As we will see in Section V-B, this represents the worst case of achievable compression rate.

Figures 4(a)-(c) show the local estimates of all nodes over time for sparsely populated scenarios, i.e., \( \lambda = \{0.0025, 0.005, 0.01\} \) nodes/s. We see that the convergence of the local estimates towards the global result is strongly dependent on the population density. For low values of \( \lambda \), Figure 4(a), nodes are not able to estimate correctly the expected global result. As the arrival rate increases, Figure 4(b), a clearer trend can be seen towards convergence, and at \( \lambda = 0.01 \) nodes/s nodes are able to locally estimate the global result. Still, for any of the low arrival rates, some nodes do not gain sufficient knowledge about other nodes or even do not disclose anyone so that their local estimates remain 0 or 1 (see Figures 4(a)-(c)). As the arrival rate further increases, nodes converge earlier to the global result, and outliers disappear. Thus, we omit results for \( \lambda = \{0.07, 0.15, 0.3\} \) nodes/s for the sake of brevity.

Figures 5(a)-(c) illustrate the proportion of shared nodes over time. In sparser scenarios, Figure 5(a), the proportion of shared nodes is kept below 20 %. Furthermore, information on shared nodes is continuously lost when nodes leave the system. Therefore, we are not able to observe clear convergence of the local estimates towards the global results, Figure 4(a).
With increase of the arrival rate, $\lambda = 0.005$ nodes/s, up to 60\% of all nodes in the system are shared leading to a clearer convergence towards the global result, Figure 5(b). At around $t = 7000$ s when approximately 30\% of all nodes have already been shared, the trend towards the mean $E(x) = 0.75$ becomes apparent. We note that approximately 60\% of nodes are already shared around $t = 4000$ s by some of the nodes in the system. A further investigation shows that indeed the knowledge is kept only in two nodes which have shared around 130 nodes each. However, both of these nodes leave the system in $t \in (4600, 4700)$ s so their accumulated knowledge is lost and the process is restarted. This is reflected in Figure 4(b) as well as the trend towards the global result does not become clear before $t = 7000$ s when the vast majority of nodes has shared at least 30\% of all nodes. Finally, Figure 5(c) shows an even clearer convergence trend; already at $t = 1000$ s when approximately 30\% of all nodes have been shared most nodes approach the mean $E(x) = 0.75$. Thus, we conclude that even in dynamically changing environments there is a correlation between the percentage of shared nodes and the convergence to the global result.

The observation that disclosing approximately 30\% of nodes is sufficient for achieving precise estimate of the global result is further confirmed in Figures 6(a)-(c), which show the change in deviation $\Delta$ with respect to the proportion of shared nodes for the denser Östermalm scenarios with $\lambda = \{0.07, 0.15, 0.3\}$ nodes/s. As the arrival rate increases, the deviation $\Delta$ significantly decreases once 30\% of the nodes have been shared, from 15\% for $\lambda = 0.07$ nodes/s, Figure 6(a), to below 6\% for $\lambda = 0.3$ nodes/s, Figure 6(c).

We further evaluate the performance of the system in steady state, i.e., once the average number of nodes in the area stays unchanged despite the arrivals and departures in the system. We then aggregate results from a 1000 nodes. We subsequently exclude the sparse scenario with $\lambda = 0.0025$ nodes/s from consideration as the trace does not comprise 1000 nodes after the steady state has been reached, and local estimates do not converge to the global result.

Table II shows the average and maximum deviation $\Delta$ including 95\% confidence intervals as well as the information overhead in the steady state depending on the arrival rate $\lambda$. Note that minimum deviation values are omitted as they are zero in all cases. These results clearly show that the denser the scenario, the smaller the deviation $\Delta$, i.e., the accuracy increases. Note that for $\lambda = 0.005$ nodes/s and $\lambda = 0.01$ nodes/s some nodes are still completely wrong regarding their local estimate when they leave the system. As the arrival rate increases, both the average and the maximum deviation are
steadily decreasing. Finally, for the most densely populated scenario, $\lambda = 0.3$ nodes/s, the maximum deviation never exceeds 3%. Moreover, the information overhead is between 93% and 94% for $\lambda = \{0.005, 0.01, 0.07\}$ nodes/s, which results from the fact that often the same nodes meet again and do not exchange new information. On the one hand, this underlines the low processing load on application layer and thus DiVote’s scalability in sparse scenarios. On the other hand, depending on the voting distribution and its mean value, sequences of consecutive 1s may be shorter in the D-GAP vector for storing votes resulting in a lower compression ratio. In Figure 7(a), the curves for $\lambda = 0.005$ nodes/s and $\lambda = 0.01$ nodes/s show an erratic trend as the scenarios are too sparsely populated to show clear convergence. With the increase of the arrival rate, however, the compression ratio also increases, and for $\lambda = 0.3$ nodes/s, the average compression ratio amounts to 92%. This results from the fact that most nodes in the system have been shared, thus almost all bits in the D-GAP are set to 1. The compression ratio of the D-GAP for storing votes approaches 25% as the arrival rate increases, Figure 7(b). This behavior is strongly dependent on the chosen voting distribution as well as on its mean value $E(x) = 0.75$. As mentioned earlier, the first node entering the system always votes for zero, while the following three nodes vote for one, and subsequently the distribution applies to all further nodes. Hence, the D-GAP for storing votes converges to sequences of three consecutive 1s, which are interrupted by a 0. Applying Equation 1, three consecutive 1s and one 0 require 3 bits for storage whereas a plain bit vector would require 4 bits. Therefore, the achievable compression ratio converges to a value of 0.25 while more and more nodes are shared, which represents the worst case in terms of compression ratio. As we show in Section V-C, when the voting distribution follows a different distribution, the achievable compression ratio increases.

### B. Effect of scenario

We now investigate the impact of different topologies by comparing the performance of the Östermalm and the Subway scenario. Table III shows the average and maximum deviation $\Delta$ including their confidence intervals, as well as the information overhead in steady state. Again, the minimum deviation values are omitted as they are zero in all cases. These results clearly show that both scenarios exhibit high accuracy, namely achieving an average deviation of 1% for the Östermalm and the Subway scenario. The lower deviation in the Subway scenario is due to the bursty arrivals in the system. This also results in a lower information overhead of approximately 77% compared to 90% in case of the Östermalm scenario as node storing votes, Figure 7(b). For each shared node, a 1 is set in the D-GAP, which results in long sequences of consecutive 1s and increases compression. On the other hand, depending on the voting distribution and its mean value, sequences of consecutive 1s may be shorter in the D-GAP vector for storing votes resulting in a lower compression ratio. In Figure 7(a), the curves for $\lambda = 0.005$ nodes/s and $\lambda = 0.01$ nodes/s show an erratic trend as the scenarios are too sparsely populated to show clear convergence. With the increase of the arrival rate, however, the compression ratio also increases, and for $\lambda = 0.3$ nodes/s, the average compression ratio amounts to 92%. This results from the fact that most nodes in the system have been shared, thus almost all bits in the D-GAP are set to 1. The compression ratio of the D-GAP for storing votes approaches 25% as the arrival rate increases, Figure 7(b). This behavior is strongly dependent on the chosen voting distribution as well as on its mean value $E(x) = 0.75$. As mentioned earlier, the first node entering the system always votes for zero, while the following three nodes vote for one, and subsequently the distribution applies to all further nodes. Hence, the D-GAP for storing votes converges to sequences of three consecutive 1s, which are interrupted by a 0. Applying Equation 1, three consecutive 1s and one 0 require 3 bits for storage whereas a plain bit vector would require 4 bits. Therefore, the achievable compression ratio converges to a value of 0.25 while more and more nodes are shared, which represents the worst case in terms of compression ratio. As we show in Section V-C, when the voting distribution follows a different distribution, the achievable compression ratio increases.

<table>
<thead>
<tr>
<th>Arrival Rate $\lambda$ [NODES/S]</th>
<th>Avg. $\Delta$ [%]</th>
<th>Max. $\Delta$ [%]</th>
<th>IO [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.005</td>
<td>14.91±1.19</td>
<td>100</td>
<td>93.30±0.4</td>
</tr>
<tr>
<td>0.01</td>
<td>7.19±0.73</td>
<td>100</td>
<td>94.05±0.2</td>
</tr>
<tr>
<td>0.07</td>
<td>2.06±0.08</td>
<td>6.3</td>
<td>93.98±0.3</td>
</tr>
<tr>
<td>0.15</td>
<td>1.01±0.05</td>
<td>3.07</td>
<td>89.55±0.2</td>
</tr>
<tr>
<td>0.3</td>
<td>0.79±0.04</td>
<td>2.43</td>
<td>66.28±0.5</td>
</tr>
</tbody>
</table>

Fig. 6. Deviation $\Delta$ depending on the proportion of shared nodes for the Östermalm scenario: (a) $\lambda = 0.07$ nodes/s, (b) $\lambda = 0.15$ nodes/s, and (c) $\lambda = 0.3$ nodes/s.
Figure 7. CDF of the compression ratio for storing (a) shared nodes and (b) votes in the Östermalm scenario under different arrival rates \( \lambda \).

Table III

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>AVG. ( \Delta ) [%]</th>
<th>MAX. ( \Delta ) [%]</th>
<th>IO [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Östermalm</td>
<td>1.01±0.05</td>
<td>3.07</td>
<td>89.55±0.2</td>
</tr>
<tr>
<td>Subway</td>
<td>0.79±0.04</td>
<td>3.27</td>
<td>76.80±0.8</td>
</tr>
</tbody>
</table>

departures are also bursty. As the sojourn time of nodes is lower in the Subway than in the Östermalm scenario, nodes exchange less messages but those exchanged are useful for advancing the knowledge of other nodes.

Figure 8 shows the CDF of the compression ratio for storing shared nodes and votes in both scenarios. The Subway scenario exhibits higher compression ratio as nodes are quicker to disclose all other nodes in the system due to the smaller and more confined area, in which mobility occurs, Figure 8(a). Due to the same fact, the compression ratio of the D-GAPs for storing votes more closely approaches a value of 0.25 in the Subway scenario as apparent from Figure 8(b).

C. Effect of voting distribution

Finally, we compare the performance of DiVote for different voting distributions. We consider a deterministic as well as a uniform voting distribution, which can be seen as a more realistic representation of votes of pedestrians in an area. We choose three different mean values of the distribution (\( \mathbb{E}(x) = 0.25, 0.5, \) and 0.75), and we perform five simulation runs for each mean value.

Table IV shows the average compression ratio with 95% confidence intervals for \( \mathbb{E}(x) = 0.25, 0.5, \) and 0.75 in the Östermalm scenario. These results obtained for the uniform distribution are contrasted with the deterministic distribution. As the mobility trace is unchanged during simulation runs, the compression ratio for the D-GAP vector for shared nodes is independent of the proportion of nodes that vote for one. However, uniformly distributing the votes leads to higher compression ratios compared to the deterministic distribution as longer sequences of 1s and 0s are possible.

The gain in terms of achievable compression ratio becomes more apparent from Table V, which shows the average compression ratio and their 95% confidence intervals for the Subway scenario. As expected, in case of the deterministic distribution the compression ratio approaches a value of 0% for \( \mathbb{E}(x) = 0.5 \) as the D-GAP expands to a bit vector of alternating 0 and 1. However, when votes are uniformly distributed, longer sequences of 1s and 0s become possible resulting in a compression ratio of 19% even for an average global result of \( \mathbb{E}(x) = 0.5 \).
VI. CONCLUSION

In this paper, we presented DiVote, a distributed voting protocol in the context of urban polling, which is suitable for environments, in which nodes exhibit high mobility. DiVote relies on device-to-device communication to exchange voting information. The proposed DiVote protocol exhibits the following main features:

- **Privacy**: By using a cryptographic hash function, votes cannot be related to the corresponding node identities.

- **Convergence speed**: The dynamism due to mobility imposes tight constraints on the convergence speed of the algorithm. Consequently, DiVote immediately updates the local estimate. Simulation results obtained when applying DiVote to realistic pedestrian mobility traces show that even in sparse scenarios local estimates quickly converge to the global result after having shared 30% of all nodes in the system.

- **Accuracy**: At the same time, accuracy of local estimates is ensured as DiVote avoids to erroneously count votes multiple times, which is of decisive importance as the same node may be encountered several times. In dense scenarios, the local estimate does not deviate by more than 3% from the global result after the system reaches the steady state.

- **Scalability**: Rather than storing shared nodes and their votes in plain bit vectors, DiVote uses D-GAP compression to be scalable in terms of required storage capacity. For realistic voting distributions, at least 19% compression is achieved. Furthermore, the processing overhead introduced at the application layer is very low as only a fraction of the received messages (approximately 30%) has to be processed even in the densely populated scenarios.

Prospectively, we intend to theoretically analyze DiVote and compare it with other existing schemes. We will further investigate more voting distributions and compare the D-GAP compression with other compression algorithms.

### Table V

<table>
<thead>
<tr>
<th>E(x)</th>
<th>Uniform Distribution</th>
<th>Deterministic Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AVG. CR of D-GAPS</td>
<td>AVG. CR of D-GAPS</td>
</tr>
<tr>
<td></td>
<td>Shared nodes Votes</td>
<td>Shared nodes Votes</td>
</tr>
<tr>
<td>0.25</td>
<td>0.33 ± 0.01</td>
<td>0.98 ± 0.0005</td>
</tr>
<tr>
<td>0.5</td>
<td>0.19 ± 0.01</td>
<td>0.01 ± 0.0005</td>
</tr>
<tr>
<td>0.75</td>
<td>0.33 ± 0.01</td>
<td>0.25 ± 0.0003</td>
</tr>
</tbody>
</table>

### Acknowledgment

The authors would like to thank the German Research Foundation (DFG) (research fellowship, GZ: DA 1687/2-1) for their financial support.

### References


77
Joint Optimization of User Association and User Satisfaction in Heterogeneous Cellular Networks

Farah Moety, Mustapha Bouhtou, Taoufik En-Najjary, Ridha Nasri
Orange Labs Networks - Chatillon, France
Email: {farah.moety, mustapha.bouhtou, taoufik.enajjary, ridha.nasri}@orange.com

Abstract—In this paper, we address the joint problem of user association and resource allocation in wireless heterogeneous networks. Therefore, we formulate an optimization approach considering two objectives, namely, maximizing the number of served User Equipments (UEs) and maximizing the sum of the UE utilities. Precisely, the aim is to associate UEs with the optimal Radio Access Technology (RAT) and to allocate to these UEs the optimal Resource Units (RUs) based on their requested services and contracts. Our problem is challenging because it is mixed integer non-linear optimization. To tackle this difficulty, we provide a Mixed Integer Linear Programming (MILP) re-formulation of the problem that makes it computationally tractable. Various preferences for user association and resource allocation are conducted by tuning: on the one hand, the weights associated with different services and contracts; on the other hand, the weights associated with the considered two objectives. The optimal solution of the MILP problem is computed for a realistic network scenario and compared with legacy solution. Extensive simulation results show that the proposed optimization approach improves the overall network performance while considering the UE requested service and contract; it outperforms legacy solutions in terms of user satisfaction. Moreover, it provides an efficient distribution of UEs on the different RATs.

I. INTRODUCTION

5G networks will consist of heterogeneous networks interoping in a clever way offering seamless network connectivity to the user equipment (UE). In such heterogeneous network, a challenging joint problem of user association and resource allocation arises. Hence, determining which Base Station (BS) over which Radio access Technology (RAT) a given UE will be associated, and how many Resource Units (RUs) it will be allocated, all substantially affect the network performance. In the present article, we tackle the joint problem of user association and resource allocation in wireless heterogeneous networks. We thus formulate an optimization approach that jointly maximizes the number of served UEs and the sum of their utilities. This is achieved by associating UEs with the optimal BS/RAT and allocating to these UEs the optimal RUs based on their requested services and contracts.

The present approach provides benefits for both UEs and network operator. Each UE is associated with the optimal RAT that guarantees its required Quality of Service (QoS) according to the contract between this UE and the network operator. The network operator in turn can better exploit its radio resources and satisfy more UEs.

In the state-of-the-art, different approaches were proposed to tackle the user association problem. First, the network-centric approach [1], where the network takes decisions transparently to UEs in a way to optimize overall network performance. Second, the user-centric approach [2], [3], where each UE selfishly strives to improve its own performance. Third, the hybrid approach [4] combines the two previous ones in such a way it does not only aim to achieve the best possible network performance, but it also considers the UE's requirement.

Authors in [1] formulated a network-centric approach for user association in a heterogeneous wireless network scenario, with various technologies and operators, as a linear optimization problem. They used a utility function with various merit parameters, reflecting the requirements of both the UEs and the network itself. In the aforementioned paper, the capacity of a BS/RAT is in terms of the number of UEs associated with this BS/RAT and not in terms of RUs. Authors in [2], [3] modeled the user association problem in wireless heterogeneous networks as a non-cooperative game, in which UEs select the BS/RATs in a distributed manner to increase their own individual throughput. Additionally, authors in [2] proposed a reinforcement learning method to find the optimal strategy of the UEs in a case where UEs have no information about other UEs. Authors in [4] formulated a hybrid approach for the user association problem. Deriving network information was formulated as a Semi-Markov Decision Process (SMDP), so as to guide UEs' decisions in a way to satisfy operator interests. Moreover, UEs combine their needs and preferences with the signaled network information, and each UE selects the BS/RAT to be associated with, in a way to maximize its own utility. However, a heavy computational load was required to find optimal solutions.

Moreover, in the literature, the problem of user association and resource allocation has not been studied jointly except in a few recent cases [5]. Authors in [5] proposed an integer linear programming model for the joint problem of user association and resource allocation with the objective of maximizing the number of associated UEs and the minimum granted utility.

The authors studied the case of only one BS with multi-RATs and the UE's utility depends only on throughput. The key contributions of our work are as follows:

- We formulate a hybrid optimization approach that jointly maximizes the number of served UEs and the sum of their utilities. The particularity of the proposed approach that it does not only consider the network overall performance (optimal user association and optimal RUs allocation), but also, and as one of the most relevant aspect, the UEs' preferences (requested QoS and contract).

- Our problem formulation allows us to investigate various preferences for user association and resource allocation by tuning: on the one hand, the weights associated with the different service classes and contracts; on the other hand, the weights associated with the considered two objectives.
• Starting from a mixed integer non-linear formulation of the problem, we provide a Mixed Integer Linear Programming (MILP) re-formulation of the problem that makes it computationally tractable. We compute the optimal solution of the MILP problem for a realistic network scenario and compare its performance with legacy solutions.

The rest of the paper is organized as follows. In Section II, we describe the network model. In Section III we present the proposed optimization approach and the formulation of the joint problem of user association and resource allocation as a MILP problem. In Section IV we provide extensive simulation results. Conclusions and perspectives are given in Section V.

II. NETWORK MODEL

We consider the downlink of a heterogeneous wireless network composed of $N_b$ BSs with $N_T$ co-localized RATS. The indices $i \in I = \{1, \ldots, N_b\}$, and $j \in J = \{1, \ldots, N_T\}$, are used throughout the paper to designate a given BS and a given RAT, respectively. We term by $k \in K = \{1, \ldots, N_u\}$, the index of a given UE where $N_u$ is the number of UEs in the network.

A. Network Resources

In each RAT, the radio resource is divided into elementary RUs. Typically, in 4G wireless networks (e.g., LTE technology), the Resource Block (RB) is the smallest RU that can be scheduled. The RB consists of 12 consecutive subcarriers for one sub-frame duration (1 ms). In 3G wireless networks (e.g., HSPA technology), codes and power are treated as RUs. In this work, we consider that all codes have the same power and only codes are thus treated as RUs. Moreover, the capacity of a RAT $j$ is limited by the number of available RUs and it is denoted by $R_j$.

B. Data Rate

The perceived throughput of UE $k$, denoted by $\gamma_k$, is the sum of the perceived throughput of this UE from the BSs/RATs of the network. Let $\varphi_{i,j,k}$ be the perceived throughput of UE $k$ from BS $i$ over RAT $j$ per one RU and $\lambda_{i,j,k}$ be the number of RUs assigned to UE $k$ associated with BS $i$ over RAT $j$. When a UE is not associated with a given BS/RAT, this UE is not assigned any RU from the corresponding BS/RAT ($\lambda_{i,j,k} = 0$). Thus, $\gamma_k$ is given by:

$$\gamma_k = \sum_{i \in I, j \in J} \lambda_{i,j,k} \varphi_{i,j,k}, \quad \forall k \in K.$$  (1)

Let $\nu_{i,j,k}$ be the Signal-to-Interference-plus-Noise Ratio (SINR) of UE $k$ from BS $i$ over RAT $j$, and $w_j$ be the bandwidth per RU. Based on Shannon’s formula, the theoretical throughput that can be attained for UE $k$ from BS $i$ over RAT $j$ per RU, is given by:

$$\varphi_{i,j,k} = w_j \log(1 + \nu_{i,j,k}), \quad \forall i \in I, \forall j \in J, \forall k \in K.$$  (2)

For instance, for LTE, $w_1$ is the bandwidth per one RB; for HSDPA, $w_2$ is the chip rate over the spreading factor. The SINR $\nu_{i,j,k}$ is given by [6]:

$$\nu_{i,j,k} = \frac{G_t}{G_t(a + 1)\text{ISR}_{i,j,k} + L_{i,j,k} \frac{\alpha}{P}}, \quad \forall i \in I, \quad \forall j \in J, \forall k \in K,$$  (3)

where $G_t$ is the transmit antenna gain and $a$ is the orthogonality factor (e.g., $a > 0$ in 3G wireless networks, $a = 0$ in 4G networks). $L_{i,j,k}$ is the path loss detected by UE $k$ from BS $i$ over RAT $j$, $P_N$ is the noise power and $P$ is the power per RU, and $\text{ISR}_{i,j,k}$ is the Interference to Signal Ratio of UE $k$ from BS $i$ over RAT $j$. $\text{ISR}_{i,j,k}$ is given by [6]:

$$\text{ISR}_{i,j,k} = \sum_{i' \in I, i' \neq i} \frac{L_{i,j,k}}{\pi_{i',j,k}}, \quad \forall i \in I, \forall j \in J, \forall k \in K,$$  (4)

where $\pi_{i',j,k}$ is the percentage of occupied resources in the interfering BS $i'$ over RAT $j$.

C. Utility

We consider two types of traffic classes: non real-time and real-time classes. We use the well-known concept of utility function which maps the UE perceived throughput with the level of UE satisfaction or QoS. The index $s$ is used throughout the paper to designate a given class of a service. Let $U_k^s$ denote the utility function of UE $k$ with class $s$ service.

a) Class A traffic: includes non real-time services that are generated by traditional data applications such as file and mail download, web (e.g., Twitter, Facebook), etc. These applications are assumed to be tolerant in delay variation and adapt their rate to available resource by means of a transport protocol like TCP. Thus, the elasticity of these services can be modeled by concave utility functions [7]. In this work, we assume that UEs with non real-time services have an exponential utility function given by:

$$U_k^a(\gamma_k) = 1 - e^{(-\gamma_k/\gamma^c)},$$  (5)

where $\gamma^c$ is the comfort throughput demand of the UE (i.e., the mean throughput beyond which, UE satisfaction exceeds 63% of maximum satisfaction). The satisfaction increases slowly as the throughput exceeds the comfort throughput demand.

b) Class B traffic: includes real-time services that are generated by real-time video and voice applications. These services are partially elastic and usually characterized by a minimum, an average and a maximum data rate requirement [8]. The elasticity of these services can be modeled by a sigmoidal-like function [7]. Therefore, in this work, we consider the following sigmoidal utility function for UEs with real-time services [9]:

$$U_k^b(\gamma_k) = d_1 \left(\frac{1}{1 + e^{b(\gamma_k - \gamma^a)}} - d_2\right).$$  (6)

where $\gamma^a$ represents the average throughput demand of class B service. $b$ is a positive constant that determines the shape of the sigmoid. $d_1 = \frac{1}{1 + e^{b\gamma^a - \gamma^a}}$ and $d_2 = \frac{1}{1 + e^{b\gamma^a}}$.

We note that, the utility functions of the UEs with the same service class are differentiated according to the contract between these UEs and the network operator.

D. Contract

In this work, we consider that the network operator provides two differentiated types of contracts, namely, regular ($\mathcal{R}$) and premium ($\mathcal{P}$). They differ in their QoS, with the premium contract being the most expensive one but also the one guaranteeing higher UE satisfaction level. As a matter of fact, for the same offered average throughput, premium UEs
perceive a level of satisfaction lower than that of regular UEs, as shown in Figure 1. The index $t$ is used throughout the paper to designate a given contract.

![Figure 1. Utility functions for UEs with different service class and contract.](image)

III. OPTIMIZATION PROBLEM

A. Problem Formulation

The proposed optimization approach consists of finding an optimal user association and an optimal RU allocation that jointly maximize the number of served UEs and the sum of their utilities. It takes into account the required QoS of the UE’s requested service and the contract between this UE and the network operator. Let $K^{s,t}$ denote the set of UEs with class $s$ service and contract $t$.

The design variables in our maximization problem are as follows:

- The user association with the network BSs over a given RAT.
- The number of RUs assigned to a given UE associated with a given BS/RAT.

Let $\Theta$ be the matrix, with elements $\theta_{i,j,k}$, defining the user association with the network BSs over a given RAT; and $\theta_{i,j,k}$ be a binary variable that indicates whether or not UE $k$ is associated with BS $i$ over RAT $j$.

$\theta_{i,j,k} = \begin{cases} 1 & \text{if UE } k \text{ is associated with BS } i \text{ over RAT } j, \\ 0 & \text{otherwise}. \end{cases}$

Let $\Lambda$ be the matrix, with elements $\lambda_{i,j,k}$, defining the amount of RUs allocated to a given UE from a given BS/RAT. $\lambda_{i,j,k}$ is an integer variable that indicates the number of RUs assigned to UE $k$ associated with BS $i$ over RAT $j$.

In the following, we define the constraints on the decision variables and the utility functions. We start by defining the constraints on the user association and the RUs as follows:

$$\sum_{i \in I, j \in J} \theta_{i,j,k} \leq 1, \quad \forall k \in K. \quad (7)$$

$$\lambda_{i,j,k} \leq R_j, \quad \forall i \in I, \forall j \in J. \quad (8)$$

$$\lambda_{i,j,k} \geq \theta_{i,j,k}, \quad \forall i \in I, \forall j \in J, \forall k \in K. \quad (9)$$

$$\lambda_{i,j,k} \leq R_j \theta_{i,j,k}, \quad \forall i \in I, \forall j \in J, \forall k \in K. \quad (10)$$

Constraints (7) state that a given UE can be associated with only one BS over one RAT. Constraints (8) ensure that the limit on the number of RUs for each RAT, $R_j$, is not exceeded. Constraints (9) ensure that $\lambda_{i,j,k}$ is not equal to zero when $\theta_{i,j,k}$ is equal to one. When a UE is associated with a given BS/RAT, these constraints ensure that this UE is assigned a number of RUs from the corresponding BS/RAT. Constraints (10) force $\lambda_{i,j,k}$ to be equal to zero when $\theta_{i,j,k}$ is equal to zero. When a UE is not associated with a given BS/RAT, these constraints prevent this UE from being assigned a number of RUs from the corresponding BS/RAT.

Let us introduce some notations to define the constraints on the utility functions. Let $u^{\min,s,t}_{k}$ be the minimum required utility for class $s$ service with contract $t$, and $u^{\max,s,t}_{k}$ be the maximum utility that the operator is willing to offer for service $s$ with contract $t$. The constraints on the utility functions are as follows:

$$U^{s,t}_{k} \geq u^{\min,s,t}_{k} \theta_{i,j,k}, \quad \forall i \in I, \forall j \in J, \forall k \in K^{s,t}, \forall s \in \{A, B\} \forall t \in \{P, R\}, \quad (11)$$

$$U^{s,t}_{k} \leq u^{\max,s,t}_{k} \theta_{i,j,k}, \quad \forall i \in I, \forall j \in J, \forall k \in K^{s,t}, \forall s \in \{A, B\}, \forall t \in \{P, R\}. \quad (12)$$

In case a given UE is associated with a given BS/RAT, constraints (11) ensure that the minimum required level of satisfaction for this UE according to its service and contract, is guaranteed. In other words, if the minimum required level of satisfaction for a given UE cannot be guaranteed by a given BS/RAT, this UE will not be associated with this BS/RAT to preserve the overall network performance. In case a given UE is associated with a given BS/RAT, constraints (12) ensure that the maximum level of satisfaction for this UE according to its service and contract, is not exceeded. In fact, we add these constraints to ensure an effective use of the RUs available in the network. For instance, for class A service, as the utility function is concave, the satisfaction increases slowly as the throughput exceeds the comfort throughput demand (as shown in figure 1). Thus, limiting the satisfaction to a maximum level, prevents, some UEs from being assigned relatively a high number of RUs without increasing considerably their satisfaction. Similarly, for class B service, the utility function is a sigmoidal-like function, which is concave for a throughput higher than the average throughput demand.

Let $\gamma^{A,t}_{k}$ and $\gamma^{B,t}_{k}$ denote the comfort throughput demand for class A service and contract $t$, and $\gamma^{s,t}_{k}$ denote the average throughput demand for class B service and contract $t$. The expressions of the utility functions of UEs with class A and B services, and contract $t$, are respectively given by:

$$U^{s,t}_{k}(\gamma_{k}) = 1 - e^{(\frac{-\gamma_{k}}{\theta \gamma_{k}})}, \quad \forall k \in K^{A,t}, \forall s \in \{A\}, \forall t \in \{P, R\}. \quad (13)$$

$$U^{s,t}_{k}(\gamma_{k}) = d_{1}^{s,t} \left( 1 + \frac{1}{e^{\frac{\gamma_{k}}{\theta \gamma_{k}}} - 1} \right) - d_{2}^{s,t}, \quad \forall k \in K^{B,t}, \forall s \in \{B\}, \forall t \in \{P, R\}. \quad (14)$$

where $d_{1}^{s,t} = \frac{1}{1 + e^{\frac{\gamma^{s,t}_{k}}{\theta \gamma_{k}}} - 1}$ and $d_{2}^{s,t} = \frac{1}{1 + e^{\frac{\gamma^{s,t}_{k}}{\theta \gamma_{k}}} - 1}, \forall s \in \{B\}, \forall t \in \{P, R\}$. The integrality constraints for the decision variables $\theta_{i,j,k}$ and $\lambda_{i,j,k}$ are respectively given by:

$$\theta_{i,j,k} \in \{0, 1\}, \quad \forall i \in I, \forall j \in J, \forall k \in K. \quad (15)$$

80
\[ \lambda_{i,j,k} \in \mathbb{N}, \quad \forall i \in I, \forall j \in J, \forall k \in K. \]  

(16)

To eliminate some trivial cases that must not be included in the solution, we add the following constraints:

If UE \( k \) is not covered by BS \( i \) over RAT \( j \), then

\[ \theta_{i,j,k} = 0. \]  

(17)

The equation (17) prevents a given UE from being associated with a BS/RAT if this UE is not in its coverage area. We note that the coverage area of a given BS/RAT is defined as the geographical area where the received SINR of each UE is above a given minimum threshold.

Therefore, our approach can be formulated as an optimization problem \((P)\) that consists of jointly maximizing the weighted sum of the number of served UEs in the network and their utilities subject to the aforementioned constraints. Consequently, problem \((P)\) is given by:

Maximize \[ \sum_{k \in K^{s,t}, \ s \in \{A, B\}, \ t \in \{P, R\}} \alpha^{s,t} \left( \beta_1 \sum_{i \in I, \ j \in J} \frac{\theta_{i,j,k}}{K^{s,t}} + \beta_2 U_k^{s,t} \right), \]  

subject to: \((1)\) and \((7)\) to \((17)\),

where \( \alpha^{s,t} \) are the weighting factors corresponding to the UEs with different services and contracts. Tuning these factors, allows to privilege some UEs based on their service classes and contracts. This in turns allows to consider various preferences for user association and RUs allocation. It is usually assumed that \( \sum_{s \in \{A, B\}} \sum_{t \in \{P, R\}} \alpha^{s,t} = 1 \), and that \( \alpha^{s,t} \in [0,1] \). In particular, when \( \alpha^{s,t} \leq \alpha^{A,R} > \alpha^{A,P} > \alpha^{A,R} \), UEs with class \( B \) service and premium contract are the most privileged UEs, followed by UEs with class \( B \) service and regular contract, then UEs with class \( A \) service and premium contract, and finally UEs with class \( A \) service and regular contract have the lowest privilege.

Similarly, \( \beta_1 \) and \( \beta_2 \) are the weighting factors representing the relative importance of the two objectives (namely, the number of served UEs, and the sum of UE utilities). Moreover, \( \beta_1 + \beta_2 = 1, \beta_1 \in [0,1] \). In particular, when \( \beta_1 \) is equal to 0, we only focus on the maximizing the number of served UEs, and as \( \beta_1 \) decreases and \( \beta_2 \) increases more importance is given on the maximization of the sum of UE utilities.

B. From non-linear to linear optimization problem

Problem \((P)\) is a non-linear mixed integer optimization problem. The non-linearity comes from the expression of the UE utility functions (exponential and sigmoidal functions). Solving such problem is a very challenging task. In this section, we explain how to transform problem \((P)\) into a MILP problem. A MILP problem consists of a linear objective function, a set of linear equality and inequality constraints and a set of variables with integer restrictions. Generally, MILP problems are solved using a linear-programming based branch-and-bound approach [10]. The idea of this approach is to solve Linear Program (LP) relaxations of the MILP and to look for an integer solution by branching and bounding on the decision variables provided by the LP relaxations.

1) Methodology: Equations (1) show that the possible values of UE’s throughput are discrete and not continuous. This implies that the possible values of UE’s utility are also discrete, which allows us to transform problem \((P)\) into a MILP problem \((P_1)\).

Let us introduce some notation to describe the MILP reformulation. Let \( n \in \mathbb{R}^j = \{1, \ldots, R_j\} \), be the possible number of RUs that a UE can obtain when associated with a given BS over RAT \( j \). Let \( \eta_j^{i,k,n} \in \{0,1\} \) be the throughput of UE \( k \), associated with BS \( i \) over RAT \( j \) and assigned \( n \) RUs. Let \( \mu_j^{i,k,n} \) be the utility of UE \( k \), with class \( s \) service and contract \( t \), associated with BS \( i \) over RAT \( j \) and assigned \( n \) RUs. First, we compute all the possible values of the throughput a UE can obtain from the different BS/RATs \((i.e., \text{the values of} \ \eta_j^{i,k,n})\). Second, we compute all the possible values of the UE’s utility from the different BS/RATs \((i.e., \text{the values of} \ \mu_j^{i,k,n})\).

Once we know all the possible values of the UE’s utility from the different BS/RATs, we reformulate the considered problem of user association and RUs allocation as a linear knapsack problem, where RUs of a given UE are the objects to be chosen and the BS/RATs are the knapsacks in such a way to: (i) jointly maximize the number of served UEs and the sum of their utilities; (ii) take into account the required QoS of the UE’s requested service and the contract between this UE and the network operator.

With the linear Knapsack reformulation, we thus introduce a new binary variable \( x_{i,k,n}^j \), equating to one if \( n \) RUs are assigned to UE \( k \) associated with BS \( i \) over RAT \( j \), and zero otherwise. Therefore, \( \lambda_{i,j,k} \) is given by:

\[ \lambda_{i,j,k} = \sum_{n \in \mathbb{R}^j} n x_{i,k,n}^j, \quad \forall i \in I, \forall j \in J, \forall k \in K. \]  

(19)

Consequently, our optimization problem is formulated as a MILP problem \((P_1)\) and it is given by:

Maximize \[ \sum_{k \in K^{s,t}, \ s \in \{A, B\}, \ t \in \{P, R\}} \alpha^{s,t} \left( \beta_1 \sum_{i \in I, \ j \in J} \frac{\theta_{i,j,k}}{K^{s,t}} + \beta_2 \sum_{i \in I, \ j \in J, \ n \in \mathbb{R}^j} x_{i,k,n}^j \mu_{i,k,n}^{s,t} \right), \]  

subject to: \((7)\) to \((10)\), \((15)\) to \((17)\) and \((19)\),

\[ \sum_{i \in I, \ j \in J, \ n \in \mathbb{R}^j} x_{i,k,n}^j \leq 1, \quad \forall k \in K, \]  

(21)

\[ x_{i,k,n}^j \leq \theta_{i,j,k}, \quad \forall i \in I, \forall j \in J, \forall k \in K, \forall n \in \mathbb{R}^j, \]  

(22)

\[ \sum_{n \in \mathbb{R}^j} x_{i,k,n}^j \mu_{i,k,n}^{s,t} \geq \min_{s,t} \theta_{i,j,k}, \quad \forall i \in I, \forall j \in J, \forall k \in K^{s,t}, \forall s \in \{A, B\}, \forall t \in \{P, R\}, \]  

(23)

\[ \sum_{n \in \mathbb{R}^j} x_{i,k,n}^j \mu_{i,k,n}^{s,t} \leq \max_{s,t} \theta_{i,j,k}, \quad \forall i \in I, \forall j \in J, \forall k \in K^{s,t}, \forall s \in \{A, B\}, \forall t \in \{P, R\}, \]  

(24)

\[ x_{i,k,n}^j \in \{0,1\}, \quad \forall i \in I, \forall j \in J, \forall k \in K, \forall n \in \mathbb{R}^j. \]  

(25)

Problem \((P_1)\) is equivalent to problem \((P)\). Precisely, constraints \((21)\) state that a given UE can be assigned a given number of RUs from only one BS over one RAT. Constraints

81
(22) force $x_{i,k,n}^j$ to be equal to zero when $\theta_{i,j,k}$ is equal to zero. These constraints prevent a UE from being assigned a number of RUs if he is not associated with a given BS/RAT. In fact, constraints (22) are called valid inequalities as they help us to find the optimal solution. Constraints (23) and constraints (24) replace constraints (11) and (12), respectively. Finally, constraints (25) are the integrality constraints for the variable $x_{i,k,n}^j$.

IV. PERFORMANCE EVALUATION

A. Evaluation Methodology

We consider the realistic positioning of the 4G and 3G network BSs for the district 14 of Paris-France [11]. The network topology is composed of 18 cells ($N_0=18$) with two co-located RATs, namely LTE and HSDPA. The positioning of UEs follows a random uniform distribution as shown in Fig. 2. For simplicity, we assume that each considered cell has an Omni-directional radiation pattern. We also assume that all interfering BSs has the same percentage of occupied resources and thus $\gamma_{i,j} = \gamma_i$ (cf. Eq. (4)). The simulated LTE system bandwidth is 10 MHz, therefore we have 50 RBs available in each cell. We assume that two R Bs are used for special information (e.g. signaling, etc.), and we thus have 48 RBs available in each cell. For HSDPA, the system bandwidth is 5 MHz, therefore we have 16 codes available in each cell. We assume that two codes are reserved for special information (e.g. signaling, etc.), and we thus have 14 codes available in each cell. The simulation parameters and the pathloss model follow that in [12], which are summarized in Tab. I.

The path loss between the BS/RAT and the UE is computed according to the Cost 231 extended Hata model considering a urban environment [12], with a carrier frequency $f_1$ of 2600 MHz for 4G and a carrier frequency $f_2$ of 2100 MHz for 3G. The shadowing [dB scale] is represented by a random variable following normal distribution with a mean of 0 dB and a standard deviation of 10 dB. The coverage radius is 500 m for LTE and 700 m for HSDPA. Table II shows how much UEs are covered by the considered BS/RATs for the considered number of UEs in the network in these simulations. For instance, for 90 UEs in the network, 21 UEs are covered by RAT 2, where only 11 of them are also covered by RAT 1. These 11 UEs have the choice to be associated with one of the two RATs or blocked, while the remaining 10 UEs can only be associated with RAT 2 or blocked.

Table III shows the simulation parameters of UEs with different service classes and contracts. First, it shows the percentage of these UEs in the network. Second, it depicts the QoS demands of UEs regarding their service classes and contracts. QoS demands are expressed in terms of: i) comfort throughput $\gamma^{c,A,t}$ (cf. Eq(13)) for UEs with class A service and contract $t$, and ii) average throughput $\gamma_t^{u,B}$ (cf. Eq(14)) for UEs with class B service and contract $t$. Table IV shows the values of the minimum required utility $u_{t}^{min}$ (cf. constraints (23)) and the values of the maximum utility $u_{t}^{max,A,t}$ (cf. constraints (24)) for each service class and contract used in our simulations. Indeed, since UEs with class A service adapt to resource availability, and require no QoS guarantees, their minimum utility requirement is zero ($u_{t}^{min,A,R/P} = 0$). However, UEs with class B service are characterized by a minimum data rate requirement; decreasing below a certain threshold will result in a drop in the QoS. Therefore, we consider that the minimum required level of utility to be guaranteed, is equal to 10% ($u_{t}^{min,B,R/P} = 10\%$). As mentioned in Section III-A, limiting the satisfaction to a maximum level, prevents some UEs from being assigned a relatively high number of RUs without increasing considerably their satisfaction. For UEs with class A service, the maximum

---

Table I

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage radius</td>
<td>500 m</td>
</tr>
<tr>
<td>Carrier frequency</td>
<td>2600 MHz</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>10 MHz</td>
</tr>
<tr>
<td>Number of RUs per cell</td>
<td>$R_1 = 48$</td>
</tr>
<tr>
<td>Orthogonality factor ((\gamma_i))</td>
<td>0.5</td>
</tr>
<tr>
<td>Percentage of occupied resources ((\gamma_i))</td>
<td>80%</td>
</tr>
<tr>
<td>Transmit power</td>
<td>10 W</td>
</tr>
<tr>
<td>Antenna</td>
<td>Omni-directional</td>
</tr>
<tr>
<td>Noise Figure</td>
<td>9 dB</td>
</tr>
<tr>
<td>Environment</td>
<td>Urban</td>
</tr>
<tr>
<td>Pathloss model</td>
<td>Cost 231 extended Hata model</td>
</tr>
<tr>
<td>Shadowing standard deviation</td>
<td>10 dB</td>
</tr>
</tbody>
</table>

Table II

<table>
<thead>
<tr>
<th>Number of UEs in the network</th>
<th>90</th>
<th>180</th>
<th>270</th>
<th>360</th>
<th>540</th>
<th>720</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAT 1 = 4G</td>
<td>11</td>
<td>22</td>
<td>45</td>
<td>56</td>
<td>66</td>
<td>88</td>
</tr>
<tr>
<td>RAT 2 = 3G</td>
<td>21</td>
<td>52</td>
<td>84</td>
<td>128</td>
<td>171</td>
<td></td>
</tr>
</tbody>
</table>

Table III

<table>
<thead>
<tr>
<th>Class A</th>
<th>Class B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Percentage of UEs</td>
</tr>
<tr>
<td>Regular</td>
<td>40%</td>
</tr>
<tr>
<td>Premium</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>Throughput of UEs</td>
</tr>
<tr>
<td>Regular</td>
<td>$\gamma_t^{u,B} = 1$ Mba</td>
</tr>
<tr>
<td>Premium</td>
<td>$\gamma_t^{u,B} = 2$ Mba</td>
</tr>
</tbody>
</table>

Table IV

\[
\begin{align*}
\gamma^{c,A,t} & = 1\text{ Mba} \\
\gamma_t^{u,B} & = 1\text{ Mba}
\end{align*}
\]
utility is equal to $u_{\text{max}, A, R/P} = 77.70\%$, which corresponds to an average throughput equal to 1.5 times the comfort throughput demand. We consider that UEs with class B service are more privileged than UEs with class A service, thus their maximum utility is higher and it is equal to $u_{\text{max}, B, R/P} = 95\%$.

### Table IV
Minimum and Maximum utility for each service class and contract.

<table>
<thead>
<tr>
<th>Class</th>
<th>Regular/Premium</th>
<th>$u_{\text{min}, \cdot}$</th>
<th>$u_{\text{max}, \cdot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>Regular/Premium</td>
<td>0%</td>
<td>77.70%</td>
</tr>
<tr>
<td>Class B</td>
<td>Regular/Premium</td>
<td>10%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Toward studying the performance of the proposed approach, we investigate the optimal solutions obtained by tuning: on the one hand, the weights associated with different service classes and contracts; on the other hand, the weights associated with the considered two objectives (maximizing the number of served UEs and maximizing the sum of the UE utilities). We thus consider three settings illustrated in Table V. Setting S1 matches the case where all UEs are equally important in the optimization, and both objectives are equally important. Setting S2 matches the case where UEs with class B service and premium contract are the most privileged UEs, followed by UEs with class C service and regular contract, then UEs with class A service and premium contract, and finally UEs with class A service and regular contract have the lowest privilege. Moreover, both objectives are equally important in this setting. Finally, setting S3 is similar to S2 in the privilege of the different service classes and contracts. However, it is different from other settings in the relative importance of the two objectives, where more importance is given to maximizing the number of served UEs.

We compute the optimal solution of the MILP problem \((P1)\) using the CPLEX V12.6.0.0 solver running on a computer equipped with an Intel(R) Xeon(R) CPU L5630, 4 cores, and a clock rate of 2.13 GHz. This tool provides the optimal solution using the branch and cut approach [13] (which consists of a combination of a cutting plane method with a branch-and-bound algorithm). The solver configuration used is the default setting. All the solutions are thus provided at the optimum. The input data for the CPLEX solver are generated using MATLAB. Thus, in MATLAB, we implement the considered heterogeneous network topology of the 14th district of Paris. We adopt the Monte Carlo method by generating 10 snapshots with different random uniform UEs distribution. After doing the calculations for all the snapshots, we provide the 95% Confidence Interval (CI) for each simulation result. We compare the performance of the MILP solutions for the considered settings with an existing approach for user association and resource allocation presented in the sequel.

### B. Existing Approach

In legacy cellular networks, UEs are associated with the BS/RAT delivering the Highest Received Power (HRP) of pilot signals [14] with a prioritization for the last network generation. Moreover, RUs are shared equitably between UEs meaning that all UEs get a similar number of RUs.

In the present paper, we devise a reference model denoted by HRP and based on legacy networks considering a higher priority association for RAT 1 (4G). The reference model HRP works as follows:

- The UE measures the received power of pilot signals for the covering BSs/RATs, and keeps them in a descending order queue for each RAT.
- Starting with RAT 1 queue, the UE is associated with the first BS/RAT 1 that positively confirmed its request among the list in this queue. A UE request may be blocked for different reasons like limitations on the capacity at the BS/RAT 1 and the minimum required received power (-85dBm [15]).
- In case of a rejection from all BSs in the RAT 1 queue, the UE repeats the same procedure for RAT 2 queue. Similarly, a UE request may be rejected for limitations on the resource capacity.

Once the association is done with a given BS/RAT, the UE shares equitably the RUs with other UEs associated with the same BS/RAT.

### C. Simulation Results

We now investigate the optimal solutions obtained by the considered settings, and the solution of HRP. First, we start by evaluating the percentage of served UEs. Second, we present the users satisfaction, which is expressed by the average utility per UE per service class per contract. Finally, we show how the considered optimization approach load balances the UEs on the different RATs.

1) **Percentage of served UEs:** Figure 3 shows the percentage of served UEs with different service classes and contracts as a function of the number of UEs in the network, for the solutions of the considered settings. Intuitively, as the number of UEs in the network increases, the percentage of served UEs decreases because of the limitations on the available RUs at the BSs/RATs.

For the case where all UEs and both objectives are equally important in the optimization, Figure 3(a) shows that the percentage of served UEs with class A service (both regular and premium contracts) is higher than that of UEs with class B service (both regular and premium contracts). In fact, the objective function of problem \((P1)\) consists of jointly maximizing the number of served UEs and the sum of their utilities. Moreover, for the same offered throughput, UEs with class A service have a higher utility compared with UEs with class B service (as shown in Figure 1). Therefore, UEs with class A service are much more accepted than others in this setting.

Setting S2 overcomes this issue by privileging UEs with class B service. Precisely, Figure 3(b) shows that the percentage of served UEs with class B service (both premium and regular contracts) has increased, while the percentage of served UEs with class A service (both premium and regular contracts) has decreased. We note that, although the percentage of served UEs with class B service (both premium and regular) has increased in setting S2, this percentage is still relatively low for relatively high number of UEs in the network. This is because, for the same level of satisfaction, UEs with class B service are more exigent in throughput than UEs with class A service (as shown in Figure 1). Moreover, a UE with class B service is served only if it will be allocated a given number of RUs...
Table V
Three studied settings.

<table>
<thead>
<tr>
<th>Settings</th>
<th>Weighting coefficients value associated with different service classes and contracts</th>
<th>Objectives (number of served UEs and sum of UEs utilities)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>$\alpha_{R^H}=\alpha_{B^H}=\alpha_{A^H}=\alpha_{P^H}=0.25$</td>
<td>$\beta_1 = \beta_2 = 0.5$</td>
</tr>
<tr>
<td>S2</td>
<td>$\alpha_{R^H}=0.4$, $\alpha_{B^H}=0.3$, $\alpha_{A^H}=0.2$, $\alpha_{P^H}=0.1$</td>
<td>$\beta_1 = \beta_2 = 0.5$</td>
</tr>
<tr>
<td>S3</td>
<td>$\alpha_{R^H}=0.4$, $\alpha_{B^H}=0.3$, $\alpha_{A^H}=0.2$, $\alpha_{P^H}=0.1$</td>
<td>$\beta_1 = 0.99$, $\beta_2 = 0.01$</td>
</tr>
</tbody>
</table>

that satisfies its minimum required utility (cf. constraint 23), otherwise this UE will be blocked.

Since more importance in setting S3 is given to maximizing the number of served UEs, Figure 3(c) shows that in this setting the percentage of served UEs with different service classes and contracts has increased compared with the two previous settings. Yet, for a relatively high number of UEs in the network (e.g., ≥ 540), the network operator can deploy small cells or WiFi access points to accommodate the remaining UEs that could not be served in the macro cells.

We note that, in HRP model, all UEs with different service classes and contracts are served, except for a very low percentage of UEs (< 1%) that are blocked for a high number of UEs in the network. This is because, in HRP model, the UE’s demands are not taken into account. Thus, the HRP model accepts UEs and shares equitably the RUs between them as long as the limit on the capacity (in terms of RUS) at the BS/RAT is not exceeded, and the minimum required received power for association is verified. However, in the next results, we show how the solutions of the proposed optimization approach leverage the tradeoff between enhancing the network overall performance while satisfying the UE’s demands and blocking some UEs.

2) User satisfaction: Figure 4 shows the average utility per UE for each service class and contract, for the solutions of the considered settings and for the reference model HRP. For all the solutions, the average utility per UE per service per contract has a decreasing function. This is because, as the number of UEs in the network increases, the UEs will be allocated a lower number of RUs, which lower their achieved throughput and thereby lower their average utility. Moreover, for the solutions of the considered settings, this is also because of the percentage of served UEs, which is a decreasing function of the number of UEs in the network (as shown in Figure 3). The blocked UEs, having a utility equals zero, contribute in reducing on average the utility per UE in the network.

For UEs with class B service and premium contract, Figure 4(a) shows that the optimal solutions of the considered settings S1, S2 and S3 outperform HRP. For a low number of UEs in the network, S1, S2 and S3 solutions have relatively same performance. As the number of UEs in the network increases, the gap between the these three solutions increases, with S2 having the highest average utility per UE, followed by S3 then S1. This is because, UEs with class B service and premium contract are the most more privileged in setting S2 and S3. This increases the number of served UEs with class B service and premium contract for solutions S2 and S3 (as shown in Figure 3), and thereby increases the average utility per UE. Yet, in setting S3, more importance is given to serve UEs than to maximize their utilities. This causes the average utility per UE for solution S3 to be lower than that of solution S2.

Moreover, we noticed that the performance of HRP solution degrades drastically with the increase of the number of UEs in the network. In particular, the average utility per UE with class B service and premium contract degrades from 39% to 0.20%, when the number of UEs in the network increases from 90 to 720. In fact, HRP model treats equally all the UEs in the network. With a high number of UEs in the network, each UE will thus be allocated a relatively low number of RUs. This decreases the achievable throughput for all UEs; decreasing below a certain threshold, the satisfaction of UEs with class B service will drop drastically (as shown in Figure 1). Compared to all other type of UEs, HRP solution gives the lowest average utility per UE with class B service and premium contract.

For UEs with class B service and regular contract, Figure 4(b) shows that the optimal solutions of the considered settings S1, S2 and S3 outperform HRP. Precisely, S1 and S2 solutions have relatively same performance and an average utility per UE greater than S3. This is explained by the fact that more importance in S3 is given to serve UEs than to maximize their utilities.

Figure 4(c) shows the average utility per UE with class A service and premium contract. For relatively low number of UEs in the network, all solutions have the same performance. As the number of UEs in the network increases, S1 shows the highest average utility per UE followed by S2, then S3, and finally HRP. As UEs with class A service and premium contract have low privilege in S2 and S3, their average utilities
are lower than that of S1.

Figure 4(d) shows the average utility per UE with class A service and regular contract. For a number of UEs in the network less than 270, HRP solution has the highest average utility per UE followed by S1, S2, then S3. Between 270 and 540 UEs in the network, S1 has the highest average utility per UE followed by HRP, then S2 and finally S3. For a number of UEs in the network greater than 540, S1 has the highest average utility per UE and S2, S3 and HRP have same performance. In fact, in HRP model, RUS are equitably shared between UEs, thus the lower the number of UEs in the network, the higher the number of RU's allocated to each UE, thus the higher the average utility per UE. Moreover, for the same offered throughput, UEs with class A service and regular contract has the highest utility compared with other UEs (cf. Figure 1). We note that, S2 and S3 have relatively the lowest average utility per UE with class A service and regular contract, because these UEs have the lowest privilege in these settings. Moreover, the gap between these two solutions and S1 solution increases with the increase of UEs in the network. This is due to the decrease in the number of served UEs with class A service and regular contract for solutions S2 and S3 (as shown in Figure 3).

We note that, in our studied simulations, we limited the satisfaction of UEs to 77.70% for class A service and to 95% for class B service. The effect of these limitations (cf. constraints (24)) on the maximum utility can be clearly seen. For instance, the maximum achievable utility per UE with class A service and regular contract is equal to 64% for the three settings (as shown in Figure 4(d)). Moreover, the present results show that lowering the maximum satisfaction for UEs with class A service, plays a role in enhancing the satisfaction of UEs with class B service. Particularly, for 90 UEs in the network, for S3, the average utility per UE equals 64% for UEs with class A service and regular contract (see Figure 4(d)) and it equals 90% for UEs with class B service and premium contract (see Figure 4(a)). However, for HRP solution, the former is 87.34% (see Figure 4(d)) and the latter is 39% (see Figure 4(a)). This reveals the importance of our optimization approach that takes into consideration the UE's requested service and the contract between this UE and the network operator.

In conclusion, by tuning the weighting coefficients, we obtain different points located on the Pareto frontier presenting all the compromises between the satisfaction of UEs with different services and contracts on the one hand, and the two objectives on the other hand. The network operator can thus fine-tune the model to reflect its own decision preferences. For instance, the operator has the choice to privilege the satisfaction of some UEs based on their services and contracts, to give more importance for maximizing the number of served UEs, or to balance the tradeoff between maximizing the number of served UEs and maximizing their utilities.

3) UEs distribution on different RATs: Figure 5 shows the percentage of UEs associated per BS over both RATs, for the solutions of the considered settings and for HRP solution, for the case of 270 UEs in the network. For the solution of the considered settings, UEs are efficiently distributed between the two available RATs. Whereas, for HRP solution, the majority of UEs are associated with RAT 1. For instance, in S3, 66% of UEs are associated with RAT 1 and 33% of UEs are associated with RAT 2. However, in HRP solution, 88% UEs associated are with RAT 1 and 22% are associated with RAT 2. In fact, in HRP solution, UEs are associated with the BS/RAT delivering the highest received power of pilot signals with a prioritization for RAT 1. Thus, covered by both RATs, UEs are more likely associated with the prioritized RAT (which is RAT 1) as long as the minimum required level of received power is verified. This causes the rush on RAT 1, and thereby reduces the average utility per UE, specially for a relatively high number of UEs in the network.

D. Discussion

In this study, we assume the existence of a central entity (CE) that has a complete control of the network state and elements (such as UEs and BSs/RATs). This entity can be easily introduced to the current wireless access networks [16]. The CE senses the network state information such as radio channel conditions, QoS demand of UEs, etc. After collecting the necessary information (e.g., using the IEEE 1900.4 standard [17]), the CE intervenes at regular time intervals (periodic
intervention) and provides the optimal solution of the proposed optimization problem. Precisely, the CE guides the UEs to be associated with the appropriate BS/RAT. Moreover, it diffuses to the network BSs/RATs their optimal RUs allocation.

In fact, the computation time of the optimal solution has a great impact on the periodic intervention of the CE. In the present simulations, the optimal solutions of the different settings are obtained using CPLEX solver. All the solutions were provided for a gap-to-optimality equals zero. The gap-to-optimality metric expresses the gap between the obtained integer solution and the optimal solution estimated by the solver. We noticed that the proposed optimization approach has a reasonable computation time. Particularly, the computation time of the optimal solutions varies on average between 1s and 26s depending on the number of UEs in the network and the considered setting. Therefore, the present study brings a value in providing an optimal solution of the challenging joint problem of user association and resource allocation in heterogeneous networks that has low computational complexity for a realistic network scenario.

We now investigate the impact of varying the gap-to-optimality metric on the computation time of the obtained solution. We consider the solution of setting S3 where we have 180 UEs in the network. Table VI shows that the mean computation time of setting S3 solution decreases as the gap-to-optimality increases. In particular, for a mean gap to optimality equals 0%, the mean computation time equals 17.86 s, and with an increase of the gap to optimality to 4.13%, the mean computation time decreases to 1.68 s. Therefore, a network operator has the option to choose the operation point of the network. For instance, the operator can choose the gap-to-optimality, that provides a near-optimal solution within a very short time. This also helps in reducing the interval time of the periodic intervention of the CE. In future work, more advanced techniques in the optimization process will be applied to further reduce the computation time of the optimal solution.

Table VI

<table>
<thead>
<tr>
<th>Gap-to-optimality</th>
<th>Mean gap-to-optimality [%]</th>
<th>Mean computation time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.92</td>
<td>0.26</td>
</tr>
<tr>
<td>1</td>
<td>1.62</td>
<td>1.95</td>
</tr>
<tr>
<td>2</td>
<td>4.13</td>
<td>1.68</td>
</tr>
</tbody>
</table>

V. CONCLUSION

In this paper, we proposed a hybrid approach with the aim of optimizing not only the overall network performance but also the UE satisfaction regarding its QoS demand and contract. We thus formulate an optimization problem considering double objectives, namely, maximizing the number of served UEs and maximizing the sum of the UE utilities. Starting from a mixed integer non-linear formulation of the problem, we provide a MILP reformulation of the problem that makes it computationally tractable. Different settings reflecting various preferences were carried out by tuning on the one hand, the weights associated with different services and contracts; on the other hand, the weights associated with the considered two objectives. The optimal solution of the MILP problem is computed for a realistic network scenario and compared with legacy solution. The extensive simulation results show that the proposed approach outperforms legacy solution in terms of user satisfaction. For instance, when adequately tuned, for a relatively high number of UEs in the network, the proposed approach offers for UEs with real-time service (class B) and premium contract an average utility of 50%, whereas the existing solution can not offer them more than 0.20% of satisfaction. Moreover, it leverages the tradeoff between distributing UEs on the different RATs and blocking some others, in a way that enhances the overall performance of the network. Furthermore, the present study brings a value in providing an optimal solution of this challenging problem that has a low computational complexity for realistic network scenarios. For future work, we plan to study the dynamics of the network. In particular, we need to take into consideration the mobility of UEs, the arrival and departure of the UEs in the network. Moreover, we plan to examine heuristic algorithms to solve this challenging problem.

REFERENCES

[14] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation v8.4.0. TS 36.211, 3rd Generation Partnership Project (3GPP), 2008.
Joint Resource Allocation and User Association for Heterogeneous Cloud Radio Access Networks

Ying Loong Lee*, Li-Chun Wang†, Teong Chee Chuah‡, Jonathan Loo†

*Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Selangor, Malaysia
Email: lee.ying.loong12@student.mmu.edu.my, tcchuah@mmu.edu.my
†Department of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu 300, Taiwan R.O.C.
Email: lichun@g2.nctu.edu.tw
‡School of Science and Technology, Middlesex University, London NW4 4BT, United Kingdom
Email: J.Loo@mdx.ac.uk

Abstract—Cloud radio access networks (C-RANs) have been regarded as a promising architecture for energy-efficient fifth generation systems. In this paper, a new joint remote radio head (RRH) activation, user-RRH pairing and resource allocation strategy is proposed for heterogeneous C-RANs (H-CRANs). We first formulate an optimization problem to maximize the energy efficiency of H-CRANs. Then, a low-complexity suboptimal solution is developed. Our proposed mechanism consists of three key procedures: 1) RRH activation is performed based on greedy RRH selection; 2) user-RRH pairing is performed based on the channel quality; 3) the resource allocation problem is solved by dual decomposition. Simulation results show that the proposed strategy can improve energy efficiency significantly.

Keywords—H-CRAN; RRH activation; user association; resource allocation; energy efficiency

I. INTRODUCTION

The fifth generation (5G) mobile cellular systems are expected to provide gigabit data rates to mobile users for broadband applications, which requires much higher capacity compared to the fourth generation (4G) systems. However, achieving such a high capacity requires high power consumption by both base stations and user equipment for the downlink and uplink, respectively. Thus, energy efficiency becomes the key design goal for 5G systems.

To meet the energy efficiency requirement, a new architecture known as cloud radio access network (C-RAN) has been proposed [1]. A C-RAN consists of a set of remote radio heads (RRHs) and a centralized baseband unit (BBU) pool. These RRHs are connected to the BBU pool via the fronthaul links. The RRHs generally serve as radio frequency (RF) transmitters and receivers, which only performs basic RF functionalities, whereas the BBU pool performs baseband signal processing and upper-layer functionalities [2], [3]. When the C-RAN co-exists with the macrocell base station (MBS), it is called heterogeneous C-RAN (H-CRAN) as shown in Fig. 1. The main feature of the H-CRAN is that control signals are transmitted by the MBS to the mobile users, which can facilitate the mobility management of small cell networks.

In the literature, a number of related studies for C-RANs [4]–[11] have been carried out. In [4], the power consumption of C-RANs was minimized via RRH activation and beamforming. A power allocation scheme for multiple input multiple output (MIMO)-based C-RANs was proposed in [5] to maximize energy efficiency. In [6], power allocation and beamforming are jointly investigated for C-RANs. Subchannel allocation is further considered in [7], together with power allocation and beamforming were jointly studied to improve the energy efficiency of C-RANs. In [8], the authors proposed a cross-layer resource allocation, RRH activation and beamforming scheme for C-RANs to minimize power consumption. Optimal resource allocation for C-RANs was studied in [9] under a coordinated multipoint transmission (CoMP) framework with power consumption and fronthaul capacity constraints. RRH activation and user association for improving energy efficiency of C-RANs was investigated in [10]. In [11], a subchannel and power allocation scheme was proposed for maximizing the energy efficiency of H-CRANs.

To the best of our knowledge, joint consideration of RRH activation, user association, and subchannel and power allocation for improving energy efficiency of H-CRANs is not available in the literature. Most of the related studies [4]–[8], [10], [11] have not considered the constraints of the limited capacity of fronthaul links. In fact, high-capacity fronthaul links such as optical fiber may incur high deployment costs especially in ultra-dense small cells. Also, energy efficiency is not investigated in [4], [6], [8] and [9]. In the literature, load-dependent...
fronthaul power consumption models have not been considered in investigating the energy efficiency performance of H-CRANs. Therefore, we are motivated to study a joint RRH activation, user-RRH pairing and resource allocation to improve the energy efficiency of H-CRANs under constrained fronthaul capacity based on a load-dependent fronthaul power consumption model.

In this paper, we develop a comprehensive mechanism for RRH on/off mechanism, user selection\(^1\) and radio resource allocation for H-CRANs. We focus on the downlink of an H-CRAN whereby the power is mainly consumed by the network infrastructure (e.g. RRHs, BBU pools and fronthaul links). The proposed strategy is formulated as an optimization problem that maximizes energy efficiency of an H-CRAN subject to the limited fronthaul capacity constraint. The problem is then transformed and solved using an iterative algorithm in which RRH activation is performed based on greedy selection and user-RRH pairing is performed based on channel quality. The resource allocation problem is solved by dual decomposition. The proposed strategy is evaluated and compared with several baseline schemes in terms of energy efficiency.

The remainder of this paper is organized as follows. Section II introduces the system model and presents the problem formulation. The solution algorithm is proposed in Section III. In Section V, performance evaluation of the proposed strategy is presented. Finally, Section VI concludes this paper.

II. SYSTEM MODEL AND PROBLEM FORMULATION

Fig. 1 shows the system model of an H-CRAN [3] consisting of a BBU pool connected to an MBS and a set of RRHs (small cells). In the considered architecture, an MBS transmits control signaling to CRAN, whereas data transmission is performed by the RRHs. Denote \(S\), \(U\) and \(K\) as the sets of RRHs, mobile users and subchannels, respectively. Further, we define \(a_s\) as the activation indicator of RRH \(s\), whereby \(a_s = 1\) indicates that RRH \(s\) is activated and \(a_s = 0\) when it is deactivated.

Also, \(b_{su}\) is defined as the pairing indicator of RRH \(s\) with user \(u\) whereby \(b_{su} = 1\) if user \(u\) is paired with RRH \(s\); otherwise \(b_{su} = 0\). For resource allocation among users associated with each RRH, let \(\omega_{ku}\) be the assignment indicator of subchannel \(k\) to user \(u\) whereby \(\omega_{ku} = 1\) if subchannel \(k\) is allocated to user \(u\); otherwise \(\omega_{ku} = 0\). Also, \(p_{sk}\) is defined as the nonnegative transmission power of RRH \(s\) on subchannel \(k\). To ease analysis, the following assumptions are made: 1) The network is perfectly synchronized; 2) All RRHs share the entire channel bandwidth available; 3) Each subchannel experiences flat and slow fading.

The power consumption model of an H-CRAN basically consists of the power consumed by the RRHs, the fronthaul links, the backhaul link and the BBU pool\(^2\). The power consumption of each RRH is given as [12]:

\[
P_{RRH,s} = a_s \left( P_{0,s} + \eta_s \sum_{u \in U} b_{su} \sum_{k \in K} \omega_{ku} p_{sk} \right) + (1 - a_s) P_{sleep,s},
\]

where \(P_{0,s}\) is the static power consumption of RRH \(s\) when it is activated, \(P_{sleep,s}\) is the total power consumed by RRH \(s\) when it is deactivated and \(\eta_s\) is the slope of the load-dependent power consumption of RRH \(s\).

We adopt a power consumption model in [13] whereby the total power consumed by fronthaul links are proportional to the network traffic carried to their associated RRHs. The power consumption model of a fronthaul link can be expressed as:

\[
P_{fh,s} = a_s \left( P_{c,s} + \beta_s R_s \right).
\]

In (2), \(P_{c,s}\) is the constant power consumption of a fronthaul link which is given as \(P_{c,s} = P_{fh,s} + \frac{\tau_{s} P_{sw,s}}{\eta_{sw,s}}\) [14]. For RRH \(s\), \(P_{fh,s}, \tau_s, P_{sw,s}\) and \(\eta_{sw,s}\) are the power consumed by the fronthaul transceiver, the percentage of the load-independent power consumption of the fronthaul aggregation switch, the maximum power consumption of the switch and the number of ports of the switch, respectively. \(R_s\) is the total network traffic of the fronthaul link associated with RRH \(s\) and \(\beta_s\) is the power consumed per bit/s by a fronthaul link which is written as \(\beta_s = \frac{1}{\eta_{sw,s}} \frac{P_{sw,s}}{\tau_s}\) [14], where \(R_{fh,s}\) is the maximum traffic load that can be carried by the switch in the fronthaul link associated with RRH \(s\), that is, the fronthaul capacity of RRH \(s\).

The power consumption model in [13] can also be adopted for the backhaul link between the BBU pool and the MBS (cf. Fig. 1). However, since the backhaul link only carries control signaling between the BBU pool and the MBS, we assume that the control traffic constantly consumes a fixed amount of the backhaul bandwidth. Thus, the power consumed by the backhaul link can be assumed to be constant.

We express the total power consumption of the H-CRAN as:

\[
P = \sum_{s \in S} (P_{RRH,s} + P_{fh,s}) + P_{bh}.
\]

The rate utility function of the H-CRAN is written as:

\[
R = \sum_{s \in S} \alpha_s \sum_{u \in U} b_{su} w_u \sum_{k \in K} \omega_{ku} R_{sku},
\]

where \(w_u\) is the weighting coefficient corresponding to the data rate achievable by user \(u\) which can be adjusted to achieve different notions of fairness [15]. \(R_{sku}\), the

\(^1\)In this paper, the terms ‘user selection’, ‘user association’ and ‘user-RRH pairing’ are interchangeable.

\(^2\)We exclude the power consumption of the MBS since the MBS does not take part in data transmission under our system model. Nonetheless, it can easily be included into our power consumption model as the MBS is always activated for control signaling, thereby incurring static power consumption.
data rate achievable by user $u$ associated with RRH $s$ on subchannel $k$, is expressed as:

$$R_{sku} = B \log_2 \left(1 + \frac{p_{sk}g_{sku}}{\sum_{i \in S \setminus \{s\}} a_i p_{ik} g_{iku} + N_0}\right),$$  

(5)

where $B$ is the bandwidth of a subchannel, $g_{sku}$ is the downlink channel gain between RRH $s$ and user $u$ on subchannel $k$, and $N_0$ is the additive white Gaussian noise (AWGN) power.

The utility function that corresponds to the energy efficiency of the H-CRAN is defined as:

$$U_{EE} = \frac{R}{P},$$  

(6)

It is noteworthy that energy efficiency is defined as a ratio of the total transmission rate to total power consumption. Thus, by setting $w_u = 1$ for all $u \in U$, (4) becomes the total transmission rate of the H-CRAN and (6) is equivalent to the energy efficiency of the H-CRAN.

The main objective of this paper is to maximize (6). Thus, the joint RRH activation, user-RRH pairing and resource allocation problem for an H-CRAN can be formulated as follows:

$$\max_{a,b,\omega,p} U_{EE}(a,b,\omega,p) = \frac{R(a,b,\omega,p)}{P(a,b,\omega,p)},$$  

(7)

subject to:

$$a_s \sum_{u \in U} b_{su} \sum_{k \in K} \omega_{ku} p_{sk} \leq P_{max,s} \ \forall s \in S,$$  

(7a)

$$a_s \sum_{u \in U} b_{su} \sum_{k \in K} \omega_{ku} R_{sku} \leq R_{th,s} \ \forall s \in S,$$  

(7b)

$$a_s b_{su} \sum_{k \in K} \omega_{ku} R_{sku} \geq R_{min,u} \ \forall s \in S, u \in U,$$  

(7c)

$$\sum_{s \in S} a_s b_{su} = 1 \ \forall u \in U,$$  

(7d)

$$a_s \sum_{u \in U} b_{su} \omega_{ku} \leq 1 \ \forall s \in S, k \in K,$$  

(7e)

$$p_{sk} \geq 0 \ \forall s \in S, k \in K,$$  

(7f)

where $a = [a_1, \ldots, a_{|S|}]$, $b = [b_{11}, \ldots, b_{|S| \times |U|}]$, $\omega = [\omega_{11}, \ldots, \omega_{|S| \times |U|}]$, and $p = [p_{11}, \ldots, p_{|S| \times |K|}]$. Constraint (7a) ensures that the total transmission power of each RRH $s$ does not exceed the maximum allowable transmission power, $P_{max,s}$. Constraint (7b) is the fronthaul capacity constraint whereby the total transmission rate of each RRH $s$ must not exceed the fronthaul capacity, $R_{th,s}$. Each user $u$ is ensured in constraint (7c) that its minimum bit rate, $R_{min,u}$, is achieved. In constraint (7d), each user is ensured to only associate with one RRH. Constraint (7e) ensures that no two or more users associated with the same RRH will receive the same subchannels. Constraint (7f) is a nonnegative transmission power constraint. In fact, (7) is a nonconvex mixed-integer programming problem, which is generally NP-hard. To solve (7), we show in the next section that it can be solved efficiently using an iterative greedy algorithm.

### III. Solution Algorithm

In this paper, we propose a low-complexity suboptimal iterative greedy algorithm, as depicted in Algorithm 1, to efficiently solve (7). In this algorithm, RRH activation is performed based on the greedy approach similar to that in [10] which deactivates the RRH that has the least contribution to the total energy efficiency iteratively; user-RRH pairing is performed based on signal-to-interference-plus-noise ratios (SINRs); subchannel and power allocation is performed based on dual decomposition.

**Algorithm 1** Greedy RRH activation and SINR-based user-RRH pairing

1. Set $a_s = 1$ for all $s \in S$, $U_{old} = 0$ and $S_a = S$.
2. Set $b_{su} = 1$ for all $s \in S_a$ and $u \in U$ such that the received wideband SINR from RRH $s$ to user $u$ is the highest.
3. Solve (7) for $\omega$ and $p$ (See Algorithm 2).
4. Calculate $P$, $R$ and $U_{EE}$ using (3), (4) and (6), respectively.
5. while $U_{EE} > U_{old}$ do
6. Set $U_{old} = U_{EE}$.
7. Evaluate the throughput of RRH $s$, i.e. $R_s$ and calculate $\frac{R_s}{P}$ for all $s \in S_a$.
8. Deactivate RRH $s$ such that its corresponding $\frac{R_s}{P}$ is the smallest among all RRHs. Then, set $a_s = 0$ and $S_a = S_a \setminus \{s\}$.
9. Set $b_{su} = 1$ for all $s \in S_a$ and $u \in U$ such that the received wideband SINR from RRH $s$ to user $u$ is the highest.
10. Solve (7) for $\omega$ and $p$ (See Algorithm 2).
11. Calculate $P$, $R$ and $U_{EE}$ using (3), (4) and (6), respectively.
12. end while

In Steps 1-4 of Algorithm 1, all RRHs are assumed to be activated, i.e., $a_s = 1$ for all $s \in S$ and the set of active RRHs, $S_a$ includes all RRHs. Each user $u$ is paired with RRH $s$ with the largest received wideband SINR. The wideband SINR received by each user from each RRH can be estimated as:

$$\Gamma_{su} = \sum_{i \in S_a \setminus \{s\}} \frac{p_i g_{iu}}{R_{iu} + N_0} \quad \forall s \in S, u \in U,$$  

(8)

where $p_i$ is the total power transmitted by RRH $s$ and $g_{iu}$ is the average channel gain received by user $u$ from RRH $s$ on the entire channel bandwidth. Thus, $b_{su} = 1$ if RRH $s$ provides the largest wideband SINR to user $u$. This SINR-based user association ensures that the channel quality experienced by the users is higher. Thus, the users can achieve a higher throughput, hence possibly a higher energy efficiency performance. Then, we can proceed to solve (7) for $\omega$ and $p$. After that, we can calculate the total power consumption $P$, the weighted sum rate $R$ and the energy efficiency $U_{EE}$ using (3), (4) and (6), respectively.

Next, in Steps 5-11 of Algorithm 1, we first set $U_{old} = U_{EE}$. Then, we evaluate the throughput of each RRH and evaluate $\frac{R_s}{P}$ for all $s \in S_a$ where $R_s = a_s \sum_{u \in U} b_{su} \sum_{k \in K} R_{sku}$. Then, RRH $s$ is deactivated such that it corresponds to the smallest $\frac{R_s}{P}$ among all RRHs in $S_a$. The key idea of this method is to deactivate the RRHs that contribute the least to the energy efficiency.
because their achievable throughput is very low and unlikely to improve the energy efficiency if other RRHs are deactivated instead. Again, the SINR-based user-RRH pairing is performed and the corresponding $P$, $R$ and $U_{EE}$ are calculated. If the new $U_{EE}$ is larger than $U_{old}$, Steps 5-11 are repeated until this condition does not hold. In this way, the RRHs that contribute the least to the overall energy efficiency will be deactivated if only the energy efficiency is improved.

To solve (7) in Steps 3 and 10 of Algorithm 1, we first rewrite (7) assuming that RRH activation and user-RRH efficiency is improved.

$$\max_{\omega, p} U_{EE}(\omega, p) = \frac{R(\omega, p)}{P(\omega, p)}.$$  \hspace{1cm} (9)

subject to:

\begin{align*}
\sum_{u \in U_{s}, k \in K} \omega_{su} p_{sk} & \leq P_{\max, s} \forall s \in S_{s} \hspace{1cm} (9a) \\
\sum_{u \in U_{s}, k \in K} \omega_{su} R_{sku} & \leq R_{\text{th}, s} \forall s \in S_{s} \hspace{1cm} (9b) \\
\sum_{k \in K} \omega_{su} R_{sku} & \geq R_{\text{min}, u} \forall s \in S_{s}, u \in U_{s} \hspace{1cm} (9c) \\
\sum_{u \in U_{s}, k \in K} \omega_{su} & \leq 1 \forall s \in S_{s}, k \in K, \hspace{1cm} (9d) \\
p_{sk} & \geq 0 \forall s \in S_{s}, k \in K, \hspace{1cm} (9e)
\end{align*}

where $S_{s}$ is the set of activated RRHs, $U_{s}$ is the set of users associated with RRH $s$. It is noted in (12) that $R_{sku} = B \log_{2} \left( 1 + \frac{\sum_{k \in K} \eta_{u} p_{sk} + \beta_{s} R_{sku}}{\sum_{k \in K} \sum_{u \in U_{s}} \omega_{su} p_{sk}} \right)$ which is equivalent to (5) and $R_{s} = \sum_{s \in S} \sum_{u \in U_{s}} \sum_{k \in K} R_{sku}$.

To solve (9), the nonlinear fractional programming approach in [16] is used. Without loss of generality, we let $\psi = R(\omega, p)$ and $\psi^{*} = \max_{\omega, p} \frac{R(\omega, p)}{P(\omega, p)} = \frac{R^{*}(\omega, p^{*})}{P^{*}(\omega, p^{*})}$ where $\{\omega^{*}, p^{*}\}$ is the optimal solution to (9). Then, we obtain the following theorem.

**Theorem 1.** $\psi^{*}$ is achieved if and only if

$$\max_{\omega, p} R(\omega, p) - \psi^{*} P(\omega, p)$$

$$= R(\omega^{*}, p^{*}) - \psi^{*} P(\omega^{*}, p^{*}) = 0,$$ \hspace{1cm} (10)

where $R(\omega, p) \geq 0$ and $P(\omega, p) > 0$.

**Proof:** Refer to [16] for a similar proof. \hfill \blacksquare

By Theorem 1, (9) can be equivalently expressed as:

$$\max_{\omega, p} f(\omega, p) = R(\omega, p) - \psi^{*} P(\omega, p),$$ \hspace{1cm} (11)

subject to (9a)-(9e). However, $\psi^{*}$ has to be found for (11). As such, we employ Dinkelbach’s method in [16] and design an iterative algorithm to solve (11). The iterative algorithm is summarized in Algorithm 2.

In Algorithm 1, the outer loop updates $\psi$ in each iteration $\{\omega, p\}$ obtained from the previous iteration until the convergence is achieved, i.e., $R(\omega, p) - \psi P(\omega, p) < \epsilon$ where $\epsilon$ is a very small positive value. In the inner loop, the following problem is solved:

$$\max_{\omega, p} f(\omega, p) = R(\omega, p) - \psi P(\omega, p),$$ \hspace{1cm} (12)

subject to (9a)-(9e). In the following theorem, we show that Algorithm 2 always converges to the optimal solution to (12).

**Theorem 2.** The solution obtained from Algorithm 2 always converges to the global optimal solution to (12).

**Proof:** Refer to [16] for a similar proof.

In the inner loop of Algorithm 2, the problem in (12) can be solved by dual decomposition. Firstly, the Lagrangian function of (12) can be written as follows:

$$L(\omega, p, \lambda, \phi, \alpha) = \sum_{s \in S_{s}, u \in U_{s}} w_{u} \omega_{su} R_{sku}$$

$$- \psi \sum_{s \in S_{s}, u \in U_{s}} \sum_{k \in K} \omega_{su} \left( \eta_{u} p_{sk} + \beta_{s} R_{sku} \right)$$

$$- \psi \sum_{s \in S_{s}} \sum_{k \in K} \omega_{su} p_{sk}$$

$$+ \sum_{s \in S_{s}} \lambda_{s} \left( P_{\max, s} - \sum_{u \in U_{s}} \sum_{k \in K} \omega_{su} p_{sk} \right)$$

$$+ \sum_{s \in S_{s}} \phi_{s} \left( R_{\text{th}, s} - \sum_{u \in U_{s}} \sum_{k \in K} \omega_{su} R_{sku} \right)$$

$$+ \sum_{s \in S_{s}} \sum_{u \in U_{s}} \sum_{k \in K} \omega_{su} R_{sku} - R_{\text{min}, u} \right).$$ \hspace{1cm} (13)

where $\lambda = \{\lambda_{1}, \ldots, \lambda_{|S_{s}|}\}$, $\phi = \{\phi_{1}, \ldots, \phi_{|S_{s}|}\}$ and $\alpha = \{\alpha_{1}, \ldots, \alpha_{|S_{s}|, |U_{s}|}\}$ are vectors of nonnegative Lagrange multipliers corresponding to constraints (9a), (9b) and (9c) respectively.

The Lagrangian dual function of (13) can be expressed as

$$D(\lambda, \phi, \alpha) = \max_{\omega, p} L(\omega, p, \lambda, \phi, \alpha).$$

The dual optimization problem can thus be formulated as follows:

$$\min_{\lambda, \phi, \alpha} D(\lambda, \phi, \alpha),$$ \hspace{1cm} (14)

subject to $\lambda_{s} \geq 0$ and $\phi_{s} \geq 0$ for all $s \in S_{s}$, and $\alpha_{su} \geq 0$ for all $s \in S_{s}$ and $u \in U_{s}$.

In this way, the convex problem in (14) can be solved using convex optimization techniques. However, the solution to a dual problem only gives the upper bound of the primal problem if the latter is not convex. Therefore, there may exist a nonzero duality gap between (14) and...
Theorem 3. The duality gap between (14) and (12) approaches zero if $|K|$ is sufficiently large.

Proof: The proof is similar to those in [11], [17].

By Theorem 3, the solution to (14) will approximate that to (12) if the number of subchannels is sufficiently large. As such, we can solve (14) by dual decomposition. The solution approach is similar to that in [18]. Firstly, assuming that the equal transmission power has been allocated by each RRH $s$ on each subchannel $k$, then optimal subchannel allocation can be performed as follows:

$$\omega_{ku} = \begin{cases} u^* = \arg \max_{u \in U_s} m_{sku} & \forall s \in S_s, k \in K, \\ 0 & \text{otherwise} \end{cases}$$

where $m_{sku} = (w_u - \psi \beta_s + \phi_s + \alpha_{su}) R_{sku} - (\psi \eta + \lambda_s)p_{sk}$.

Proof: See Appendix A for the derivation of (15).

Let $q(s,k)$ indicates the user associated with RRH $s$ that is allocated subchannel $k$, i.e., $q(s,k) = u \in U_s$, whereby $\omega_{ku} = 1$, notation $\omega_{ku}$ can be removed and (13) can be rewritten as:

$$\mathcal{L}(p, \lambda, \phi, \alpha) = \sum_{s \in S_s} \sum_{k \in K} w_{hq}(s,k) R_{skq}(s,k)$$

$$- \psi \sum_{s \in S_s} \left( P_{b,s} + \sum_{k \in K} (\eta_s p_{sk} + \beta_s R_{skq}(s,k)) \right)$$

$$+ \sum_{s \in S_s} \lambda_s \left( P_{max,s} - \sum_{k \in K} p_{sk} \right)$$

$$+ \sum_{s \in S_s} \phi_s \left( R_{th,s} - \sum_{k \in K} R_{skq}(s,k) \right)$$

$$+ \sum_{s \in S_s} \alpha_{su} R_{skq}(s,k) - \sum_{s \in S_s} \sum_{u \in U} \alpha_{su} R_{min,u}.$$

Using the Karush-Kuhn-Tucker (KKT) conditions [19], optimal power allocation can be derived (See (17) at the top of the next page). Note that $[x]^+$ is equivalent to $\max(0,x)$. With (15) and (17), the dual problem in (14) can be solved iteratively using a subgradient method [20] where the Lagrange multipliers are iteratively updated as follows:

$$\lambda_{s}^{(t+1)} = \lambda_{s}^{(t)} - \delta_1 \left( p_{max,s} - \sum_{u \in U_s, k \in K} \omega_{ku} p_{sk} \right) \forall s \in S_s.$$

$$\phi_{s}^{(t+1)} = \phi_{s}^{(t)} - \delta_2 \left( R_{th,s} - \sum_{u \in U_s, k \in K} \omega_{ku} R_{sku} \right) \forall s \in S_s,$$

$$\alpha_{su}^{(t+1)} = \alpha_{su}^{(t)} - \delta_3 \left( \sum_{k \in K} \omega_{ku} R_{sku} - R_{min,u} \right) \forall s \in S_s, u \in U_s,$$

where $\delta_1$, $\delta_2$ and $\delta_3$ are positive step sizes corresponding to (18), (19) and (20), respectively, which satisfy infinite travel conditions [20]; $\lambda_{s}^{(t)}$, $\phi_{s}^{(t)}$ and $\alpha_{su}^{(t)}$ are the respective $\lambda_s$, $\phi_s$ and $\alpha_{su}$ at the $t$-th iteration. For each update of the Lagrange multipliers, the subchannel and power allocation are recomputed again using (15) and (17). The process is repeated in the inner loop of Algorithm 2 until convergence or the predetermined maximum number of iterations, $T_{max}$, is reached.

IV. COMPLEXITY ANALYSIS

In Algorithm 2, the maximum number of iterations required to solve (9) is $I_{max} T_{max}$ where $I_{max}$ is the maximum number of iterations for the outer loop and $T_{max}$ is the maximum number of iterations for the inner loop. Therefore, the maximum number of iterations for Algorithm 1 can be estimated as $|S| I_{max} T_{max}$. Hence, the asymptotic complexity of our proposed algorithm is of $O(|S| I_{max} T_{max})$. The proposed scheme is intended to be executed by the BBU pool periodically in order to keep up with the channel variations.

V. RESULTS AND DISCUSSION

A single-cell H-CRAN network, which consists of an MBS with a macrocell radius of 500 m and six picocell RRHs, is considered. The RRHs are randomly distributed within the macrocell since small cells are deployed at random locations. We set the number of subchannels to 100 with each having a bandwidth of 180 kHz, following the 3GPP specifications [21]. Here, we assume that all fronthaul links are identical, therefore $R_{th,s} = R_{th}$, $P_{b,h} = P_{b}$, $P_{sw,s} = P_{sw}$, $\tau_s = \tau$ and $n_{port,s} = n_{port}$ for all $s \in S$. The power parameters of the RRHs are set according to the power consumption of the picocells as follows: $P_{b,h,s} = 30$ dBm, $P_{b} = 0.8$ W, $P_{sw} = 4.3$ W and $n_{port} = 4$ [12]. For the fronthaul links, we set the following parameters as in [14]: $P_{f} = 3$ W, $P_{f} = 300$ W, $\tau = 0.8$ and $n_{port} = 24$. The power consumed by the backhaul link is assumed to be 13.25 W. For channel modeling, we consider Rayleigh fading, which are independently and identically distributed (i.i.d.) with zero mean and unit variance. We also follow the 3GPP specifications [22] by considering log-normal shadowing which is also i.i.d. with zero mean and a standard deviation of 10 dB, and the small cell path loss model: $140.7 + 36.7 \log d$ where $d$ is the distance between the RRH and the user in km. The noise power spectral density and noise figure are set to -174 dBm/Hz and 9 dB [22] respectively. All the users are uniformly distributed within the network, which is a practical user distribution. For the proposed scheme, $\delta_1$.

$^3$This value is calculated using the same power consumption model as the fronthaul link. The relevant parameters are set similar to those of the fronthaul link, except that the backhaul capacity and the control traffic carried in the backhaul link are assumed to be 100 Mb/s and 10 Mb/s, respectively.
\[ p_{sk} = \left[ B(w_{u}(s,k) - \psi)\beta_s - \phi_s + \alpha_{sk}(s,k) \right] \left( \lambda_s + \psi \eta_s \right) \ln 2 - \frac{\sum_{\mathcal{S} \setminus \{s\}} p_{sk} g_{skq}(s,k) + N_0}{g_{skq}(s,k)} \] \quad \forall s \in \mathcal{S}_a, k \in \mathcal{K}. \quad (17)

\[ \delta_2 \text{ and } \delta_3 \text{ are set following the square summable but not summable rule [20] with } \delta_1 = \delta_2 = \delta_3 = 0.001 \text{ at the first iteration.} \]

Fig. 2 shows the energy efficiency achieved by the proposed scheme for the H-CRAN with different fronthaul capacities. It is observed that the energy efficiency improves with the fronthaul capacity as the latter approaches the capacity that can be supported by an RRH on the wireless channel.

Next, we compare the proposed scheme with the following several baseline schemes.

- Full activation: All RRHs are activated and energy efficiency of the H-CRAN is optimized via resource allocation, which similar to that in [11] except that fronthaul capacity constraints are considered.

- Sequential activation: RRHs are deactivated one by one following an ascending index order. Deactivation halts when the energy efficiency cannot be further improved.

Fig. 3 shows the energy efficiency performance of the H-CRAN with 100 Mb/s fronthaul capacity. The proposed scheme outperforms the baseline schemes with average energy efficiency gains of 4.16% and 18.2% over the sequential deactivation and full activation schemes, respectively. The full activation and sequential deactivation schemes do not find out which RRHs are the most suitable to be deactivated, unlike our proposed scheme, thus resulting in inferior energy efficiency performance.

VI. CONCLUSION

In this paper, we have proposed a joint RRH activation, user-RRH pairing and resource allocation scheme for maximizing the energy efficiency of H-CRANs. We have formulated an optimization problem that maximizes energy efficiency of an H-CRAN subjecting to the limited fronthaul capacity. We have designed an iterative algorithm that performs greedy RRH activation, SINR-based user-RRH pairing and solved the resource allocation problem by dual decomposition. Simulation results have demonstrated that the proposed scheme provides an average energy efficiency gain of 4.16%–18.2% compared to the baseline schemes. In future, we will consider a more comprehensive resource allocation model that involves macrocell and RRH users.

APPENDIX A.
DERIVATION OF (15)

Let (13) be rewritten as follows:

\[ \mathcal{L}(\mathbf{p}, \omega, \lambda, \phi, \alpha) = \sum_{s \in \mathcal{S}_a} \sum_{k \in \mathcal{K}} \sum_{u \in \mathcal{U}} L_{sku}(p_{sk}, \omega_{ku}, \lambda_s, \phi_s, \alpha_{su}) \]

\[ - \psi \left( \sum_{s \in \mathcal{S}_a} (P_{0,s} + P_{c,s}) + \sum_{s \in \mathcal{S}_a} P_{\text{sleep},s} + P_{bh} \right) \]

\[ + \lambda_s P_{\text{max},s} + \sum_{s \in \mathcal{S}_a} \phi_s R_{bh,s} - \sum_{s \in \mathcal{S}_a} \sum_{u \in \mathcal{U}_s} \alpha_{su} R_{\text{min},u} \]

where

\[ L_{sku}(\omega_{ku}, p_{sk}, \lambda_s, \phi_s, \alpha_{su}) = \omega_{ku} \left( (w_u - \psi)\beta_s - \phi_s + \alpha_{su} \right) R_{sku} - (\psi \eta + \lambda_s) p_{sk} \]

Given the values of the Lagrange multipliers and that power allocation has been performed, then, for each RRH \( s \) and each subchannel \( k \), the user associated with RRH \( s \) that gives the largest value of \( L_{sku}(\omega_{ku} = \)
user $u^*$ will be allocated subchannel $k$:  

$$u^* = \arg \max_{u \in U_k} L_{sku} (\omega_{ku} = 1, p_{sk}, \lambda_s, \phi_s)$$  

$$= \arg \max_{u \in U_k} \left( (u_w - \psi \beta_k - \phi_s + \alpha_{su}) R_{sku} - (\psi \eta + \lambda_s) p_{sk} \right)$$  

\( \forall s \in S, k \in K \)

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for the constructive comments and suggestions to further improve the quality of the paper.

REFERENCES


Performance-Oriented Association in Large Cellular Networks with Technology Diversity

Abishek Sankararaman  
Dept. of ECE, Univ. of Texas at Austin

Jeong-woo Cho  
School of ICT, KTH, Sweden

François Baccelli  
Dept. of ECE, Univ. of Texas at Austin

Abstract—The development of mobile virtual network operators, where multiple wireless technologies (e.g. 3G and 4G) or operators with non-overlapping bandwidths are pooled and shared is expected to provide enhanced service with broader coverage, without incurring additional infrastructure cost. However, their emergence poses an unsolved question on how to harness such a technology and bandwidth diversity. This paper addresses one of the simplest questions in this class, namely, the issue of associating each mobile to one of those bandwidths. Intriguingly, this association issue is intrinsically distinct from those in traditional networks. We first propose a generic stochastic geometry model lending itself to analyzing a wide class of association policies exploiting various information on the network topology, e.g. received pilot powers and fading values. This model firstly paves the way for tailoring and designing an optimal association scheme to maximize any performance metric of interest (such as the probability of coverage) subject to the information known about the network. In this class of optimal association, we prove a result that the performance improves as the information known about the network increases. Secondly, this model is used to quantify the performance of any arbitrary association policy and not just the optimal association policy. We propose a simple policy called the Max-Ratio which is not-parametric, i.e. it dispenses with the statistical knowledge of base station deployments commonly used in stochastic geometry models. We also prove that this simple policy is optimal in a certain limiting regime of the wireless environment. Our analytical results are combined with simulations to compare these policies with basic schemes, which provide insights into (i) a practical compromise between performance gain and cost of estimating information and; (ii) the selection of association schemes under environments with different propagation models, i.e. path-loss exponents.

I. INTRODUCTION

In traditional operated mobile networks, each user (mobile) is obliged to subscribe to a particular operator and has access to the base stations owned by the operator (or to Wi-Fi access points administered by the operator). A new paradigm known as mobile virtual network operators (MVNO) is currently reshaping the wireless service industry. The idea is to provide higher service quality and connectivity by pooling and sharing the infrastructure of multiple wireless networks. A recent remarkable entrant such as Google is testing the water in the US market under the name of “Project Fi”, whose main feature is improved coverage provided through outsourcing infrastructure from its partners, T-Mobile, Sprint and their Wi-Fi networks. In the meantime, the European Commission has been ruling favorably for MVNOs since 2006, so as to make the European wireless market more competitive [1], thereby facilitating investment in MVNOs in Europe. These virtual operators can take advantage of the hitherto impossibility to cherry-pick different network operators which use separate bandwidths, and even different wireless access technologies, for improvement of user experience. It is reported [2] that the market share of these operators, especially in mature markets, ranged from 10% (UK and USA) to 40% (Germany and Netherlands) as of 2014. However, these unprecedented diversities in terms of bandwidths and wireless technologies raise a challenging question on how to harness them in large-scale wireless networks.

In the rest of the paper, we use the terminology “technology diversity” to refer to (i) several networks operated on orthogonal bandwidths and (ii) different cellular technologies (e.g. 3G and 4G), both of which can be shared by MVNOs.

Notably, the de facto standard association policy in existing wireless networks consists in associating each user equipment (UE) with the nearest base station (BS) or access point where one typical aim is to maximize the likelihood of being covered or connected. One of the main points of the present paper is that this is no longer optimal in these emerging virtual networks, as further discussed in Section I-A. The subtle distinction arising from technology diversity is illustrated in Fig. 1, where $r_1^A$ and $r_1^B$ (respectively, $r_2^A$ and $r_2^B$) denote the distances to the nearest and second-nearest BSs of technology $A$ (respectively, technology $B$) from the UE located at the origin. Also, we assume that $r_1^A < r_2^A$, i.e. the nearest BS of technology $A$ is the nearest to the UE, and there are only four BSs as shown in Fig. 1, which are identical except that they operate on different technologies (i.e. non-overlapping bandwidths). In the single technology case ($A = B$), the UE can simply associate with the BS at $r_1^A$. However, if $A \neq B$, e.g. the two technologies operate on different bandwidths, the locations of the strongest interferers, $r_2^A$ and $r_2^B$ (the second-nearest BSs), may overturn the choice of technology $A$ when the strongest interferer of technology $B$ is much farther from the UE than that of technology $A$, i.e. $r_2^B \gg r_2^A$, thus boosting the signal-to-noise-ratio (SINR) of technology $B$. In light of this example, optimal association in such networks requires sophisticated policies adaptively exploiting available information.
We can further generalize the above example and envisage a practical scenario where each UE can obtain the information about several received pilot signals of nearby BSs, as in 3G and 4G cellular networks, which can be translated into a vector of distances. In this paper, we are interested in investigating the following question.

Q: How much performance gain is achievable theoretically by tailoring the association policy and how much of it can we achieve in practice by exploiting available information?

**Main Contributions:** To tackle this association problem, we propose a stochastic geometry model of multi-technology wireless networks which partly builds upon [3]. This leads to a generic analytical framework lending itself to associating UEs to BSs in such a way that various performance metrics are optimized in the presence of the diversities alluded to above, and for various degrees of available information at the UE. In theoretical terms, the proposed framework paves the way for structural results on the partial ordering of optimal policy performance (see Section III) and a methodology for quantifying the performance of various association policies in a mathematically tractable manner. From the practical viewpoint, the results provide a mathematical edifice not only replacing exhaustive simulations but also usable for instance to analyze parsimonious association scheme, such as the max-ratio algorithm defined in the paper. We also prove asymptotic optimality of this pragmatic policy, which uses only the distances to the nearest and second-nearest BSs. Remarkably, all association schemes discussed in this work are underpinned by a user-centric approach leveraging the information about the network that is typically available at each UE in existing networks, thereby dispensing with any need for centralized coordination.

In the rest of the paper, after discussing the specificity of our problem with respect to previous work, we describe the notion of information exploited for the association in Sections II and III in order to characterize optimal association policies which in turn ameliorate performance indices, which are founded upon an underlying stochastic model of BSs and diverse types of information including fading values and distances to the BSs across different technologies. After establishing the optimality of the max-ratio algorithm under a limiting regime and deriving a versatile formula for computation of resulting performance in Sections IV and V, we derive tractable expressions for performance metrics of several association schemes and evaluate them in Sections VI and VII. Proofs for all the results are deferred to the appendix of the full paper [4].

**A. Related Work**

The policy of associating each UE to the nearest BS or the BS with the strongest received power has been taken for granted in the vast literature on cell association. This is for instance the case in the stochastic geometry model of cellular networks [3]. The rationale is clear. This leads to the highest connectivity for each UE to choose the nearest BS unless it is possible to exploit the time-varying fading information, which is often unavailable in practice. Even with the recent emergence of heterogeneous wireless networks, also called HetNet, the rule is still valid in terms of coverage probability. That is, a UE is more likely to be covered if it associates with a BS whose received long-term transmission power (called pilot power) is the strongest. A stochastic geometry model to exploit this heterogeneous transmission powers of BSs belonging to multiple tiers in HetNets along with fading information has been investigated in [5].

However, from the perspective of load balancing between cells, the rule is invalid in general because each UE might be better off with a lightly-loaded cell rather than heavily-loaded one irrespective of the distances to them. In particular, in HetNet scenarios, it is important to distribute UEs to macro-cells and micro-cells so that they are equally loaded. The optimal association in the HetNet setting is inherently computationally infeasible, i.e. NP-Hard, whereas the potential gains from load-aware association schemes are much higher [6]. To tackle this problem, a few approximate or heuristic algorithms were proposed based on convex relaxations [6], [7] and non-cooperative and evolutionary games [8], [9]. Most of these algorithms are iterative in nature, requiring many rounds of messaging between UEs and BSs for their convergence.

It must be stressed that the multiple technology setting studied here is a largely unexplored territory where the validity of the standard rule to associate with the nearest BS is undermined, which is unprecedented in the literature as exemplified in Fig. 1. Lastly, while there have been considerable work adopting stochastic geometry models for analyzing given algorithms in large wireless networks, our work is a radical turnaround in the way of harnessing the model: we investigate new opportunities to tailor and design such algorithms to optimize the performance.

**II. Stochastic Network Model**

In this paper, we consider adapting association schemes to ameliorate any performance metric in a downlink cellular network that is a function of the SINR received at a single typical UE. To this aim, we first describe a generic stochastic model of the network and define the general performance metric that is induced by an association policy of the UE of interest, which are assumed to be decoupled from those of other UEs.¹ Note that we retain our stochastic network model in the most generic form for easier mathematical manoeuvrability of key results in Section III, which in fact holds for for a large class of point processes (PPPs). For instance, the information structure $\mathcal{F}_t$ is simplified later in Section V.

**A. Network Model**

We consider $T$ different technologies where $T$ is finite. The BS locations of technology $i \in [1, T]$ are assumed to be a realization of a homogeneous Poisson-Point Process (PPP) $\phi_i$ on $\mathbb{R}^2$ of intensity $\lambda_i$ independent of other PPPs. The typical user, from whose perspective we perform the analysis, is

¹Note that extending this framework and results therein to the case where the association policy of a user is affected by those of other users (e.g. load-balancing in HetNet) is mathematically far more challenging and thus is left to future work.
assumed to be located at the origin, without loss of generality. Denote by $r_i^j \in \mathbb{R}_+$ the distance to the $j$th closest point of $\phi_i$ to the origin, or equivalently the $j$th nearest BS, where ties are resolved arbitrarily. Hence $r_i^j$ denotes the distance to the closest point (BS) of $\phi_i$ from the origin.

Each BS of technology $i$ transmits at a fixed power $P_i$. The received power at a UE from any BS is however affected by fading effects and signal attenuation captured in the propagation model, typically through the path-loss exponent. We assume independent fading, i.e. the collection of fading coefficients $H_i^j$, which denotes the corresponding value from the $j$th nearest BS in technology $i$ to the UE, are jointly independent and identically distributed according to some distribution function. We model the propagation path loss through a non-increasing function $l_i(\cdot): \mathbb{R}_+ \rightarrow \mathbb{R}_+$, where $i \in \{1, 2, ..., T\}$, i.e. the propagation model for each technology is determined by a possibly different attenuation function. Hence, the signal power received at the typical UE from the $j$th BS of technology $i$ is $P_i H_i^j l_i(r_i^j)$. For mathematical brevity, we henceforth consider the point process $\phi_i$ of technology $i$ where each point is marked with an independent mark denoting the fading coefficient between the point (BS) to the origin (UE). We can assume that all the random variables belong to a single probability space denoted by $(\Omega, \mathcal{F}, \mathbb{P})$ [10].

**B. Information at a UE**

Another point at issue in this paper is the tradeoff between the cost of "information" available at UE and the performance gain attained by the association policy making use of that information. For easier presentation of results, e.g. Theorem III.1, the notion of information is encapsulated in a sigma-field $\mathcal{F}_i$ which is a sub-sigma algebra of the sigma-algebra $\mathcal{F}$ on which the marked point processes $\phi_i$ are defined. A sub-sigma algebra $\mathcal{F}'$ of $\mathcal{F}$ is such that $\mathcal{F}' \subseteq \mathcal{F}$. An example of information is $\mathcal{F}_i = \sigma \left( \cup_{j=1}^{J} \phi_i(B(0, w)) \right)$, which corresponds to the sigma-field generated by the point process up to distance $w$ from the origin. In other words, the UE can estimate BS locations of different technologies $r_j^i$ such that $r_j^i \leq w$. However, note that, we use the information sigma algebra $\mathcal{F}_i$ more generally, which could potentially include fading and shadowing and not just the distances as given in the above example.

**C. Association Policies**

An association policy governs the decisions on which technology and BS the typical user (who is located at the origin) should associate with. More formally, an association policy $\pi$ is a measurable mapping, i.e. $\pi : \Omega \rightarrow \{1, T\} \times \mathbb{N}$ which is $\mathcal{F}_i$ measurable. As stated before, we assume that all additional random variables needed by the policy $\pi$ are $\mathcal{F}_i$ measurable.

The interpretation of the policy $\pi$ being $\mathcal{F}_i$ measurable is that a typical UE decides to choose a technology and a BS to associate with based only on the information obtainable in the network. It is important to note that while our discussion in this paper mainly revolves around optimal policies denoted by $\pi^*$, our methodology for the performance evaluation in Section V can be applied for any (suboptimal) policy.

**D. Performance Metrics**

All performance metrics considered in this work are functions of SINR (Signal to Interference plus Noise Ratio) received at the typical UE. The SINR of the signal received at the origin from the $j$th nearest BS of technology $i$ is:

$$\text{SINR}_{0,j} = \frac{P_i H_i^j l_i(r_i^j)}{N_0 + \sum_{k \in \mathbb{N} \setminus \{j\}} P_k H_k^j l_k(r_k^j)},$$

where $N_0$ is the thermal noise power which is a fixed constant for each technology $i \in \{1, 2, ..., T\}$. In order to encompass a general set of most useful performance metrics in wireless networks, the performance of different association policies are evaluated through non-decreasing functions of the SINR observed at the typical UE. Formally, let $p_i(\cdot) : \mathbb{R}_+ \rightarrow [0, 1]$ be a non-decreasing function for each $i \in \{1, 2, ..., T\}$ which represents the metric of interest if the typical UE associates with technology $i$. Since $\pi$ takes values in two coordinates $[1, T] \times \mathbb{N}$ (Section II-C), we divide them into separate coordinates which are denoted by $\pi(0) \in [1, T]$ and $\pi(1) \in \mathbb{N}$, respectively corresponding to the technology and BS chosen by the policy. Then the performance of the association policy $\pi$ when the information at the typical UE is quantified by $\mathcal{F}_i$ is then given by:

$$\mathbb{E}[p_{\pi(0)}(\text{SINR}_0^\pi)].$$

(1)

The subscript $I$ refers to the fact that the information present at the typical UE is $\mathcal{F}_i$. The performance metric $\mathcal{R}_i^\pi$ is averaged over all realizations of the BS deployments, fading variables, and any additional random variables used in the policy $\pi$.

Two well-known examples of performance metrics used in practice are coverage probability and average achievable rate. Coverage probability corresponds to setting the function $p_i(x) = 1(x \geq \beta_i)$, which is the chance that the SINR observed at a UE from technology $i$ exceeds a threshold $\beta_i$. The other common performance metric of interest, average achievable rate, is defined as $p_i(x) = R_i \log_2(1 + x)$, where the parameter $R_i$ is the bandwidth of technology $i$. All results on optimal association policy and performance evaluation are stated on the assumption of a general function $p_i(x)$.

**III. Optimal Association Policy**

The optimal association policy denoted by $\pi^*$ is

$$\pi^*_i = \arg \sup_{\pi} \mathcal{R}_i^\pi,$$

(2)

where the supremum is over all $\mathcal{F}_i$ measurable policies. From a practical point of view, the optimal association policy is

<table>
<thead>
<tr>
<th>Notation</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_i$</td>
<td>Point process corresponding to technology $i \in [1, T]$</td>
</tr>
<tr>
<td>$\lambda_i$</td>
<td>Intensity (density) of point process $\phi_i$</td>
</tr>
<tr>
<td>$p_i(\cdot)$</td>
<td>Performance function when associated with technology $i$</td>
</tr>
<tr>
<td>$\mathcal{F}_i$</td>
<td>Information available at the typical UE</td>
</tr>
<tr>
<td>$J_i$</td>
<td>$\arg \sup_{\Theta} E[p_i(SINR_{0,i}^\pi)</td>
</tr>
<tr>
<td>$I^*$</td>
<td>The technology chosen by an association policy $\pi$</td>
</tr>
<tr>
<td>$R_i^\pi$</td>
<td>Average performance of optimal policy $\pi^*$ with $\mathcal{F}_i$</td>
</tr>
</tbody>
</table>
the one that maximizes the performance of the typical UE among all policies having the same “information”. In this setup of optimal association, however, we always assume that the typical UE has knowledge of the densities \( \lambda_i \) of the different technologies and the fact that they are independent PPPs although several fundamental results can be easily extended to more general point processes.

Since we are interested in maximizing an increasing function of the SINR of the typical UE, the optimal association rule is clearly to pick the pair of technology and BS which yields the highest performance conditional on \( F_1 \).

**Proposition 1.** The optimal association algorithm when the information at the typical UE is given by the filtration \( F_1 \) is such that

\[
\pi^*_1(0) = \arg \max_{i \in [1:T]} \sup_{j \geq 1} E[p_i(\text{SINR}_0^{(j)})|F_1], \\
\pi^*_1(1) = \arg \max_{j \geq 1 \in [1:T]} \sup_{i \in [1:T]} E[p_i(\text{SINR}_0^{(j)})|F_1],
\]

where the UE must pick the technology \( \pi^*_1(0) \) and the \( \pi^*_1(1) \)-th nearest BS to the origin in \( \phi_{\pi^*_1(0)} \).

The performance of the optimal association is

\[
R^*_1 = E[\sup_{i \in [1:T]} E[p_i(\text{SINR})|F_1]].
\]

Since \([1, T]\) and \( \mathbb{N} \) are countable sets, the order of the maxima in (4) does not matter. An important point to observe is that the optimal association given in (3) depends on the choice of the performance metric \( \{p_i(\cdot)\}_{i=1}^T \). Hence, the optimal association rule would be potentially different if one was interested in maximizing coverage probability as opposed to maximizing rate-related metrics for instance.

**A. Ordering of the Performance of the Optimal Association**

In this sub-section, we prove an intuitive theorem (Theorem III.1) stating that “more” information leads to better performance.

**Theorem III.1.** If \( F_{i1} \subseteq F_{i2} \), then \( R^*_1 \leq R^*_2 \) where the association rule is the optimal one given in (3).

This theorem establishes a partial order on the performance of the optimal policy under different information scenarios at the UE for any performance functions \( \{p_i(\cdot)\}_{i=1}^T \). Hence, the optimal association rule would be potentially different if one was interested in maximizing rate-related metrics as opposed to maximizing rate-related metrics for instance.

**B. Optimal Association in the Absence of Fading Knowledge**

The following lemma is quite intuitive and affirms that the optimal strategy for a UE in the absence of fading knowledge is to associate to the nearest BS of the optimal technology.

**Lemma 1.** If the information \( F_1 \) at the typical UE does not contain the fading random variables, then \( j_1 = \arg \sup_{j \geq 1} E[p_i(\text{SINR}_0^{(j)})|F_1] = 1 \) and hence \( \pi^*_1(1) = j^*_{\pi^*_1(0)} = 1 \).

**C. Examples of Information**

One common class of information is the “locally estimated information” which a UE may attain through measurements of (i) received long-term receive pilot signals, which can be easily converted into distances of BS, and (ii) instantaneous received signals, from which fading coefficients can be computed. For example, the knowledge of the distances to BSs no farther than \( w \) from the UE is quantified through the sigma-algebra \( F_w = \sigma(\cup_{i=1}^w F_i) \), where \( F_i = \sigma(\phi(B(0,i))) \) is the sigma algebra generated by the stochastic process \( \phi(B(0, i)) \).

Furthermore, in case the UE is capable of estimating fading information, one can opt for the sigma-field generated by the marked stochastic process \( \phi(B(0, w)) \), denoted as \( F_w^{H} \), where each point (BS) is marked with a fading coefficient between the BS and the UE. Here the superscript \( H \) refers to the sigma-field generated by the marked-point-process.

In existing networks, the most practical example is the knowledge of the nearest \( L \) BSs of each technology, denoted by \( r_i^* = [r_i^*1, .., r_i^*L] \), i.e. the \( L \)-dimensional vector of the distances. In terms of sigma-algebra, it can be defined as \( F_{\pi}^* = \sigma(\cup_{i=1}^w F_i) \), where \( F_i = \sigma(\phi(B(0, r_i^*))) \) is the sigma-field of the \( L \) nearest BS of each technology. One particularly intriguing scenario is complete information about the BS deployments, i.e. \( L = \infty \). Denote by \( F_{\infty} \) the sigma-field for this information scenario and \( R^*_\infty \) as the performance obtained by the optimal policy knowing the entire network. Since \( F_{\infty} \) is the maximal element among all sub-sigma algebras of \( F \), it follows from Theorem III.1 that \( R^*_\infty \) is the upper bound of all achievable performances. To strike a balance between the performance of interest and estimation cost at UE, each MVNO can evaluate \( R^*_\infty \) to see how much the association policy with \( L \)-distances stack up against the upper bound \( R^*_\infty \).

**IV. MAX-RATIO ASSOCIATION POLICY**

While the parametric framework in Section III paves the way for designing the association policy maximizing various metrics, the optimal schemes encapsulated in (2) and (3) are amenable to tractable analysis only with the knowledge about the underlying PPPs \( \phi_i \), i.e. their intensities \( \lambda_i \). On the other hand, it is less conventional at the present time, if not unrealistic, to assume that the densities \( \lambda_i \) are available at the UE in a real network. More importantly, in certain deployment scenarios, it is highly likely that the BS distribution follows a non-homogeneous point process with density (intensity) varying with the location over the network, thereby invalidating the homogeneous PPP assumption.

From the computational perspective, the optimal association can often demand substantial processing power of the UE particularly when the resulting association tailored for a specified performance metric is not simplified into a tractable closed-form expression. In this light, it is desirable to have policies that are completely oblivious to any statistical modeling assumption on the network, i.e. minimalistic policies exploiting universally available information such as distances to BSs, which can be computed from received pilot signal powers in 3G and 4G networks. To address these issues, we propose a max-ratio association policy. This policy has access to the ratio \( r_{i1}/r_{i1} \) information for each technology \( i \), i.e. the information \( F_1 = \sigma(\cup_{i=1}^L r_{i1}/r_{i1}) \). The max-ratio association is formally described by

\[
i^* = \max_{i \in [1:T]} r_{i1}/r_{i1}, \quad j^* = 1.
\]
This ratio maximization implies that we place a high priority on a technology where simultaneously the distance to the nearest BS $r_1$ is smaller and that to the second-nearest BS $r_2$ is larger than other technologies. Note also that the above expression can be easily rearranged into the ratio of the received pilot powers of the nearest and second-nearest BSs when the BS transmission powers within each technology is the same. We show in Theorem IV.1 that although this policy per se is a suboptimal heuristic, it is optimal (in the sense of (3)) under a certain limiting regime of the wireless environment.

**Theorem IV.1.** Let the noise powers $N_i^0 = 0$ for all technologies $i$ and the performance function for all technologies $p_i(\cdot) = p(\cdot)$ for all $i$. Consider the family of power-law path-loss functions $\{p^{(\alpha)}(\cdot)\}_{\alpha \geq 2}$ where $p^{(\alpha)}(x) = x^{-\alpha}$. Let $k$ be any integer greater than or equal to 2. If the information at the UE is the $k$-tuple of the nearest distances of each technology $i$, i.e., $F_1 = \sigma(U_1^T(r_1^1, \ldots, r_k^1))$, then

$$\pi_\alpha(0) \xrightarrow{\alpha \to \infty} \arg \max_{i \in [1,T]} \frac{r_1^i}{r_2^i} \text{ a.s.,}$$

where $\pi_\alpha(0)$ is the optimal association as stated in (3). Recall $\pi_\alpha(1) = 1$, $\forall \alpha$ from Lemma 1.

This theorem states that max-ratio association is optimal in cases where the signal is drastically attenuated (i.e. large path-loss exponents) with distance, e.g., metropolitan or indoor environments where the exponent reach values higher than 4, e.g. $\alpha \in [4, 7]$. It is noteworthy that $\alpha$ at higher frequencies as in LTE networks tends to be higher (See, e.g. [11, Chapter 2.6] and references therein). In addition, another remarkable implication of this theorem is that it suffices for the asymptotic optimality to exploit the reduced information $r_1^i/r_2^i$ per technology in lieu of the given original information, i.e. $r_1^i$ and $r_2^i$. Also, any supplementary information on distances (or received pilot powers) to the third-nearest or farther BSs is superfluous and does not influence the optimality of the association. In Sections VI, we show that this association brings about surprisingly tractable expressions for key performance indices.

V. Framework for Performance Analysis

In Section III, we compared the performance of the optimal association policy under different information scenarios by establishing a partial order on them without explicitly computing the performance $R^\pi$. However, in order to quantify its impact on $R^\pi$ without resorting to exhaustive simulations, one is also interested in its explicit expression for a given policy $\pi$, which may be an optimal policy as in Section III or a suboptimal one as in Section IV. We demonstrate how to explicitly compute $R^\pi$ in an automatic fashion (in Theorem V.1) for any arbitrary policy $\pi$ belonging to a large class of policies, called generalized association, which constitutes another part of our contribution.

A. Generalized Association

In the rest of the paper, we restrict our discussion to a class of association policies $\pi$ that are optimized over information with a special structure $F_1$, incorporating what is conventionally available in cellular networks. That is, in order to answer the question posed in I, we assume that the form of information that a UE has about each technology $i$ is a vector $r_i \in \mathbb{R}^L$. For instance, if the mobile is informed of the smallest two distances of each technology and their instantaneous signal powers, then $r_i$ is a 4-dimensional vector with 2 dimensions representing the distances and the other 2 dimensions corresponding to the instantaneous fading powers. That is, we adopt this reduced notation as a surrogate for the sigma-algebra notation in Section III for simplicity of the exposition. Formally, we assume that the association policy $\pi = (\pi_i(\cdot))_{i=1}^I$, according to which a mobile chooses a technology to associate with is given by

$$i^* = \arg \max_{i \in [1,I]} \pi_i(r_i, \lambda_i),$$

where $j_i$ is the index of BS of technology $i$ to which the UE associates conditioned on selection of $i$, i.e. $i = i^*$, and $r_i$ is the $L$-dimensional vector of observation for technology $i$.

It is noteworthy that when the technologies are operated on overlapping bandwidths, the above form of association may be extended to a more general form $\pi_i((r_{i1})_1^{j_i}, (\lambda_{i1})_1^{j_i})$, where each association policy utilizes not only the information regarding technology $i$ but also that about all other technologies. Envisioning this extension is easily justifiable because the desirability of technology $i$ (represented by $\pi_i(\cdot)$) is affected by the interference inflicted by other technologies. However, we leave it as future work and focus our discussion onto the restricted class of information $F_1$ in (6) which covers the most interesting scenarios of non-overlapping bandwidths.

B. Performance Computation of the Generalized Association

For each technology $i$, we denote by $f_i(r_i)$ the probability density function (pdf) of the information vector $r_i$ of technology $i$. For instance, if $L = 1$ and each mobile has knowledge about the location of the nearest base-station $r_1^i$, then it follows from the property of a PPP that $f_i(r_1^i)$ is the Rayleigh distribution with parameter $1/\sqrt{2\pi\alpha}$. As for the max-ratio policy, $f_i(r_i) = f_i((r_1^i, r_2^i))$ becomes the distribution of the nearest and second-nearest BSs characterized by the underlying PPP of technology $i$. We also denote by $f_i^*(r)$ the pdf of the vector $r_i$ conditioned on the event that technology $i$ is selected, i.e. $i^* = i$.

Denote by $f_{s_i}(\cdot)$ the pdf of $\pi_i(r_i, \lambda_i)$ and by $F_{s_i}(\cdot)$ the cumulative density function (cdf) of $\pi_i(r_i, \lambda_i)$. To put it simply, $f_{s_i}(\cdot)$ is the cdf of a function $\pi_i(\cdot)$ of the given information $r_i$, rather than that of $r_i$ itself. For example, in case of the max-ratio policy, $f_{s_i}(\cdot)$ is the distribution of $r_2^i/r_1^i$. To prove the main theorem, we first need to delineate the interplay between the distribution of optimal technology $f_i^*(\cdot)$, its original distribution $f_i(\cdot)$, and the (cumulative) distribution of the association policy $F_{s_i}(\cdot)$. We have the following lemma from a direct application of Bayes’ rule and independence of the point processes $\phi_i$.

**Lemma 2.** The probability density function $f_i^*(r)$ is given by

$$f_i^*(r) = f_i(r) \cdot \frac{1}{P[i^* = i]} \cdot \prod_{j=1, j \neq i}^T F_{s_j}(\pi_j(r_i, \lambda_i)).$$

(7)
The following theorem finally presents a direct method for computing the performance of any generalized association policy \( \pi \). Recall that the performance of a policy \( \pi \) is given by \( \mathbb{E} [ \text{SNR}_i^\pi ] \).

**Theorem V.1.** The performance of the association algorithm \( \pi \) under information \( F_i \) denoted by \( \mathbb{R}^\pi_i \) is given by:

\[
\sum_{i=1}^{T} \int_{r \in \mathbb{R}^T} \mathbb{E} [ p_i (\text{SNR}_i^{\pi_i}) | r ] f_i (r) \prod_{j=1, j \neq i} F_{r_j} (r_i) \, dr,
\]

where \( \mathbb{E} [ p_i (\text{SNR}_i^{\pi_i}) | r ] \) corresponds to the performance obtained by associating to technology \( i \) conditioned on the information about technology \( i \). The UE has the vector \( r \).

This theorem states that we need only two expressions, information distribution \( f_i (\cdot) \) and policy distribution \( f_{r_i} (\cdot) \), in order to derive the performance metric. As exemplified earlier, while \( f_i (\cdot) \) is usually a simplistic expression thanks to properties of PPP, mathematical manipulability of \( f_{r_i} (\cdot) \) relies highly on the complexity of the association policy.

**VI. COMPUTATIONAL EXAMPLES**

In this section, we leverage the results in Section V to derive several performance metrics in selected practical scenarios where the association policy utilizes information \( F_i \) restricted to a vector of distances to BSs \( (r_i) \) and BS densities \( (\lambda_i) \) as shown in (6). Note however that one can directly compute the performance \( (\mathbb{R}^\pi_i \) in Theorem V.1) with the probability density function of any association policy \( (f_i (r)) \) by exploiting Lemma 2. We show that the resulting performance expressions are mathematically tractable and lend themselves to quantifying the performance of large-scale wireless networks.

For the rest of this section, we consider two representative metrics: (i) coverage probability \( p_i (x) = 1 (x \geq \beta_i) \) and (ii) average achievable rate where, to simplify the exposition, we assume the bandwidths of different technologies are the same, i.e. \( p_i (x) = p (x) = \log_2 (1 + x) \). However, Theorem V.1 can be used to compute the performance of any arbitrary non-decreasing function \( p_i (\cdot) \). We also assume that the fading variable \( H_i^\pi \) is exponential, i.e. Rayleigh fading, with mean \( \mu_i^{-1} \) and the path-loss function \( l_i (r) = r^{-\alpha} \) for \( \alpha > 2 \) for all \( i \in [1, T] \). These assumptions have often been adopted for analysis of wireless systems [11] and espoused in stochastic geometry models [10], [12].

Let us denote by \( \varsigma_i (j; r, \lambda, P, \beta) \) the coverage probability of a UE at the origin served by the \( j \)th nearest BS to the origin where the BSs are spatially distributed as a PPP of intensity \( \lambda \). Here each BS transmits at power \( P \) and we are interested in the probabilistic event that the received SINR exceeds the threshold \( \beta \). The vector \( r \) denotes the vector of distances to BSs, based on which the association decision will be made. More formally,

\[
\varsigma_i (j; r, \lambda, P, N_0, \beta) = \mathbb{E} \left\{ 1 \left( \frac{P H_j}{N_0 + \sum_{k \neq j} P H_k} \geq \beta \right) \right\} | r |.
\]

Likewise, we denote by \( r (j; r, \lambda, P) \) the expected rate received by a typical UE at the origin when it is being served by the \( j \)th nearest BS to the origin where the BSs are distributed as a PPP of intensity \( \lambda \) and transmitting at power level \( P \):

\[
r (j; r, \lambda, P, N_0) = \mathbb{E} \left\{ \log_2 \left( 1 + \frac{P H_j}{N_0 + \sum_{k \neq j} P H_k} \right) \right\} | r |
\]

\[
= \int_{t \geq 0} \varsigma_i (j; r, \lambda, P, N_0, 2^t - 1) dt.
\]

Therefore, once we derive an expression for the coverage probability, the average achievable rate expression follows immediately from the calculation of the simple integral in (10).

In the sequel, we first compute technology-wise expressions, (9) and (10), which are in turn plugged as \( p_i (\cdot) \) into Theorem V.1 to yield coverage probability metric \( \mathcal{R}^p \) and average achievable rate metric \( \mathcal{R}^r \), respectively.

**A. Optimal Association Policy**

Recall that in the absence of knowledge of fading information, the optimal association policy is to choose technology \( i^* \) such that:

\[
i^* = \arg \max_{i \in [1, T]} c_i (1; r_i, \lambda_i, P_i, N_0, \beta_i),
\]

\[
i^* = \arg \max_{i \in [1, T]} r (1; r_i, \lambda_i, N_0, P_i),
\]

respectively for coverage probability and average achievable rate. Note also that it follows from Lemma 1 that it is unconditionally optimal to choose the nearest BS for each technology, i.e. \( j^* = 1 \). Thus our discussion in this section is focused on the choice of technology \( i \).

In this example, we investigate two cases where the UE has knowledge of the distances to the nearest \( r_1^i \) or up to the second-nearest BSs \( [r_1^i, r_2^i] \) along with the densities of technologies \( \lambda_i \) while being oblivious to the information about fading \( H_i^\pi \). In comparison with the standard rule to associate with the nearest BS, this example demonstrates how our proposed framework can be used not only to design an optimal association algorithm maximizing a performance index but also to compute the resulting performance improvements arising from the additional knowledge of distances and densities. The following theorem delineates, among all technologies \( i \in [1, T] \), which technology yields the best coverage probability metric.

**Theorem VI.1.** If the UE has the knowledge about \( r_1^i \), for all \( i \in [1, T] \), the association rule (11) with the following expression maximizes the coverage probability:

\[
c_i (1; r_1^i, \lambda, P, N_0, \beta) = e^{-\mu N_0 \alpha \beta^{-1} P^{-1}}
\]

\[
\exp \left\{ -2 \pi \lambda \int_{u=r_1}^{\infty} \frac{1}{1 + \beta^{-1} (u/r_1)^\alpha} u du \right\}.
\]

If the UE has the knowledge about \( r_1^i \) and \( r_2^i \), for all \( i \in [1, T] \), the association rule (11) with the following expression maximizes the coverage probability:

\[
c_i (1; r_1^i, r_2^i, \lambda, P, N_0, \beta) = e^{-\mu N_0 \alpha \beta^{-1} P^{-1}}
\]

\[
\exp \left\{ -2 \pi \lambda \int_{u=r_2}^{\infty} \frac{1}{1 + \beta^{-1} (u/r_1)^\alpha} u du \right\}.
\]
To better understand the practical implications of (13), we can consider the case where the thermal noise and threshold terms are identical, i.e. $N_0 = 0$ and $\beta_i = \beta$. Since both the first and second factors in the right-hand side of (13) are decreasing functions with respect to $r_1$, the above policy gives preference to smaller $r_1$ among all technologies $i \in [1, T]$, which is in line with our intuition.

However, for approximately similar values of $r_1$, it also reveals that the optimal policy tends to choose technology $i$ with lower density $\lambda_i$ because the right-hand side of (13) decreases with $\lambda$. The observation is in best agreement with our intuition again because technology $i$ with high density $\lambda_i$ implies that there are more interfering BSs on the average. On the other hand, the standard rule leads to higher chance of association with the technology with large $\lambda_i$ because the nearest BS is more likely to belong to the technologies consisting of higher number of BSs. Thus it can be deduced that in case of heterogeneous BS densities, the standard rule leads to very poor coverage performance because of its tendency to associate with the most *populous* technology, whereas the above equation reveals the optimality of associating with *sparsely populated* technology, which sheds light on the complex optimization to be carried out by MVNOs. Likewise, the optimal policy exploiting the additional information of $r_2$ exhibits similar tendencies in (14) by preferring a technology $i$ having large $r_2$, i.e. the technology with smallest dominant interfering power.

In order to compute the optimal performance metric $R^{op}$ resulting from the association rule maximizing the coverage probability, we first need to derive the probability distribution of $\mathbb{P}(r; \lambda_i, P_i, N_0, \beta)$, which is in turn plugged into Theorem VI.1. The CDF $F_{r_i}(y) = \mathbb{P}[r; \lambda_i, P_i, N_0, \beta] \leq y]$ can be simplified into the following expression by using the fact that the nearest distance $r$ of BSs distributed as a PPP is Rayleigh distributed with parameter $\frac{1}{\sqrt{2} \lambda_i}$.

**Lemma 3.** The CDF $F_{r_i}(\cdot)$ is given by

$$F_{r_i}(y) = e^{-\ln(\frac{1}{2})(\int_{r_i}^{y} \frac{1}{\sqrt{2} \lambda_i v} e^{-v} dv)}^{-1}. \quad (15)$$

Finally, plugging (15) into (8), we get the following theorem on the coverage probability maximized by the optimal association policy.

**Corollary 1.** The coverage probability resulting from the optimal association exploiting the knowledge of $r_1$ is

$$R^{op} = \sum_{i=1}^{T} \int_{r \in \mathbb{R}} c_p(1; r, \lambda_i, P_i, N_0^i, \beta_i) 2\pi \lambda_i r e^{-\pi \lambda_i r^2} \prod_{j=1,j \neq i}^{T} \left( \frac{1}{c_p(1; r, \lambda_j, P_j, N_0^j, \beta_j)} \right)^{-\frac{1}{2}(\int_{r_j}^{y} \frac{1}{\sqrt{2} \lambda_j v} e^{-v} dv)}^{-1}.$$ 

**B. Max-Ratio Association Policy**

Recall that in the absence of fading information, the Max-Ratio algorithm described in Section IV is to choose technology such that $i^* = \max_{i \in [1, T]} r_2^i/r_1^i$ with the nearest BS in the chosen technology, i.e. $i^* = 1$. Although we saw in Section VI-A that the density information play a crucial role in performing optimal association, we know from Theorem IV.1 that the simple non-parametric policy of Max-Ratio is optimal in the limit of large path-loss. In this section, we also show that this policy is tractable and yields expressions for key performance metrics (Corollary 2 and 3). The simplistic form of the policy distribution $F_{r_i}(\cdot)$ in the following lemma alludes to ensuing tractable results in this section.

**Lemma 4.** The law $F_{r_i}(\cdot)$ for the max-ratio algorithm is:

$$F_{r_i}(x) = \mathbb{P}[r_2^i/r_1^i \leq x] = 1 - 1/x^2. \quad (16)$$

**Corollary 2.** The coverage probability performance $R^{op}$ of the max-ratio algorithm is given by

$$2 \sum_{i=1}^{T} \int_{t \geq 1} c_p \left( 1; \frac{r_2}{r_1} = \lambda_i, P_i, N_0^i, \beta_i \right) \frac{1}{t} \left[ 1 - \frac{1}{t^2} \right]^{T-1} dt, \quad (17)$$

where

$$c_p \left( 1; \frac{r_2}{r_1} = \lambda_i, P_i, N_0^i, \beta_i \right) = \int_{u=0}^{\infty} c_p \left( 1; [u, \alpha], \lambda_i, P_i, N_0^i, \beta_i \right) 2(\pi \lambda_i)^2 u^3 t^4 e^{-\lambda_i(\alpha + u)t^2} du,$$

in which $c_p \left( 1; [u, \alpha], \lambda_i, P_i, N_0^i, \beta_i \right)$ is given in (14).

Since the max-ratio does not optimize a particular performance metric but merely compares the ratio $r_2^i/r_1^i$, the average achievable rate expression can be obtained directly from the integral transform in (10), which in turn is plugged into Theorem IV.1 to yield the following corollary.

**Corollary 3.** The average achievable rate of the max-ratio algorithm is

$$R^p = 2 \sum_{i=1}^{T} \lambda_i \int_{t \geq 1} c_p \left( 1; \frac{r_2}{r_1} = \lambda_i, P_i, 2^{n_i^p} - 1 \right) \frac{1}{t} \left[ 1 - \frac{1}{t^2} \right]^{T-1} dt dv, \quad (19)$$

where $c_p \left( 1; \frac{r_2}{r_1} = \lambda_i, P_i, 2^{n_i^p} - 1 \right)$ is given in (18).

To get more intuition about the formula, we present the following theorem.

**Theorem VI.2.** In the Interference-limited regime (i.e. $N_0^0 = 0$ for all $i \in [1, T]$), if the path-loss function is given by $l_i(r) = r^{-\alpha}$ for some $\alpha > 2$ and all $i \in [1, T]$, the coverage probability and the average achievable rate of the max-ratio algorithm are respectively given by

$$R^{op} = \left\{ \begin{array}{ll} \int_{x=0}^{1} 2(T-1)x^{(1-x)^2} \frac{1}{1 + \beta x + (\alpha, x)} dx, & T \geq 2, \\ \int_{x=0}^{1} 2(T-1)x^{(1-x)^2} \frac{1}{1 + \beta x + (\alpha, x)} dx, & T = 1, \end{array} \right. \quad (20)$$
\[ R^\alpha = \begin{cases} \sum_{i=1}^{T} \int_{x=0}^{1} \frac{2(T-1)x^2(1-x)^{T-2}}{1+(T-2)x^{T-1}} \phi(\alpha, \beta_i, x) \cdot dx, & T \geq 2 \\ \int_{t>0} \frac{1}{1+(T-2)t^2} \cdot \phi(\alpha, \beta_i, t) \cdot dt, & T = 1 \end{cases} \]

where the function \( \phi \) is given by

\[ \phi(\alpha, \beta_i, x) = \int_{u \geq y^{-2/\alpha}} \frac{1}{1 + x^{-u^{\alpha/2}}} \cdot du. \]

Since the case \( T = 1 \) in the above theorem corresponds to the standard rule to associate with the nearest BS in the presence of only one technology, Equations (20) and (21) reduce to much simpler expressions compared to those in the literature, e.g. Sections III-D and IV-C in the work [3]. On the other hand, as \( T \) becomes larger, inside integrand in (20), the distribution \( 2T(T-1)x^3(1-x^3)^{T-2} \) (additional \( T \) cancels out the summation operation) is gradually skewed toward the origin \( x = 0 \), around which \( \phi(\alpha, \beta_i, x) \) approaches 0. It is easy to show that the coverage probability \( R^\alpha \) approaches one with higher technology diversity, i.e. \( T \rightarrow \infty \). Though it is not realistic to envision such a large number of technologies or operators, from which each UE can cherry-pick its optimal BS, this theorem demonstrates how much UEs can potentially benefit from the this diversity pooled by MVNOS.

Contrary to the standard association which tends to pick more populous technologies (i.e. large \( \lambda_i \)), giving rise to higher number of interferers, the max-ratio policy counterbalances this pathological behavior by ensuring that the strongest interferer \( r^*_i \) is located relatively further. At the same time, the overall performance of max-ratio algorithm critically relies on large path-loss constant \( \alpha \), whereas with this caveat, Theorem IV.1 states that the algorithm is asymptotically optimal as \( \alpha \rightarrow \infty \) for any increasing performance function \( p_i(\cdot) = p(\cdot) \) in the interference-limited regime.

VII. SIMULATIONS AND NUMERICAL RESULTS

In this section, we provide more insights into our framework and results by performing simulations and noticing their trends. In performing the simulations, we take as performance metrics, the coverage probability with \( p_i(x) = 1(x \geq \beta_i) \) and the average rate with \( p_i(x) = \log_2(1 + x) \).

A. Diminishing Returns with Increasing Information

We first observe through simulations that the optimal association for any good class of performance metrics (made precise in the sequel) exhibits the law of diminishing returns. The term, diminishing returns, is used in the context where the additional gains or improvements in performance of optimal association reduces as the information increases. Fig. 2 shows that coverage probability with the optimal association exploiting the knowledge of the nearest \( k \) BS of each technology. As we move on the x-axis, we are increasing the information known at the UE and observe that the gains saturate drastically. Remarkably, beyond learning the 2 nearest BSs per technology, there is no tangible improvement in the coverage probability.

This implies that in practice, it is sufficient for each UE to learn the nearest two BSs per technology which will yield almost all the optimal performance possible with the full information about the topology.

We present a simple argument why one would expect to see diminishing returns for any performance metric. Assume we have some “good” performance metric functions \( \{p_i(\cdot)\}_{i=1}^n \), i.e. \( \mathbb{E}[p_i(\text{SINR}^l_i)] \) is upper-bounded for all \( i \in [1, T] \) and all \( j \in \mathbb{N} \). Let \( \{F_n\}_{n \in \mathbb{N}} \) be a filtration of information \( \mathcal{F} \) such that \( i \in F_n \subseteq F_{n+1} \subseteq \mathcal{F} \) for all \( n \). Denote by \( \mathcal{F}_n^\infty \) as the limit of \( F_n \), i.e. \( \mathcal{F}_n^\infty = \cup_{n \geq 1} F_n \) and let \( R_n^\alpha = \mathbb{E}[\sup_{j \geq 1} \max_{i \in [1, T]} \mathbb{E}[p_i(\text{SINR}^l_i)] | \mathcal{F}_n^\infty] \) be the performance of the optimal association policy under information \( \mathcal{F}_n^\infty \). Theorem III.1 then gives that the sequence \( \{R_n^\alpha\}_{n \geq 1} \) is non-decreasing and \( R_n^\alpha \rightarrow \infty \). Any such sequence of bounded and non-decreasing numbers contains a sub-sequence \( \{R_{n_i}^\alpha\} \), such that the gains \( \Delta_i = R_{n_i}^\alpha - R_{n_i+1}^\alpha \) decreases with \( i \). Therefore, the law of diminishing returns property holds.

B. Comparison of Schemes and Technology Diversity

The first two graphs in Fig. 3 compare the coverage probability of various association schemes with path-loss exponent \( \alpha = 4 \) for different number of technologies, \( T = 5 \) and \( T = 8 \). We observe in all graphs that the Max-Ratio association scheme outperforms the optimal association policy under the case when only the nearest BS distances are known. More importantly, the Max-ratio association performs almost as well as the optimal association under the knowledge of nearest 2 BSs per technology for this typical value of path-loss exponent, not to mention that it outperforms the nearest BS association significantly, particularly when the technology diversity is higher, i.e. \( T = 8 \).

The rightmost graph in Fig. 3 depicts the average achievable rate for path-loss exponents \( \alpha \in [2.5, 7] \), which empirically corroborates the statement of Theorem IV.1 that Max-Ratio is the optimal policy when nearest \( k \geq 2 \) BS per technology are known in the high path-loss regime. Remarkably, Max-Ratio and the optimal association with two nearest BS distances performs almost equally (indistinguishable in the graph) for \( \alpha \geq 5 \). That is, a simple non-parametric policy like the max-ratio performs as well as the optimal association policy in which the entire network topology is known (the best possible
performance) even in the finite path-loss case. It is also noted that the random BS association, which is the only policy oblivious to technology diversity, results in poor performance in all cases. Thus it is beneficial for MVNOs to leverage the technology diversity in any possible manner by all means.

As shown in Fig. 4, the coverage probability tends to one as $T$ goes to infinity, as discussed in Section VI-B. The performance of Max-Ratio algorithm however reaches one quicker than nearest BS association. This shows that Max-Ratio exploits this diversity better than the conventional scheme to associate to the nearest BS.

VIII. CONCLUSIONS

In this work, we explored the potential to boost the service performance of wireless networks without incurring additional infrastructure cost by capitalizing on a new form of diversity, which can be either several networks operated on orthogonal bandwidths or multiple wireless technologies pooled by some mobile virtual network operators. We proposed a generic stochastic geometry model for designing association policies proactively optimizing desired performance metrics. We also showed that the most important metrics can in turn be evaluated via a generic formula. Combined with another result characterizing the gradual increase of performance with respect to the amount of information, the framework provides a theoretical upper bound on the given metric, which can be used to determine the balance between the cost of estimating information at a mobile and the performance gain. Lastly, we devised a pragmatic association scheme exploiting only two received pilot powers, whose asymptotic optimality is established under a limiting regime of high path-loss. As shown in the simulations, this scheme can serve as an alternative to the standard rules in urban or metropolitan environments with severe signal attenuation which better exploits the new form of diversity.

IX. ACKNOWLEDGMENTS

This work was supported by an award from the Simons Foundation (2017982) to The University of Texas at Austin.

REFERENCES

Bridging the Gap Between QoE and User Engagement in HTTP Video Streaming

Christian Moldovan  
University of Duisburg-Essen  
Modeling of Adaptive Systems  
Essen, Germany  
Email: christian.moldovan@uni-due.de

Florian Metzger  
University of Duisburg-Essen  
Modeling of Adaptive Systems  
Essen, Germany  
Email: florian.metzger@uni-due.de

Abstract—On video streaming platforms, users expect a high Quality of Experience (QoE). In contrast, service and content providers aim at high User Engagement, most notably because their revenue is usually dependent on it. In order to satisfy users, it is critical to know how QoE is related to the User Engagement. However, no model for this relation exists yet. Current approaches to managing QoE are usually based on traffic analysis. However, this will become more difficult in the future since the encryption of Internet traffic progresses.

In this paper, we present an approach to bringing QoE and User Engagement together with video streaming as the use case. We do this by fitting existing measurement data of User Engagement to obtain a model. Furthermore, we extend the existing queuing model for QoE and investigate the correlation between QoE and User Engagement in a simulation. Hereby, we model different scenarios where we quantify network bandwidth and video requirements.

Our results show that there is a strong correlation between QoE and User Engagement. Additionally, we observe that simple buffer policies, which do not rely on buffer information, can still perform well. These revelations open the way for new approaches to QoE Management in the future Internet.

1. Introduction

In the past few years, TCP-based video streaming services, like Netflix or YouTube, have garnered enormous popularity amounting to a significant share of the Internet’s traffic (compare e.g. [1]). For both, service providers as well as network operators, it is in their best interest to optimize the QoE of those video streams in order to satisfy their customers.

Although service providers typically can directly derive QoE metrics from the data that flows between them and the service users, network operators usually lack this option. In order to get a faint picture of the current QoE conditions in the network, operators have to go the extra mile and intensively conduct traffic analyses and deploy middleboxes to directly investigate and influence traffic conditions. In the future, this will become even more challenging, as almost all services and protocols are slowly moving to deploy full end-to-end encryption, eliminating many traffic analysis approaches.1

In contrast, there is another class of metrics that could deem worthwhile, namely context factors. Using this kind of indirect measures and metadata of various origin, an astonishing amount of information can be revealed on the traffic of interest. Moreover, many of these factors could still be collected with end-to-end encryption.

Of particular interesting variant of context factors are User Engagement metrics, which give a description of the amount a user interacts with a given service. For example, in the case of video streaming, an engagement metric would be the (relative) duration of a user watching a specific video. Service providers already use such metrics, e.g. for the successful placement of advertisements or to keep users invested in their platform. Since direct QoE measures might become harder to obtain in the future, the question is, if such User Engagement metrics can be used as a replacement and, if at all possible, which trade-offs have to be taken.

This paper aims to further this exact research question of the possible correlation between User Engagement and QoE for on-demand video streaming. In addition, the correlation and the interdependence between those two metrics needs to be evaluated. Speaking of HTTP streaming, the impact of the client’s buffering policy on the QoE and on the User Engagement needs to be investigated.

We conduct this investigation via a queuing model that describes the video player behavior in terms of stalling periods for arbitrary network conditions and video characteristics. The results are mapped to QoE according to our QoE model. Further, we propose a model for User Engagement that is established by fitting existing measurement results. Based on these models, we analyze the correlation between QoE and User Engagement numerically. In addition, we use measurement results from a real environment for the comparison of different buffer policies. Thus, the main contributions of this paper are as follows:

- We present an approach to align QoE and User Engagement.
- We show that our queuing model can be applied to any video with a generally independently distributed bit rate.
- We show that the choice of the buffer policy has no significant impact on either the QoE or the User Engagement.
- An important implication of our results is that User Engagement will be useful for QoE management in the future.

The remainder of the paper is structured as follows. Section II briefly covers the importance of QoE and User Engagement in the context of future Internet developments. In Section III we provide background and related work. Section IV describes our system and user model with Section V presenting and assessing results from this model and further simulations. Finally, Section VI concludes the paper and outlines future work.

II. QoE MANAGEMENT IN THE FUTURE INTERNET

Many contradicting visions exist for a future version of the Internet, but they can be roughly divided into two schools of thought: The “clean slate” approach, that essentially wants to have a differently organized network structure created completely independent from the current structure, and the evolutionary approach, aiming to iterate on and alter current network architectures with endeavors such as Content-Aware Networking (CAN) (cf. e.g. [2] and [3]).

One of the chief goals thereof is to create the capability to select the right QoE strategy inside the network depending on the type of content. In order to achieve this, the network has to be aware of the content that the flows are transporting. This information can be attained from several angles. Either by developing new protocols and mechanisms that intentionally involve the network with the transport and thereby revealing enough metadata to it, or by using Deep Packet Inspection (DPI), which is a more likely case. The network can then start to optimize TCP-based video streaming by taking into account the client’s buffering behavior.

One aspect of a future Internet is often underrepresented in research, although this development is currently observable in today’s Internet: The transition to fully end-to-end encrypted transmissions. A series of events in the recent past makes this development pretty self-evident, including:

- The Standardization of the HTTP/1.1-successor HTTP/2 in [4]. While, in contrast to earlier specifications, its final form does not mandate the use of Transport Layer Security (TLS) any more, major browser and server vendors have agreed to enforce TLS in their implementations nonetheless.2
- Both Google and Mozilla intent to phase out unencrypted HTTP usage completely3. Statements from major Internet organizations also strongly discourage the further use of insecure protocols for most applications, including the World Wide Web Consortium (W3C)4, the Internet Architecture Board (IAB)5, and the Internet Engineering Task Force (IETF) [5].
- Major service providers, including the YouTube video service, already migrated much of their infrastructure to use HTTPS by default in order to reduce the attack surface of its users and to reduce the influences of middleboxes in networks. The performance cost and overhead for servers as well as to the connection is becoming more and more negligible [6] due to ongoing optimization efforts.7

Therefore, in the near-to-mid-term future almost all data will be transported in an encrypted manner making traffic analysis much more difficult. This is a fact, that has to be kept closely in mind for future research and also impacts the aforementioned QoE management aspect as it prevents one from successfully employing DPI.

But this might make other metrics that can still be derived from encrypted transmissions much more interesting to use. In the case of User Engagement metrics, many of them can not only be measured through monitoring but also by capturing simple flow and TCP characteristics, e.g., the length and throughput of a flow and if it is aborted prematurely. With a clear relationship between User Engagement and QoE, QoE-aware network management and dimensioning could then be easily conducted on a per-source or per-destination basis, if the User Engagement metrics detect issues for a specific source or destination.

III. BACKGROUND AND RELATED WORK

This paper touches the topics of HTTP streaming, User Engagement, as well as the monitoring and management of Quality of Service (QoS) and QoE. Each of these three will be tackled here.

A. HTTP Video Streaming Background

HTTP video streaming has been taking up a large portion of Internet traffic in recent years. Unlike past Real-time Transport Protocol (RTP)-based streaming approaches, this newer technique employs the reliable TCP as transport protocol, bringing along very distinct characteristics.

When a user opens up a video player of such a streaming platform, the player will issue an HTTP request for the video file. Video data is then progressively downloaded and put into a playback buffer, meaning that playback can be started before the whole video file is downloaded. After an initial stalling period and after the buffer contains a certain amount of video data the playback starts. Further

---

2http://daniel.haxx.se/blog/2015/03/06/tls-in-http/
3https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/
4https://groups.google.com/a/chronium.org/forum/%23/topic/blink-dev/2LXKVWYkOus/
5https://w3c.github.io/webappsec/specs/powerfulfeatures/
6https://www.iab.org/2014/11/14/iab-statement-on-internet-confidentiality/
7https://islsfastyet.com/
on, if not enough data is being transmitted in time, stalling will occur. This is the key differentiator to RTP streaming, where in the same scenario, frames would have been dropped or corrupted but the playback would have continued. Stalling events lead to dissatisfaction for users which can be objectively measured as QoE.

In adaptive video streaming videos are partitioned into independently playable segments and each video will be provided in several quality levels and thus in several download volumes. This gives the player another degree of freedom and allows to reduce the video quality in order to minimize stalling events. A longer overview on HTTP streaming is given in [7] or [8]. Furthermore, e.g., [9] presents a model-based approach to evaluate the buffering characteristics of different players and strategies. This paper only takes non-adaptive video streaming into account, and leaves adaptive streaming for future work.

B. Related Work on User Engagement and QoE in non-live Video Streaming

The authors of [10] measure QoE-metrics and User Engagement from various sites, different types of content (short Video On Demand (VOD), long VOD, and live video), and also distinguish other kinds of parameters. Their results show that a high buffering ratio lowers User Engagement, with the impact being stronger for short videos. Similarly, a high bit rate has a significant impact in the live scenario while it does not in VOD. In [11] traffic during a single live event is measured and the impact of QoE metrics on User Engagement is analyzed. Their results show that the buffering ratio and the bit rate have a high impact on User Engagement. Further, they noted that the video play time may depend on various other factors such as user behavior. A correlation between QoE and User Engagement was also recognized.

A 2014 paper [12] conducted a large scale measurement study that looked at the abandonment rate — which can be another appropriate User Engagement metric — for mobile video streaming. Using data from the study, a model is proposed that can predict User Engagement in mobile video streaming with a high accuracy based on network statistics.

In two further publications, Balachandran et al. [13], [14] measured User Engagement and video session quality and run machine learning algorithms on it. Through this effort, they highlight the challenges of obtaining a robust video QoE model from such metrics. And finally, a paper from Krishnan et al. [15] puts viewer behavior in relation to video quality metrics. Of note is the observation that an increase in the initial delay of a video stream also directly leads to a higher abandonment rate.

User Engagement can be defined in many different ways. E.g., time spent on a website, abandonment rate, interactions, click rate, attention paid, number of comments. It is interesting from the perspective of the content provider and the service provider since high User Engagement leads to a higher number of ad views or sales. For video streaming services, we need an easily measurable, objective metric that describes how much content users consume and how willing they are to view ads. Therefore, we define User Engagement as the view time of a video. An overview of models for User Engagement metrics for a number of online services is given in [16]. In the remainder of this paper we look at average values of the User Engagement on a per video level. In addition, users might abandon a service because of stalling, thereby reducing User Engagement. Using this definition, it seems plausible that video streaming platforms, content providers, or video service providers generate revenue based on User Engagement making it a critical metric.

C. QoE Management Related Work

So far, all these papers have looked at the significance of specific User Engagement metrics but lack in terms of mapping measured QoS values to a specific QoE and how to facilitate this information for network and QoE management aspects in light of the future Internet development. The following publications investigate this from an Internet Service Provider (ISP)’s point of view.

For an ISP it is generally more difficult to estimate the video streaming QoE in its network and may require invasive measures, such as DPI. However, this is possible with the approach suggested in [17] which was also successfully deployed in the network of a large European mobile operator [18]. Data gained from such monitoring endeavors can be further utilized, e.g., to enable flow-based traffic management for improving QoE via SDN as presented in [19].

Once such influence factors from all network layers have been collected they can be mapped to QoE according to existing QoS-QoE models and relationships. As this is not without challenges, [20] surveys current research activities on QoE management with a focus on wireless networks where QoE management has mostly just been considered in terms of resource scheduling and resource allocation decisions. Furthermore, the involved technology continuously advances and introduces new challenges. Such as the migration of services to the cloud [21], and in particular cloud gaming [22], [23]. Home gateways are also a starting point to optimize QoE at a small scale. [24] shows that even with just very basic knowledge of the users service requirements, a significant improvement in QoE can be achieved through methods such as application prioritization and traffic shaping.

In contrast, managing QoE based on User Engagement estimates the user’s QoE with objective metrics. For example, [25] defines a reception ratio as the ratio between download throughput and video encoding rate. For some ISPs this may already be sufficient to determine whether stalling occurs or not and how the user reacts in response. [17] concludes that this ratio cannot be directly related to the QoE, yet it is still a good indicator if there are problems in the network. Both [11] and [26] investigate and review different engagement measures and how they are impacted by QoE metrics.
The problem with many such QoE management approaches is that for some services the models are not fully understood or there may be further, hidden influence factors which are not captured by the employed methodology, e.g., recency effects. Additionally, even if the models are well established, e.g. for HTTP video streaming, it may still be difficult to measure the related parameters. Similarly, looking at research efforts involving video streaming engagement, often the reasons are unclear why a user stops watching the video. It may stem from quality issues in the streaming process, but it may very well also just be that the user lost interest in that particular content.

While many QoE metrics might not be measurable in an HTTPS environment, User Engagement can be measured more easily at large scale. Additionally, the effects of low quality will be directly visible in this User Engagement measures. Since there seems to be a lot of value in investigating the relation between QoE and engagement, the aim of this work is to bridge those two fields together, preparing the way to combine their advantages in new models. An overview of our contribution and its relationship with previous and future work is presented in Figure 1.

IV. SYSTEM AND USER MODEL

In the following we discuss our system model, QoE and User Engagement in on-demand HTTP video streaming. The buffer of the video player is modeled as a queuing model with network bandwidth patterns and video characteristics (frame size, frame rate) as input and stalling patterns as output. The presented QoE model maps key performance parameters to QoE. Further, we introduce a new model for User Engagement in which a performance parameter is mapped to play time. Additionally, we point out the limitations of our model.

A. Player Model

The video player determines which frame is played out at which point in time. On application level, video frames are downloaded in order into a buffer with rate \( \lambda \). Downloaded frames are replayed from the buffer with a frame rate \( \mu \). If \( a < 1 \) (i.e. \( \mu > \lambda \)), frames are replayed faster than they are downloaded, which will lead to an empty buffer. In this case, the player pauses the replaying process, which is called stalling. In order to resume the replaying process, a condition has to be met that is determined by the players policy. E.g. playback resumes when the number of frames in the buffer surpass the predefined buffer size \( \delta \).

B. QoE Model

B. QoE Model

The QoE model is provided in [17] and is based on subjective experiments in which test subjects assessed QoE for short videos with varying stalling patterns. In order to objectively assess QoE, we quantify it in terms of Mean Opinion Score (MOS). The MOS is defined as a value between 1.5 and 5 with 1.5 corresponding to the lowest QoE value and 5 to the highest. For the QoE \( Q \), the subjective results of [17] show \( Q(L, N^*) = 3.5 \cdot e^{-(\alpha L + \beta) N^*} + 1.5 \) with \( \alpha \) and \( \beta \) being the parameters for the user’s sensitivity to stalling. This relation is depicted in Figure 2.

C. User Engagement

User Engagement describes the activity or attention of users in a system. As described in Section III-B, for video streaming we restrict the definition of User Engagement to the average amount of time (in minutes) users watch a video. In [10], several data sets were collected and analyzed. In Figure 3 we take a closer look at one of their data sets: LvodA which contains long VoD clips with a length of about 35 min to 60 min. In each data point users with the same ratio of buffering events are related to an average play time. We fitted a nonlinear curve to these data points in least-squares sense using MATLAB, which provides us with a fitting function

\[
U(R) = 4.2712 \cdot e^{-0.5435 \cdot R} + 25.9000 \cdot e^{-0.0339 \cdot R}
\]
This function maps the ratio of buffering events $R$ to the average play time in minutes. For the fitting, we chose a double exponential decay since this is commonly used for describing spontaneous human behavior (e.g. in [28]). The Pearson correlation coefficient for this fit is 0.996 (Spearman 0.997). The RMSE is 0.659 min (normalized RMSE 0.092 min). This indicates that the fit is very accurate.

### D. Impact of Player Policies on QoS and QoE

In this section we present an analytical approach to calculating key QoS parameters for the player model described in IV-A. We do this by extending an existing $M/M/1$ model from [27] to an $M/G/1$ model for three buffer policies which are presented in [29]: the D-policy, the n-policy, and the T-policy.

A detailed mean value analysis of the steady state was derived in [27] for an $M/M/1$ model. Similarly, we derive the key performance metrics from Figure 4 as follows. The video download starts at $t_0$ with bytes being downloaded at rate $\lambda$. If the downloaded bytes amount to $d$ at $t_1$, the video is being replayed from the buffer with video bit rate $\mu$ while the downloading continues. This means the data in the buffer is being removed at a rate of $\mu - \lambda$ until the data is the buffer is reduced to 0 at $t_2$. Then the replaying stalls and the process repeats. The average length of stalling events is $L = t_2 - t_1 = \frac{d}{\mu - \lambda}$. Further, $L$ is identical to the idle period. The ratio of buffering events is $R = \frac{t_2 - t_1}{t_2 - t_0} = 1 - a$. For the average length of the busy period (or playing period) we yield $B = t_2 - t_0 = \frac{d}{\mu - \lambda}$. If we look at the frequency of busy periods during the video replaying, we get the normalized buffering ratio $N^* = \frac{1}{R} = \frac{L}{B}$. For the QoE, this leads to

$$Q(L, N^*) = 3.5 \cdot e^{-\alpha \frac{L}{B} + \beta \frac{1}{N^*}}.$$

Next, we extend the $M/M/1$-model from above for the n-policy, the D-policy and the T-policy. In [29], the authors derive the distributions and the means of the busy and idle periods of queuing models for these three policies. In the following, we adapt these results for HTTP video streaming for the case of a generally distributed service process. For the sake of clarity, when variables for specific policies are discussed, they have the policy name as their index, e.g. for the D-policy, we use the notation $L_D$, $B_D$, $N^*_D$ instead of $L$, $B$, $N^*$.

1) $M/G/1$ with D-policy: With a D-policy, the idle period ends if the sum of the service times of the units in the queue amounts to $D$. For the specific case of video streaming, this policy means that stalling ends after the data in the buffer amounts to a certain play time $D$. This policy guarantees that the length of the busy period is at least $D$. For the D-policy, it is

$$E[L_D] = \frac{1}{\lambda}(M(D) + 1)$$

with $M(D)$ being the renewal process of $D$,

$$E[B_D] = \frac{M(D) + 1}{\mu - \lambda}$$

$$E[N^*_D] = \frac{1}{B_D} = \frac{\mu - \lambda}{M(D) + 1}.$$
It follows
\[ Q(L_D, N_D^*) = 3.5 \cdot e^{-(\lambda T)(\frac{1}{2}+\frac{\beta}{\alpha+\beta}e^{-\lambda T})} + 1.5. \]

If we assume the buffer size \( d \) does not change during a video session, then \( M(D) = d - 1 \) is constant as well. Thus \( L = L_D \) and \( N^* = N_D^* \) are equal for \( M/M/1 \) and \( M/G/1 \) with D-policy. Therefore, \( Q(L, N^*) = Q(L_D, N_D^*) \) is equal for both models.

2) \( M/G/1 \) with \( n \)-policy: With the \( n \)-policy, the idle period ends if \( n = d^* \cdot \mu \) bytes are in the queue. For the \( n \)-policy, it is
\[ E[L_n] = \frac{n}{\lambda} = \frac{d^*}{a}, \]
\[ E[B_n] = \frac{1}{\mu(1 - \alpha)} = \frac{1}{1 - \alpha}, \]
\[ E[N_n^*] = \frac{1}{B_n} = \frac{1 - a}{d^*}. \]

Since \( E[L_n] \) and \( E[N_n] \) are equal for \( M/M/1 \) and \( M/G/1 \) with \( n \)-policy, \( Q(L, N^*) = Q(L_n, N_n^*) \) is also equal for both models.

3) \( M/G/1 \) with T-policy: With the T-policy, when an idle period starts, a timer is started. If the timer reaches \( T \), the systems verifies whether a unit arrived during the idle period. If it did arrive, the busy period is started. Otherwise, the timer is restarted. The probability that no unit arrives during the idle period \( T \) is \( e^{-\lambda T} \). If we can assume \( e^{-\lambda T} = 0 \), this policy guarantees that the length of each stalling period is exactly \( T \). For the T-policy, it is
\[ E[L_T] = \frac{T}{1 - e^{-\lambda T}}, \]
\[ E[B_T] = \frac{\lambda T}{(1 - e^{-\lambda T})(\mu - \lambda)}, \]
\[ E[N_T^*] = \frac{1}{B_T} = \frac{(1 - e^{-\lambda T})(\mu - \lambda)}{\lambda T}. \]

It follows
\[ Q(L_T, N_T^*) = 3.5 \cdot e^{(1+\frac{1}{2})(\alpha+\beta+\frac{1}{\alpha+\beta}e^{-\lambda T})} + 1.5. \]

If we can assume \( e^{-\lambda T} = 0 \) and choose \( T = d^* \) this leads to \( Q(L_T, N_T^*) = Q(L, N^*) \). In Figure 5 we see that the impact of the policies is small for \( T = d^* \). Consequently, the service process has no impact on the QoE under the assumption of a Markovian arrival process. This means that only the mean and not the variance of the video bit rate matters for the QoE, assuming an \( M/G/1 \) model. This result was also observed in simulation results that will be presented in Section V.

E. Model Limitations

The QoE model that we use is based on subjective experiments of short video clips (\( T = 30 \) s). Nevertheless, we use this model for steady state analysis in Section IV-D and for our simulation in Section V-C for a movie with a length of 12 min. However, subjective experiments for long videos are currently missing in literature and the model needs to be validated against subjective experiments with longer videos. New objective and subjective user tests are necessary in order to provide a general QoE function that takes the duration of a video into account. This issue is a current research topic and is discussed in greater detail in [30] and [31].

While we map the ratio of stalling events to the User Engagement, there are also other factors (such as startup delay [15] or the video bit rate [11]) which influence engagement. Measurement studies for engagement need to consider those factors too, in order to derive a complete model. Future work should include such factors in order to refine the proposed model. Nevertheless, this work is an important first step in identifying the relationship between QoE and User Engagement.

V. RESULTS

This section takes a closer look at the relation between QoE and User Engagement by discussing analytic results for the queuing model described in IV-A. In addition, we look at the simulation of the download of a real video in a real network and compare it with the analytic results.

A. Analytic Results

First, we focus on the D-policy as it reflects current video player implementations of HTTP streaming, and investigate the impact of reception rate (i.e., offered load) and different buffer sizes on QoE and User Engagement. Later, in Section V-C, we compare the different policies in terms of QoE and User Engagement.

Figure 6 shows how the offered load (or ratio between network bit rate and video bitrate) \( a \) is related to the MOS value and the play time for different buffer sizes \( d^* \) (e.g., a value \( a = 0.5 \) means that the bandwidth is half of the video bit rate). We notice that increasing the offered load \( a \) leads to an increasing average MOS and User Engagement. It should be noted that MOS values lower than 2.5 are not considered acceptable by most users [32]. In addition, we see that an increasing buffer size...
leads to higher MOS with the optimum being reached at $Q^* = \lim_{d^* \to \infty} Q(L, N^*) = e^{-\alpha \frac{1}{2a}}$ as shown in [27]. In contrast, the buffer size does not have an impact on the User Engagement. A large difference between QoE and User Engagement is that for $a < 0.4$ the MOS is 1.5 and does not change while increase the play time is noticeable. This is because the QoE model that we use is based on short video clips while the user engagement model is based on long videos. User Engagement has been observed to be lower for shorter videos [15].

Next, we investigate how the QoE value is related to the User Engagement. In Figure 7 we calculated the User Engagement and the QoE for various offered loads. We observe that an increase in QoE always leads to an increase in User Engagement. Since our model for User Engagement does not take the buffer into account, more research is necessary in order to identify its impact. Furthermore, we notice that for very low QoE values, it is difficult to estimate the User Engagement as users may react differently in such scenarios.

The Pearson correlation coefficient is 0.981 (Spearman 0.994). A larger buffer leads to a higher mean play time for the same QoE. This can be explained by the fact that an increase in buffer size leads to an increase in QoE, but not to an increase in mean play time. This means that users will abort videos much earlier if the QoE is low. Therefore, it is critical to ensure a high QoE if User Engagement is to be maximized.

B. Simulation Environment

In order to compare our analytic results with measurement results, we simulated replaying a real video using a real network trace that was recorded in [33]. For this simulation we chose the video “Tears of Steel” in a low spatial resolution ($320 \times 180$). It is a 12 min short movie with a variable bit rate. The network trace was recorded by downloading a large file via HTTP using a UMTS stick while driving on a highway. The resulting trace has a strongly fluctuating bit rate. In total, we used 30 different traffic patterns that were created by adding a temporal shift to the original traffic pattern in [34]. We simulated different network capacities by adjusting the video bit rate, resulting in various offered loads $a$.

In our simulation, video frames are downloaded with a rate that is based on the effective network capacity and the size of the video frame in a best effort manner. Video frames are replayed at a constant rate of 24 frames per second until the video ends. If a stalling event occurs, it is resolved according to the given buffer policy. The simulator is implemented in MATLAB and is available online.

C. Simulation Results

In the following, we compare the simulation results to the analytic results. Figure 8 shows the impact of the

8https://github.com/ChristianMoldovan/HAS-Simulator

Figure 6. QoE value in an $M/G/1$ system. The offered load quantifies the ratio between average network bandwidth and video bit rate.

Figure 7. User Engagement in relation to the QoE in an $M/G/1$ system for $d = 2, 10, 30$ according to Equation 1 in Section IV-C.
offered load $a$ on the MOS for the n-policy, the D-policy and the T-policy. It is clearly visible that the policy does not impact the QoE value significantly. In addition, the real traces lead to a higher QoE than the $\text{M/G}/1$ model. This is mainly due to video specific attributes, i.e. the distribution of frame sizes. Nevertheless, we consider the $G$-distribution a reasonably good approximation for the distribution of frame sizes in videos. While more advanced models may lead to more realistic results, they cannot be solved analytically.

In Figure 9 we investigate how the buffer policies impact the User Engagement that was calculated based on the rate of buffering events $R$ according to Equation 1 in Section IV-C. The main observation is that the policies have almost no impact on the mean play time. This means that since the T-policy does not require any information from the player, it provides a solid alternative to the other policies. This is particularly the case if hiding such information becomes a common practice in the future.

VI. CONCLUSION

In this paper, we studied QoE and User Engagement for an $\text{M/G}/1$ queuing model and for real measurement data. We achieved this by first creating a fit that relates video quality metrics to User Engagement. We showed analytically that $\text{M/G}/1$ with n-policy, D-policy and T-policy is equal to the $\text{M/M}/1$ queuing model in terms of QoE and User Engagement. An interesting observation was that the T-policy does not perform much different than other policies, while in contrast it does not rely on buffer information. This may open new approaches to QoE management and should be investigated in future work.

Furthermore, we noticed a strong correlation between QoE and User Engagement which indicates that User Engagement monitoring is important for QoE management. Therefore, collaboration between the community of User Engagement researchers and the community of QoE researchers will be necessary in the future. A more precise relationship between User Engagement and QoE may be established through future subjective and objective experiments.

ACKNOWLEDGEMENTS

This work was partly funded by Deutsche Forschungsgemeinschaft (DFG) under grants HO 4770/1-2 (DFG Ökonet: Design and Performance Evaluation of New Mechanisms for the Future Internet – New Paradigms and Economic Aspects).

REFERENCES


A Markov Model for Evaluating Resource Sharing Policies for DASH Assisting Network Elements

Jan Willem Kleinrouweler*, Sergio Cabrero*, Rob van der Mei†, Pablo Cesar‡

*Centrum Wiskunde & Informatica †VU University ‡Delft University of Technology
Amsterdam, The Netherlands Amsterdam, The Netherlands Delft, The Netherlands

Email: {j.w.m.kleinrouweler, cabrero, r.d.van.der.mei, p.s.cesar}@cwi.nl

Abstract—In this paper, we present a model for evaluating bandwidth sharing policies, that can be applied to networks that handle both video streaming traffic, as well as other traffic. Video streaming is a demanding network application. In crowded networks, resources need to be properly divided in order not to diminish the streaming experience. However, in network deployments with a large number of users, the streaming performance cannot be obtained straightforwardly from a sharing policy. Therefore, we propose a Markov model that is compatible with Dynamic Adaptive Streaming over HTTP (DASH), the major technology for video streaming over the Internet. If DASH is combined with in-network resource management, its performance can be significantly improved. Nevertheless, resource sharing policies need to be configured. This requires evaluation of many different configurations. Real deployments or network simulations demand many system resources and time. In contrast, our model can quickly evaluate many configurations, and for each configuration output the expected video bitrate and number of changes in video bitrate. These two parameters play an important role in the Quality of Experience of the viewer. In this paper, we demonstrate how our model can be used to analyze and optimize resource sharing policies. As such, our model is a useful tool for network administrators and allows them to better provision and configure their networks.

I. INTRODUCTION

Video streaming over the Internet is a popular network application. However, due to the demand that video streaming puts on a network, its performance and resulting quality of experience (QoE) is largely dependent on how the network is handled. In managed networks, bandwidth sharing policies determine how much bandwidth is available for video streaming, and how bandwidth is available for other traffic. In this paper, we investigate how sharing decisions for these policies affect the streaming performance. We present a performance model that can be used to determine the streaming performance, that is compatible with Dynamic Adaptive Streaming over HTTP (DASH) and network-assisted DASH.

DASH (sometimes referred to as HTTP adaptive streaming) is the major technology for delivering video over the Internet. It has been adopted by major content providers, such as YouTube and Netflix. The client-pull based technology relies on HTTP for transport, and it is known to be firewall friendly and it allows for scalable distribution using content delivery networks (CDNs). Although this technology has advantages over traditional UDP based streaming, it has the drawback that the streaming performance is highly sensitive to other traffic in the network. DASH players have difficulties to maintain a stable high quality stream when a network connection is shared with other DASH players, or when there is (non-video) background traffic. This results in the Quality of Experience (QoE) of the viewer being lowered, measured in terms of video bitrate and number of changes in video quality during playback [1][2].

DASH streaming performance can be significantly improved by using DASH assisting network elements (DANEs). DANEs are in-network elements with knowledge about the network, the current load, and specifically the number of active DASH players. Based on these factors, a DANE partitions the available bandwidth in resources dedicated to DASH streaming and resources for the remaining traffic. Furthermore, it will divide the bandwidth among DASH players, when there are multiple DASH players active at the same time. The DANE will enforce these sharing rules by means of traffic shaping and signaling target bitrates to the players.

In small networking environments (e.g. a home- or small business network) DANEs can provide highly personal network management. Sharing rules are typically ad-hoc and require network administrators to have specific knowledge about the users, their devices, and their applications. Nevertheless, the effects of the sharing rules on the resulting streaming performance are clear. This is different when a large number of users shares a network infrastructure. It is not straightforward how sharing policies propagate considering dynamics in the number of video streams and variations in background traffic.

To determine the streaming performance given a policy, the network environment that is controlled by the DANE has to be evaluated. Real deployments or network simulations with a large number of nodes are unpractical (i.e. they are time consuming or require powerful hardware) and are sometimes even unfeasible. However, to find the optimal policy it is required to quickly evaluate many different configurations. This cannot be offered by actual deployments or network simulations.

The contribution in this paper is twofold. First, we propose a Markov model that can be used to evaluate the streaming performance given a sharing policy. Our model distinguishes DASH flows from other traffic, and allows to specify how bandwidth is shared between these two types of traffic. The result of the model is the mean streaming bitrate and how often this bitrate changes. For these two factors it has been identified that they largely contribute to the Quality of Experience of the viewer [1]. Secondly, we apply our model to demonstrate how decisions on bandwidth sharing affect the video quality, and

978-0-9883045-1-2/16 $31.00 © 2016 ITC
show how our model can be used to obtain an optimal resource sharing policy.

It is impossible to provide a definitive answer on how to optimize video quality, since it largely depends on the local configuration. Changing parameters like capacity, demand on the network, and optimization goal may lead to different outcomes. Therefore, we intend our model to be used as an analytical tool, targeted to network administrators that would like to better manage video traffic in their networks. Based on analyses with the model that we presented in this paper, network administrators can optimize resource management while being aware of the resulting video streaming performance. Compared to real deployments and network simulators, applying our model requires less system resources and time. As such, it allows for evaluation of many different configurations prior to deployment, leading to better configured DANEs.

The remaining of this paper is structured as follows. In Section II we discuss related work and provide a background on video streaming using DANEs. Section III characterizes the different traffic flows and identify different sharing policies. In Section IV we formulate the Markov model that can be used to evaluate sharing policies. In Section V we demonstrate how our model can be used to analyze and optimize streaming performance. Section VI concludes this paper.

II. BACKGROUND

To determine the video streaming performance in a network, given a sharing policy, it is required to perform many evaluations. Because real deployments and simulations are often expensive, we proposed a model that can be used to perform such evaluations [3]. To the best of our knowledge this model is the only model specifically on evaluating DASH streams in networking environments with DANEs, though it shows similarities to the use case from Elayoubi and Roberts to evaluate in-network video transcoding and caching services [4]. However, their evaluation is primarily targeted at cache performance, and does not take changes in video bitrate as an output parameter.

Our previous model allows to specify different groups (e.g. different encoding schemes, devices, or type of user) and accurately outputs mean video bitrate and bitrate changes [3]. However, applicability of this model is limited to deployments with only DASH traffic. As such, it only focusses on resource sharing between DASH players, and not on the impact of background traffic on video quality. In many networks a video streaming service would be deployed in the presence of other traffic in the network. In this paper we aim to close this gap and evaluate how bandwidth sharing between DASH streams and background traffic affects streaming performance. Furthermore, we apply our model to three different traffic flows and identify different sharing policies and analyze the impact on streaming performance.

The goal of a resource sharing policy is to increase the streaming performance of DASH players, and thus to improve the quality of experience of the viewer. The four main factors that contribute to the QoE are: stalling, initial delay, quality and resolution, and bitrate switches (i.e. changes in video quality) [1]. Due to the nature of adaptive streaming, stalling can largely be avoided and the initial delay can be kept short, by changing to low-bitrate video when needed. Furthermore, if DANEs provide Quality of Service for DASH streams, then these events are even more unlikely. The video bitrate and bitrate changes depend on how the DANE handles the current demand, and variations in demand, on the network. This is how the resource sharing policy in the DANE manifests itself and influences the QoE of the viewer. Therefore, in this paper we will use video bitrate and bitrate changes as metrics to measure streaming performance.

To optimize the streaming performance the video bitrate must be maximized. This means that it is important to spend time on higher layer representations when possible [2][5]. Although changing to a higher bitrate representation is often appreciated, if it is followed up by a switch back to a lower bitrate representation shortly after, this effect diminishes. Often switching between different representations, known as bitrate instability, has been identified to negatively impact the QoE [6][7][8]. In the remaining of this section we will give a background on related technologies, for the convenience of the reader.

Dynamic adaptive streaming over HTTP (sometimes referred to as HTTP adaptive streaming) is the dominant technology for video streaming over the Internet. In DASH, the a video file is split up into short segments, typically with a duration between two and ten seconds. Each segment is encoded at multiple bitrates and resolutions. A manifest file describes the representation and the order of the segments. DASH protocols are designed to be client-pull based with HTTP as the transport protocol. This means that the segment files, together with the manifest, are put on an HTTP server. A DASH player first downloads the manifest to obtain full knowledge of the stream. Based on an adaptation algorithm, the player then downloads segments in representations it sees fit. Adaptation algorithms can include factors like current network bandwidth, buffer level, device type, screen resolution, and battery level.

In most DASH architectures the adaptation algorithm is built into the player. This has the major advantage that server infrastructures are stateless (i.e. normal HTTP), and thus allow for more scalable distribution compared to server-push technologies. However, it has been shown that current DASH-capable solutions have difficulties selecting a bitrate, and that they suffer instability and unfair resource sharing when there is background traffic, or when multiple DASH players share a bottleneck link [9][10][11]. Instability and unfairness are the result of a mismatch between the bursty ON/OFF download behavior of DASH players and the TCP transport protocol [12].

A plethora of adaptation algorithms have been proposed in the literature. However, solving problems with bandwidth sharing in the players remains difficult due to a lack of knowledge on network usage. Network devices potentially have this knowledge, and in combination with their capabilities of doing in-network traffic control, they can be leveraged to form a better adaptation mechanism. Network devices that have a minimum knowledge about DASH traffic, and assist DASH players to improve their streaming performance, are referred to as DASH assisting network elements [13]. Several implementations have been presented, including traffic shaping at the residential gateway [14]. DASH-aware proxy
servers [15][16][3], and Software Defined Networking (SDN) based implementations [17][18]. DANE implementations aim to eliminate the problems introduced by the client-pull mechanism, and provide a bridge between the traditional server-push based streaming and DASH technology. In the Server and Network Assisted DASH (SAND) architecture the communication between clients and DANEs, and between different DANEs in a network, is being standardized [13].

III. SYSTEM DESCRIPTION

We first shortly discuss the networking environments that we think are interested to apply our models to. Secondly, we characterize the different types of traffic and identify different sharing policies.

A. Use case

Resource management is typically deployed in networks where bandwidth is scarce. This can occur in cases where the available bandwidth is little, or when a network is shared among a large number of users. In both such cases, an overseeing entity can provide a better division of bandwidth between flows in order to optimize the networking experience. In small deployments, different flows can be addressed separately and DANEs can provide highly personalized traffic management. In this work we are interested in the scenario where a large number of users shares a network connection, and a DANE is deployed to optimize the QoE for a video streaming service. A popular example of such a scenario is a public Wi-Fi network offered in a social event. The event organization, or venue, could enrich the experience by offering video feeds from other parts of the event or with additional content.

B. Traffic characteristics

In our analysis we are primarily focussed on video streaming performance. Therefore, we distinguish between DASH video traffic and other traffic. We characterize the two types of traffic as follows:

Type 1 flows are for DASH streams that belong to the video streaming service we are trying to optimize. For each individual stream the DANE knows in which bitrates the stream is available. Based on the current demand and the selected sharing policy, the DANE selects one of those available bitrates and reserves a matching amount of bandwidth. The target bitrate is communicated to the DASH player, that will use this bitrate in the requests for the next segments. DASH is client-pull based, and the player is responsible for selecting the bitrate. It selects the bitrate in the HTTP requests for a next video segment. This poses the restriction that the video bitrate can only be changed every \( x \) seconds, where \( x \) is the duration of a segment. As a result, if a DASH player receives multiple target bitrates in between two segment requests, only the last communicated bitrate will be effective.

Type 2 flows are for the remaining flows that do not belong to the DASH streaming service. Since our primary focus is on video quality, we will refer to Type 2 flows as background traffic. How much bandwidth is reserved depends on the demand on the network. Type 2 flows represent the aggregate of all background traffic, for which the throughput is obtained by monitoring the network in the DANE. Because it takes time to propagate the new bandwidth reservations, and install the new queuing configuration for traffic control, changes in background traffic are processed as averages over longer intervals (e.g. a few seconds). This will also prevent the DANE from responding to aggressively short peaks in demand. Additionally, shifting to a different bandwidth reservation level occurs in discrete steps. Smaller steps in combination with more frequent updates, will result in a better match to the actual demand. However, such decisions have an effect on quality and stability of DASH stream. In this paper we will, among other factors, evaluate what is the impact of reservation step size and change interval.

C. Sharing policies

Depending on the network where a DANE is deployed in, a different type of sharing policy can be adopted. In this paper, we specify three types of policies that will be discussed below in detail: DASH priority, background traffic priority, and mixed mode. For each type of policy, we define the admittance of DASH players to the network and the priority of DASH traffic over background traffic. The available bandwidth is divided into three zones: reserved for DASH, reserved for background traffic, and a shared zone. The two reserved zones are put in place to prevent starvation of either type of traffic. These zones should be configured to represent the absolute minimum bandwidth that is made available to DASH and background traffic. The shared zone is divided between DASH players and background traffic depending on the selected policy. We only consider policies where once a DASH player is admitted to the network, it will receive enough resources to finish the stream. Stalling, or forcefully stopping a stream by the DANE, as result of a too high demand on the network has such a strong impact on QoE, that these cases should be avoided.

The different types of policies define how DASH traffic interacts with background traffic. They are defined as follows:

**DASH priority:** In this policy DASH traffic is treated as most important. DASH streaming can take as much of the bandwidth in the shared zone as it requires. This decision has two major effects. First, it means that DASH players are allowed in the network as long as they fit in the reserved zone for DASH plus the shared zone. This means that the bandwidth that is required for all players to stream at the lowest available bitrate cannot exceed the bandwidth that is available in the DASH zone plus that is available in the shared zone. The second effect is on the video bitrate for the active players. In DASH priority mode, A DANE selects the highest bitrate that is possible, and can take up all the bandwidth in the shared zone while doing this. Only when there is bandwidth left, the DANE allocates more resources for background traffic. This policy will be most effective in scenarios where DASH traffic is the primary type of traffic, and network access for other services is only provided for convenience.

**Background priority:** This policy is the opposite of DASH priority, and should be considered in environments where a video streaming service should interfere as little as possible with the "original" traffic. As an implication of the rule that the DASH streams should not be stopped by the DANE once...
allowed into the network, the number of DASH flows is limited to how many DASH flows would fit in the zone reserved for DASH. Allowing more DASH flows would risk the possibility that the background demand cannot be satisfied. The DANE only selects higher bitrate than the lowest when the demand from background traffic allows it (i.e. when the demand is low).

**Mixed:** In this mode, the policy for admitting DASH flows into network is taken from DASH priority mode, and combined with the constraint that the bitrate for DASH flows is determined by the background demand. This means that DASH flows are admitted to the network as long as they fit the shared zone, but can only get to higher video bitrates when the background demand is low enough. However, to provide a bit more flexibility, a target bitrate could be specified. If a target bitrate is specified, the highest bitrate that does not exceeds this target bitrate, and would fit in the shared bandwidth zone, will be selected. Higher bitrates can only be selected when the demand for background traffic would allow this. Setting the target bitrate to the highest available bitrate would yield the DASH priority policy. However, setting it to the lowest bitrate does not equal the background priority policy, as the maximum number of DASH flows that is allowed into the network could be higher in mixed mode.

**IV. PERFORMANCE MODEL**

In this section we present the model that can be used to determine the streaming performance. For the purpose of traceability we adopt a Markov model.

**A. Markov model**

We define a two dimensional Markov process to describe the population of the two traffic types. The Markov process is defined as the vector \((n_d, n_{bg})\), where \(n_d\) describes the number of DASH flows, and \(n_{bg}\) represents the current demand from background traffic. The state space of the Markov process is denoted by \(S\). We use the notation \(n_{[d,bg]}(\bar{x})\) to refer to the number of DASH flows, or the intensity of background demand, represented by a state \(\bar{x}\).

In the horizontal dimension, we model the number of DASH flows that are active in the network. The state space is horizontally bounded by the maximum number of DASH flows that can admitted in the network, without affecting the continuity of the stream. Let \(C\) be the capacity of the channel, \(Z_{dash}\) the bandwidth zone that is guaranteed to be reserved for DASH flows, \(Z_{bg}\) be the bandwidth zone reserved only for background traffic, and \(B_{min}\) the lowest available bitrate for DASH videos. The maximum number of DASH flows then becomes \((C - Z_{bg})/B_{min}\) for DASH priority- and mixed mode. For background priority mode, this maximum is \(Z_{dash}/B_{min}\).

The vertical dimension represents the demand of background traffic on the network. As defined in Section III-C, the demand changes with discrete sized steps. A transition from \(n_{bg}\) to \(n_{bg} + 1\) represents increasing the demand of background traffic by one step. In our analysis we will make a distinction between background demand, and how much bandwidth is actually reserved for background traffic. The Markov process describes the background demand, and is vertically bounded by the maximum number of background demand steps: \(C/B_{bg}\), where \(B_{bg}\) is the size of a step. Note that the background demand could overlap with \(Z_{bg}\), to allow for the channel to be fully utilized when there are no (or a low number of) DASH players.

The vector \((n_d, n_{bg})\) is a Markov process with the following transition rates:

- \(n_d\) increases to \(n_d + 1\) at rate \(\lambda_d\),
- \(n_d\) decreases to \(n_d - 1\) at rate \(\lambda_d/\beta_d\),
- \(n_{bg}\) increases to \(n_{bg} + 1\) at rate \(\lambda_{bg}\),
- \(n_{bg}\) decreases to \(n_{bg} - 1\) at rate \(\lambda_{bg}/\beta_{bg}\).

The above listed transition rates for states \(\bar{x} \in S\) define transition matrix \(Q\). In this paper we make the simplifying assumption that both DASH flows and background demand are described by an Erlang process. This allows us to analytically obtain the steady state probabilities \(\pi(n_d, n_{bg})\) by using the Erlang multi-rate loss formula, as follows:

\[
\pi(n_d, n_{bg}) = \frac{1}{G} \cdot \frac{(\lambda_d\beta_d)^{n_d}}{n_d!} \cdot \frac{(\lambda_{bg}\beta_{bg})^{n_{bg}}}{n_{bg}!},
\]

where

\[
G = \sum_{\bar{x} \in S} \frac{(\lambda_d\beta_d)^{n_d(\bar{x})}}{n_d(\bar{x})!} \cdot \frac{(\lambda_{bg}\beta_{bg})^{n_{bg}(\bar{x})}}{n_{bg}(\bar{x})!}.
\]

For DASH flows we have shown in previous work that this would be appropriate [3]. However, different networking environments could yield different background traffic patterns, that possibly require a Markov process with different transition rates for transitions \(n_{bg} \pm 1\). Nevertheless, it would still be possible to numerically solve \(\pi Q = 0\) in short time, as the size of \(S\) is not too large. Furthermore, the methods to obtain the streaming performance that we will describe below, still apply.

**B. Streaming performance**

The bitrate of the DASH streams, and how often this bitrate changes, depends strongly on how a DANE selects bitrates for the streams. Depending on the number of DASH flows and the background demand, a DANE will select the highest possible bitrate from the set of available bitrate, that satisfies the constraints put there by the sharing policy. We determine the bitrate by averaging over the states based on the steady state probabilities.

If the expected number of players is defined as:

\[
E[N_d] = \sum_{\bar{x} \in S} \pi(\bar{x})n_d(\bar{x}),
\]

then expected bitrate of the DASH players becomes:

\[
E[B_d] = \sum_{\bar{x} \in S} \pi(\bar{x})n_d(\bar{x})q_d(\bar{x}),
\]

where \(q_d(\bar{x})\) is the bitrate that is selected by the DANE for DASH players in state \(\bar{x}\). In practice, the encoding scheme for DASH videos typically have bitrate steps that are not of the same size. They are chosen such that every next step in
bitrate will give a similar improvement in quality. Equation 4 can straightforwardly be modified to express mean quality level, by letting \( q_d(x) \) define quality level instead of bitrate.

The number of bitrate switches closely relates to how often the Markov process transitions between states that have a different bitrate selected for DASH players. Changing the bitrate of a DASH stream is technically limited by the size of DASH segments. Let \( t_{seg} \) be the size of the segments in seconds, then DASH streams can only change bitrate every \( t_{seg} \) seconds. We include this behavior in our analysis by only observing the Markov process every \( t_{seg} \) seconds. Via uniformization of the Markov chain we can obtain the probabilities that the process transitions from state \( x \) to state \( y \) in \( t_{seg} \) seconds, \( P_{x,y} \). The expected number of bitrate switches per minute then becomes:

\[
E[Q_d] = \frac{60}{t_{seg}}E[N_d] \sum_{x,y \in S} \pi(x)P_{x,y}g(x \rightarrow y),
\]

where \( s(x \rightarrow y) \) is the number of DASH stream that make a bitrate switch when transitioning from \( x \) to \( y \). DASH streams only switch bitrate when the bitrate in \( x \) is different from the selected bitrate in \( y \). Furthermore, only streams that are active in both states have to make a switch. The number of streams that make a switch on the transition \( x \rightarrow y \) then becomes:

\[
s(x \rightarrow y) = \min(n_a(x), n_a(y)) \cdot \min(1, |q_d(x) - q_d(y)|).
\]

C. Background demand

In the Markov process, we model the demand of background traffic on the network. However, in some conditions the actual bandwidth that is reserved for background traffic cannot match the demand, because the sharing policy prescribes a higher priority for DASH traffic. In those cases, the actually reserved bandwidth, denoted as steps by \( b_{bg} \), will be lower than \( n_{bg} \). Similar to the selected bitrates for DASH, \( b_{bg} \) has to be computed for each state in \( S \). The bandwidth that is assigned to background traffic for state \( x \), given \( B_{bg} \) as the background demand step size, is defined as:

\[
b_{bg}(x) = \frac{C - n_a(x)q_d(x)}{B_{bg}}.
\]

This means that background traffic will get all the remaining bandwidth assigned. This potential over-reservation of bandwidth will allow for faster transfers. Given the reserved bandwidth in each state, we can obtain the expected bandwidth that is assigned to background traffic, by averaging over all states weighted by the steady state distribution:

\[
E[B_{bg}] = \sum_{x \in S} \pi(x)b_{bg}(x)B_{bg}.
\]

In states where the demand exceeds the actual reservation, \( n_{bg}(x) > b_{bg}(x) \), there is under-reservation. Under-reservation could lead to traffic being blocked in the network, and should be kept as low as possible. The probability that a state is encountered where there is under-reservation, can be found by:

\[
E[U_{bg}] = \sum_{x \in T} \pi(x)
\]

where \( T = \{y \mid y \in S \land b_{bg}(y) < n_{bg}(y)\} \), the subset of all states that have under-reservation.

In this section we perform a model based evaluation where we demonstrate how the different parameters in our model affect the streaming performance and the resource allocation for background traffic. Further, we perform a weighted sum optimization to construct the final sharing policy. We want to stress that the analysis below contains only a few example scenarios. Different environments require different input parameters, and may lead to different results. For this reason, we propose our model as an analytical tool, and allow it to be adapted to the environment.

A. Scenario

We consider the scenario of a single public Wi-Fi hotspot that is used by a large number of users at the same time. Users can access the Internet for any purpose, but a DASH based video-on-demand service is offered. A DANE is deployed to manage network resource sharing. The capacity of the network will be \( C = 26000 \) kbit/s. This fairly well represents the maximum throughput of a 54 mbit/s Wi-Fi network, that is about half the theoretical throughput. The background demand is configured as \( \lambda_{bg} = 0.1 \) and \( \beta_{bg} = 50 \), and will result in a mean demand of 10 mbit/s, while showing reasonable variability. Furthermore, such a demand in background traffic will leave enough bandwidth to deploy a video streaming service. The two zones that are reserved for either DASH- or background traffic are set to \( Z_{dash} = Z_{bg} = 6000 \) kbit/s, to guarantee bandwidth for either type of traffic. The mean video duration is set to 140 seconds, and is available in the following bitrates: \( \{300, 400, 500, 750, 1000, 1500, 2000, 3000, 4000, 6000\} \). A segment size of 4.0 seconds is used.

B. Sharing policies

The different policies defines how many DASH players are allowed into the network, and how much of the shared bandwidth can be reserved for DASH players. In Figure 1 we compare the DASH priority policy with the background priority policy, in terms of the expected video quality. The available video bitrates are mapped to quality levels, 0 representing the lowest bitrate, 9 representing the highest.

Figure 1 shows that for a lower DASH demand, the DASH priority policy provides an increase of about one quality level. However, for \( \lambda_d \geq 0.2 \) the video quality will be much higher in favor of background priority mode, where the difference is at most four quality levels. This might seem counterintuitive at first, but is actually an effect of the limited admission of DASH players into the network for the background priority policy. Background priority keeps the number of DASH flows in the network low. This means, that when is a low demand from background traffic, the DASH streams are switching to higher bitrates, instead of allowing more flows into the network. The difference in the expected number of DASH players is illustrated in Figure 2.

Figure 2 shows that for \( \lambda_d > 0.1 \) the DANE is denying DASH players service, and thus keeps the number of players in the network limited. Although we did not work it out in detail for the sake of brevity, the video quality of DASH streams in background priority mode could also go down to the lowest
quality level. This happens when the demand of background traffic would be higher. In those cases, the probability that background traffic leaves bandwidth in the shared zone for DASH becomes very small.

In mixed mode, the quality of DASH streams can be expected to be in between that of DASH priority mode and background priority mode, as long as the quality of DASH flows in DASH priority exceeds background priority. For a higher DASH demand, the target bitrate is not likely to be reached due to a too large number of DASH flows in the network. In these cases, the video bitrate will be similar to the one in DASH priority policy.

Consider a moderate DASH demand with $\lambda_d = 0.08$ (in the remaining of this paper we will use this arrival rate for DASH flows). By increasing the target bitrate in mixed mode, the bitrate of the DASH streams can be increased. The set of target bitrates is equal to the available bitrates (i.e. it would not make sense to have other targets, because a target in between two available bitrates will result in the selecting the lowest of the two). Figure 3 displays the increase in video quality level when increasing the video bitrate. Interestingly, this increase is only little.

The reason that the increase in quality level is only little, is because every step to a higher bitrate is done by all players. Therefore, for the extra demand that is put on the network, the step in bitrate has to multiplied by the number of DASH players. Since the configuration in this example represents a moderately loaded network, there is not that much bandwidth left for increasing the video bitrate. Increasing the target bitrate will thus have a bigger effect in the states where the number of DASH players is lower, but when the load on the network is higher DASH players have to switch to lower bitrates anyway.

C. DASH stream stability

Having only a marginal increase in bitrate when increasing the target bitrate, as displayed in Figure 3, does not mean that setting the target bitrate is of little importance. Video bitrate is only one of the factors that contribute to the quality of experience of the viewer. Another important factor is the number of changes, and the size of these changes, during playout of a video stream. Figure 4 shows that the number of quality level changes (per minute) can be reduced by more than a half, when selecting the target bitrate of 1500 kbit/s.

At target bitrate 1500 kbit/s, the mean bitrate of DASH flows is close to 1500 kbit/s as well. Therefore, it is more likely...
that DASH flows have to make smaller bitrate switches, or no switches at all, compared to the other target bitrates. Figure 4 also shows a difference in the number of bitrate switches, and the number of quality levels that is switched in between, for target bitrates up until 1500 kbit/s. This means that at least for some of the bitrate switches, the size of the switch was bigger than one quality level. We expect that the difference between the number of switches and the number quality levels becomes smaller, when increasing the target bitrate, as an effect of the sensitivity of to changes in background demand. For the lower target bitrates, background traffic has higher priority over DASH traffic compared to the higher target bitrates. This difference in sensitivity to background traffic is shown in Figure 5. This figure shows the number of bitrate switches for target bitrates \{300, 1500, 6000\}, while increasing the speed of variations in background demand.

\[ \text{Fig. 5. Effect of background demand instability on DASH stability. A} \]
\[ \text{comparison of target bitrates (mixed mode) 300 kbit/s, 1500 kbit/s, and 6000} \]
\[ \text{kbit/s.} \]

For each point, \( \lambda_{bg} \) and \( \beta_{bg} \) are chosen such that they result in a background demand of 10 mbit/s (i.e. \( \beta_{bg} = 5/\lambda_{bg} \)). This figure very well shows that lower target bitrates are much more effected by instability in the background demand. It also shows that at the highest target bitrate shows almost no increase in bitrate switches. This is to be expected, as selecting the highest target bitrate equals the DASH priority policy, for which it was the goal that background demand has the as little effect on DASH as possible. Based on this analysis, we can conclude that if video bitrate stability is an important, selecting higher target bitrates would be better. However, selecting a high target bitrate might negatively impact the overall performance of a DANE. For example, increasing the video bitrate lowers the available bandwidth for background traffic, which at some point could be problematic. We will address these conflicting objectives in the next section.

### D. Optimizing performance

A typical approach to improve video quality is to increase the bitrate. However, if we consider the same configuration as above, then the increase in bitrate that can be accomplished by increasing the target bitrate is less than one quality level (from 4.26 at for target bitrate 300 kbit/s, to 5.07 for target 6000 kbit/s, a difference of only 0.8 quality levels). Therefore, in this scenario it might not be interesting to increase the target bitrate for the goal of a higher video bitrate. However, as we showed in Figure 4, the number and size of the bitrate switches can be greatly reduced. While increasing the target bitrate, one must be careful though not to have a too large background traffic under-reservation. Not reserving enough bandwidth for background traffic may have the negative side-effect that some of the background traffic has to blocked. In Figure 6 we plot the probability that background traffic demand the actually reserved bandwidth for background traffic. The figure shows a steep curve from target bitrates higher than 750 kbit/s.

\[ \text{Fig. 6. Effect of target bitrate (mixed mode) on under-reservation of} \]
\[ \text{bandwidth for background traffic.} \]

From the examples above, we can distill that there are three objectives for which we can optimize: video bitrate, number of bitrate switches, and background traffic under-reservation. However, optimizing for one objective could potentially conflict with another objective. For example, increasing the video bitrate also increases bandwidth under-reservation, that should be kept low. Furthermore, increasing the video bitrate is only beneficial for target bitrates up to 1500 kbit/s. Therefore, optimizing the sharing policy becomes a trade-off between bitrate, bitrate switches, and background traffic under-reservation. We employ the weighted sum optimization method to find the optimal target bitrate given these three objectives.

We define three optimization objective functions, \( Q_b(t), Q_s(t), Q_u(t) \), that define how well each target bitrate scores in terms of video bitrate, quality switches, and background demand under-reservation. Each function \( Q_s(t) \), is a linear projection of the best possible value (in this scenario) to 0.0, to the worst possible value to 1.0. Given three weights for each optimization objective, \( w_b = 0.25, w_s = 0.25, w_u = 0.5 \), we can formulate the optimization function as follows:

\[
\min_{t \in B} f(t) = w_b Q_b(t) \cdot w_s Q_s(t) \cdot w_u Q_u(t)
\]

where \( B = \{300, 400, 500, 750, 1000, 1500, 2000, 3000, 4000, 6000\} \), the set of target bitrates. Note that the weights in this example are chosen arbitrarily, as we are not network administrators and have no particular goals for a network

\[
\min_{t \in B} f(t) = w_b Q_b(t) \cdot w_s Q_s(t) \cdot w_u Q_u(t)
\]

subject to

\[
t \in B
\]

\[
\text{where} \ B = \{300, 400, 500, 750, 1000, 1500, 2000, 3000, 4000, 6000\}, \text{the set of target bitrates. Note that the weights in this example are chosen arbitrarily, as we are not network administrators and have no particular goals for a network}
\]

\[
\min_{t \in B} f(t) = w_b Q_b(t) \cdot w_s Q_s(t) \cdot w_u Q_u(t)
\]

subject to

\[
t \in B
\]

\[
\text{where} \ B = \{300, 400, 500, 750, 1000, 1500, 2000, 3000, 4000, 6000\}, \text{the set of target bitrates. Note that the weights in this example are chosen arbitrarily, as we are not network administrators and have no particular goals for a network}
\]
Fig. 7. Weighted sum optimization function including video bitrate, bitrate switches, under-reservation probability of background traffic (lower is better).

VI. CONCLUSION

Video streaming is a popular networking application. However, it requires relatively high bandwidth for the duration of the stream, and therefore it can put a significant demand on the network. In busy networks, that are used by many users simultaneously, it is important that network resources are properly divided among the services, to maintain a satisfying experience. In this paper we have proposed a performance model to evaluate resource sharing policies, that can be applied to network connections that need to handle a high demand, containing both video streaming and other traffic. Our model is consistent with DASH, the technology that nowadays dominates the video streaming market, and DANEs that are the executors of the resource sharing policies. As such, our model provides the streaming bitrate, and the number and size of switches in bitrate over time, as a measure of quality of experience. Furthermore, it allows to estimate the resources that are reserved for DASH and for background traffic, as well as provide an indication of how likely it is that not enough resources are allocated.

With the large interest of research in managing network resources including DASH, and also due to the standardization efforts of the Server and Network Assisted DASH (SAND), it will be likely that the number of DANE deployments will grow in the future. Our model will then be a useful tool for network administrators, that want to better configure their networks. Decisions on resource sharing are not necessarily straightforward, and thus require tools for analysis. By making an analysis with our model similar to the one we performed in this paper, it can be understood what are the implications of sharing decisions on streaming performance or on non-video services. Compared to real deployments or network simulators, our model is more accessible as it allows for a quick evaluation of many configurations.

In previous work, we validated an early version of our model for evaluating DASH streaming performance, using our streaming testbed including real DASH streams [3]. That model showed high accuracy, but was limited to networks with only DASH traffic. In this paper we build on top of that fundament, but the model presented in this paper accounts for background traffic as well. However, as the concept of how to analyze the Markov process to obtain streaming performance could be transferred to the current model, we expect that this model will yield high accuracy as well.

Nevertheless, future efforts will be on providing an evaluation of our model that compares it to real deployments. Furthermore, we will will study what background traffic patterns are common in, for example, public Wi-Fi hotspots. By analyzing traces from such networks, we will be able to reconfigure our model, and apply it to optimize resource sharing in these networks. Moreover, computing the streaming performance with our model is a matter of seconds on modest PC hardware. This potentially allows the model to be used online, and select the optimal resource sharing given the current demand. The next steps with our model are geared towards this direction.

REFERENCES


Mobile Live Video Upstreaming

Philip Lundrigan, Mojgan Khaledi, Makito Kano, Naveen Dasa Subramanyam, and Sneha Kasera
Department of Computer Science
University of Utah
{philipbl, mojgankh, makito, naveends, kasera}@cs.utah.edu

Abstract—We design and build a system we call mobiLivUp, that utilizes nearby smartphones to improve live wide-area video upstreaming. In mobiLivUp, to distribute the video to nearby devices, the video streaming device creates a small wireless network using Wi-Fi Direct. Other devices then connect to this network. Parts of the video stream are sent to these connected devices, which then upload their parts to a location in the wide-area network using their cellular connections. We develop algorithms and methods to effectively distribute video data to nearby nodes and for incentivizing cooperation from these nodes. We test our system through trace-driven simulation and implementation in various settings. Our experiments show that, in general, mobiLivUp increases the aggregate video throughput, depending on the number of nodes forwarding data and their data rates.

I. INTRODUCTION

Smartphones have opened up new possibilities, from the way we organize our lives, to the way we communicate with each other. With smartphone cameras capable of capturing high definition video, we are able to communicate in more powerful ways. An example of this is live streaming video from a mobile device. Live streaming video allows people to share in an experience together, at the same time. Live streaming has become a popular activity for consumers with the advent of apps like Periscope [1] and Meerkat [2]. In such apps, a user shares a link to their live stream with friends through social media like Facebook or Twitter. Friends can connect and watch the video stream in real-time, communicating with the person live streaming through comments and likes. Comcast has recently announced that it will allow customers to live stream video from their XFINITY Share mobile app to another user’s cable box [3]. Another example of a use for live streaming video is live action sports using small recording devices, like the GoPro Hero [4]. Some of these small devices are now equipped with transmitters that can broadcast video to cellular capable devices for uploading in real-time. Live streaming video opens up possibilities to new applications that have not been possible before.

While live video can be upstreamed at a high quality over high bandwidth Wi-Fi links, the same cannot be said about video transmission over cellular links for the following reasons. First, a cellular connection cannot always support the video data rate. Cellular networks have a difficult time keeping up with the demands of smartphone data usage [5] and video streaming perpetuates this problem. Furthermore, cellular networks (as with most networks) have been optimized for fast download speeds, but not upload. Second, spatio-temporal variations in cellular channel conditions can reduce the data rate in unpredictable ways. Even when standing still, the data rate of a cellular connection can vary dramatically [6] [7] over time. This makes it hard for any variable video rate algorithm to work effectively. Third, smartphone video quality is increasing faster than that of cellular connections. Each year, new phones are equipped with better cameras and this is not slowing down. Ultra HD or 4K cameras are already starting to be commonplace in new smartphones [8]. Higher quality video requires a higher cellular data rate.

In this paper, we address the increasingly important problem of upstreaming live video over cellular links in the absence of infrastructure Wi-Fi networks. Specifically, we design and build a system, that we call mobiLivUp, that utilizes nearby smartphones and their cellular bandwidth collectively to effectively increase the live upstream bandwidth. Due to the nature of cellular networks, each mobile device is guaranteed some slice of the cellular bandwidth. The use of multiple devices to transmit on the wireless medium in parallel, without interfering, can enhance the aggregate video transmission rate.

mobiLivUp takes advantage of a smartphone’s multiple wireless interfaces. In mobiLivUp, to distribute the video to nearby devices, the video streaming device creates a small wireless network, using Wi-Fi Direct [9]. Other devices then connect to this Wi-Fi Direct network. Parts of the video stream are sent to these connected devices, which then upload their parts to a location in the wide-area network using their cellular connections. The following advantages are gained by using multiple cellular connections:

i. The total aggregate throughput increases as more cellular connections are used. Each cellular connection contributes its data rate which when combined together, allows the streaming device to stream video as if it had a higher data rate.

ii. The throughput is stabilized as more cellular connections are used. As stated earlier, cellular networks can have highly fluctuating data rates. This effect is minimized by combining multiple cellular connections together. For example, when more cellular connections are used and spatial diversity is higher, the less likely it is for each connection to fluctuate in the same way, causing the aggregate data rate to be more smooth than the individual fluctuating data rates.

1Does not require a Wi-Fi infrastructure network.
Using multiple cellular connections allows for cellular provider diversity. When two or more different cellular providers are used, the difference in cell tower placement and network particularities will lead to a more stable video streaming experience. For example, in a given location, one cellular provider might have a dead spot, but another cellular provider might not [10].

Using multiple mobile devices to cooperatively download stored data (but not upload live video) has been studied in past work. To the best of our knowledge, mobiLivUp is the first system that deals with uploading live video data by using multiple neighboring nodes in real-world scenarios. Cooperative live video uploading presents challenges that have not been addressed in past work. The most critical of these challenges is to ensure that video streaming content traveling through multiple paths arrive at the destination in the correct sequence. Without intervention, the video data will arrive out of sequence, rendering the video unplayable. Receive buffers can be used to relieve this problem to a certain point, but if the receive buffer is too big, the video playback will not be live. Buffering alone can not solve this problem. mobiLivUp uses a novel algorithm to intelligently distribute video packets between cooperating neighboring devices, responding quickly to changes in cellular data rates. Our distribution algorithm also incorporates a mechanism to incentivize devices to forward data. With the incentive model, nodes that participate in mobiLivUp are paid for their usage and everyone benefits as a result.

We evaluate mobiLivUp in three ways. First, we implement mobiLivUp in the ns3 simulator to test and evaluate the method by which we distribute data to forwarding nodes. We also use the simulator to test our incentive model and demonstrate its functionality. Second, we create a wireless testbed which allows us to test the system under controlled conditions, but with more realistic conditions than simulation. Third, we deploy our system while traveling on a commuter train. Our experiments show that, in general, mobiLivUp increases the aggregate video throughput, depending on the number of nodes forwarding data and their data rates. In the case of the commuter train experiment, with two smartphones, mobiLivUp increases the video throughput by up to 88.6% in comparison to that obtained using only one smartphone.

II. RELATED WORK

The idea of using multiple nearby devices in cooperation to share cellular connections has been used for cellular offloading [10] and video downloading. MicroCast [11] is an example of using Wi-Fi or Bluetooth to cooperatively download a video using multiple cellular connections. The use cases of our system and MicroCast seem similar, but there are significant differences. The biggest difference is that MicroCast does not deal with live video data. This allows for different parts of the video to be indefinitely buffered by individual phones. MicroCast is concerned about getting the whole video to all devices, which leads to their focus on a dissemination algorithm. Our system is only concerned about getting the video from the video source to a destination. We use other devices as the means to an end, whereas MicroCast uses the devices as clients. Lastly, MicroCast uses a modified wireless driver, but our system uses standard Wi-Fi Direct. This increases the deployability of our system since no changes need to be made to the operating system for it to work. MicroCast also assumes that all peers remain static during the video download, whereas with our system, we do not make that assumption. We assume a setting in which peer nodes can come and go.

MultiPath TCP (MPTCP) [12] allows for data to be transferred by different interfaces on a device. However, MPTCP is limited to the number of interfaces that are on the device, while our system can scale up to the number of devices in close proximity that are willing to cooperate. One could use emulation to add an interface for each wireless peer to allow for normal use of MPTCP. As far as we know, interface emulation for wireless clients has not been implemented on Android and is beyond the scope of this project. In a recent work, Lim et al. [13] use MPTCP in a simulation framework for stored video and file transfer applications, but not for transmitting live video. SCTP [14] is also limited to the number of interfaces on one device and is not necessarily optimized for live video transmission.

Both MPTCP and MicroCast use some form of queuing to schedule which connection should download a specific segment of data. With MPTCP, a shared receive buffer is used between subflows. With MicroCast, each participating device has a backlog of segments to be downloaded. A new segment is assigned to the device with the smallest backlog. In mobiLivUp, data queuing takes place on separate devices and not on the device where the data is coming from. As a result, explicit feedback by each device is required. In mobiLivUp, this feedback is indirectly obtained from the gatherer, and the splitter pushes data to the participating devices at specific rates, adjusting the rates as necessary.

Carrier aggregation [15] is a technology, part of LTE-Advanced, that allows for a device to receive multiple bands, increasing that device’s data rate and therefore throughput. We use multiple cellular devices, each receiving their own band, to transmit the data. However, there are two important differences. First, with carrier aggregation, the amount of bands dedicated to one device is controlled by the network and not by the device itself. Our system gives control to the device for the number of bands it wants to use to upload video. Second, our system allows for both spatial diversity and cellular provider diversity. The devices used in our system can be at better radio locations and on different cellular providers, increasing the robustness of our system.

Like our system, Quality-Aware Traffic Offloading (QATO) [10] allows a mobile node to offload its data upload to another nearby mobile node with a better network connection. QATO also suggests using Wi-Fi Direct for transmission of data from the source to the neighboring node. However, QATO only uses one mobile node amongst its neighbors to offload uploading. While this limits the benefits obtained from QATO, QATO does not have to deal with
splitting and gathering. Importantly, QATO only experiments with stored data, such as pictures, for uploading. Our system is built for live video transmission.

Link-alike [16] studies the idea of having a wireless device distribute upload data to multiple residential wired connections for a higher aggregate upload rate. However, Link-alike has wired connections. Also, mobiLivUp deals with live video data which is not tolerant to delays, whereas Link-alike deals with general uploaded data which can be buffered as needed.

III. ARCHITECTURE COMPONENTS

mobiLivUp consists of four major components: a splitter, a forwarder, a gatherer, and a client. Figure 1 shows how each component connects to the others. A data source (the video streaming application) passes data to the splitter. The splitter broadcasts its availability, allowing nearby peers who are willing to forward data to connect. To keep the splitting transparent to the client, a gatherer server is needed to act as a proxy. The gatherer also collects statistics about each forwarder and sends feedback to the splitter.

Each component is described below in more detail.

A. Splitter

The splitter takes data from the data source and splits it between forwarding nodes. The splitter is started when a user wants to stream video. When the splitter is first started, it connects to the gatherer. The connection between the splitter and gatherer stays open during the life of the splitter, acting as a control channel for the splitter and gatherer. The gatherer sends feedback to the splitter about each forwarder.

Using Wi-Fi Direct [9], forwarding nodes connect to the splitter. Wi-Fi Direct is a technology used primarily by smartphones to connect point to point, without having to be connected to the same wireless network. With mobiLivUp, the splitter acts as an access point allowing forwarding nodes to connect to it.

When a forwarding node connects to the splitter, information is sent to the forwarder, such as the internal IP address, a unique identifier for the forwarding node, and the IP address and port number of the gatherer, allowing the forwarding node to forward data packets from the splitter to the gatherer. A forwarding node in return sends the cost of using it to forward data. This cost metric is explained in the Forwarder section below. The splitter determines which forwarding nodes it will use based on the cost and the data rate that forwarding node can offer, as described in Section V-B. The splitter itself acts as a forwarding node, using its cellular connection to distribute data to forwarding nodes. The throughput of each forwarding node is sent as feedback from the gatherer to the splitter. This throughput estimate is end-to-end, taking into account both the Wi-Fi Direct connection and the cellular connection. The feedback loop also incorporates loss, as loss affects the throughput. This means that no matter which wireless interface is the bottleneck, the splitter will be able to adjust accordingly.

The splitter determines how much data to send to each forwarder based on a distribution algorithm which is described in Section V-A.

B. Gatherer

The gatherer, a server running in the cloud, receives data from each of the forwarding nodes and the splitter, combines the data together, and sends it to the client. It makes sure the data packets are in the correct order before sending them to the client. This is a necessary step since the packets can become out of order due to the different paths and network conditions they are traveling. The gatherer collects throughput statistics for each of the forwarding nodes and sends it as feedback to the splitter.

C. Forwarder

A forwarding node looks for a splitter to connect to through Wi-Fi Direct. Once it has found a splitter to connect to and received information about where to forward data, it advertises its cost to the splitter. This cost represents the cost of forwarding the data which affects the user’s data plan, bandwidth, and battery life. The cost can adapt with respect to all of these values and change as necessary. The idea is that a forwarder user will charge more if their battery is low or if they have a small amount of data left in their data plan. If the splitter selects the forwarder, it forwards incoming data from the splitter to the gatherer, through its cellular connection. The forwarder is using both of its wireless interfaces at the same time – Wi-Fi to receive data from the splitter and cellular to send data to the gatherer.

IV. WI-FI DIRECT

mobiLivUp uses Wi-Fi Direct to create a local wireless network for the distribution of video data from the splitter to forwarders. It is important to understand the capabilities of this wireless network as it has a significant role in the performance of the system. Wi-Fi Direct allows for an easy configuration of a wireless network by automating the access point setup, authentication, and association, but ultimately, the underlying 802.11 protocol is the same as a traditional Wi-Fi setup (clients connected to an access point).

To test the capability of the Wi-Fi Direct link and smartphones, we send UDP traffic between two Android smartphones at different rates and monitor loss. We find that the loss increases non-linearly as the data rate increases (see Figure 2).
We run an Android version of Wireshark on the sender and determine that the loss is not happening at the sender. To rule out losses on the wireless link, we run similar experiments using a laptop sending to another laptop at the same data rates. In these experiments, we observe that there is no loss. We conclude that the losses are happening at the receiver buffer of the smartphone. Based on these findings, 7 Mbps is the highest data rate with reasonable amount of loss (0.42%). While the high receiver loss is an interesting finding, it is not a problem for our system given that cellular upload rates are typically below 7 Mbps [17]. Moreover, the end-to-end feedback loop from splitter to gatherer in our system can account for any bottleneck along the path and the distribution algorithm will adjust appropriately.

V. DISTRIBUTION ALGORITHM AND INCENTIVE MODEL

Two important aspects of our system are how to distribute data to forwarding nodes and how to motivate forwarders to participate in our system. Using both of these components, mobiLivUp is able to dynamically select which forwarding nodes to distribute data to and utilize those selected forwarding nodes fully.

A. Distribution Algorithm

As described in the previous section, the splitter takes data from the video source and distributes the packets to itself and other forwarding nodes that were selected based on our incentive model. To suitably distribute video packets among different nodes based on their cellular data rates, we assign a weight\(^2\) to each node. There are different approaches on how to update the weights according to the network conditions. The goal of updating the weights is to maximize the throughput of all of the forwarding nodes, thus maximizing the utility of each node. Because of the dynamic nature of cellular networks, it is not always possible to have the weights set to the optimal values. In this situation, there is a trade-off between being aggressive and reacting too quickly to temporary changes.

In the context of our distribution algorithm, we treat the splitter node as a forwarder node. We assume that the data source will be able to adapt the video to match the aggregate throughput of the forwarders, to a certain limit. In other words, if we are only able to send at 1 Mbps, in aggregate, we expect the data source to degrade the video to match it.

Our distribution algorithm is shown in Figure 3. The feedback that the gatherer sends to the splitter contains the actual throughput for each of the forwarders. The splitter also knows the expected data rate of each forwarding node based on the input data rate of the video source and the weight of each forwarding node. The splitter chooses which forwarders to use based on their actual throughput and our incentive model. Let \( E_i \) and \( A_i \) be the expected throughput and actual throughput for the \( i \)th selected forwarding node. mobiLivUp’s distribution algorithm determines how to adjust the expected throughput, \( E_i \), updating the weight for that forwarding node, causing the \( i \)th forwarder to receive \( E_i \) from the data source. The algorithm contains three states: increasing, reducing, and constant. Each state is explained below (see lines 12–20 of Figure 3):

1. **Increase** A node is in the increasing state when \( A_i \geq E_i \). When a forwarding node is in the increase state, its expected throughput gets increased by a small constant value, \( \Delta \), such that \( E_i = A_i + \Delta \).

2. **Constant** A forwarding node is in the constant state when \( A_i < E_i \) and \( A_i > A'_i \), where \( A'_i \) is the previous actual throughput of the \( i \)th forwarder. In this state, the expected value, \( E_i \), stays the same.

3. **Reduce** Reduce state is when \( A_i < E_i \) in which \( E_i = A_i \times (1 - \sigma) \), where \( \sigma \) is the decrease factor.

We discuss the exact values of \( \Delta \) and \( \sigma \) in section VII-A. This algorithm is similar to additive increase, multiplicative decrease (AIMD) technique, but it has the important distinction of containing a third state, constant. This is an important discovery as we implement and test this algorithm. Under certain conditions, a forwarder’s throughput can be increasing, but less than \( E_i \) (\( A_i < E_i \)). There are many reasons why this occurs. Some of the factors that affect this are how much the expected throughput value \( E_i \) is increased by in the increase

\[ \text{while splitter is running do} \]
\[ \text{receive feedback from gatherer} \]
\[ \text{determine selected forwarders based on feedback} \]
\[ \text{if } A_{\text{total}} \geq V_{\text{max}} \text{ then } // \text{Equalize state} \]
\[ j = \text{forwarder with highest actual throughput} \]
\[ k = \text{forwarder with lowest actual throughput} \]
\[ E_j = A_j - \Delta \]
\[ E_k = A_k + \Delta \]
\[ \text{continue} \]
\[ \text{for all } i \text{ in selected forwarders do} \]
\[ E_i = \text{expected throughput for } i\text{th forwarder} \]
\[ A_i = \text{actual throughput for } i\text{th forwarder} \]
\[ \text{if } A_i \geq E_i \text{ then } // \text{Increasing Rate state} \]
\[ E_i = A_i + \Delta \]
\[ \text{else if } A_i > A'_i \text{ then } // \text{Constant Rate state} \]
\[ // \text{Do nothing} \]
\[ \text{else } // \text{Reducing Rate state} \]
\[ E_i = A_i \times (1 - \sigma) \]
\[ A'_i = A_i \]

\[ \text{Fig. 3. Distribution algorithm} \]

\[ \text{Fig. 2. Loss characteristic as the send data rate increases. The shaded region is the ideal throughput with minimal loss.} \]
state \((\Delta)\), how big the window for calculating throughput is at the gatherer, and how often feedback is sent to the splitter from the gatherer (feedback interval). With only two states, a forwarding node’s throughput could be increasing such that \(A'_i < A_i < E_i\), but still be considered in the reduce state and be incorrectly lessened. The extra state, constant, takes care of this scenario.

Figure 4 compares using a two state AIMD distribution algorithm to the three state distribution for different feedback intervals. We use a cellular trace with three forwarding nodes for both distribution algorithms to calculate the average throughput for a given feedback interval. See section VII for how we collect the cellular trace.

When the feedback interval is below 500 ms, the two state distribution algorithm performs poorly relative to the three state distribution algorithm. This is because the feedback is coming too quickly, not allowing enough time for a forwarding node’s throughput to increase to the expected value, causing the distribution algorithm to classify some forwarding nodes in the reduce state. It is only when the feedback interval goes above 500 ms that you seem performance similar to the three state approach. The 500 ms value is specific to this particular cellular trace and can change based on the network conditions. Using the three state approach allows any feedback interval to work with peak performance. For our simulations and experiments we use a feedback interval value of 100 ms.

When the maximum video data rate is reached (a value set by the video streaming application), such that \(A_{total} \geq V_{max}\), where \(A_{total}\) is actual throughput of each forwarder summed together and \(V_{max}\) is the max video rate, the distribution algorithm stops and enters a phase where it tries to equalize the data rates of each of the forwarding nodes (see lines 5–10 of Figure 3). This is done by lowering the expected throughput of the highest forwarder \((E_{high} = A_{high} - \Delta)\) and raising the expected throughput of the lowest forwarder \((E_{low} = A_{low} + \Delta)\). If the forwarder with the lowest expected throughput is able to support the extra throughput \((A_{total} \geq V_{max})\), then the process will be done again. If the forwarder is unable to handle the extra throughput \((A_{total} < V_{max})\), then the total throughput of each forwarder will not be greater than the maximum video data rate and the distribution algorithm will start again. By equalizing the throughput of each selected forwarder, the whole system is more stable. Rather than having a few forwarders at high throughput (possibly close to their limits), it is better to equally share the throughput across many forwarders so all are below their respective limits. If any fluctuations occur on a link, the other forwarders will be able to absorb the extra throughput. By proactively equalizing the rates, we are protecting against later adjustments (and loss) due to under-performing forwarders. mobiLivUp uses coarse granular feedback and hence is robust to temporary disruptions in cellular connectivity. For the same reason, it uses less cellular bandwidth for controlling the system.

B. Incentive Model

In mobiLivUp, we propose a pricing based method that provides incentives for the forwarding nodes to cooperate. We assume that all nodes are rational and selfish. A forwarding node’s main goal is to maximize its profits but not to harm others. We use a simple auction to model cooperative bandwidth sharing. In our auction, the splitter holds the auction among \(n\) nearby forwarding nodes called players. Each player \(i\) has an individual private value \(c_i\) which is the cost of sending one unit of data to the gatherer using the cellular connection. \(c_i\) depends on various parameters including available cellular bandwidth, cellular data rate, and battery level on the phone. Our auction works as follows.

The splitter sends a request to the forwarding nodes and the forwarding nodes reply by sending their bids, \(c_i\), to the splitter. The splitter determines the allocation rule and the payment mechanism based on the received bids and the received feedback from the gatherer about the actual data rates of the forwarders. Due to the changes in the actual data rates, the splitter holds auction every time it receives feedback from the gatherer. The forwarding nodes can also change their bids in each auction and it is possible that the number of forwarding nodes varies in different auctions because of the mobility of mobile phones. Our auction provides incentive for forwarding nodes to cooperate by implementing a \textit{dominant equilibrium}. In this setting, each forwarding node’s best strategy is to report its actual cost \(c_i\), regardless of other player’s strategies. Each player’s utility is defined as its total received payment minus its cost of participation.

\textbf{Problem Formulation And Solution:} In our auction, the splitter must consider both the cost and the actual data rate in selecting forwarding nodes. Let \(w_i \in [0, 1]\) be the weight that is assigned to the player \(i\) based on the gatherer feedback. We define a score \(s_i\) for forwarding node \(i\), as \(s_i(c_i, w_i) = \frac{c_i}{w_i}\). Note that in this equation, \(s_i\) depends on both the cost \(c_i\) and the actual data rate \(w_i\). The splitter uses these scores to select a set of forwarding nodes. We obtain the utility of player \(i\) with score \(s_i\) from the following formula: \(u_i(s_i) = p_i(a_k) - c_i\).

The number of players that are selected by the splitters depends on a budget limit \(B\), the maximum amount that the splitter can pay per second. Depending on the type of the service, the splitter may choose different values for \(B\).
The splitter determines the allocation and payment based on the scores of forwarding nodes under the following conditions: 

**Optimal.** The mechanism should maximize the total score, i.e., total actual data rates divided by total costs. 

**Incentive Compatibility.** There is no selfish forwarding node that has an incentive to lie about the cost, \( c_i \). 

**Individual Rationality.** The utility of all forwarding nodes should be non-negative to provide incentive for them to participate in the game. 

Our problem description is as follows:

\[
\max \sum_{i=1}^{n} a_i s_i \\
\text{s.t.} \\
\forall i, c_i' \in C, p_i(a_s) - c_i \geq p_i(a_{s_i}) - c_i \\
\forall i, p_i(a_s) - c_i \geq 0 \\
\sum_{i=1}^{n} p_i(a_s) \leq B
\]

(1) (2) (3)

Here, \( a_s \in \{0, 1\} \) represents the allocation to player \( i \) with score \( s_i \), when the splitter assigns data to the player, \( a_s = 1 \), otherwise, \( a_s = 0 \). Also, \( p_i(a_s) \) represents the amount that the splitter pays to the player \( i \) under allocation rule \( a_s \). Equation 1 provides incentive compatibility for cost\(^4\). Equation 2 is for individual rationality, and Equation 3 captures the budget limit of the splitter. The splitter orders the forwarding nodes based on their scores decreasing scores \( (s_i) \). Then, it selects the largest number of forwarding nodes \( \{1, 2, \ldots, k\} \) such that \( \forall i \in \{1, 2, \ldots, k\}, c_i < \frac{Bw_i}{\sum_{i=1}^{n} w_i} \).

The payment, \( p_i(a_s) \), is obtained from the following formula:

\[
P_i(a_s) = \min \left( \frac{w_i c_{k+1}}{w_{k+1}}, \frac{Bw_i}{\sum_{i=1}^{n} w_i} \right)
\]

(4)

Here, \( k + 1 \) is the index of the player with the largest score after the selected \( k \) players. In equation 4, the payment increases linearly with the forwarding node’s actual data rate, \( w_i \). The proofs of incentive compatibility and individual rationality have been left out due to space constraints.

VI. Implementation

The splitter and forwarder components of our system are written as an Android application on Samsung Galaxy S4 phones. For the data source component, we used Spydroid [18], an open source project that can stream a smartphone’s camera video to a client either through HTTP or RTSP. The gatherer component is implemented on a server in Python. When the splitter is initially started, it creates a TCP connection with the gatherer. This connection acts as a way for control messages to be sent back and forth between the two components. On this connection, feedback is sent from the gatherer to the splitter. For a client to connect to a data source, it connects through the gatherer. The gatherer forwards all messages from a client to the splitter. The splitter then forwards messages to the data source.

When the RTSP connection has been established and data starts being sent to the splitter from the data source, the splitter distributes packets based on the available data rate of each of the forwarders. To keep track of the ordering of the packets, a sequence number is added to each packet. RTP contains a sequence number, but we did not want to be dependent on any particular application protocol so we do not use it. The splitter also adds the MAC address of the forwarding node it is using to send the packet. This allows the gatherer to keep statistics of how each forwarding node is performing. The gatherer calculates the throughput of each forwarding node and sends this back as feedback to the splitter. Based on the feedback, the splitter changes the weightings of the forwarding nodes to match their actual data rate. The gatherer sends feedback to the splitter every 100 ms. The weight of the forwarding node and the send time are also added to the packet header for debugging and data collection purposes.

The gatherer buffers received packets to make sure they are in order. Once packets are in the right sequential order, it sends the ordered packets to the client. Since we are using RTP, which uses UDP, there is no way of telling if a packet has been lost or if it is still in transition. This presents a problem to the gatherer to know if it should wait for a missing packet to arrive or skip it. To solve this, individual queues are kept for each forwarding node. When a packet is received, it is put into its corresponding queue. If sequence numbers of all the heads of the different queues is greater than the sequence number of the packet in question, then we know the packet must have either been lost or severely reordered by the network and can be counted as a loss. This allows the gatherer to skip that packet and continue to put the rest of the packets in order.

VII. Evaluation

We evaluate mobiLivUp using ns-3 simulations and an implementation on smartphones. There are three purposes for evaluating mobiLivUp in simulation: (i) to test our design with a large number of forwarding nodes (in comparison to only two nodes in our implementation) under various conditions, (ii) to determine the best parameters for the distribution algorithm, and (iii) to develop and understand the interaction between the video distribution algorithm and the incentive mechanisms. We collect and use cellular data traces and Wi-Fi Direct traces to give us realistic wireless characteristics such as data rates and loss.

We also implement the splitter and forwarder as Android applications in Java on two Samsung Galaxy S4 phones. We implement our gatherer on a server in Python. We use our implementation in the following two ways. First, we create an emulation platform that emulates cellular forwarding nodes using Wi-Fi, i.e., the forwarding nodes transmit data using Wi-Fi (and not a cellular network). Here, as in our simulation, we determine the transmit data rate of the forwarding nodes using our traces. However, unlike our simulation, we use real live video transmission from our Android phones to the

---

\(^4\)Since the data rate of each forwarding node is obtained from the gatherer feedback, we do not need to provide incentive compatibility for the data rate.
gatherer. The purpose of this emulation is to validate our implementation in a more realistic setting (compared to the simulation), while still having a controlled environment.

Finally, we use our implementation with Wi-Fi Direct and cellular links in a real-world live video transmission during a commuter train ride to evaluate the benefits of our approaches in real time under real conditions. We present the results of our simulation, implementation, and commuter train ride below.

A. Simulation-Based Evaluation

We implement mobiLivUp in the ns3 simulator. We first determine the best increase rate and decrease factor for our distribution algorithm.

**Increase Rate (Δ), Decrease Factor (σ):** Using a cellular trace, we test our algorithm with a variety of forwarding nodes and increase rates. Figures 5a and 5b show the average throughput and loss of the cellular trace compared to the rate increase amount (in Kbps). The different lines represent different amounts of forwarding nodes used in the simulation. We observe that the larger the increase amount higher the throughput and higher the loss. Intuitively, the higher the increase value, the more aggressive the probing gets, leading to increased throughput but more loss. Interestingly, the throughput levels out at around 400 Kbps, however the loss rate does not. Based on these findings, we choose an increase rate of 200 Kbps in our evaluation. It has an increased throughput compared to lower values, but the loss rate is still low.

Similar to the increase rate, we use our simulation to determine the best decrease factor. Figures 5c and 5d show the average throughput and loss compared to changes in the decrease factor. Our results show that a decrease factor between 0.2 and 0.25 allows us the best combination of high throughput and low loss.

We now test our distribution algorithm under various network conditions. Due to space constraints, we only show our results for one of the cellular traces (the same one we use for determining our increase rate and decrease factor). Table I shows the overall results using different amounts of forwarding nodes. Figure 6a shows the specific throughput of each forwarding node for the five forwarder case.

Compared to the individual phone with the highest throughput, mobiLivUp performs 79.8% better when using three forwards, 150.7% better when using five forwards, and 238.5% better when using ten forwards. Figure 6a gives a clearer idea of how much each forwarder contributes to the overall throughput. The splitter node contributes the most, followed by forwarder one. All the other nodes contribute about the same amount of throughput to the aggregate throughput.

To understand how our distribution algorithm works with mobility, we use the SLAW mobility model [19] to determine when nodes connect and disconnect to the splitter over the course of a one hour simulation. Forwarder one stays connected to the splitter for the whole simulation. Forwarder two comes in at 15 minutes and leaves at 18 minutes. Forwarder three connects at the start of the simulation and disconnects after 51 minutes. Forwarder four comes in at 53 minutes and stays until the simulation ends. Finally, forwarder five comes in at 45 minutes and leaves at 52 minutes. Figure 6b shows how our algorithm adapts as forwards connect and disconnect.

To demonstrate and test the function of our distribution algorithm when it has achieved the maximum throughput, we run a simulation using the following conditions. The splitter and all forwarding nodes that connect have 1400 Kbps of available throughput. The splitter starts out alone sending below the maximum video rate of 1500 Kbps. Forwarder one, two, three, and four connect at 15, 25, 35, and 45 seconds respectively. As each forwarder connects, the aggregate throughput is well above the maximum video data rate. As a result, the throughput of each node is equalized. This behavior can be seen in Figure 6c.

The results in Figure 7 show how the incentive model interacts in the simulator. In this simulation, only three forwarding nodes are used. Figure 7a shows the data rate limit of each of the forwarding nodes (thick line) in relation to the data rate the distribution algorithm has selected (thin line). Figure 7b shows the score of each forwarding node (see Section V-B). As the scores of the forwarding nodes change (based on how much data rate they can provide and their cost), the splitter selects different forwards that can maximize the aggregate data rate and stay under its budget. In Figure 7b, this occurs at about 14 seconds and 21 seconds. This shows that the splitter is able to dynamically select the best set of forwards that can provide the best value to it.

B. Wireless Testbed Evaluation

In this section, we look at the performance of mobiLivUp under three different scenarios. For the testbed, instead of using cellular connections to send data to the gatherer, Wi-

---

**TABLE I**

**SUMMARY OF SIMULATION RESULTS**

<table>
<thead>
<tr>
<th># of Forwarders</th>
<th>Individual Phone (Kbps)</th>
<th>Aggregate Goodput (Kbps)</th>
<th>Jitter (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2010.23</td>
<td>3615.27</td>
<td>3.07225</td>
</tr>
<tr>
<td>5</td>
<td>2197.61</td>
<td>5518.57</td>
<td>1.99732</td>
</tr>
<tr>
<td>10</td>
<td>2573.50</td>
<td>8711.93</td>
<td>1.32504</td>
</tr>
</tbody>
</table>

**TABLE II**

**SUMMARY OF EXPERIMENT RESULTS**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Method</th>
<th>Median Goodput (Kbps)</th>
<th>Median Jitter (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Varied Data Rate</td>
<td>One Phone mobiLivUp</td>
<td>2242</td>
<td>0.067</td>
</tr>
<tr>
<td>Indoor Cell Trace</td>
<td>One Phone mobiLivUp</td>
<td>553</td>
<td>11.670</td>
</tr>
<tr>
<td>Train</td>
<td>One Phone mobiLivUp</td>
<td>1838</td>
<td>0.448</td>
</tr>
</tbody>
</table>

---
Fi is used.\textsuperscript{5} Wi-Fi Direct is still used to connect the cellular devices together. The testbed allows for the data rate of individual forwarding nodes to be set for a specific amount of time. Using the testbed, we can emulate different data rates of each forwarding node based on the incentive model.

conditions, allowing us to see how the system reacts. The first scenario changes the data rate of each of the wireless nodes (Section VII-B1), testing how our system reacts to large changes in cellular conditions. In the second scenario, we playback a cellular trace (Section VII-B2) to test how the system responds to real conditions.

To measure the effectiveness of mobiLivUp, we use two metrics: goodput and jitter. Goodput is the rate at which the gatherer can send usable data to the client (in order video data). Goodput captures the effect of out of order packets because out of order packets need to be buffered before they can be sent to the client. Having only two devices available to us, we do not evaluate the incentive model using the wireless testbed.

Table II summarizes the data for each scenario, showing the median for goodput and jitter. We present a more detailed discussion of the results below.

1) Varied Data Rate: In this scenario, the data rate of the splitter and forwarder start at 360 Kbps and 6 Mbps, respectively. After 60 seconds, the splitter’s data rate increases to 6 Mbps. After 120 seconds, the forwarder’s data rate decreases to 360 Kbps. Based on a study of throughput characteristics of cellular networks [6], we select 360 Kbps and 6 Mbps because they roughly represent the lower bound and average cellular bandwidth. The big difference in data rate between the splitter and gatherer help to illustrate any potential problems with out of order packets. The results are shown in Figure 8a.

The purpose of this scenario is to see how quickly the system can respond to changes in data rate. In this case, the changes occur at the 60 and 120 second marks. This scenario also tests how much the goodput is affected by these network changes. The median goodput is 2200 Kbps. The aggregate goodput is not much higher than the splitter or forwarder’s goodput, indicating that they roughly represent the lower bound and average cellular network conditions.

\textsuperscript{5}Our data rates are low enough and do not saturate the Wi-Fi channel. Any data rate loss due to wireless devices sharing the channel will not affect the results of the testbed.
throughput. This is due to the low data rate of the splitter and forwarder, which is by design of the scenario. The results show that mobiLivUp is able to transition from one forwarder to another with minor dips in goodput.

2) Indoor Cellular Trace: The data rates used in this scenario are calculated from a cellular trace. We collected data to measure the maximum throughput of two cellular links at the same time. The maximum throughput was then entered into the wireless testbed to emulate the cellular environment.

In this particular trace, the data rates change dramatically and at a rapid pace. The trace was collected deep inside a building where cellular connectivity fluctuates. The rapid changes make it harder for the distribution algorithm to determine the data rate of each forwarding node and as a result, at some points, the aggregate goodput is less than an individual forwarding node’s throughput.

The throughput of the splitter and forwarder and the goodput are shown in Figure 8b. mobiLivUp is able to maintain a higher goodput compared to a single phone 67.98% of the time with an average goodput of 1067 Kbps, compared to 553 Kbps when using just one phone. The jitter of mobiLivUp is higher than with one phone as expected due to it using multiple paths. This high jitter can be overcome by waiting to play the video for some amount of time. In this specific scenario, waiting about one second to play the video is acceptable.

C. Train Ride

We tested our system on a train that travels to different cities in the area. The train travels up to 79 mph and makes a stop roughly every eight miles. We traveled a total of 90 miles, collecting data along the way, using two cell phones, one acting as a splitter/forwarder and another acting as a forwarder. Figure 8c shows the throughput of each of the nodes as well as the goodput. The goodput exceeds the throughput of both the forwarding nodes, except for two instances. The displayed results only show a small portion of the data that we collected on the train, while Table II takes into account all of the data collected on the train. The results of this experiment show that mobiLivUp is capable of producing almost ideal aggregate throughput in real network conditions. mobiLivUp performs better than one phone 88.6% of the time with a 67% improvement in goodput.

In summary, in all of our experiments, mobiLivUp has a higher goodput when compared to the performance of a single phone, with the exception of the varied data rate experiment, in which the single phone and mobiLivUp perform comparably on average. Using mobiLivUp, we can have a much higher quality of video but need to introduce a playout delay of only a few seconds to deal with the higher jitter.

VIII. CONCLUSIONS

mobiLivUp is a novel system that provides greater aggregate upload of real-time video data through using cellular connections of nearby neighbors. We designed a novel distribution algorithm to distribute live video to nearby devices at their maximum capacity. Using an incentive model, we can select nearby devices to participate, in which both parties benefit.

REFERENCES

PlanetIgnite: A Self-Assembling, Lightweight, Infrastructure-as-a-Service Edge Cloud

Andy Bavier  
PlanetWorks, LLC and Princeton University  
acb@cs.princeton.edu

Rick McGeer  
US Ignite  
rick.mcgeer@us-ignite.org

Glenn Ricart  
US Ignite  
glenn.ricart@us-ignite.org

Abstract—PlanetIgnite is a general-purpose, Infrastructure-as-a-Service, self-assembling, lightweight edge cloud on virtualized infrastructure with support for single-pane-of-glass distributed application configuration and deployment. This is an entirely new concept. PlanetLab[32], GENI[7], [22], and SAVI[19] are general-purpose IaaS edge clouds, but require top-down installation and dedicated hardware resources at each site and do not offer single-pane-of-glass application deployment. Seattle[11] is a lightweight self-assembling edge cloud that offers single-pane-of-classification and control, but developers are restricted to using a subset of Python. PlanetIgnite is a Containers-as-a-Service Edge Cloud which offers Docker Containers to each PlanetIgnite user. A PlanetIgnore node is an off-the-shelf Ubuntu 14.04 Virtual machine with Docker installed, meaning it can be installed on any edge node where a VM with a routable v4 address is available. Adding a PlanetIgnite node to the infrastructure is simple: a site wishing to host a PlanetIgnite node simply downloads the image; on boot, the new PlanetIgnite node registers with the PlanetIgnite portal, which runs a series of acceptance tests. Once complete, the image is registered and the node is added to the set of PlanetIgnite sites.

I. INTRODUCTION AND MOTIVATION

Computing systems have traditionally been logically centralized systems, either running on a single computer or with components separated by a few hundred microseconds. However, tomorrow’s landscape will be dominated by truly distributed systems – systems whose components are separated by tens to hundreds of milliseconds. This sea change will occur – indeed, has already started – because of the dramatic drop in price of computation, storage, and sensors relative to communication.

The cost for communication has dropped dramatically over the past twenty years: the amount of bandwidth that cost $100 in 1997 now costs only $4. However, the cost of computation has dropped even more dramatically: the amount of computation that could be purchased for $100 in 1997 now costs three cents. Both have dropped dramatically, but computation has dropped by two orders of magnitude more than communication[14]. This has a profound impact, because the price:performance of computation governs the rate of data production, consumption and storage. The price:performance of communication governs the rate of its transmission.

The general effect of programs is to reduce data taken from sensors or produce data to be consumed by people. The dramatic drop in the price of computation relative to communication means that in the future it will make much more sense to send programs to data and to people rather than to send data to programs or connect people to distant programs, an inversion of both the traditional role of networks and common IT practice. These price/performance trends imply that the future of networks is distributed systems, exploiting ubiquitous computation to minimize data transmission.

A network tuned to distributed systems will accelerate the trend towards thin clients as personal systems. Thin clients and Cloud applications have a number of advantages over traditional fat-client applications: data is protected against local failure, is available everywhere there is network connectivity, and is device-independent. Collaboration is a natural, almost built-in feature of Cloud applications. However, fat-client systems such as personal computers are still in widespread use, in part because connectivity is not yet ubiquitous and because today’s centralized cloud imposes bandwidth limits and latency lower bounds on client-server communications. Only applications which are either latency-insensitive or whose data can be cached on the client can be used on thin clients. The utility of owning a personal fat-client computing devices is being steadily eroded by utility and cloud computing, with the principal limitations on the latter being the programmability of the browser and the responsiveness of the network, primarily due to latency concerns. A network with POPs everywhere dramatically increases the responsiveness of the network and relieves pressure on the browser platform.

The Edge Cloud combines the advantages of the Cloud with low latency and high bandwidth. This enables a broad range of services and applications to migrate to the cloud, which are currently restricted to the client for bandwidth or latency reasons. An edge cloud also permits a broad range of truly distributed services such as Content Distribution Networks[39], [13], Wide-Area Storage Systems[27], cloud-based interactive fat-data applications such as the Ignite Distributed Collaborative Visualizer[9], [8], [14], overlay multicast trees[12], wide-area measurement systems, distributed key-value stores[35], distributed map-reduce systems for in situ data reduction from distributed sensors. Many of these systems have been developed and deployed on standalone edge clouds such as the Global Environment for Network Innovations (GENI) and PlanetLab. Infrastructure-as-a-Service (IaaS) Edge Clouds such as GENI and PlanetLab have spread slowly, in part because hosting a PlanetLab node or GENI Rack is a
substantial burden. Both rely on a dedicated hardware investment, which in the case of the GENI Rack costs in the tens of thousands of dollars[3], [2], [23], which must be purchased, installed, maintained and refreshed. Even when a rack is donated to an institution, it can take weeks or months before the institution installs it. The value of an edge cloud is largely dependent on its ubiquity; it is therefore of great value to dramatically lower the cost of installing and maintaining an edge cloud node.

Technology has also changed substantially in the 13 years since PlanetLab was first conceived. In 2002, virtualization technology was nascent, and the Cloud didn’t exist. Indeed, in some sense PlanetLab and Emulab[40], [36] were the world’s first Clouds. In 2002, building an infrastructure required distributed and maintaining dedicated, specialized hardware. But today, clouds and virtualized infrastructure are ubiquitous, and changing the role of hardware resources is commonplace. This opens up a new possibility in deploying infrastructure — as a set of virtual machines which can run on any offered hosts. PlanetIgnite is an update of the PlanetLab/GENI Experiment Engine architecture which does exactly that.

The remainder of this paper is organized as follows. In Section II we describe the architecture of PlanetIgnite. We also describe how each constituent of PlanetIgnite (an application developer, an application deployer, and a host city) sees the infrastructure. In Section III we discuss the implementation of the PlanetIgnite Node Image. In Section IV, we discuss implementing the PlanetIgnite portal and node instantiation and management software. In Section V, we discuss the on-node services available to PlanetIgnite developers, including the foundational Slice Control Service. In Section VI, we conclude and discuss future work.

II. PLANET IGNITE: A LIGHTWEIGHT EDGE CLOUD

PlanetIgnite is intended as a general-purpose, Infrastructure-as-a-Service, self-assembling, lightweight edge cloud on virtualized infrastructure with support for single-pane-of-glass distributed application configuration and deployment. This is an entirely new concept. PlanetLab, GENI, and SAVI are general-purpose IaaS edge clouds, but do not offer single-pane-of-glass application deployment, and require top-down installation and dedicated hardware resources at each site. Seattle is a lightweight self-assembling edge cloud that offers single-pane-of-class configuration and control, but developers are restricted to using a subset of Python. We propose an edge cloud as lightweight and easy to install as Seattle, but with the general-purpose properties or PlanetLab or GENI, which we call PlanetIgnite.

PlanetIgnite is based on our previous experience with the GENI Experiment Engine (GEE)[5], [6], [4]. The GENI Experiment Engine is a Containers-as-a-Service infrastructure that runs on the GENI infrastructure. The GEE offers Docker Containers as a Service, one per GEE node, to each user. It offers single pane-of-glass control through the use of off-the-shelf configuration tools, and a number of services instantiated in each user container, including an internode messaging service. The goal of PlanetIgnite is to use the GENI Experiment Engine as the prototype of a self-assembling self-growing edge cloud, extending its footprint well beyond the current GENI-based platform and to make instantiation of applications on the infrastructure far more easy and automatic. To a current GEE developer PlanetIgnite is intended to appear to be almost indistinguishable from the GEE, save that there will be many more GEE nodes; to a user/deployer of a pre-written PlanetIgnite application it is intended to appear to be a local cloud that instantiates applications immediately; to a PlanetIgnite site it is intended to be a VM image installed, as any other. We consider each of these PlanetIgnite constituents in order.

A. PlanetIgnite Application Developer

To use the GEE, a user logs in to the GEE portal using her GENI credentials. The GEE portal stores no user information or credentials; instead, OpenID[34] is used to call back to the GENI portal, and the user’s email is the userid for the purposes of the GEE. The user is then directed to a dashboard, where, with the click of a button, she can allocate a GEE Slice. When this process is completed (within a few seconds), a download link to a zip file appears on her dashboard. The user then downloads the file to her computer. The zip file contains files which include directions on using the site, Ansible and Fabric configuration files, Slice Control Service playbooks, and slice-specification authentication credentials.

Once the user has downloaded and unpacked the slice file, she is immediately able to ssh into slivers in the usual fashion, and configure them in the usual way. A user will also be able to use any ssh tool of her choosing to populate or control her slice. However, use of the Fabric file downloaded from the GEE site makes upload and execution as easy and quick (roughly) as uploading a Python program to the Google App Engine.

Fabric is one solution to single pane-of-glass control of a slice. It is simply a Python wrapper around ssh commands, which automates the execution of both remote and local commands. We have pre-loaded the Fabric file with a number of commands to both introduce the user to Fabric and to give them out-of-the-box functionality on the site. For example, typing: “fab nmap” runs a script on each host that reports the reachable IP addresses on the private network.

A second solution, with somewhat different semantics, is Ansible. Ansible uses YAML as a declarative description of the node configuration. Rather than issuing ssh commands to the nodes to install and configure software, the user writes a YAML description of the final state of the node, and Ansible issues the necessary commands to build and configure the node. Both Ansible and Fabric are supported and used by, the GEE.

The user can tear down her experiment by using the “free slice” button on her dashboard. No configuration of the slice is required: the user simply runs her experiment.
Indeed, if the software the experiment requires is pre-installed on a basic Ubuntu 14.04 LTS distribution, the user need not install any software at all.

1) The Slice Directory: As mentioned above, once an application developer has allocated a slice, she is able to download a zip file, which, when unzipped, contains all of the configuration and authentication information needed to access and manipulate her slice from her local laptop. Here, we detail the contents of the slice file: the contents offer a concrete guide to the services available to the developer.

<table>
<thead>
<tr>
<th>File</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_rsa</td>
<td>Private key for slice access</td>
</tr>
<tr>
<td>id_rsa.pub</td>
<td>Public key corresponding to id_rsa</td>
</tr>
<tr>
<td>ansible.cfg</td>
<td>Configuration file for Ansible</td>
</tr>
<tr>
<td>ansible-hosts</td>
<td>Inventory of hosts for use with Ansible playbooks</td>
</tr>
<tr>
<td>fabfile.py</td>
<td>Starter Fabric file with configuration information</td>
</tr>
<tr>
<td>ssh-config</td>
<td>SSH Configuration file for ssh and ssh-tool access to the slice</td>
</tr>
<tr>
<td>slice-hosts.yaml</td>
<td>Example Ansible playbook</td>
</tr>
<tr>
<td>message-server.yaml</td>
<td>Ansible playbook to set up a message server in the slice</td>
</tr>
<tr>
<td>message-client.yaml</td>
<td>Ansible playbook to install a message client in the slice</td>
</tr>
<tr>
<td>README.txt</td>
<td>File detailing the contents and use of the directory</td>
</tr>
</tbody>
</table>

The first file is just the authentication information required to use the slice. The third and fourth files are used to configure Ansible: ansible.cfg directs Ansible to use slice<slice_num> as the remote user, and ansible-hosts is the inventory file for ansible. So, for example, $ ansible -i ./ansible-hosts -m ping nodes runs the ping module on every ansible host, verifying the ability to connect to it. fabfile.py is a Fabric file, pre-loaded with authentication information and with a few sample commands. $ fab uptime runs the uptime command on all hosts. ssh-config is a standard ssh configuration file for the nodes in the slice, with helpful shorthand access. $ ssh -F ./.ssh-config ig-uwashington logs in to the GEE node at the University of Washington.

The three .yaml files are Ansible playbooks. The first is simply an example playbook, which runs the ping module on every node in the slice. It verifies connectivity for the user, offers a very simple example of how to write playbooks for this infrastructure, and creates a local Python file with a dictionary of hostnames and ip addresses for subsequent use by Python programs running locally or in the slice. ansible-playbook -i ./ansible-hosts slice-hosts.yaml creates this file, slice-hosts.py. The second playbook installs the Beanstalk[37] client on every node in the slice; the third installs a Beanstalk server on every node chosen as a message_server in the Ansible playbook.

We describe Beanstalk in more detail below. Here, we focus on the architecture of providing the end user with scripts to install optional services rather than pre-configuring them in the slice. First, this keeps slices lean: if a user doesn’t want a service, she simply doesn’t run the script. Second, this makes the slices highly customizable: the user is free to make her own choices for a messaging server rather than taking ours. Third, it makes adding services a lightweight and easy process: we simply add an Ansible playbook to the slice tarball.

B. PlanetIgnite User/Application Deployer

One of the motivating factors for PlanetIgnite was to broaden the footprint of US Ignite, and make it much easier for communities to use GENI services and offer instantly-deployable GENI applications. A key part of this integration is with the Ignite Smart Gigabit App Store. A PlanetIgnite User/Application deployer follows a similar path to a GEE Developer through allocating a slice. However, once the slice is allocated, he deploys an application simply by choosing a pre-packaged application from the Ignite Gigabit App Store.

1) Smart Gigabit App Store: A smart and connected communities end user can also start an application on GENI or PlanetIgnite through the US Ignite Smart Gigabit App Store. The Gigabit App Store’s graphical user interface allows them to choose an application to invoke. The App Store will then collaborate with GEE to instantiate that application on one of the nearest eligible GENI or PlanetIgnite VM instance and provide a link to its browser control port. The browser may provide the application as a high-bandwidth and/or low-latency web app. Alternatively, for some applications, the URL returned will be a URL for a RESTful API instance called by a client application. Finally, for applications using the security and protection of network slicing, the URL will contain the information needed to establish an encrypted tunnel to the slice, or, the slice itself may appear at an end-point specified by the user during the application invocation when slicing can be extended to the end-user. The US Ignite Gigabit Application Store will be built on top of the existing Collaborative Community Exchange (CCX) which can be seen in operation at https://us-ignite.org/apps. Instead of getting only a description of the application, a button or link will allow the application to be invoked for the user. During the invocation time, the user will be informed of the slice name being created and its expiration date and time and the specific PlanetIgnite node on which it is being instantiated.

Like GENI, the US Ignite Smart Gigabit App Store will provide single sign-on to apps via Shibboleth[25] and InCommon[17]. Citizens of US Ignite testbed communities will be given extended credentials upon application to their Community Coordinator or designee, and may have to provide evidence of their identity. This approach will automatically allow most college and university faculty,
staff, and students to have access to the Gigabit App Store since they already have InCommon credentials. The CCX currently federates with Mozilla Persona [26] for identity management and will transition to InCommon when Persona is decommissioned in November.

C. PlanetIgnite Node Site

PlanetIgnite is a self-configuring programmable and sliceable server infrastructure designed to self-assemble an interoperable and interconnected software-defined locavore applications platform. GENI and PlanetIgnite servers are in-community and will have direct access to gigabit access networks and services with no backhaul charges. By keeping bits in-community, ultra-low-latency apps will become possible. And communities can become more digitally self-sufficient by bringing the cloud to the community instead of running their apps in a distant cloud.

A PlanetIgnite node can be installed anywhere using a standard open source image. Communities can add PlanetIgnite facilities as they can afford them. Local companies or universities may turn over some of their own VMs to PlanetIgnite instances during off-peak hours. Individual donors can put up one or more servers on their gigabit access networks to help serve the community. Charities might contribute resources to keep gigabit apps of use to low-income and underserved communities available. Libraries may choose to run PlanetIgnite facilities to serve their own patrons, either in or outside the library. PlanetIgnite nodes can be projected into low-income neighborhoods to bridge the digital divide.

PlanetIgnite will reduce the cost of entry for a community to add a platform to execute server apps at a location near the user where there is a clear gigabit path to the user and ultra-low latency. Where the access infrastructure permits, network slicing will be used to provide a protected path for public safety, medical, or sensitive information. Where the access structure does not have this capability built-in, encrypted channels will be used instead until a full sliced infrastructure becomes available.

Mechanically, to become a PlanetIgnite site, a community will simply go to the PlanetIgnite portal and fill out a short web form, download the PlanetIgnite VM image and a boot script derived from the web form. The boot script will contain site configuration parameters. The node will automatically register with PlanetIgnite. Qualification test will be done automatically.

III. THE PLANETIGNITE NODE IMAGE

The PlanetIgnite node image is an Ubuntu 14.04 node image which uses Docker as a container manager. An ongoing issue with PlanetLab, one of our immediate predecessors, is that the set of virtual machine images that may currently be instantiated by users on PlanetLab is limited to those that have been manually installed by PlanetLab’s administrators. Creating new images on PlanetLab is not an automated process, and installing new images requires manual intervention. As the set of available images is limited, they often differ from the operating system that a developer is using locally, leading to software compatibility issues when a service is deployed from a local setting to a distributed PlanetLab setting. For example, the stock images available on PlanetLab often lack the specific version of libraries or Java Development Kit (JDK) that a PlanetLab user is expecting. Pushing new images with software patches, such as security updates, is seldom done due to the manual intervention required with image management. Researchers have consistently requested a way of custom-deploying their own filesystem images with the latest Linux distributions and flexible control.

Docker[24] is an open platform for building, shipping, and running distributed applications based on LXC[21]. Docker supports a layered image structure where templates are unioned together to create an image. This allows for easy and space-efficient extension of existing images to form new images. Users can locally instantiate Docker images, so their local environment may be identical to the PlanetIgnite environment, eliminating software compatibility issues. A wide selection of existing Docker images is available, leading to a useful starting point for PlanetIgnite users and developers.

Docker includes a registry that stores images, facilitating both storage of private images and sharing of public images. This eliminates the step where PlanetIgnite staff is required to manually install and deploy new images. Image creation with Docker is sufficiently lightweight that it also may solve the software distribution problem, in the respect that a PlanetIgnite user may build custom packages directly into his or her Docker image. By making the image update and deployment process less painful, the frequency and likelihood that new images will be created with patches and security updates will increase.

We leverage Docker on PlanetIgnite to automate image building and distribution tasks. As an initial step we will allow users to select from a number of Docker images that have been vetted by PlanetIgnite staff as providing environments sufficient for the majority of users (e.g., for major distributions like Ubuntu and CentOS). Ultimately we will enable users to supply their own custom Docker images.

We have successfully used Docker in PlanetIgnite’s immediate precursor and prototype, the GEE. We use Docker as the container manager service on the slice. Docker is essentially an overlay on a Linux container solution, either using libvir[20] and LXC or using the built-in libcontainer library. Despite its relative youth – the first release was in March of 2013 – it has become an extremely popular virtualization solution, with over 16,000 deployed images on DockerHub. Its primary use is to provide isolation for multiple processes running within a virtual machine, and this has been responsible for most of its uptake. Docker’s web page advertises that “Dockerized’ apps are completely portable and can run anywhere” but currently support is limited to Linux. A Dockerized application is independent of the underlying
flavor of Linux. Each Docker “virtualized application” carries only its libraries, without an underlying guest OS. This gives significant size savings. The Ubuntu 14.04 Docker container is about 255 MB, compared to at least 1 GB of disk space for an Ubuntu VM.

A. Monitoring With collectd

Monitoring is essential to determine the performance and correct functioning of a Slice. Historically, PlanetLab has provided the CoMon tool[29] in order to return information about Slices, such as the amount of memory used, number of processes running, CPU utilization, bandwidth, and so on. CoMon has been decommissioned and this service is no longer available to PlanetLab users, leaving a void to be filled.

The collectd tool[16] is a mechanism for collecting performance data. The focus of collectd is on modularity and expandability. Collection is done via plugins; a robust set of plugins is already available. New plugins may be written and deployed to expand the scope of collection, allowing collectd to be extended to collect PlanetLab-specific performance data. In addition to collecting data, plugins are also used to store and push data, and collectd includes a Network Plugin that may be used to push data to a central server. This provides us with an end-to-end system from data collection on the individual nodes to storage and aggregation on a central server. Another technology, that may be used either in conjunction with collectd or used on its own, is BigQuery[38]. BigQuery is an append-only database service. Its particular focus is on massive datasets, making it an optimal choice for storing data collected by large sets of distributed machines. It supports an SQL-like language for querying the database, allowing users to easily craft tools for generating custom reports. BigQuery is integrated with Google App Engine and Google Spreadsheets, further facilitating reporting of data. By outsourcing the data storage to a distributed service like BigQuery, we gain the advantage of scalability while avoiding the pitfall of having to maintain a reliable scalable service ourselves. We have prototyped a distributed collection system using BigQuery, and a set of views for data analysis. We propose to build a CoMon replacement using collectd and BigQuery.

IV. THE PLANETIGNITE BACK END

Though Docker is primarily used in the enterprise IT space to scale individual applications seamlessly within a VM, the functions of the Docker Engine are quite similar to those of the Node Manager of PlanetLab. Its principal functions are to instantiate and deploy containers and populate them with images. It was easily adapted to managing a PlanetLab-style multi-tenant container node.

The Docker Engine comes in two parts: an on-node Docker daemon, which creates, manages, and destroys the containers, and populates them with images; and a client that issues Docker commands to the daemons. Our base installation for a node image is a GEE-customized version of an Ubuntu-based Docker image, available on DockerHub at gee-project/phusion-baseimage. We use Ansible playbooks as the interface to Docker to create and delete containers and build the slice zip file from templates.

The value of Ansible and Docker was easy to see: the Ansible slice-creation YAML file was only 57 lines of markup, and the script created the slice tarball was only 18 lines of bash.

This remarkable economy is also due to our ability to configure slivers post-instantiation through the use of Fabric and/or Ansible commands and scripts. To install the GEE Message Service we wrote a Fabric command which installed the appropriate server package, started it, and installed the Python client libraries on the hosts. This combination of three tools – Docker for slice management and image manipulation, Ansible for slice creation and post-creation customization, and Fabric for post-creation customization and experiment control – led us to name this the Fabric, Ansible, Docker (FAD) architecture for embedded distributed infrastructures. A second simplification is due to the embedded nature of the GEE. Since the GEE is embedded, its containers run in VMs allocated by the underlying infrastructure. Connectivity to the VMs is maintained by the underlying infrastructure, relieving the GEE from maintaining and repairing this connectivity.

A. The PlanetIgnite Portal

The interface for a user to create and manage slices is through the GEE Portal, at http://www.gee-project.org. The Portal itself runs in two Docker containers inside a VM on the Stanford VICCI[31] cluster. We use Docker both for its convenience as an execution environment and to gain hands-on experience with features such as inter-container networking, which we will employ for services deployed in slices.

The first container has a Mongo [33] database, which is used to register users, slices, and slice manipulation requests. No credential information for the user is stored; the only records are the user name, email, and the slice, if any, which he has created. In addition to its usual tasks, the database is designed to be an intermediate representation for stateful processes, primarily slice creation and deletion. When a user makes a slice request (other than renewal, which is handled entirely by the database itself), the portal issues a request into the database which a daemon process subsequently services; the slice status is kept in a database field. This architecture was chosen to permit the portal to respond instantly to a user request, without waiting for back-end processes to complete. The second VM contains the webserver and associated scripts. Database requests are made through the networking architecture of Docker, and the connections are made at boot time for the two containers. Use of Docker within a VM has had a number of benefits, in addition to familiarizing us with the slivers’ execution environment. The first is that we are able to use the portal VM itself as a test system. We actually maintain two sets of Docker containers, one for test and one for production, and use other Docker-based hosts on the VICCI cluster as a test production system. This has meant that any enhancements to or tests of the portal can
be run in a nearly-perfect *in situ* environment, leading to rapid debug and reliability cycles.

1) The Developer and User Facing Portal:

Authentication and User Access: Authentication and user access were questions that we considered carefully. We wanted to offer the GEE to any user with GENI access, without maintaining a separate database of authentication information. This was chosen for reasons of user convenience, maintainability, and user security. Users, once they have registered with GENI, should not need to add themselves to a separate database. Further, delegating authentication promotes maintainability, and not keeping user authentication information afforded attackers one fewer place to obtain ssh keys and passwords. To authenticate users we used an OpenID callback to the GENI portal, obtaining the minimum information needed to create and maintain user slices—the user’s email address, which was the only indexing information used in the GEE portal database.

Optional Pre-Allocation: The “five-minute rule” has dominated our design consideration. Delay in use of PlanetLab slices after allocation was due to sliver configuration and key propagation. This is a much more rapid process in the FAD-based GEE, but it is still nonzero; further, a number of scenarios (such as, for example, use in tutorials) envision the creation of multiple slices more or less simultaneously. We serialize slice creation requests, to avoid excessive network traffic to the GEE nodes, using a daemon on the GEE portal to continuously service incoming requests. Since slice creation is serialized and creation of each slice takes on the order of tens of seconds, we optionally maintain a bank of pre-created slices as a buffer against heavy node creation time.

Use-once Keys: We used a use-once, or “burner” key for two reasons: speed and security. Speed is obvious: we have pre-propagated the key. Security is nearly as obvious: if a user’s slice is compromised, or the use-once key is discovered, all that is compromised is the user’s slice. The GEE portal retains no credential from the user, and so cannot compromise any user credentials. Similarly, compromise of a user’s ssh key won’t result in an attacker gaining access to a GEE slice.

Use-once keys are the infrastructure equivalent of hotel room cardkeys; they are allocated when the slice is instantiated, used only to access the slice, and are destroyed when the slice is de-allocated. As a result, they come with fewer security concerns than do standard keys, just as a hotel is completely unconcerned with travelers departing with cardkeys in their pockets.

The Site Facing Portal: The Site-Facing Portal is designed to permit sites to easily register as a PlanetIgnite site and automatically download the image and install it on a VM. While the portal will export a visual interface for manual entry, registration will primarily be done automatically through the downloaded and installed image.

When a PlanetIgnite image comes up on a local site, it will offer two configuration options. The first is a simple boot script, which takes as an argument the routable IP address of the VM, configures the local network interfaces, and then uses a REST http interface to automatically register the image with the portal. The portal not only register the image but invokes, through the local node manager, a series of tests which determine whether the node is up, functioning, and not behind a firewall which blocks access to required ports. If the node passes these tests, it is added to the database of available PlanetIgnite nodes.

V. The PlanetIgnite Developer Services

PlanetIgnite will follow the GENI Experiment Engine in providing a number of services to developers on the PlanetIgnite nodes. Here, we detail the initial services.

A. The PlanetIgnite Slice Control Service

The foundational service is the PlanetIgnite Slice Control Service, which permits PlanetIgnite slices to be configured and controlled through a single pane of glass. It is the foundational service because we leverage this service to bootstrap the other services that we offer.

Scalability of control for a distributed application is critical. Slice management and configuration was the focus of a large number of early PlanetLab efforts[1], [10]. Despite a number of early efforts for unified desktop orchestration, most early experimenters used a combination of Perl, ssh, and Python scripts for experiment orchestration and control. The emergence of Cloud platforms and the need for scalable orchestration, configuration and management of very large-scale systems has given rise to a number of open-source and commercial tools for these purposes. We use two as the basis for the Slice Control Service, Fabric and Ansible. Both Fabric and Ansible employ Python wrappers over ssh. As with most configuration management and orchestration software, both distinguish between controllers and nodes. A controller executes configuration commands to configure the nodes. Both are agentless: they require no agent on the nodes themselves. Fabric requires only OpenSSH on the nodes; Ansible requires both OpenSSH and Python 2.4 or later.

Fabric is a set of Python libraries which wrap sftp for file transfer and OpenSSH for command execution. As this implies, it offers an imperative semantics for node orchestration and configuration management. Ansible also offers a declarative semantics for known tasks in its Playbook abstraction.

Both Ansible and Fabric have roles to play in coordinating wide-area experiments and distributed applications. Ansible requires installation of Python-based software on the desktop; in contrast, Fabric requires only the installation of a Python library through pip or easy_install. Our solution was to support both, through the definition of skeleton files which incorporated slice information and rudimentary commands, making it easy for experimenters to extend.

The inclusion of Ansible and Fabric in our workflow turned out to have substantial benefits for Slice deployment and configuration, and significant simplification
of both the core of the GENI Experiment Engine and deployed slices. Rather than pre-installing a great deal of software on the experiment nodes, we could simply incorporate the relevant Ansible or Fabric commands in the files we downloaded to the user. This insight led to the fundamental idea of the Slice Control Service as the foundation on which the other services could be bootstrapped. Rather than pre-loading services into slices, or building services slices, we could simply build Fabric commands or Ansible playbooks to instantiate and install the service. We have tested this approach with the Slice Message Service.

B. The Slice Message Service

The Slice Message Service is used to route job control messages within a slice; this is a common feature of many Cloud systems, and a number of systems are available. The Message Service is a server which can be loaded into the slice, and a client library; a user activates the server on whichever nodes in the slice she prefers through a Fabric command. We searched for a message service that is well-documented, simple, configures automatically, has a rich set of client libraries, and can be enabled with a service start command.

We chose Beanstalk[37]. Beanstalk has libraries in a variety of languages, notably including Python. It installs as a service on Ubuntu, with a configurable port. It has a simple put/get interface and supports a wide variety of use models, including pub/sub.

As with many Message Service systems, Beanstalk is configured for a single-tenant environment. Its primary use case is to coordinate tasks within a data center. Its use mode is not a multi-tenant provider who offers messaging-as-a-service, like IronMQ[18], but rather that each job or service instantiate its own messaging server accessible only from its own nodes: security is assumed at the slice level. This dictated our deployment choice: rather than instantiating a GEE-wide messaging service, we offer the developer an Ansible playbook to turn the service on in the appropriate slivers, and choose the appropriate server site.

C. The Slice Storage Service

PlanetLab users have historically relied on multiple community provided tools and services for deploying data to slivers and getting data back from experiments. These include CoDeploy[28], Stork[10], PLuSH[1], and a variety of ad-hoc ssh scripts. To simplify this common use-case (as well as replace tools that are no longer maintained), we will use Syndicate[27] to give each slice a shared, read-write private storage volume and make available public read-only volumes of popular datasets.

Syndicate is a scalable software-defined storage system, under development at Princeton, that combines existing CDNs and cloud storage into a coherent, wide-area read/write storage medium. With Syndicate, we will augment existing PlanetLab-hosted CDNs (like Coral and CoBlitz[30]) with commodity cloud storage (like Dropbox and Amazon S3) to give each slice a shared private storage space. Syndicate readers pull data from one another via the CDNs, thereby scaling up aggregate read bandwidth beyond what a single sliver can provide. Syndicate writers push data into commodity cloud storage to keep it highly available in the face of node failures. To keep slice data secure, Syndicate implements end-to-end encryption and cryptographic signatures to guarantee data confidentiality, integrity, and authenticity.

Deploying and gathering data with Syndicate is straightforward. Syndicate’s client is a FUSE file system, so users deploy data to slivers simply by copying data in, and retrieve sliver-generated data simply by copying data out.

PlanetLab users have frequently requested a global filesystem for sharing data. In addition to moving data between slivers, we will use Syndicate to expose existing, public datasets and repositories as read-only volumes. This will save users the time and effort of having to push data to upwards of 1000 nodes, while ensuring that slivers always have access to it even if they are re-imaged. For example, we will use Syndicate to expose geo-IP databases, software repositories, and public scientific datasets as mountable volumes.

D. The Slice Reverse Proxy Service

Intra-slice traffic on the PlanetIgnite will primarily be through a network private to the slice. Routable IPs are notoriously scarce on PlanetLab nodes, and GENI member institutions have been unable to devote large banks of routable IP addresses to GENI slices. Our goal is to permit sites to easily download and instantiate PlanetIgnite nodes. If a typical site has an eight-core dual-socket node - a typical InstaGENI worker node - we should be able to accommodate between 80 and 160 slivers on this node. This will require a /25 or /24 to give each sliver a routable IP, and many of our candidate sites will not have a /24 sitting in their hip pocket to hand to us. Clearly, we cannot count on being able to give a routable IP to each sliver.

Though the private network suffices for intra-slice traffic, a number of PlanetLab slices and services offered distributed public services. Clearly, for such services to use PlanetIgnite, some method must be found to enable public-facing services at each site.

We don’t have enough IP addresses to offer each public-facing service its own routable IP, and it isn’t really feasible to assign each its own port: an HTTP service that isn’t on port 80 faces multiple logistical problems, from firewalls to configuration of client-side software. IPv6 is the obvious solution, but it isn’t implemented on many campuses.

If our developers are to offer public-facing services on GEE nodes, we must find a way to give them all access to the same port on the same v4 address. The solution we hit upon was to multiplex the HTTP ports and isolate at the URL level using the PlanetIgnite Reverse Proxy. The Reverse Proxy Service operates a reverse proxy in a sliver on each GEE site. HTTP PUT, GET, and POST requests of the form http://<hostname>/<slicename>/<request>
are caught by the reverse proxy and sent to the http server in the slice’s sliver over the GEE private network; the returned value is sent back to the requester.

VI. CONCLUSIONS

PlanetIgnite is currently near-deployment. We plan a graceful transition from the current GEE, using the PlanetIgnite technology themselves to automatically deploy PlanetIgnite nodes onto ExoGENI and SAVI slices, Chameleon and CloudLab nodes. Once we have this process smoothly automated, we will transition this onto a subset of PlanetLab’s existing nodes, then offer this to early beta communities.

At the end of the day, PlanetLab, GENI, the GENI Experiment Engine, and PlanetIgnite are about an idea: it should be as easy for a developer to ship a program to a computational element as it is for a user to download data over the Internet today. In this work, we have largely solved the major technological problem in doing this: we have presented the developer with an homogeneous execution environment across the wide area, much as cellphone OS’s created a homogeneous, ubiquitous client. This is, however, only half the battle: what remains is the far more challenging enterprise of persuading sites to permit untrusted third parties to run programs on their sites: in effect, to ask each site to host a node on a nationwide distributed Cloud.

Solving this problem involves a threefold strategy: increasing the value of hosting a Cloud site to the host institution; radically reducing the cost of hosting a Cloud site; and developing the web of agreements, acceptable use policies, and liability structures to reduce the risk to the host institution of hosting a Cloud site.

This paper is targeted at the second element, much as our companion paper[15] addressed the first. We do not neglect the third. At the moment, we use the GENI Acceptable Use Policy, but will work with sites and our partners to derive one suitable for a worldwide infrastructure.

We are committed to offering these services wherever a Docker node can be instantiated. Our view of the cloud is that it should reach as close to the edge as is feasible given the underlying implementation technology. As a result, we have presented the developer with an homogeneous execution environment across the wide area, much as cellphone OS’s created a homogeneous, ubiquitous client.

ACKNOWLEDGEMENTS

The authors thank Leigh Stoller and Rob Ricci of the University of Utah and Niyi Riga and Mark Berman of the GPO for much logistical assistance in setting up and maintaining the GEE Slicelets; Marshall Brinn of the GPO and Nick Bastin of Barnstomer Softworks for productive conversations; Chip Elliott of the GPO for years of guidance; Joe Kochan and the staff at US Ignite; the technical team at US Ignite, particularly Nishal Mohan and Scott Turnbull; and Jack Brassil of the InstaGENI project and HP Labs for invaluable help. This project was partially funded by the GENI Project Office under subaward from the National Science Foundation.

REFERENCES


Stochastic Dynamic Cache Partitioning for Encrypted Content Delivery

Andrea Araldo†, György Dán‡, Dario Rossi†
† LRI, Université Paris Sud, Orsay, France – araldo@lri.fr
‡ School of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden – gyuri@ee.kth.se
† Télécom ParisTech, Paris, France – dario.rossi@telecom-paristech.fr

Abstract—In-network caching is an appealing solution to cope with the increasing bandwidth demand of video, audio and data transfer over the Internet. Nonetheless, an increasing share of content delivery services adopt encryption through HTTPS, which is not compatible with traditional ISP-managed approaches like transparent and proxy caching. This raises the need for solutions involving both Internet Service Providers (ISP) and Content Providers (CP): by design, the solution should preserve business-critical CP information (e.g., content popularity, user preferences) on the one hand, while allowing for a deeper integration of caches in the ISP architecture (e.g., in 5G femto-cells) on the other hand.

In this paper we address this issue by considering a content-oblivious ISP-operated cache. The ISP allocates the cache storage to various content providers so as to maximize the bandwidth savings provided by the cache: the main novelty lies in the fact that, to protect business-critical information, ISPs only need to measure the aggregated miss rates of the individual CPs and do not need to be aware of the objects that are requested, as in classic caching. We propose a cache allocation algorithm based on a perturbed stochastic subgradient method, and prove that the algorithm converges close to the allocation that maximizes the overall cache hit rate. We use extensive simulations to validate the algorithm and to assess its convergence rate under stationary and non-stationary content popularity. Our results (i) testify the feasibility of content-oblivious caches and (ii) show that the proposed algorithm can achieve within 10% from the global optimum in our evaluation.

I. INTRODUCTION

It is widely known that content delivery over the Internet represents a sizeable and increasing fraction of the overall traffic demand. Furthermore, most of the content, including video, is carried over HTTP connections; this evolution of the last decade was not among those forecasted for the IP hourglass model evolution [1], and is rather a choice of practical convenience. This evolution has a tremendous practical relevance, to the point that HTTP was newly recognized and proposed [2] as the new de facto “thin waist” of the TCP/IP protocol family.

In very recent times, we are on the verge of yet another shift of the thin waist: we indeed observe that the fraction of traffic delivered through HTTPS has already passed 50% [3], and it is expected to increase, as the IETF Internet Architecture Board (IAB) recommends “protocol designers, developers, and operators to make encryption the norm for Internet traffic” [4]. Besides the IAB recommendation, Content Providers (CP) are already heavily relying on encryption to both protect the privacy of their users, as well as sensitive information (related to user preferences) of their own business.

This evolution toward an all-encrypted Internet creates a tussle between security and efficiency. Today’s Internet heavily relies on middleboxes such as NATs (to combat the scarcity of IPv4 addresses) and transparent or proxy caches [5] (to relieve traffic load). However, some of these middleboxes will simply fail to operate in tomorrow’s Internet with end-to-end encryption: for example, end-to-end encryption renders caching useless, since multiple transfers of the same object generate different streams if the same object is encrypted with different keys. At times where the design of the new 5G architecture strives to reduce latency, increase the available bandwidth and better handle mobility, this tradeoff is especially unfortunate, as distributed caches represent a natural way to reduce latency, reduce bandwidth usage and to cope with mobility avoiding long detours to anchor points.

This architectural evolution calls for a redesign of the current operations involving both Internet Service Providers (ISP) and Content Providers (CP): by design, the solutions should preserve business-critical CP information (e.g., content popularity, user preferences) on the one hand, while allowing for a deeper integration of caches in the ISP architecture (e.g., in 5G femto-cells) on the other hand.

In this paper we address this issue by proposing a content-oblivious algorithm that manages the storage space of an ISP cache that delivers encrypted content: the algorithm dynamically partitions the cache storage among various CPs so as to maximize the cache hit rate (hence, the bandwidth savings). The most important feature of the algorithm is that in order to protect business-critical information of the CPs, the ISP only needs to measure the aggregated miss rates of the individual CPs. We prove that relying on the aggregated miss rates only, our algorithm converges close to the optimal allocation that maximizes the overall cache hit rate, and provide a bound on the gap compared to the optimal allocation. Extensive simulations under realistic scenarios show the feasibility and good performance of the proposed algorithm.

The rest of the paper is organized as follows. Sec. II reviews related work. Sec. III describes the system model and presents our cache partitioning algorithm. Sec. IV provides analytic results, and Sec. V evaluates the performance of the algorithm. Sec. VI concludes the work.
The problem of cache partitioning has first been considered in the context of partitioning CPU cache among competing processes [6]. Clearly, the cache workload created by CPU jobs differs very much from the characteristics of traffic requests from a user aggregate, so that our technique significantly differs in their design choices: i.e., there is no notion of privacy within the CPU that would force the scheduler to be agnostic of the instruction patterns, unlike in our case.

Related to our work are recent works in the area of in-network caching, which is an established technique that continues to receive significant interest in recent years [7]. They generally consider the cache as a whole, but for a few exceptions [8], [9] that partition a cache among different applications and video quality levels, respectively, and assume that the served content is observable by the ISP. Our work differs from these works, as we consider a cache partitioned among CPs, and the ISP can observe the aggregate miss rates of the CPs only. A recent work [10] proposes to share a cache managed by an ISP among different CPs in order to achieve fairness. Partitioning of the storage is done using a pricing scheme based on the value that each CP gives to cache space. Unlike [10], our proposed algorithm aims to maximize the cache hit rate, and does not involve payments, which may make its adoption less controversial than the one of the ISP. Our work is the first to propose an effective way to maximize the efficiency of an ISP-managed cache, by partitioning it among multiple CPs, without being aware of the content that the CPs serve and without involving payments.

Closer in scope to ours are recent works that target ISP/CDN cooperation [12], [13], [14], [15]. These not only show that ISPs have strong incentive in investing in caching to reduce the traffic on their critical paths, but also show that the other Internet actors, i.e., CPs and users, would benefit from ISP in-network caching. The game theoretical study in [12] shows that caches are inefficient when operated by CPs, since CP content placement and ISP traffic engineering are often not compatible. Solutions are proposed in [13], [15], which however require ISPs to share with the CP confidential information, such as topology, routing policies or link states, and as such are arguably highly impractical. Conversely, [14] fosters an ISP-operated cache system, but requires the ISP to be able to observe every object requested by the users, which is arguably equally impractical since CPs purposely hide this confidential information via HTTPS. In contrast with these previous works, our solution does not yield to any leaking of business critical information. Furthermore, our solution is not limited to a single CP, unlike [13].

Finally, from a technical viewpoint, our work is aligned with recent industry efforts in the Open Caching Working Group (OCWG) [16]. The mission of the OCWG is to develop standards, policies and best practices for a new layer of content caching within ISP networks, which can coexist with HTTP/3 and provide shared access to storage for many CPs. Our work fits the OCWG requirements and as such is, we believe, of high practical relevance.

### III. Stochastic Dynamic Cache Partitioning

#### A. System Model and Problem Formulation

We consider a cache with a storage size of $K$ slots (e.g., in units of MB) maintained by an operator and shared by $P$ content providers (CPs). The operator is not aware of what content the individual slots store and only decides how to partition the slots among CPs.

We denote by $\theta_p \in \mathbb{Z}_{\geq 0}$ the number of cache slots allocated to CP $p$, which it can use for caching its most popular contents. We define the set of feasible cache allocation vectors

$$\Theta \triangleq \{ \theta \in \mathbb{Z}_{\geq 0}^P \mid \sum_{p=1}^P \theta_p \leq K \} \subset \mathbb{Z}_{\geq 0}^P. \tag{1}$$

We consider that the arrival of requests for content can be modeled by a stationary process, and the number of arrivals over a time interval of length $T$ can be bounded by some positive constant $A(T)$. This assumption is reasonable as requests are generated by a finite customer population, and each customer can generate requests at a bounded rate in practice. Upon reception of a request for a content of the CPs that share the cache, the request can either generate a cache hit (for content stored in the CP partition at time of the request) or a cache miss (otherwise). Formally, we denote the expected cache miss rate (i.e., expected number of misses per time unit) of CP $p$ when allocated $\theta_p$ slots of storage by $L_p(\theta_p)$. We make the reasonable assumption that $L_p$ is decreasing and strictly convex on $[0 \ldots K]$. This assumption corresponds to that having more storage decreases the miss intensity (in expectation) with a decreasing marginal gain, and each CP would in principle have enough content to fill the entire storage. For convenience we define the cache miss intensity vector

$$\bar{L}(\theta) \triangleq (L_1(\theta_1), \ldots, L_P(\theta_P))^T.$$  \tag{2}

Finally, we define the overall expected cache miss intensity

$$L(\theta) \triangleq \sum_{p=1}^P L_p(\theta_p). \tag{3}$$

Motivated by the increasing prevalence of encrypted content delivery, we assume that the operator cannot observe what content an individual request is for, but it can observe the number of content requests received by a CP and the corresponding number of cache misses.

Given a static cache partitioning $\theta \in \Theta$, the observed number of content requests and the number of cache misses would form a stationary sequence when measured over subsequent time intervals. The objective of the operator is to find the optimal allocation $\theta^{OPT}$ that minimizes the overall expected cache miss intensity, i.e.,

$$\theta^{OPT} \in \arg \min_{\theta \in \Theta} L(\theta), \tag{4}$$

based on the measured cache miss intensity. In what follows we propose the Stochastic Dynamic Cache Partitioning (SDCP) algorithm that iteratively approximates the optimal allocation.
### Stochastic Dynamic Cache Partitioning (SDCP) Algorithm

The proposed SDCP algorithm is an iterative algorithm that is executed over time slots of fixed duration $T$. The pseudo code of the algorithm is shown in Alg. 1. For simplicity we present the algorithm assuming that $P$ is even, but the case of an odd number of CPs can be handled by introducing a fictitious CP with zero request rate. At time slot $k$ the algorithm maintains a virtual cache allocation $\theta^k$. The virtual allocation is an allocation of $K'$ storage slots among the CPs, i.e.,

$$\theta^k \in \mathcal{C} \triangleq \left\{ \theta \in \mathbb{R}^P | \theta \in K' \right\},$$

where $K' = K - P/2$. (6)

We will justify the introduction of $K'$ and of $\mathcal{C}$ in the proof of Lemma 6.

In order to obtain from $\theta^k$ an integral allocation that can be implemented in the cache, we define the center-point function $\Gamma : \mathbb{R}^P \rightarrow \mathbb{R}^P$, which assigns to a point in Euclidean space the center of the hypercube containing it, i.e.,

$$\Gamma(\theta) \triangleq (\gamma(\theta_1), \ldots, \gamma(\theta_P))^T, \quad \forall \theta \in \mathcal{C},$$

where we use $\lfloor \cdot \rfloor$ to denote the floor of a scalar or of a vector in the component-wise sense. Furthermore, we define the perturbation vector $D^k = (D_1^k, \ldots, D_P^k)^T$ at time slot $k$, which is chosen independently and uniformly at random from the set of $-1, 1$ valued zero-sum vectors

$$D^k \in Z \triangleq \{ z \in \{-1, 1\}^P | z^T \cdot 1_P = 0 \}.$$ (8)

Given $\Gamma$ and $D^k$ the algorithm computes two cache allocations to be implemented during time slot $k$,

$$\theta^k \triangleq \frac{1}{T} D^k,$$

$$\bar{\theta}^k \triangleq \frac{1}{T} \hat{\theta}^k - \frac{1}{T} \frac{1}{\hat{\theta}^k} D^k.$$ (9)

The algorithm first applies allocation $\theta^k$ for $T/2$ amount of time and measures the cache miss rate $\gamma^k$ for each provider $p = 1, \ldots, P$. It then applies allocation $\bar{\theta}^k$ during the remaining $T/2$ amount of time in slot $k$ and measures the cache miss rates $\gamma^k$. The vectors of measured cache misses $\gamma^k \triangleq \gamma^k, \ldots, \gamma^k$ and $\gamma^k \triangleq \gamma^k \ldots, \gamma^k$ are used to compute the impact $\delta y^k \triangleq \gamma^k - \gamma^k$ of the perturbation vector on the cache miss intensity of CP $p$, or using the vector notation $\delta y^k \triangleq \gamma^k - \gamma^k$.

Based on the measured miss rates, the algorithm then computes the allocation vector $\theta^{k+1}$ for the $(k+1)$-th step. Specifically, it first computes (line 11, where $\sigma$ denotes the Hadamard product) the update vector $\delta \theta^k$, which we show in Cor. 12 to match in expectation a subgradient of the miss intensity vector (Lemma 8).

We first provide definitions and known results (Sec.IV-A) and a bound on the optimality gap (Sec.IV-D).
A. Preliminaries

Let us start by introducing the forward difference defined for functions on discrete sets.

**Definition 1.** For a function \( F : \mathbb{Z}^n \rightarrow \mathbb{R}^q, q_1, q_2 \geq 1 \) the forward difference is

\[
\Delta_n F(x) \triangleq F(x + n \cdot 1_{q_1}) - F(x), \forall x \in \mathbb{Z}^n, n \in \mathbb{Z} \setminus \{0\}.
\]

By abuse of notation, we will simply use \( \Delta F(x) \) to denote \( \Delta_1 F(x) \).

The forward difference is convenient for characterizing convexity using the following definition [17].

**Definition 2.** A discrete function \( F : \mathbb{Z} \rightarrow \mathbb{R} \) is strictly convex iff \( x \rightarrow \Delta F(x) \) is increasing.

Furthermore, for a class of functions of interest we can establish the following.

**Lemma 3.** Let \( F : \mathbb{Z} \rightarrow \mathbb{R} \) decreasing and strictly convex, \( x \in \mathbb{Z} \) and \( n \in \mathbb{Z} \setminus \{0\} \) we have

\[
\Delta_n F(x) > n \Delta F(x). \tag{11}
\]

**Proof:** We first show that \( \forall x, y \in \mathbb{Z} \) such that \( y > x \), the following holds

\[
\Delta_n F(y) > \Delta_n F(x) \quad \text{if } n > 0, \tag{12}
\]

\[
\Delta_n F(y) < \Delta_n F(x) \quad \text{if } n < 0. \tag{13}
\]

For \( n > 0 \) we can use Def. 2 to obtain

\[
\Delta_n F(y) = \sum_{i=0}^{n-1} [F(y+i+1) - F(y+i)] = \sum_{i=0}^{n-1} \Delta F(y+i) > \sum_{i=0}^{n-1} \Delta F(x+i) = \Delta_n F(x), \tag{14}
\]

which proves (12).

For \( n < 0 \) algebraic manipulation of the definition of the forward difference and (14) gives

\[
\Delta_n F(y) = -\Delta_{|n|} F(y-|n|) < -\Delta_{|n|} F(x-|n|) = \Delta_n F(x),
\]

which proves (13). To prove (11) for \( n > 0 \), observe that, thanks to Def. 2, each of the \( n \) terms of the last summation in (12) is lower bounded by \( \Delta F(x) \). For \( n < 0 \) via algebraic manipulation we obtain

\[
\Delta_n F(x) = -\sum_{i=1}^{n} \Delta F(x-i) > -\sum_{i=1}^{n} |n| \Delta F(x) = -|n| \Delta F(x),
\]

which proves (11) as \( |n| = -n \).

Since SDCP generates virtual configurations whose components are not necessarily integer, we have to extend the discrete functions \( L_p \) to real numbers. Thanks to Theor. 2.2 of [18], we have the following existence result.

**Lemma 4.** Given a discrete decreasing and strictly convex function \( F : \mathbb{Z} \rightarrow \mathbb{R} \), there exists a continuous and strictly convex function \( \tilde{F} : \mathbb{R} \rightarrow \mathbb{R} \) that extends \( F \), i.e., \( \tilde{F}(x) = F(x), \forall x \in \mathbb{Z}. \) We call \( \tilde{F} \) the interpolant of \( F \).

Finally, we formulate an important property of the Euclidean projection \( \varphi \).

**Lemma 5.** There is a unique function \( \varphi \) satisfying (10). Furthermore, \( \varphi \) satisfies

\[
\|\varphi(\theta) - \varphi(\theta')\| \leq \|\theta - \theta'\|, \forall \theta \in \mathcal{C}, \theta' \in \mathcal{C} \cap \mathbb{R}^p_{\geq 0}, \tag{15}
\]

i.e., \( \varphi(\theta) \) is no farther from any allocation vector than \( \theta \).

**Proof:** Observe that \( \mathcal{C} \cap \mathbb{R}^p_{\geq 0} \) is a simplex, and thus closed and convex. Hence, the Euclidean projection \( \varphi(\theta) \) is the unique solution of (10) [19]. Furthermore, the Euclidean projection is non-expansive (see, e.g., Fact 1.5 in [20]), i.e., for \( \theta, \theta' \in \mathcal{C} \) it satisfies \( \|\varphi(\theta) - \varphi(\theta')\| \leq \|\theta - \theta'\| \). Observing that if \( \theta' \in \mathcal{C} \cap \mathbb{R}^p_{\geq 0} \) then \( \varphi(\theta') = \theta' \) proves the result.

B. Consistency

We first have to prove that during each time slot the configurations \( -\theta^{(k)} + \delta \hat{g}^{(k)} \) that SDCP imposes on the cache are feasible. This is non-trivial, as the operators used in computing the allocations are defined on proper subsets of \( \mathbb{R}^p \). The following lemma establishes that the allocations computed by SDCP always fall into these subsets.

**Lemma 6.** The allocations \( \theta^{(k)} \) are consistent in every time slot, as they satisfy

(a) \( \theta^{(k+1)} \in \mathcal{C} \),

(b) \( \theta^{(k)} \in \mathcal{C} \cap \mathbb{R}^p_{\geq 0} \),

(c) \( +\hat{g}^{(k)} - \theta^{(k)} \in \Theta \).

**Proof:** Recall that \( \theta_0 \in \mathcal{C} \cap \mathbb{R}^p_{\geq 0} \). To show (a) observe that

\[
\hat{g}^{(k)} \cdot \mathbf{1}_p = \sum_{j=1}^{P} \delta y_{p}^{(k)} \cdot D_{p}^{(k)} - \sum_{j=1}^{P} \delta y_{p}^{(k)} \cdot D_{p}^{(k)} = 0, \tag{16}
\]

and thus if \( \theta^{(k)} \in \mathcal{C} \), then \( \theta^{(k)} - \delta \hat{g}^{(k)} \in \mathcal{C} \). The definition of the Euclidean projection (10) and (a) together imply (b). Finally, observe that

\[
1 \cdot \theta^{(k)} = 1 \cdot |\theta| + \frac{P}{2} \leq 1 \cdot \theta + \frac{P}{2} \leq K' + \frac{P}{2} = K, \tag{k, 17}
\]

which proves (c). Note that the above motivates the choice of \( K' \) in the definition of the set of virtual allocations \( \mathcal{C} \), as if \( K' > K - \frac{P}{2} \) then \( +\hat{g}^{(k)} - \theta^{(k)} \in \Theta \) may be violated due to the use of the mapping \( \gamma \) and \( D_{p}^{(k)} \) in (9).

C. Convergence

To prove convergence of SDCP, we first consider the relationship between the measured miss rates \( y_{p}^{(k)} \) and \( y_{p}^{(k)} \) and the expected miss intensities \( L_p(\theta^{(k)}) \) and \( L_p(\theta^{(k)}) \), respectively. We define the measurement noise

\[
+\varepsilon(k) \triangleq \theta + \hat{g}(k) - L_p(\theta^{(k)}),
\]

\[
-\varepsilon(k) \triangleq -\hat{g}(k) - L_p(\theta^{(k)}), \tag{18}
\]

and the corresponding differences

\[
\delta_\varepsilon(k) \triangleq +\varepsilon(k) - \varepsilon(k), \tag{19}
\]

\[
\delta_\varepsilon(k) \triangleq -\varepsilon(k) - \varepsilon(k).
\]
Observe that $D^{(k)}$, $\bar{\gamma}^{(k)}$ and $-\bar{\gamma}^{(k)}$ are random variables and form a stochastic process. Using these definitions we can formulate the following statement about the measured miss rates.

**Lemma 7.** The conditional expectation of the measurement noise and its difference satisfy

$$E[\xi^{(k)}(\theta)] = 0_p.$$  

(20)

**Proof:** Observe that due to the stationarity of the request arrival processes we have $E[\xi^{(k)}(\theta)] = 0$ and $E[-\xi^{(k)}(\theta)] = 0$, which due to the additive law of expectation yields the result.

Intuitively, this is equivalent to saying that the sample averages provides an unbiased estimator of the miss rates. In what follows we establish an analogous result for the update vector $\bar{\gamma}^{(k)}$ with respect to a subgradient of the interpolant $\bar{L}$ of the expected miss intensity $L$, which itself is a discrete function. We define and characterize $\bar{L}$ in the following lemma, which recalls known results from convex optimization.

**Lemma 8.** Given the interpolants $L_p$ of the expected miss intensities $L_p$ of the CPs and defining the interpolant of $L$ as $L(\theta) \triangleq \sum_{p=1}^{P} L_p(\theta_p)$, $\forall \theta \in \mathbb{R}^p$, $\bar{L}$ is strictly convex and admits a unique minimizer $\theta^*$ in $C \cap \mathbb{R}^p$.

**Proof:** Recall that each interpolant $L_p$ of $L_p$ is strictly convex as shown in Lemma 4. The strict convexity of $\bar{L}$ can then be obtained applying Theor. 1.17 of [21]. Then, we observe that $\theta^*$ is the solution to a convex optimization problem with a strictly convex objective function, which is unique (Sec. 4.2.1 of [22]).

For completeness, let us recall the definition of a subgradient of a function from this section, e.g., [23]).

**Definition 9.** Given a function $L : \mathbb{R}^p \to \mathbb{R}$, a function $\bar{g} : C \subseteq \mathbb{R}^p \to \mathbb{R}^p$ is a subgradient of $L$ over $C$ iff

$$L(\theta^*) - L(\theta) \geq \bar{g}(\theta)^T \cdot (\theta^* - \theta), \forall \theta, \theta^* \in C.$$

We are now ready to introduce a subgradient $\bar{g}(\theta)$ for the interpolant of the expected cache miss intensity $\bar{L}$.

**Lemma 10.** The function

$$\bar{g}(\theta) \triangleq \Delta \bar{L}^{(k)}([\theta]) - \frac{1}{p} \cdot \Delta \bar{L}([\theta]) \cdot 1_p$$

(21)

is a subgradient of $\bar{L}$ over $C \cap \mathbb{R}^p$.

**Proof:** Observe that for $\theta, \theta^* \in C$

$$\bar{g}(\theta)^T \cdot (\theta^* - \theta) = \Delta \bar{L}^{(k)}([\theta])^T \cdot (\theta^* - \theta) - \frac{1}{p} \cdot \Delta \bar{L}([\theta]) \cdot [1_p^T \cdot \theta^* - \theta].$$

At the same time, for $\theta, \theta^* \in C$ we have

$$1_p^T \cdot (\theta^* - \theta) = (1_p^T \cdot \theta^* - 1_p^T \cdot \theta) = K' - K' = 0.$$

Therefore, for any $\theta, \theta^* \in C$

$$\bar{g}(\theta)^T \cdot (\theta^* - \theta) = \Delta \bar{L}^{(k)}([\theta]) \cdot (\theta^* - \theta).$$

(22)

Thus, according to Def. 9, in order to show that $\bar{g}$ is a subgradient of $\bar{L}$ it suffices to show that

$$\sum_{p=1}^{P} [L_p(\theta'_p) - L_p(\theta_p)] \geq \sum_{p=1}^{P} \Delta L_p([\theta_p]) \cdot (\theta'_p - \theta_p).$$

(23)

We now show that this holds component-wise. If $[\theta'_p] - [\theta_p] = 0$, then the above clearly holds. Otherwise, if $n = [\theta'_p] - [\theta_p] \neq 0$ we apply a well known property of convex functions (Theor. 1.3.1 of [24]) to obtain:

$$\frac{L_p([\theta'_p]) - L_p([\theta_p])}{(\theta'_p - \theta_p)} \leq \frac{L_p([\theta'_p] + 1) - L_p([\theta_p] + 1)}{(n + 1 - (\theta'_p + 1))},$$

which, by Def. 1, can be rewritten as:

$$\sum_{p=1}^{P} (\theta'_p - \theta_p) \leq \sum_{p=1}^{P} (\theta'_p - \theta_p),$$

(24)

For $n > 0$ we can use the first inequality of (24) and Lemma 3 to obtain

$$L_p(\theta'_p) - L_p(\theta_p) \geq \Delta L_p([\theta_p]) \cdot (\theta'_p - \theta_p).$$

(25)

For $n < 0$ we can use the second inequality of (24) and Lemma 3 to obtain

$$L_p(\theta'_p) - L_p(\theta_p) \leq \Delta L_p([\theta_p]) \cdot (\theta'_p - \theta_p).$$

(26)

and by multiplying the first and second term of (26) by $\theta'_p - \theta_p$ (which is negative since $n = [\theta'_p] - [\theta_p]$ is negative), we obtain the result.

The subgradient $\bar{g}$ will be central to proving the convergence of SDCP, but it cannot be measured directly. The next proposition establishes a link between the update vector $\bar{g}^{(k)}$, which we compute in every time slot, and the subgradient $\bar{g}$.

**Proposition 11.** The update vector $\bar{g}^{(k)}$ is composed of the subgradient $\bar{g}$ plus a component due to the noise.

$$\bar{g}^{(k)} = \bar{g}^{(k)} + \delta k \cdot D^{(k)} = \frac{1}{P} \cdot \sum_{k=1}^{K} \bar{g}^{(k)} \cdot (\bar{L}^{(k)} - L^{(k)}) \cdot 1_p.$$  

(27)

**Proof:** We first apply (19) to obtain

$$\bar{g}^{(k)} = \frac{1}{P} \cdot \sum_{k=1}^{K} \bar{g}^{(k)} + \delta k \cdot D^{(k)} \cdot 1_p$$

(27)

Consider now a particular realization of the random variable $D^{(k)}$. We can express component $p$ of $\delta \bar{L}^{(k)}(\bar{L}^{(k)} - L^{(k)})$ as

$$I_p \left( \Pi \left( \bar{g}^{(k)} \right) + \frac{1}{2} D^{(k)} \right) - L_p \left( \Pi \left( \bar{g}^{(k)} - \frac{1}{2} D^{(k)} \right) \right)$$

Therefore, for any $\theta, \theta' \in C$

$$\bar{g}(\theta)^T \cdot (\theta' - \theta) = \Delta \bar{L}^{(k)}([\theta]) \cdot (\theta' - \theta).$$

(22)
The conditional expectation of

\[ D_p^{(k)} = -1 \]

and then assuming that it is \( D_p^{(k)} = 1 \). We thus obtain

\[ \delta L^{(k)} \circ D^{(k)} = \Delta L(\theta^{(k)}) \]

and in scalar form

\[ \delta E^{(k)} T \cdot D^{(k)} = \Delta L(\theta^{(k)}). \]

By substituting these in (27)

\[ g^{(k)} = \Delta L(\theta^{(k)}) - \frac{1}{P} \cdot \Delta L(\theta^{(k)}) \cdot 1_p \]

+ \( \delta \epsilon^{(k)} \circ D^{(k)} - \frac{1}{P} \cdot (\delta \epsilon^{(k)} \cdot T \cdot D^{(k)}) \cdot 1_p \]

and using (21), we obtain the result.

Furthermore, thanks to Lemma 7, the second term of (27), which is due to the noise, is zero in expectation, which provides the link between the update vector \( g^{(k)} \) and the subgradient \( \langle \hat{g}^{(k)} \rangle \).

**Corollary 12.** The conditional expectation of \( \hat{g}^{(k)} \) is \( E[\hat{g}^{(k)} | \theta^{(k)}] = g^{(k)} \) and thus \( \hat{g}^{(k)} \) is a stochastic subgradient of \( L \), i.e., \( E[\hat{g}^{(k)}] = g^{(k)} \).

This leads us to the following theorem.

**Theorem 13.** The sequence \( \theta^{(k)} \) generated by SDCP converges in probability to the unique minimizer \( \theta^* \) of \( L \), i.e., for arbitrary \( \delta > 0 \)

\[ \lim_{k \rightarrow \infty} Pr(\|\theta^{(k)} - \theta^*\| > \delta) = 0. \]

**Proof:**

The proof of convergence is similar to (Theor. 46 in [23]), with the difference that our proof holds for Euclidean projection-based stochastic subgradients. Let us compute

\[ \|\theta^{(k+1)} - \theta^*\|^2 = \|\varphi(\theta^{(k)} - a^{(k)} \hat{g}^{(k)}) - \theta^*\|^2 \]

\[ \leq \|\theta^{(k)} - a^{(k)} \hat{g}^{(k)}\|^2 \]

\[ = \|\theta^{(k)} - \hat{g}^{(k)}\|^2 - 2a^{(k)} \cdot (\hat{g}^{(k)})^T \cdot (\theta^{(k)} - \theta^*) \]

\[ + (a^{(k)})^2 \cdot \|g^{(k)}\|^2, \quad (28) \]

where the first inequality is due to Lemma 5. Thanks to cor 12 and Def. 9

\[ (E[\hat{g}^{(k)} | \theta^{(k)}])^T \cdot (\theta^{(k)} - \theta^*) = \left(\hat{g}^{(k)}(\theta^{(k)})\right)^T \cdot (\theta^{(k)} - \theta^*) \geq 0. \]

Recall that the number of arriving requests per time slot \( A(T) \) is bounded, and thus \( \|\hat{g}^{(k)}\|^2 \) is bounded, i.e., \( \|\hat{g}^{(k)}\|^2 \leq c \) for some \( 0 < c < \infty \). Hence, applying the expectation to (28)

\[ E \left(\|\theta^{(k+1)} - \theta^*\|^2 \right) \leq \|\theta^{(k)} - \theta^*\|^2 + c(a^{(k)})^2. \quad (29) \]

Defining the random variable

\[ z_k = \|\theta^{(k)} - \theta^*\|^2 + c \sum_{t=1}^{k} (a^{(t)})^2, \]

it can be easily verified that (29) is equivalent to the inequality

\[ E[z_{k+1}] \leq z_k \leq z_1. \]

Consequently, \( \{z_k\}_{k=1}^{\infty} \) is a supermartingale and converges almost surely to a limit \( z^* \). Recalling now one of the required properties of the step size sequence,

i.e., \( \lim_{k \rightarrow \infty} \sum_{t=1}^{k} (a^{(t)})^2 = 0 \), we have that the sequence \( \{\|\theta^{(k)} - \theta^*\|^2\} \) also converges to \( z^* \) with probability one.

We now show by contradiction that the limit \( z^* \) is equal to zero. If this were not true, then one could find \( \epsilon > 0 \) and \( \delta > 0 \) such that, with probability \( \delta > 0 \), \( \|\theta^{(k)} - \theta^*\| \geq \epsilon \) for all sufficiently large \( k \), and thus

\[ \sum_{k=0}^{\infty} a^{(k)} \cdot (E[g^{(k)} | \theta^{(k)})]^T \cdot (\theta^{(k)} - \theta^*) = +\infty, \]

with probability \( \delta \), which would imply

\[ E \left(\sum_{k=0}^{\infty} a^{(k)} \cdot \|g^{(k)}\|^2 \right) = +\infty. \]

However, this would contradict the following relation (which is obtained by a recursion on (28) and then applying the expectation)

\[ E[\|\theta^{(k+1)} - \theta^*\|^2] \leq \|\theta^{(k)} - \theta^*\|^2 - 2E \left(\sum_{k=0}^{k} a^{(k)} \cdot (\hat{g}^{(k)})^T \cdot (\theta^{(k)} - \theta^*) \right) + \]

\[ E \left(\sum_{k=0}^{k} a^{(k)} \cdot \|g^{(k)}\|^2 \right), \]

as the left hand side cannot be negative.

**D. Optimality gap**

It is worthwhile to note that the minimizer \( \theta^* \) of \( \theta \) over \( C \cap R^p_{>0} \) may not coincide with its minimizer \( \theta^{OPT} \) over \( \Theta \) for two reasons: i) \( \theta' < K \) and ii) \( \theta^{OPT} \) is forced to have integer components while \( \theta^* \) is a real vector. In what follows we show that the optimality gap \( \|\theta^{OPT} - \theta^*\|_\infty \) is bounded by a small number, compared to the number of cache slots available.

**Lemma 14.** The gap between the optimal solution \( \theta^{OPT} \) and the configuration \( \theta^* \) to which SDCP converges is \( \|\theta^* - \theta^{OPT}\|_\infty \leq (3/2)P \)

**Proof:** We observe that \( \theta^* \) is the optimal solution of the continuous Simple Allocation Problem (SAP), expressed as \( \max -\sum_{p=1}^{P} L_p(\theta_p) \), subject to \( \sum_{p=1}^{P} \theta_p \leq K \) with \( \theta \in R^p_{>0} \). \( K' \) usually referred to as volume and we denote with SAP\_conf(\( K' \) the problem above. The integer version of the SAP, which we denote by SAP\_int(\( K' \), is obtained from the problem above with the additional constraint \( \theta \in \mathbb{Z}^p \). According to Cor. 4.3 of [25] there exists a solution \( \hat{\theta} \) of SAP\_int(\( K' \) such that \( \|\theta - \hat{\theta}\|_\infty \leq P \). The solution of the integer SAP can be constructed via the greedy algorithm presented in Sec. 2 of [25]. In our case, it consists of iteratively adding storage slots, one by one, each time to the CP whose miss intensity is decreased the most by using this additional slot. Based on this, it is easy to verify that a solution \( \theta^{OPT} \) can be obtained starting from \( \theta \) and adding the remaining \( K-K' \) slots. Therefore, \( \|\theta^{OPT} - \theta\|_\infty \leq P/2 \), which implies \( \|\theta^{OPT} - \theta^*\|_\infty \leq \|\theta^* - \hat{\theta}\|_\infty + \|\theta^{OPT} - \hat{\theta}\|_\infty \leq (3/2)P \).


Algorithm 2: Conditional Step Size Sequence Computation

1: \( a = \frac{p}{[g(1)]_{1}}, \frac{k}{p} \)  
2: \( b = a/10 \)  
3: if \( k \leq k_{BS} \) : \# Bootstrap Phase
4: then
5: \( \tilde{a}(k) = a \)  
6: else if \( k \leq M \) : \# Adaptive Phase
7: then
8: Compute the miss-ratio \( m_{(k)} \) during the current iteration
9: Compute \( m_{(k)} \), i.e., the 5th percentile of the previous miss ratios \( m_{(k-1)} \), \( m_{(k-2)} \), \( m_{(k-3)} \), \( m_{(k-4)} \)
10: \( \tilde{a}(k) = a \)  
11: \( \tilde{a}(k) = \frac{a(k-1)}{2} - \frac{a(k-1)-b}{2} \)  
12: \( \tilde{a}(k) = \left\{ \begin{array}{ll} \min(a(k), \tilde{a}(k)), b & m_{(k)} \leq m_{(k-1)} \\ \tilde{a}(k) & \text{otherwise} \end{array} \right. \)  
13: end
14: \( a(k) = a(k-1) \cdot \left( 1 - \frac{1}{(k+1)} \right)^{\frac{1}{2}} \) \# Moderate Phase
15: end

V. PERFORMANCE EVALUATION

We evaluate the performance of SDCP through simulations performed in Octave. We first describe the evaluation scenario (Sec.V-A) and show how the convergence speed is impacted by the choice of the step size sequence (Sec.V-B). We then evaluate the sensitivity of SDCP to various system parameters (Sec.V-C). Finally, recognizing that content catalogs are rarely static in the real world, we investigate the expected performance in the case of changing content catalogs (Sec.V-D).

A. Evaluation Scenario

We consider a content catalog of \( 10^8 \) objects, in line with the literature [26] and recent measurements [27]. We partition the catalog in disjoint sub-catalogs, one per each CP. We assume that the content popularity in each sub-catalog follows Zipf’s law with exponent \( \alpha = 0.8 \), as usually done in the literature [28]. We use a cache size of \( K \in \{10^2, 10^3, 10^4\} \) objects (which corresponds to cache/catalog ratios of \( 10^{-4}, 10^{-3} \) and \( 10^{-2} \), respectively). In practice, the request arrival rate may depend on several factors such as the cache placement in the network hierarchy, the level of aggregation, the time of day, etc. Without loss of generality, we set the request arrival rate to \( \lambda = 10^3 \) req/s, according to recent measurements performed on ISP access networks [27]. We compare the performance of SDCP to that of the optimal allocation \( \theta^{OPT} \) (Opt), and to that of the naive solution in which the cache space \( K \) is equally divided among all the CPs and is unchanged throughout the simulation (Unif).

While we proved convergence of SDCP, the speed of convergence is crucial to let the algorithm also be of practical use: we thus consider three step size sequences, as follows. In the Reciprocal scheme, the step size is \( a(k) = a/k \), where \( a = \frac{1}{\lambda^1}, \frac{k}{P} \). Observe that, with this choice, the Euclidean norm of the first update \( a^{(1)} \cdot \tilde{g}^{(1)} \) is \( \frac{K}{P} \), which allows to change this amount of slots in the allocation, thus obtaining a broad exploration of the allocation space at the very beginning.

In the Moderate scheme, step sizes decrease slowly, to avoid confining the exploration only to the beginning. We resort to guidelines of [29], and define the step size as

\[
   a^{(k)} = a^{(k-1)} \cdot \left( 1 - \frac{1}{(k+1)} \right)^{\frac{1}{2}}
\]

where \( a \) is computed as above and \( M_0 = 100 \) is an expected value of the current allocation, which is smaller than the 5-th percentile of the miss ratios observed so far. In this case the step size is halved, unless it already equals \( b \). The intuition behind this phase is that we try to reduce the exploration extent every time we encounter a “good” allocation, i.e., an allocation that shows a small miss ratio compared to what we experienced so far. Note that we do not start immediately with the Adaptation phase, since we need to collect enough samples during the Bootstrap phase in order to correctly evaluate the quality of the current allocation. Finally, we continue with a Moderate phase, in which step sizes are updated as above and are asymptotically vanishing, thus guaranteeing convergence.

After a preliminary evaluation, we set \( \epsilon = 1/100 \) as in [29] and \( b = a/10 \). We set \( k_{BS} = M_0 \), i.e., the duration of the bootstrap and adaptive phases, to the number of iterations in 6 minutes and 1 hour, respectively. While the duration of these phases is clearly tied to the arrival rate, and are expected to require tuning when ported to a different scenario, we point out that performance achieved with these choices remains satisfying under the different scenarios we consider.

B. Convergence

We first consider a cache size of \( K = 10^3 \) and 4 CPs, receiving 13%, 75%, 2% and 10% of requests, respectively. From Fig. 1, we can observe that, after a first exploration phase, the algorithm converges to a stable allocation. It is interesting to note that the average allocation (Avg), which is obtained by averaging each component of the allocation vector throughout the iterations, is very close to the optimal one, unlike the naïve uniform allocation policy.
Second, we consider a larger scenario with cache size $K = 10^6$ and $p = 10$ CPs, one of which we is a popular CP, to which 70% of requests are directed, followed by a second one receiving 24% of requests, other 6 CPs accounting for 1% each and the remaining two CPs receiving no requests. Fig. 2 shows the step sizes and the inaccuracy of the algorithm, i.e. the distance to the optimal allocation, measured as:

$$\text{Error}(\theta) = \frac{\|\theta^{OPT} - \theta\|_\infty}{K} = \max_{j=1...p} |\theta_j - \theta^{OPT}_j|.$$  \hspace{1cm} (30)

Observe that Reciprocal steps decrease too fast, which immediately limits the adaptation of the allocation, significantly slowing down convergence. Conversely, Moderate steps remain large for an overly long time, preventing the algorithm to keep the allocation in regions that guarantee good performance. Conditional steps show the best performance since in the Adaptation phase the step sizes are sharply decreased if the current allocation is providing a small miss ratio.

C. Sensitivity Analysis

We next study how the performance of SDCP is affected by the algorithm parameters and the scenario. We first focus on the time slot duration $T$. On the one hand, a small $T$ implies that only few requests are observed in each time slot, which may result in a high noise $\epsilon^{(b)}$, $\epsilon^{(h)}$, and ultimately affects the accuracy of the update. On the other hand, a large $T$ decreases the measurement noise, but allows updates to be made less frequently, which possibly slows down convergence.

To evaluate the impact of $T$, Fig. 3 shows the miss ratio measured over 1h for the default scenario. We consider SDCP with the three step size sequences, and compare it to the Uniform and to the Optimal allocations as benchmarks. The figure shows that SDCP with the Conditional step size sequence enhances the cache efficiency significantly. We also observe that an iteration duration of $T = 10s$ (corresponding to 100 samples on average per CP) represents a good compromise between a more accurate miss ratio estimation based on more samples (with large $T$) and a larger number of iterations at the cost of lower accuracy (with small $T$).

Fig. 4 shows the cache miss rate measured over 1h for a time slot length of $T = 10s$ and for various cache sizes $K \in \{10^4, 10^5, 10^6\}$ and arrival rates $\lambda \in [1, 10^4]$. The figure confirms that the gains of SDCP hold for different cache sizes, and shows that the gain increases for large caches. To interpret the results for different arrival rates, recall that for any given time slot duration $T$, the average request rate affects the measurement noise. Fig. 4 confirms that the miss rate increases when the measurement noise is higher, i.e., for lower $\lambda$, but it also shows a very limited impact: the number of time slots in a relatively short time (in 1h, there are 360 time slots of duration $T = 10s$) allows SDCP to converge to a good cache configuration, in spite of the noise and the consequent estimation errors.

D. Changing Content Popularity

Recent studies [30] have observed that the catalog statistics vary over time. We show in this section that in order for SDCP to be robust to these variations, it suffices to periodically reinitialize the step sequence. To model changing content popularity, we adopt the model of [30], in which each object is characterized by a sequence of ON and OFF periods, with exponentially distributed duration $T_{ON}$ and $T_{OFF}$, respectively. At each time instant, an object can be ON or OFF, and only ON objects attract requests. As in [30], we set the catalog size to $3.5 \times 10^6$ and the cache size to $K = 10^4$ objects. We set the average ON and OFF duration to 1 and 9 days, respectively. On average, we maintain the overall request rate of active objects equal to our default value $\lambda = 100 \text{ req/s}$.

In Fig. 5 we compare Unif and Conditional($\tau$) that reinitialize the step sequence every $\tau$ amount of time. We consider
τ ∈ {3h, 1d, ∞}, i.e., 8 reinitializations per day, daily reinitialization, or no-reinitialization, respectively. As expected, reinitialization improves cache efficiency. Indeed, already after 3 hours of simulation, the evolution of the catalog misleads Conditional and Conditional(1d) (that overlap in this time interval) causing them to have performance worse than Unif. This is expected, since Conditional(∞) tries to converge to the optimal allocation, which is problematic in a non-stationary scenario. At the same time, it also shows that reinitializing step sequences as in Conditional(3h) is sufficient to respond to the catalog dynamics.

VI. CONCLUSION

One of the main challenges of in-network caching nowadays is its incompatibility with encrypted content. Our work represents a first step in solving this challenge by proposing a simple and therefore appealing system design: Stochastic Dynamic Cache Partitioning requires solely the knowledge of aggregated cache miss-intensities, based on which it provably converges to an allocation with a small optimality gap. Simulation results show the benefits of the proposed algorithm under various scenarios, and results obtained under complex content catalog dynamics further confirm the algorithm to be applicable in scenarios of high practical relevance.

ACKNOWLEDGEMENTS

This research was partially supported by Labex DigiCosme (ANR-11-LABEX-0045-DIGICOSME), part of the program “Investissement d’Avenir” Idex Paris-Saclay (ANR-11-IDEX-0003-02), and by the Swedish Foundation for Strategic Research through the Modane project. This work was carried out while Araldo was visiting KTH in the context of the EIT Digital project T15212 on “Programmable data plane for Software Defined Networks”. This work benefited from support of NewNet@Paris, Cisco’s Chair “NETWORKS FOR THE FUTURE” at Telecom ParisTech (http://newnet.telecom-paristech.fr). Any opinion, findings or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of partners of the Chair.

REFERENCES

Access-time aware cache algorithms

Giovanni Neglia\textsuperscript{*}, Damiano Carra\textsuperscript{†}, Mingdong Feng\textsuperscript{‡}, Vaishnav Janardhan\textsuperscript{§}, Pietro Michiardi\textsuperscript{§} and Dimitra Tsigkari\textsuperscript{*}

\textsuperscript{*}Inria, \{giovanni.neglia, dimitra.tsigkari\}@inria.fr
\textsuperscript{†}University of Verona, damiano.carra@univr.it
\textsuperscript{‡}Akamai Technologies, \{mfeng, vjanardh\}@akamai.com
\textsuperscript{§}Eurecom, pietro.michiardi@eurecom.fr

Abstract—Most of the caching algorithms are oblivious to requests’ timescale, but caching systems are capacity constrained and, in practical cases, the hit rate may be limited by the cache’s impossibility to serve requests fast enough. In particular, the hard-disk access time can be the key factor capping cache performances. In this paper, we present a new cache replacement policy that takes advantage of a hierarchical caching architecture, and in particular of access-time difference between memory and disk. Our policy is optimal when requests follow the independent reference model, and significantly reduces the hard-disk load, as shown also by our realistic, trace-driven evaluation.

I. INTRODUCTION

The hit probability is a well-known key metric for caching systems: this is the probability that a generic request for a given content will be served by the cache. Most of the existing literature implicitly assumes that a hit occurs if the content is stored in the cache at the moment of the request. In practice, however, in real caching systems the actual hit rate is often limited by the speed at which the content can serve requests. In particular, Hard-Disk Drive (HDD) access times can be the key factor capping cache performance.

As an illustrative example, Fig. 1 shows the percentage of CPU and HDD utilization, as reported by the operating system, over two days in the life of a generic caching server. As the amount of requests varies during the day, the resource utilization of the caching server varies as well: during peak hours, HDD utilization can exceed 95%. Such loads may cause the inability to serve a request even if the content is actually cached in the HDD. In case of a pool of cache servers, a solution based on dynamic load balancing may alleviate this problem by offloading the requests to another server. Nevertheless, this solution has its own drawbacks, because the rerouted queries are likely to generate misses at the new cache.

In this paper, we study if and how the RAM can be used to alleviate the HDD load, so that the cache can serve a higher rate of requests before query-rerouting becomes necessary.

The idea to take advantage of the RAM is not groundbreaking. Modern cache servers usually operate as a hierarchical cache, where the most recently requested contents are stored also in the RAM: upon arrival of a new request, content is first looked up in the RAM; if not found, the lookup mechanism targets the HDD. Hence, the RAM “shields” the HDD from most of the requests.

The question we ask in this paper is: what is the optimal way to use the RAM? Which content should be duplicated in the RAM to minimize the load on the HDD? We show that, if content popularities are known, the problem can be formulated as a knapsack problem. More importantly, we design a new dynamic replacement policy that, without requiring popularity information to be known, can implicitly solve our minimization problem. Our policy is a variant of \(q\)-LRU. In \(q\)-LRU, after a cache miss, the content is stored in the cache with probability \(q\) and, if space is needed, the least recently used contents are evicted. We call our policy \(q\)-LRU, because we use a different probability \(q\) for each content \(i\). The value \(q_i\) depends on the content size and takes into account the time needed to retrieve contents from the HDD. Simulation results on real content request traces from the Akamai’s Content Delivery Network (CDN) [1] show that our policy achieves more than 80% load reduction on the HDD with an improvement between 10% and 20% in comparison to the standard LRU.

The paper is organized as follows. In Sec. II we formalize the problem and illustrate the underlying assumptions. In Sec. III we present the policy \(q\)-LRU and prove its asymptotic optimality. We evaluate its performance under real-world traces in Sec. IV. Related works are discussed in Sec. V. Some proofs are in the companion report [2].

II. MODEL

In this section, we illustrate our main assumptions about HDD operation and content request process and then formalize our optimization problem.
A. Hard Disk Service Time

Our study relies on some assumptions about the load imposed on the HDD by a set of requests. Consider a single file-read request for content \( i \) with size \( s_i \). We call service time the time the HDD works just to provide content \( i \) to the operating system. Our first assumption is that the service time is only a function of content size \( s_i \). We denote it as \( T(s_i) \). The second assumption is that service times are additive, i.e., let \( A \) be a set of contents, the total time the HDD works to provide the contents in \( A \) is equal to \( \sum_{i \in A} T(s_i) \), independently of the specific time instants at which the requests are issued. Note that we are not assuming any specific service discipline for this set of requests: they could be served sequentially (e.g. in a FIFO or LIFO way) or in parallel (e.g. according to a generalized processor sharing). But we are requiring that concurrent object requests do not interfere by increasing (or reducing) the total HDD service time.

The analytical results we provide in Sect. III, which is the main contribution of our work, do not depend on a particular structure of the function \( T(s_i) \). Nevertheless, we describe here a specific form based on past research on HDD I/O throughput [3][4], and based on our performance study of disk access time observed in caching servers. We will refer to this specific form later to clarify some properties of the optimal policy. Furthermore, we will use it in our experiments in Sec. IV.

Considering the mechanical structure of the HDD, every time a new read is done, we need to wait for the reading arm to move across the cylinders, and for the platter to rotate on its axis. We call these two contributions the average seek time and average rotation time, and we denote them by \( \sigma \) and \( \rho \) respectively. Each file is divided into blocks, whose size \( b \) is a configuration parameter. If we read a file whose size is bigger than a block, then we need to wait for the average seek time and the average rotation time for each block.

Once the reading head has reached the beginning of a block, the time it takes to read the data depends on the transfer speed \( \mu \). As a last contribution, we have a constant delay due to the controller overhead, \( \phi \).

Overall, the function that estimates the cost of reading a file from the hard disk is given by the following equation (see Table I for a summary of the variables used):

\[
T(s_i) = (\sigma + \rho) \frac{s_i}{b} + \frac{s_i}{\mu} + \phi. \tag{1}
\]

Based on our experience on real-life production systems, the last column of Table I shows the values of the different variables for a 10'000 RPM hard drive.

We have validated Eq. (1) through an extensive measurement campaign for two different hard disk drives (10'000 RPM and 7'200 RPM). The results are shown in Fig. 2. In the figure, we actually plot the quantity \( T(s_i)/s_i \); in Sect. III we will illustrate the key role played by this ratio. The estimated value of \( T(s_i)/s_i \) has discontinuity points at multiples of the block size \( b \); in fact, as soon as the size of an object exceeds one of such values, the service time increases by an additional average seek time and an additional average rotation time. The points in the figures represent the output of our measurement campaign for a representative subset of sizes (in particular, for sizes close to the multiples of block size \( b \), where the discontinuities occur). Each point is the average value for a given size over multiple reads. From the experiments, we conclude that the function \( T(s_i) \) shown in Eq. (1) is able to accurately estimate the cost of reading a file from the HDD.

B. Query Request Process

Let \( \mathcal{N} = \{1, 2, \ldots, N\} \) denote the set of contents. For mathematical tractability, as done in most of the works in the literature (see Sec. V), we assume that the requests follow the popular Independent Reference Model (IRM), where contents requests are independently drawn according to constant probabilities (see for example [5]). In particular, we consider the time-continuous version of the IRM: requests for content \( i \in \mathcal{N} \) arrive according to a Poisson process with rate \( \lambda_i \) and the Poisson processes for different contents are independent. While the optimality results for our policy \( q_i \)-LRU are derived under such assumption, significant performance improvements are obtained also considering real request traces (see Sec. IV).

C. Problem Formulation

In general, the optimal operation of a hierarchical cache system would require to jointly manage the different storage units, and in particular to avoid to duplicate contents across

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Typical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_i )</td>
<td>Size of object ( i )</td>
<td>-</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>Average seek time</td>
<td>( 3.7 \times 10^{-3} \text{ s} )</td>
</tr>
<tr>
<td>( \rho )</td>
<td>Average rotation time</td>
<td>( 3.0 \times 10^{-3} \text{ s} )</td>
</tr>
<tr>
<td>( b )</td>
<td>Block size</td>
<td>2.0 MB</td>
</tr>
<tr>
<td>( \mu )</td>
<td>Transfer bandwidth</td>
<td>157 MB/s</td>
</tr>
<tr>
<td>( \phi )</td>
<td>Controller Overhead</td>
<td>0.5 ( \times 10^{-3} \text{ s} )</td>
</tr>
</tbody>
</table>

Fig. 2. Graph of the function \( T(s_i)/s_i \).
multiple units. On the contrary, in the case of a RAM-HDD system, the problem is usually decoupled: the HDD caching policy is selected in order to maximize the main cache performance metric (e.g. hit ratio/ rate), while a subset of the contents stored in the HDD can be duplicated in the RAM to optimize some other performance metric (e.g. the response time). The reason for duplicating contents in the RAM is twofold. First, contents present only in the RAM would be lost if the caching server is rebooted. Second, the global cache hit ratio/ rate would not be significantly improved because the RAM accounts for a small percentage of the total storage available at the server. A consequence of such decoupling is that, at any time, the RAM stores a subset \((\mathcal{M}_R)\) of the contents stored in the HDD \(\mathcal{M}_H\).\(^3\) In our work we consider the same decoupling principle. As a consequence, our policy is agnostic to the replacement policy implemented at the HDD (LRU, FIFO, Random, \ldots).

We now look at how the RAM reduces the HDD load. An incoming request can be for a content not present in the HDD (nor in the RAM because we consider \(\mathcal{M}_R \subset \mathcal{M}_H\)). In this case, the content will be retrieved by some other server in the CDN or by the authoritative content provider, and then stored or not in the HDD depending on the specific HDD cache policy. Note that the choice of the contents to be duplicated in the RAM plays no role here. Read/write operations can occur (e.g. to store the new content in the HDD), but they are not affected by the RAM replacement policy, that is the focus of this paper. We ignore then the corresponding costs. On the contrary, if an incoming request is for a content present in the HDD, the expected HDD service time depends on the set of contents \(\mathcal{M}_R\) stored in the RAM. It is indeed equal to

\[
\lambda \sum_{i \in \mathcal{M}_R, i \not\in \mathcal{M}_R} \frac{\lambda_i}{\lambda_j} T(s_i) = \sum_{i \in \mathcal{M}_R} \lambda_i \sum_{j \in \mathcal{N}} \frac{\lambda_i}{\lambda_j} T(s_i) - \sum_{i \in \mathcal{M}_R} \sum_{j \in \mathcal{N}} \lambda_j T(s_j),
\]

because, under IRM, \(\lambda_i/\sum_{j \in \mathcal{N}} \lambda_j\) is the probability that the next request is for content \(i\), and the request will be served by the HDD only if content \(i\) is not duplicated in the RAM, i.e. only if \(i \notin \mathcal{M}_R\).

Our purpose is to minimize the HDD service time under the constraint on the RAM size. This is equivalent to maximize the second term in Eq. (2). By removing the constant \(\sum_{j \in \mathcal{N}} \lambda_j\), we obtain then that the optimal possible choice for the subset \(\mathcal{M}_R\) is the solution of the following maximization problem:

\[
\text{maximize}_{\mathcal{M}_R \subset \mathcal{N}} \sum_{i \in \mathcal{M}_R} \lambda_i T(s_i)
\]

subject to \(\sum_{i \in \mathcal{M}_R} s_i \leq C\).

\(^3\) Although it is theoretically possible that a content stored in the RAM and in the HDD may be evicted by the HDD earlier than by the RAM, these events can be neglected in practical settings. For example, in the scenario considered in Sec. IV typical cache eviction times are a few minutes for the RAM and a few days for the HDD for all the cache policies considered.

where \(C\) is the RAM capacity. This is a knapsack problem, where \(\lambda_i T(s_i)\) is the value of content/item \(i\) and \(s_i\) its weight. The knapsack problem is NP-hard. A natural, and historically the first, relaxation of the knapsack problem is the fractional knapsack problem (also called continuous knapsack problem). In this case, we accept fractional amounts of the contents to be stored in the RAM. Let \(h_i \in [0, 1]\) be the fraction of content \(i\) to be put in the RAM, the fractional problem corresponding to Problem (3) is:

\[
\text{maximize}_{h_1, \ldots, h_N} \sum_{i=1}^N \lambda_i h_i T(s_i)
\]

subject to \(\sum_{i=1}^N h_is_i = C\).

From an algorithmic point of view, the following greedy algorithm is optimal for the fractional knapsack problem. Assume that all the items are sorted in decreasing order with respect to the profit per unit of size (i.e. \(\lambda_i T(s_i)/s_i \geq \lambda_j T(s_j)/s_j\) for \(i \leq j\)). The algorithm finds the biggest index \(c\) for which the sum \(\sum_{i=1}^c s_i\) does not exceed the memory capacity. Finally, it stores the first \(c\) contents in the knapsack (in the RAM) as well as a fractional part of the content \(c+1\) so that the RAM is filled up to its capacity. A simple variant of this greedy algorithm guarantees a \(\frac{1}{2}\)-approximation factor for the original knapsack problem [6, Theorem 2.5.4], but the greedy algorithm itself is a very good approximation algorithm for common instances of knapsack problems, as it can be justified by its good expected performance under random inputs [6, Sec. 14.4].

From a networking point of view, if we interpret \(h_i\) as the probability that content \(i\) is in the RAM,\(^4\) then we recognize that the constraint in Problem (4) corresponds to the usual constraint considered under Che’s approximation for cache networks [7], where the effect of the finite cache size is taken into account by imposing the expected cache occupancy to be equal to the cache size \(C\).

The last remark connects our problem to the recent work in [9], where the authors use Che’s approximation to find optimal cache policies to solve the following problem:

\[
\text{maximize}_{h_1, \ldots, h_N} \sum_{i=1}^N U_i(h_i)
\]

subject to \(\sum_{i=1}^N h_is_i = C\),

where each \(U_i(h_i)\) quantifies the utility of a cache hit for content \(i\).\(^5\) Results in [9] do not help us solve our Problem (4) because their approach requires the functions \(U_i(h_i)\) to be (i) known and (ii) strictly concave in \(h_i\). On the contrary, in our case, content popularities \(\lambda_i\) are unknown and, even if they

---

\(^4\) Since the PASTA property holds under the IRM model, then \(h_i\) is also the RAM hit probability.

\(^5\) The work in [9] actually assumes that all the contents have the same size, but their analysis can be easily extended to heterogenous sizes, as we do in Sec. III-B.
were known, the functions $U_i(h_i)$ would be $\lambda_i h_i T(s_i)$ and then linear in $h_i$. Besides deriving the cache policy that solves a given optimization problem, [9] also “reverse-engineers” existing policies (like LRU) to find which optimization problem they are implicitly solving. In Sec. III we use a similar approach to study our policy.

After this general analysis of the problem, we are ready to introduce in the next section a new caching policy $q_i$-LRU that aims to solve Problem (4), i.e. to store in the RAM the contents with the largest values $\lambda_i T(s_i)/s_i$ without the knowledge of content popularities $\lambda_i$, for $i=1,\ldots,N$.

III. The $q_i$-LRU POLICY

We start introducing our policy as a heuristic justified by an analogy with LRU.

Under IRM and Che’s approximation, if popularities $\lambda_i$ are known, minimizing the miss throughput at a cache with capacity $C$ corresponds to solving the following problem:

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{N} \lambda_i h_i s_i \\
\text{subject to} & \quad \sum_{i=1}^{N} h_i s_i = C
\end{align*}
\]  

(6)

The optimal solution is analogous to what discussed for Problem (4); set hit probabilities to one for the $k$ most popular contents, a hit probability smaller than one for the $(k+1)$-th most popular content, and hit probabilities to zero for all the other contents. The value of $k$ is determined by the RAM size.

Now, it is well known that, from a practical perspective, the traditional LRU policy behaves extremely well, despite content popularity dynamics. LRU is a good heuristic for Problem (7): it implicitly selects and stores in the cache the contents with the largest values of $\lambda_i$, even when popularities $\lambda_i$ are actually unknown.

Recall that our purpose is to store the contents with the largest values $\lambda_i T(s_i)/s_i$; then, the analogy between the two problems suggests us to bias LRU in order to store more often the contents with the largest values of $T(s_i)/s_i$. Intuitively, upon a cache miss, the newly requested content $i$ is cached with probability $q_i$, which is an increasing function in $T(s_i)/s_i$. Specifically, we define $q_i$ as follows:

\[
q_i = e^{-\beta \frac{T(s_i)}{s_i}}, \quad i \in \mathcal{N},
\]  

(7)

where $\beta > 0$ is a constant parameter. In practical cases, as discussed in section IV, we set $\beta$ such that $q_i \geq \frac{q_{\min}}{1/q_{\min}}$ for every $i \in \mathcal{N}$, so that any content is likely to be stored in the cache after $1/q_{\min}$ queries on average.

Our policy has then the same behaviour as the $q$-LRU policy, but the probability $q_i$ is not fixed, it is instead chosen depending on the size of the content as indicated in Eq. (8). For this reason, we denote our policy by $q_i$-LRU.

With reference to Fig. 2, the policy $q_i$-LRU would store with higher probability the smallest contents as well as the contents whose size is slightly larger than a multiple of the block size $b$. Note that the policy $q_i$-LRU does not depend on the model described above for the HDD service time, but it requires the ratio $T(s)/s$ to exhibit some variability (otherwise we would have the usual $q$-LRU).

Until now we have provided some intuitive justification for the policy $q_i$-LRU. This reasoning reflects how we historically conceived it. The reader may now want more theoretically grounded support to our claim that $q_i$-LRU is a good heuristic for Problem (4). In what follows we show that $q_i$-LRU is asymptotically optimal when $\beta$ diverges in two different ways. We first prove in Sec. III-A that $q_i$-LRU asymptotically stores in a cache the contents with the largest values $\lambda_i T(s_i)/s_i$, as the optimal greedy algorithm for Problem (4) does. This would be sufficient to our purpose, but we find interesting to establish a connection between $q_i$-LRU and the cache utility maximization problem introduced in [9]. For this reason, in Sec. III-B, we reverse-engineer the policy $q_i$-LRU and derive the utility function it is implicitly maximizing. We then let again $\beta$ diverge and show that the utility maximization problem converges to a problem whose optimal solution corresponds to store the contents with the largest values $\lambda_i T(s_i)/s_i$.

A. Asymptotic $q_i$-LRU hit probabilities

In [10] (and partially in the conference version [13]) it is proven that under the assumptions of the IRM traffic model, the usual $q$-LRU policy tends to the policy that statically stores in the cache the most popular contents when $q$ converges to 0. We generalize their approach to study the $q_i$-LRU policy when $\beta$ diverges (and then $q_i$ converges to 0, for all $i$). In doing so, we address some minor technical details that are missing in the proof in [10].

Let us sort contents in a decreasing order of $\frac{\lambda_i T(s_i)}{s_i}$ assuming, in addition, that $\frac{\lambda_i T(s_i)}{s_i} \neq \frac{\lambda_j T(s_j)}{s_j}$ for every $i \neq j$.

Note that the hit probability $h_i$ associated to the content $i$ for the $q_i$-LRU policy is given by the following formula (see [10])

\[
h_i(\beta, \tau_e) = q_i(\beta) \frac{1 - e^{-\lambda_i \tau_e}}{e^{-\lambda_i \tau_e} + q_i(\beta) (1 - e^{-\lambda_i \tau_e})},
\]  

(8)

where $\tau_e$ is the eviction time that, under Che’s approximation [7], is assumed to be a constant independent of the selected content $i$.

Now, by exploiting the constraint:

\[
C = \sum_i s_i h_i(\beta, \tau_e)
\]  

(9)

it is possible to express $\tau_e$ as an increasing function of $\beta$ and prove that $\lim_{\beta \to \infty} \tau_e(\beta) = \infty$. This result follows [10], but, for the sake of completeness, we present it extensively in [2].

We can now replace $q_i = e^{-\beta \frac{T(s_i)}{s_i}}$ in Eq. (9) and express the hit probability as a function of $\beta$ only as follows:

\[
h_i(\beta) = e^{\tau_e(\beta)} \left( \frac{1 - e^{-\lambda_i \tau_e}}{e^{-\lambda_i \tau_e} + q_i(\beta) (1 - e^{-\lambda_i \tau_e})} \right) + 1 - e^{-\lambda_i \tau_e(\beta)}.
\]  

(10)
Let us imagine to start filling the cache with contents sorted as defined above. Let $c$ denote the last content we can put in the cache before the capacity constraint is violated\footnote{We consider the practical case when $s_1 < C < \sum_{i=1}^N s_i$.} i.e.
\[ c = \max \left\{ k \mid \sum_{i=1}^k s_i \leq C \right\}. \]

We distinguish two cases: the first $c$ contents fill exactly the cache (i.e. $\sum_{i=1}^c s_i = C$), or they leave some spare capacity, but not enough to fit content $c+1$. Next, we prove that $q_i$-LRU is asymptotically optimal in the second case. The first case requires a more complex machinery that we develop in \cite{2}.

Consider then that $\sum_{i=1}^c s_i < C < \sum_{i=1}^{c+1} s_i$. As an intermediate step we are going to prove by contradiction that
\[ \lim_{\beta \to \infty} \frac{\beta}{\tau_c(\beta)} = \sum_{i=1}^{c+1} \frac{T(s_{i+1})}{s_{i+1}}. \] (11)

Suppose that this is not the case. Then, there exists a sequence $\beta_n$ that diverges and a number $\epsilon > 0$ such that $\forall n \in \mathbb{N}$
\[ \frac{\beta_n}{\tau_c(\beta_n)} \leq \frac{\lambda_n T(s_{i+1})}{s_{i+1}} - \epsilon \] (12)
\[ \frac{\beta_n}{\tau_c(\beta_n)} \geq \frac{\lambda_n T(s_{i+1})}{s_{i+1}} + \epsilon. \] (13)

If inequality (13) holds, then $\forall i \leq c + 1$
\[ \frac{\beta_n}{\tau_c(\beta_n)} - \frac{\lambda_n T(s_i)}{s_i} \leq \frac{\beta_n}{\tau_c(\beta_n)} - \frac{\lambda_n T(s_{i+1})}{s_{i+1}} \leq \frac{\epsilon}{s_{i+1}}. \]

From Eq. (11) it follows immediately that
\[ \lim_{\beta_n \to \infty} h_i(\beta_n) = 1 \quad \forall i \leq c + 1, \]
but then it would be
\[ \lim_{n \to \infty} \sum_{i=1}^{c+1} h_i(\beta_n)s_i = \sum_{i=1}^{c+1} s_i > C \]
contradicting the constraint (10). In a similar way it is possible to show that inequality (14) leads also to a contradiction and then Eq. (12) holds.

Because of the limit in Eq. (12) and of Eq. (11), we can immediately conclude that, when $\beta$ diverges, $h_i(\beta)$ converges to 1, for $i \leq c$, and to 0, for $i > c + 1$. Because of the constraint (10), it holds that:
\[ \lim_{\beta \to \infty} h_{c+1}(\beta) = \frac{C - \lim_{\beta \to \infty} \sum_{i \neq c+1} h_is_i}{s_{c+1}} = \frac{C - \sum_{i \leq c} s_i}{s_{c+1}}. \]

The same asymptotic behavior for the hit probabilities holds when $\sum_{i=1}^{c+1} s_i = C$, as it is proven in \cite{2}.\footnote{When $\sum_{i=1}^{c+1} s_i = C$, $h_{c+1}(\beta)$ converges to $(C - \sum_{i=1}^c s_i)/s_{c+1} = 0$.}

Proposition III.1. When the parameter $\beta$ diverges, the hit probabilities for the $q_i$-LRU policy converge to the solution of the fractional knapsack problem (4), i.e.
\[ \lim_{\beta \to \infty} h_i(\beta) = \begin{cases} 1, & \text{for } i \leq c, \\ \frac{(C - \sum_{i=1}^c s_i)/s_{c+1}, & \text{for } i = c + 1, \\ 0, & \text{for } i > c + 1. \end{cases} \]

Then the $q_i$-LRU policy asymptotically minimizes the load on the hard-disk.

B. Reverse-Engineering $q_i$-LRU

In \cite{9}, the authors show that existing policies can be thought as implicitly solving the utility maximization problem (6) for a particular choice of the utility functions $U_i(h_i)$. In particular they show which utility functions correspond to traditional policies like LRU and FIFO. In what follows, we “reverse-engineer” the $q_i$-LRU policy and we show in a different way that it solves the fractional knapsack problem. We proceed similarly to what done in \cite{9}, extending their approach to the case where content sizes are heterogeneous (see \cite{2}). We show that the utility function for content $i$ is
\[ U_i(h_i) = -\lambda_i \int_0^{1-h_i} \frac{dx}{\ln(1 + \frac{1-x}{q_i})}. \] (14)

that is defined for $h_i \in (0, 1]$ and $q_i \neq 0$. Each function $U_i(.)$ is increasing and concave. Moreover, $U_i(h_i) < 0$ for $h_i \in (0, 1)$, $U_i(1) = 0$ and $\lim_{h_i \to 0} U_i(h_i) = -\infty$.

We are interested now in studying the asymptotic behavior of the utility functions $U_i(h_i)$ when $\beta$ diverges, and then $q_i$ converges to zero. First, we note that the following inequalities are true for every $\delta > 0$ such that $q_i^\delta < 1 - h_i$:
\[ \int_0^{1-h_i} \frac{dx}{\ln(1 + \frac{1-x}{q_i})} \geq \int_0^{1-h_i} \frac{dx}{\ln(1 + \frac{1-x}{q_i^\delta})} \geq \frac{1 - h_i - q_i^\delta}{\ln(1 + \frac{1-q_i^\delta}{q_i})}, \] (15)

where the last inequality follows from the fact that the integrand is an increasing function of $x$.

Similarly, it holds
\[ \int_0^{1-h_i} \frac{dx}{\ln(1 + \frac{1-x}{q_i^\delta})} \leq \frac{1 - h_i}{\ln(1 + \frac{h_i}{(1-h_i)})} \leq \frac{1 - h_i}{\ln(1 + \frac{1}{q_i})}. \] (16)

Asymptotically, when $q_i$ converges to zero, the lower bound in Eq. (16) is equivalent to $\frac{1-h_i}{(1+\delta)\ln(1/q_i)}$. For every $\delta > 0$, we obtain the following (asymptotic) inequalities when $q_i$ converges to 0 (and then $q_i^\delta < 1 - h_i$ asymptotically):
\[ \frac{1 - h_i}{(1+\delta)\ln(1/q_i)} \leq \int_0^{1-h_i} \frac{dx}{\ln(1 + \frac{1-x}{q_i})} \leq \frac{1 - h_i}{\ln(1/q_i)}. \] (17)

\footnote{We say that $f(x)$ is equivalent to $g(x)$ when $x$ converges to 0 if $\lim_{x \to 0} f(x)/g(x) = 1$, and we write $f(x) \sim g(x)$.}
Thus, when $q_i$ converges to 0, we get
\[
\int_0^{1-h_i} \frac{dx}{\ln (1 + \frac{1}{q_i x})} \sim \frac{1 - h_i}{\ln(1/q_i)}
\]
since, otherwise, we could find an $\varepsilon > 0$ and a sequence $q_{i,n}$ converging to 0 such that for large $n$
\[
\int_0^{1-h_i} \frac{dx}{\ln (1 + \frac{1}{q_{i,n} x})} \leq (1 - \varepsilon) \frac{1 - h_i}{\ln(1/q_i,n)}.
\]
But, this would contradict the left-hand inequality in (18), which is valid for every $\delta > 0$. We conclude that, when $q_i$ converges to 0,
\[
U_i(h_i) = -\lambda_i s_i \int_0^{1-h_i} \frac{dx}{\ln (1 + \frac{1}{q_i x})} \sim -\frac{\lambda_i s_i (1 - h_i)}{\ln(1/q_i)}.
\]

Next, we consider $q_i = e^{-\beta T_i}$, and we can write
\[
U_i(h_i) \sim \frac{\lambda_i T_i(s_i)(1 - h_i)}{\beta}, \quad \text{when } \beta \to \infty.
\]
Maximizing the sum of the utilities $\sum_i U_i(h_i)$ over the hit probabilities is equivalent to maximizing $\beta \sum_i U_i(h_i) + \sum_i \lambda_i T_i(s_i)$. We conclude that, when $\beta$ diverges, the problem (6) can be formulated as follows
\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{N} \lambda_i h_i T(s_i) \\
\text{subject to} & \quad \sum_{i=1}^{N} h_i s_i = C,
\end{align*}
\]
which is exactly the formulation of the fractional knapsack problem.

IV. EXPERIMENTS

In this section, we evaluate the performance of our $q_i$-LRU policy. Here we take a numerical perspective, and design a trace-driven simulator that can reproduce the behavior of several caching policies, which we compare against $q_i$-LRU. We have used both synthetic traces generated according to the IRM and real traces collected at two vantage points of the Akamai network [1]. We proved that $q_i$-LRU is optimal under the IRM and indeed our experiments confirm it and show significant improvement in comparison to other replacement policies. For this reason, in this section we focus mainly on the results obtained with real traces. In the following, we describe our experimental methodology, show the characteristics of the real traces we use, and present the results of our evaluation.

A. Methodology and Performance indexes

The comparative analysis of different caching policies requires an environment where it is possible to reproduce exactly the same conditions for all the different policies. To do so, we adopt a trace-driven simulation approach,\(^{10}\) which allows us to control the initial conditions of the system, explore the parameter space and perform a sensitivity analysis, for all eviction policies.

Our simulator reproduces two memory types: the main memory (RAM) and the hard disk (HDD). Each object is stored in the HDD according to the LRU policy. For the RAM we consider 3 different policies: LRU, SIZE and $q_i$-LRU. They all evict the least recently requested content, if space is needed, but they adopt different criteria to decide if storing a new content after a miss:

- LRU always stores it;
- SIZE stores it if 1) its size is below a given threshold $T$, or 2) it has been requested at least $N$ times, including once during the previous $M$ hours;
- $q_i$-LRU stores it with probability $q_i$, as explained in the previous sections.

So, in addition to comparing $q_i$-LRU to the traditional LRU policy, we also consider the SIZE policy since small objects are the ones that have a bigger impact on the HDD, in terms of their service time $T(s_i)$ (see also Fig. 2). We therefore prioritize small objects, and we store objects bigger than the threshold $T$ only after they have been requested for at least $N$ times. The SIZE policy can thus be seen as a first attempt to decrease the impact of small objects on the HDD, and ultimately reduce the strain on HDD resources. With the $q_i$-LRU policy, we aim at the same goal, but modulate the probability to store an object in RAM as a function of its size, and thus of its service time.

Note that the hit ratio of the whole cache depends only on the size of the HDD and its replacement policy (LRU). The RAM replacement policy does not affect the global hit ratio. In what follows, we focus rather on the number of requests served by the RAM and by the disk. More precisely, we consider the total disk service time: this is the sum of the $T(s_i)$ of all the objects served by the HDD. Smaller disk service times indicate lower pressure on the disk.

We show the results for a system with 4 GB RAM and 3 TB HDD. We have tried many different values for the RAM size up to 30 GB, and the qualitative results are similar (not shown here because of space constraints). For the SIZE policy, we have extensively explored the parameter space (threshold $T$, number of requests $N$, and number of hours $M$) finding similar qualitative results.\(^{11}\) For the $q_i$-LRU policy, the default value of the constant $\beta$ is chosen such that $\min_{i \in X} q_i = 0.1$ (see Eq. (8)).

B. Trace characteristics

We consider two traces with different durations and collected from two different vantage points. The first trace has been collected for 30 days in May 2015, while the second trace for 5 days at the beginning of November 2015. Table II shows the basic characteristics of the traces.

\(^{10}\)As a future work, we plan to deploy our policy in a real production system. In this case, the methodology to perform a comparative analysis is substantially different.

\(^{11}\)As a representative set of results, we show here the case with $T = 256$ KB, $N = 5$ and $M = 1$ hour.
Fig. 3 shows the number of requests for each object, sorted by rank (in terms of popularity), for both traces. For the 30-day trace, there are 25-30 highly requested objects (almost 25% of the requests are for those few objects), but the cumulative size of these objects is less than 8 MB. Since they are extremely popular objects, any policy we consider stores them in RAM, so they are not responsible for the different performance we observe for the different policies.

Next, we study the relation between the size and the number of requests of each object. In Fig. 4, for each object, we plot a point that corresponds to its size (y-axis) and the number of requests (x-axis). For the 30-day trace, the plot does not include the 30 most popular objects. We notice that the 5-day trace does not contain objects smaller than 1 kB.

This is also shown in Fig. 5, where we plot the empirical Cumulative Distribution Function (CDF) for the size of the requested objects (without aggregating requests for the same object). The 30-day trace contains a lot of requests for small objects, while the 5-day trace contains requests for larger objects (e.g., see the 90-th percentile). In the 30-day trace we have then a larger variability of the ratio \( T(s)/s \) (see Fig. 2) and we expect \( q_i - \text{LRU} \) to be able to differentiate more among the different contents and then achieve more significant improvement, as it is confirmed by our results below.

![Figure 5: Cumulative fraction of the requests for objects up to a given size (for the 30-day trace, we do not include the 30 most popular objects).](image)

### Tables

**Table II**

<table>
<thead>
<tr>
<th>Traces: Basic Information.</th>
<th>30 days</th>
<th>5 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of requests received</td>
<td>( 2.22 \times 10^9 )</td>
<td>( 4.17 \times 10^8 )</td>
</tr>
<tr>
<td>Number of distinct objects</td>
<td>113.15 M</td>
<td>12.27 M</td>
</tr>
<tr>
<td>Cumulative size</td>
<td>59.45 TB</td>
<td>2.53 TB</td>
</tr>
<tr>
<td>Cumulative size of objects requested at least twice</td>
<td>20.36 TB</td>
<td>1.50 TB</td>
</tr>
</tbody>
</table>

**Table III**

<table>
<thead>
<tr>
<th>Results for the 30-day Trace with 4 GB RAM.</th>
<th>% reqs</th>
<th>bytes served</th>
<th>service time</th>
<th>( \Delta ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU RAM</td>
<td>79.61</td>
<td>159 TB</td>
<td>1058 h</td>
<td>-</td>
</tr>
<tr>
<td>LRU HDD</td>
<td>20.39</td>
<td>23 TB</td>
<td>219 h</td>
<td>-</td>
</tr>
<tr>
<td>SIZE RAM</td>
<td>84.72</td>
<td>149 TB</td>
<td>1074 h</td>
<td>+ 1.51%</td>
</tr>
<tr>
<td>SIZE HDD</td>
<td>15.28</td>
<td>33 TB</td>
<td>203 h</td>
<td>-2.74%</td>
</tr>
</tbody>
</table>

**Table IV**

<table>
<thead>
<tr>
<th>Results for the 5-day Trace with 4 GB RAM.</th>
<th>% reqs</th>
<th>bytes served</th>
<th>service time</th>
<th>( \Delta ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU RAM</td>
<td>79.61</td>
<td>159 TB</td>
<td>1058 h</td>
<td>-</td>
</tr>
<tr>
<td>LRU HDD</td>
<td>20.39</td>
<td>23 TB</td>
<td>219 h</td>
<td>-</td>
</tr>
<tr>
<td>SIZE RAM</td>
<td>84.72</td>
<td>149 TB</td>
<td>1074 h</td>
<td>+ 1.51%</td>
</tr>
<tr>
<td>SIZE HDD</td>
<td>15.28</td>
<td>33 TB</td>
<td>203 h</td>
<td>-2.74%</td>
</tr>
</tbody>
</table>

### C. Comparative analysis of the eviction policies

Tables III and IV summarize the aggregate results for the two traces we consider in our study. For the hit ratio, we see that the \( q_i - \text{LRU} \) policy can serve more requests from the RAM. On the other hand, the overall number of bytes served by RAM is smaller: this means that the RAM is biased towards storing small, very popular objects, as expected. The last column shows the gain, in percentage, in disk service time between each policy and LRU, which we take as a de-facto reference (e.g., -10% for policy “x” means that its disk service time is 10% smaller than for LRU). This is the main performance metric we are interested in. For the 30-day trace, the \( q_i - \text{LRU} \) policy improves by 23% the disk service time, over the LRU policy. For the 5-day trace, the improvement of \( q_i - \text{LRU} \) over
LRU is smaller, topping at a bit more than 7%. The reason behind this result relates to the object size distribution in the trace: as shown in Fig. 5, the trace contains objects starting from 1 kB, while, for the 30-day trace, 20% of the requests are for objects smaller than 1 kB. The impact of these objects on the overall \( T(s_i) \) is significant.

Next, we take a closer look at our policy, \( q_i \)-LRU, in comparison to the reference LRU policy. We now consider the contribution to the overall hit ratio of each object, to understand their importance to cache performance. For the 5-day trace, we sorted the objects according to their rank (in terms of popularity) and their size, and plot the difference between LRU hit ratio and \( q_i \)-LRU hit ratio. Fig. 6 shows that both policies store the same 1000 most popular objects; then, the \( q_i \)-LRU policy gains in hit ratio for medium-popular objects. Switching now to object size, both policies store the same set of small objects, while \( q_i \)-LRU gains hit ratio with the medium-size objects.

![Fig. 6. Difference between LRU hit ratio and \( q_i \)-LRU hit ratio when objects are ordered by popularity (left) and by size (right) for the 30-day trace.](image)

Fig. 7 considers the contribution to the disk service time of each object (ordered by rank or by size) and shows the difference between the service time reduction under LRU and under \( q_i \)-LRU. Clearly, medium popular objects and medium size objects contribute the most to the savings in the service time that our policy achieves.

![Fig. 7. Difference between HDD service time reduction under LRU and under \( q_i \)-LRU when objects are ordered by rank (left) and by size (right) for the 30-day trace.](image)

**D. Sensitivity analysis**

Next, we study the behavior of \( q_i \)-LRU as a function of the parameter \( \beta \), but we plot the results for the parameter \( q_{\min} = \min q_i \), that is easier to interpret, being the minimum probability according to which a content is stored in the RAM.

Figure 8 provides two different views. On the left-hand side, it shows the percentage of HDD service time offloaded to the RAM by \( q_i \)-LRU, both under the 30-day trace and a synthetic IRM trace generated using the same empirical distributions for object size and popularity as in the 30-day trace. As expected, under IRM, the improvement from \( q_i \)-LRU increases as \( q_{\min} \) decreases, i.e. as \( \beta \) increases. Interestingly, the HDD benefits even more under the 30-day trace, with more than 80% of the service offloaded to the RAM. This is due to the temporal locality effect (see e.g. [11]), i.e. to the fact that requests typically occur in bursts and then the RAM is more likely to be able to serve the content for a new request than it would be under the IRM model. We observe also that the performance of \( q_i \)-LRU is not very sensitive to the parameter \( q_{\min} \) (and then to \( \beta \)), a feature very desirable for practical purposes. The right-hand side of Fig. 8 shows the relative improvement of \( q_i \)-LRU in comparison to LRU (calculated as difference of the HDD service time under LRU and under \( q_i \)-LRU, divided by the HDD service time under LRU). While \( q_i \)-LRU performs better and better as \( q_{\min} \) decreases with the IRM request pattern, the gain reduces when \( q_{\min} \) approaches 0 (\( \beta \) diverges) with the 30-day trace. This is due also to temporal locality: when the probabilities \( q_i \) are very small, many contents with limited lifetime have no chance to be stored in the RAM by \( q_i \)-LRU and they need to be served by the HDD. Despite this effect, \( q_i \)-LRU policy still outperforms LRU over a large set of parameter values and obtain improvements larger than 20% for \( 0.02 < q_{\min} < 0.4 \).

![Fig. 8. Sensitivity analysis to the value of \( q_{\min} \).](image)

**V. RELATED WORK**

Cache replacement policies have been the subject of many studies, both theoretical and experimental. We focus here on the more analytical studies, which are closer to our contribution in this paper. Moreover, our policy is explicitly designed to mitigate the burden on the HDD, a goal not considered in most previous experimental works, despite its practical importance.

Most of the theoretical work in the past has focused on the characterization of the performance of LRU, RANDOM, and FIFO [7][12][13][14]. These works do not design caching policies to solve a specific optimization problem.
The work in [15], instead, considers a 2-level hierarchy, with the content stored in the SSD and DRAM. They propose a new policy which decreases the response time by pre-fetching the content from SSD to DRAM. To this aim, they focus on a specific type of content, videos divided into chunks, for which the requests are strongly correlated, and a request for a chunk can be used to foresee future requests for other chunks of the same content. In our work, instead, we provide a model for the $q_i$-LRU policy which does not assume any correlation on the requests arrivals, but prioritize the content that imposes a high burden on the HDD.

A different approach is taken in [16]. The authors consider that caching policies could be designed with other purposes than maximizing the local hit probability. For example, they propose a heuristic that takes into account the cost to retrieve the contents from expensive inter-domain links. Cost-aware caches have been the subject of many experimental studies [17][18][19]. While these studies are similar in spirit, none of them considers cost functions analogous to the HDD service time that is the focus of this paper. Moreover, they did not prove the optimality of the replacement policies proposed.

The most related work to ours is the cache optimization framework in [9], that we have widely discussed through the paper. We stress again here, that they assume content popularities to be known (or to be explicitly estimated) and the utility functions to be strictly concave, and this is not the case in our problem.

VI. CONCLUSION

Caches represent a crucial component of the Internet architecture: decreasing the response time is one of the primary objectives of the providers operating such caches. This objective is pursued by exploiting the RAM of the cache server, while keeping most of the content in the HDD.

In this paper, we presented a new cache replacement policy that takes advantage of the access-time difference in the RAM and in the HDD to reduce the load on the HDD, so that to improve the overall cache efficiency for a capacity constrained storage systems. Our policy, called $q_i$-LRU, is a variant of $q_i$-LRU, where we assign a different probability $q_i$ to each content based on its size.

We proved that $q_i$-LRU is asymptotically optimal, and we provided an extensive trace-driven evaluation that showed between 10% and 20% reduction of the HDD load with respect to the LRU policy.

ACKNOWLEDGMENT

This work was partially supported by the “Investments for the Future” Program reference #ANR-11-LABX-0031-01, funded by the French Government (National Research Agency, ANR). The authors would like to thank Bodossaki Foundation for having supported Dimitra Tsigkari’s internship at Inria.

REFERENCES


Asymptotically Exact TTL-Approximations of the Cache Replacement Algorithms LRU(m) and h-LRU

Nicolas Gast
Inria, France

Benny Van Houdt
University of Antwerp, Belgium

Abstract—Computer system and network performance can be significantly improved by caching frequently used information. When the cache size is limited, the cache replacement algorithm has an important impact on the effectiveness of caching. In this paper we introduce time-to-live (TTL) approximations to determine the cache hit probability of two classes of cache replacement algorithms: the recently introduced h-LRU and LRU(m). These approximations only require the requests to be generated according to a general Markovian arrival process (MAP). This includes phase-type renewal processes and the IRM model as special cases.

We provide both numerical and theoretical support for the claim that the proposed TTL approximations are asymptotically exact. In particular, we show that the transient hit probability converges to the solution of a set of ODEs (under the IRM model), where the fixed point of the set of ODEs corresponds to the TTL approximation. We further show, by using synthetic and trace-based workloads, that h-LRU and LRU(m) perform alike, while the latter requires less work when a hit/miss occurs. We also show that, as opposed to LRU, h-LRU and LRU(m) are sensitive to the correlation between consecutive inter-request times.

I. INTRODUCTION

Caches form a key component of many computer networks and systems. A large variety of cache replacement algorithms has been introduced and analyzed over the last few decades. A lot of the initial work was focused on deriving explicit expressions for the cache content distribution by using a Markov chain analysis [1]. This approach, however, is not always feasible: Even if explicit expressions can be obtained, they are often only applicable to analyze small caches, because of the time it takes to evaluate them. This gave rise to various approximation algorithms to compute cache hit probabilities and most notably to time-to-live (TTL) approximations.

The first TTL approximation was introduced for the least recently used (LRU) policy under the Independent reference model (IRM) by Che et al. in [6]. The main idea behind this approximation is that an LRU cache behaves similar to a TTL cache. In a TTL cache, when an item enters the cache, it sets a deterministic timer with initial value \( T \). When this timer expires the item is removed from the cache. If an item is requested before its timer expires, its timer is reset to \( T \). When \( T \) is fixed, an item with popularity \( p_k \) is present in the cache at a random point in time with probability \( 1 - e^{-p_k T} \) and \( \sum_{k=1}^{N} [1 - e^{-p_k T}] \) is the average number of items in the cache. The Che approximation [6] consists in approximating an LRU cache of size \( m \) by a TTL cache with characteristic time \( T(m) \), where \( T(m) \) is the unique solution of the fixed point equation

\[
m = \sum_{k=1}^{N} (1 - e^{-p_k T}).
\]

The above TTL approximation for LRU can easily be generalized to renewal requests as well as to other simple variations of LRU and RANDOM under both IRM and renewal requests, as well as to certain network setups [3], [7], [13], [14]. All of these TTL approximations have been shown to be (very) accurate by means of numerical examples, but except for LRU in [8], no theoretical support was provided thus far.

In this paper we introduce TTL approximations for two classes of cache replacement algorithms that are variants of LRU. The first class, called LRU(m), dates back to the 1980s [1], while the second, called h-LRU, was recently introduced in [13]. In fact, a TTL approximation for h-LRU was also introduced in [13], but this approximation relies on an additional approximation of independence between the different lists when \( h > 2 \). As we will show in the paper, this implies that the approximation error does not reduce to zero as the cache becomes large.

In this paper we make the following contributions:

- We present a TTL approximation for LRU(m) and h-LRU that is valid when the request process of an item is a Markovian arrival process (MAP). This includes any phase-type renewal process and the IRM model. In the special case of the IRM model, we derive simple closed-form expressions for the fixed point equations.
- Our TTL approximation for h-LRU can be computed in linear time in \( h \) and appears to be asymptotically exact as the cache size grows, in contrast to the TTL approximation in [13] for \( h > 2 \). Numerical results for the TTL approximation for LRU(m) also suggest that it is asymptotically exact.
- We prove that, under the IRM model, the transient behavior of both h-LRU and LRU(m) converges to the unique solution of a system of ODEs as the cache size goes to infinity. Our TTL approximations correspond to the unique fixed point of the associated systems of ODEs. This provides additional support for the claim that our TTL approximations are asymptotically exact and is the main technical contribution of the paper.
We validate the accuracy of the TTL approximation. We show that \( h \)-LRU and LRU(\( m \)) perform alike in terms of the hit probability under both synthetic and trace-based workloads, while less work is required for LRU(\( m \)) when a hit/miss occurs.

- We indicate that both \( h \)-LRU and LRU(\( m \)) can exploit correlation in consecutive inter-request times of an item, while the hit probability of LRU is insensitive to this type of correlation.

The paper is structured as follows. We recall the definitions of LRU(\( m \)) and \( h \)-LRU in Section II. We show how to build and solve the TTL-approximation for LRU(\( m \)) in Section III-A, and for \( h \)-LRU in Section III-B. We demonstrate the accuracy of the TTL-approximation for any finite time period in Section IV. We compare LRU(\( m \)) and \( h \)-LRU in Section V, by using synthetic data and real traces. We conclude in Section VI. This paper is complemented by a technical report [10] that contains most of the proofs.

II. REPLACEMENT ALGORITHMS

We consider two families of cache replacement algorithms: \( h \)-LRU, introduced and called \( k \)-LRU in [13], and LRU(\( m \)), introduced in [1], [9]. Both operate on a cache that can store up to \( m \) items and both are variants of LRU, which replaces the least-recently-used item in the cache. One way to regard LRU is to think of the cache as an ordered list of \( m \) items, where the \( i \)-th position is occupied by the \( i \)-th most-recently-used item. When a miss occurs, the item in the last position of the list is removed and the requested item is inserted at the front of the list. If a hit occurs on the item in position \( i \), item \( i \) moves to the front of the list, meaning the items in position \( 1 \) to \( i - 1 \) move back one position.

The \( h \)-LRU replacement algorithm: \( h \)-LRU manages a cache of size \( m \) by making use of \( h - 1 \) additional virtual lists of size \( m \) (called list 1 to list \( h - 1 \)) in which only meta-data is stored and one list of size \( m \) that correspond to the actual cache (called list \( h \)). Each list is ordered, and the item in the \( i \)-th position of list \( \ell \) is the \( i \)-th most-recently-used item among the items in list \( \ell \). When item \( k \) is requested, two operations are performed:

- For each list \( \ell \) in which item \( k \) appears (say in a position \( i \)), the item \( k \) moves to the first position of list \( \ell \) and the items in positions 1 to \( i - 1 \) move back one position.
- For each list \( \ell \) in which item \( k \) does not appear but appears in list \( \ell - 1 \), item \( k \) is inserted in the first position of list \( \ell \), all other items of list \( \ell \) are moved back one position and the item that was in position \( m \) of list \( \ell \) is discarded from list \( \ell \).

List 1 of \( h \)-LRU behaves exactly as LRU, except that only the meta-data of the items is stored. Also, an item can appear in any subset of the \( h \) lists at the same time. This implies that a request can lead to as many as \( h \) list updates. Note that there is no need for all of the \( h \) lists to have the same size \( m \).

The LRU(\( m \)) replacement algorithm: LRU(\( m \)) makes use of \( h \) lists of sizes \( m_1, \ldots, m_h \), where the first few lists may be virtual, i.e., contain meta-data only. If the first \( v \) lists are virtual we have \( m_{v+1} + \cdots + m_h = m \) (that is, only the items in lists \( v+1 \) to \( h \) are stored in the cache). With LRU(\( m \)) each item appears in at most one of the \( h \) lists at any given time. Upon each request of an item:

- If this item is not in the cache, it moves to the first position of list 1 and all other items of list 1 move back one position. The item that was in position \( m_1 \) of list 1 is discarded.
- If this item is in position \( i \) of a list \( \ell < h \), it is removed from list \( \ell \) and inserted in the first position of list \( \ell + 1 \). All other items of list \( \ell + 1 \) move back one position and the item in the last position of list \( \ell + 1 \) is removed from list \( \ell + 1 \) and inserted in the first position of list \( \ell \). All previous items from position 1 to \( i - 1 \) of list \( \ell \) move back one position.
- If this item is in position \( i \) of list \( h \), then this item moves to the first position of list \( h \). All items that are in position \( 1 \) to \( i - 1 \) of list \( h \) move back one position.

When using only one list, LRU(\( m \)) coincides with LRU, and therefore with 1-LRU.

III. TTL-APPROXIMATIONS

A. TTL-approximation for LRU(\( m \))

1) IRM setting: Under the IRM model the string of requested items is a set of i.i.d. random variables, where item \( k \) is requested with probability \( p_k \). As far as the hit probability is concerned this corresponds to assuming that item \( k \) is requested according to a Poisson process with rate \( p_k \).

The TTL-approximation for LRU(\( m \)) exists in assuming that, when an item is not requested, the time it spends in list \( \ell \) is deterministic and independent of the item. We denote this characteristic time by \( T_{\ell} \). Let \( t_n \) be the \( n \)-th time that item \( k \) is either requested or moves from one list to another list (where we state that an item is part of list 0 when not in the cache). Using the above assumption, we define an \( h + 1 \) states discrete-time Markov chain \( (X_n)_{n \geq 0} \), where \( X_n \) is equal to the list id of the list containing item \( k \) at time \( t_n \).

With probability \( e^{-p_k T_{\ell}} \) the time between two requests for item \( k \) exceeds \( T_{\ell} \). Hence, the transition matrix of \( (X_n)_{n} \) is

\[
\mathbf{P}_k = \begin{pmatrix}
0 & 1 & 0 & 1 - e^{-p_k T_{\ell}} \\
0 & 0 & 1 & e^{-p_k T_{h-1}} \\
\vdots & \vdots & \ddots & \vdots \\
0 & e^{-p_k T_{h-1}} & \cdots & 1 - e^{-p_k T_{h-1}} \\
e^{-p_k T_{\ell}} & 0 & \cdots & e^{-p_k T_{h}} \\
1 - e^{-p_k T_{\ell}} & 1 & \cdots & e^{-p_k T_{h}} \\
\end{pmatrix}
\]

The Markov chain \( X_n \) is a discrete-time birth-death process. Hence, its steady state vector \( (\pi_{k,0}, \pi_{k,1}, \ldots, \pi_{k,h}) \) obeys

\[
\pi_{k,\ell} = \pi_{k,0} \prod_{s=1}^{\ell-1} (1 - e^{-p_k T_s}) \prod_{s=1}^{h-1} e^{-p_k T_s} = \pi_{k,0} e^{p_k T_{\ell}} \prod_{s=1}^{h-1} (e^{p_k T_s} - 1),
\]

(2)
for \( \ell = 1, \ldots, h \).

Further for \( \ell \in \{1, \ldots, h\} \), the average time spent in \( \ell \) is

\[
E[t_{n+1} - t_{n} | X_n = \ell] = \int_{t = 0}^{T_{\ell}} e^{-p_{\ell} t} dt = \frac{1 - e^{-p_{\ell} T_{\ell}}}{p_{\ell}}.
\]

and \( E[t_{n+1} - t_{n} | X_n = 0] = 1/p_k \). Combined with (2), this implies that observing the system at a random point in time, that item \( k \) is in list \( \ell \) with probability

\[
\pi_{k,\ell} E[t_{n+1} - t_{n} | X_n = \ell] = (e^{p_{\ell} T_{\ell}} - 1) \frac{1}{p_k}.
\]

The expected number of items part of list \( \ell \) is the sum of the previous expression over all items \( k \). As for the Che approximation, setting this sum equal to \( m_{\ell} \) leads to the following set of fixed point equations for \( T_1 \) to \( T_h \):

\[
m_{\ell} = \sum_{k=1}^{n} \frac{(e^{p_{\ell} T_{k}} - 1)}{1 + \sum_{j=1}^{k} (e^{p_{\ell} T_{k}} - 1)}.
\]

An iterative algorithm used to determine a solution of this set of fixed point equations is presented in [10, Appendix A]. In the next section we generalize this approximation to MAP arrivals.

2) MAP arrivals: We now assume that the times that item \( k \) is requested are captured by a Markovian Arrival Process (MAP). MAPs have been developed with the aim of fitting a compact Markov model to workloads with statistical correlations and non-exponential distributions [5], [15]. A MAP is characterized by two \( d \times d \) matrices \((D_{0}(k), D_{1}(k))\), where the entry \((j, j')\) of \( D_{0}^{(k)} \) is the transition rate from state \( j \) to \( j' \) that is accompanied by an arrival and the entry \((j, j')\) of \( D_{1}^{(k)} \) is the transition rate from state \( j \) to \( j' \) (with \( j \neq j' \)) without arrival. Let \( \phi^{(k)} \) such that \( \phi^{(k)} D_{0}^{(k)} + D_{1}^{(k)} = 0 \) and \( \phi^{(k)} e = 0 \). Note, the request rate \( \lambda_k \) of item \( k \) can be expressed as \( \lambda_k = \phi^{(k)} D_{1}^{(k)} e \). Setting \( D_{1}^{(k)} = -p_k \) and \( D_{0}^{(k)} = p_k \) corresponds to the IRM case and letting \( D_{0}^{(k)} = -D_{1}^{(k)} e \) implies that item \( k \) is requested according to a phase-type renewal process characterized by \((\phi^{(k)}, D_{0}^{(k)})\).

Extending the previous section, we define a discrete-time Markov chain \((X_n, S_n)\), where \( X_n \) is the list in which item \( k \) appears and \( S_n \) is the state of the MAP process at time \( t_n \). This Markov chain has \( d(h + 1) \) states and its transition probability matrix \( P_{MAP}^{(k)} \) is given by

\[
\begin{bmatrix}
0 & -D_{1}^{(k)} & & \\
D_{0}^{(k)} T_{1} & 0 & & \\
& \ddots & \ddots & \\
& & D_{0}^{(k)} T_{h-1} & 0 \\
& & & D_{0}^{(k)} T_{h} & A_{k,h-1} & \\
& & & & D_{0}^{(k)} T_{h} & A_{k,h}
\end{bmatrix}
\]

where

\[
A_{k,\ell} = \int_{t=0}^{T_{\ell}} e^{D_{0}^{(k)} T_{\ell}} dt D_{1}^{(k)} = (I - e^{D_{0}^{(k)} T_{\ell}})(-D_{0}^{(k)})^{-1} D_{1}^{(k)}.
\]

Due to the block structure of \( P_{MAP}^{(k)} \), its steady state vector \((\bar{\pi}_{k,0}, \bar{\pi}_{k,1}, \ldots, \bar{\pi}_{k,h})\) obeys

\[
\bar{\pi}_{k,\ell} = \bar{\pi}_{k,0} \ell \prod_{s=1}^{\ell} R_{k,s},
\]

for \( \ell = 1, \ldots, h \), where the matrices \( R_{k,s} \) can be computed recursively as

\[
R_{k,h} = A_{k,h-1} (I - A_{k,h})^{-1},
\]

\[
R_{k,\ell} = A_{k,\ell-1} \left( I - R_{k,\ell+1} e^{D_{1}^{(k)}} \right)^{-1},
\]

for \( \ell = 1, \ldots, h - 1 \) and \( h > 1 \).

We also define the average time \((N_{k,\ell})_{j,j'}\) that item \( k \) spends in state \( j' \) in \((t_n, t_{n+1})\) given that \( X_n = (\ell, j) \), for \( j, j' \in \{1, \ldots, d\} \). Let \( N_{k,\ell} \) be the matrix with entry \((j, j')\) equal to \((N_{k,\ell})_{j,j'}\), then

\[
N_{k,\ell} = \int_{t=0}^{T_{\ell}} e^{D_{1}^{(k)} t} dt = (I - e^{D_{1}^{(k)} T_{\ell}})(-D_{0}^{(k)})^{-1},
\]

for \( \ell \geq 1 \) and \( N_{k,0} = (-D_{0}^{(k)})^{-1} \). The fixed point equations for \( T_1 \) to \( T_h \) given in (3) generalize to

\[
m_{\ell} = \sum_{k=1}^{n} \frac{\bar{\pi}_{k,\ell} N_{k,\ell} e}{\sum_{j=0}^{h} \bar{\pi}_{k,j} N_{k,j} e},
\]

where \( e \) is a column vector of ones. The hit probability \( h_{\ell} \) in list \( \ell \) can subsequently be computed as

\[
h_{\ell} = \frac{1}{\sum_{k=1}^{n} A_{k,\ell}} \sum_{k=1}^{n} \frac{\bar{\pi}_{k,\ell} N_{k,\ell} D_{1}^{(k)} e}{\sum_{j=0}^{h} \bar{\pi}_{k,j} N_{k,j} e},
\]

for \( \ell = 0, \ldots, h \).

B. TTL-approximation for h-LRU

1) IRM setting: As in [13], our approximation for h-LRU is obtained by assuming that an item that is not requested spends a deterministic time \( T_{\ell} \) in list \( \ell \), independently of the identity of this item. For now we assume that \( T_1 < T_2 < \ldots < T_h \). We will show that the fixed point solutions for \( T_1 \) to \( T_h \) always obey these inequalities.

We start by defining a discrete-time Markov chain \((Y_n)_{n\geq0}\) by observing the system just prior to the time epochs that item \( k \) is requested. The state space of the Markov chain is given by \{0, \ldots, h\}. We say that \( Y_n = 0 \) if item \( k \) is not in any of the lists (just prior to the \( n \)th request). Otherwise, \( Y_n = \ell \) if item \( k \) is in list \( \ell \), but is not in any of the lists \( \ell + 1 \) to \( h \).

In short, the state of the Markov chain is the largest id of the lists that contain item \( k \).

If \( Y_n = \ell \), then with probability \( 1 - e^{-p_{\ell} T_{\ell}} \), item \( k \) is requested before time \( T_{\ell} \) in which case we have \( Y_{n+1} = \ell + 1 \). Otherwise, due to our assumption that \( T_{\ell} \geq T_{\ell+1} \geq \ldots \geq T_{h} \), we have \( Y_{n+1} = 0 \) as in this case the item was discarded from
all lists. Therefore the transition probability matrix $\tilde{P}_{h,k}$ of the $h + 1$ state Markov chain $(Y_n)_{n \geq 0}$ is given by

$$
\begin{bmatrix}
  e^{-p_1 T_1} & 1 - e^{-p_1 T_1} \\
  e^{-p_2 T_2} & 1 - e^{-p_2 T_2} \\
  \vdots & \vdots \\
  e^{-p_h T_h} & 1 - e^{-p_h T_h} \\
  e^{-p_{h+1} T_{h+1}} & 1 - e^{-p_{h+1} T_{h+1}}
\end{bmatrix}
$$

(9)

Let $\tilde{\pi}^{(h,k)} = (\tilde{\pi}_0^{(h,k)}, \ldots, \tilde{\pi}_h^{(h,k)})$ be the stationary vector of $\tilde{P}_{h,k}$, then the balance equations imply:

$$
\pi^{(h,k)}_\ell = \xi_\ell \tilde{\pi}_0^{(h,k)} \prod_{s=1}^\ell (1 - e^{-p_s T_s}),
$$

(10)

for $\ell = 1, \ldots, h$, where $\xi_\ell = 1$ for $\ell < h$ and $\xi_h = e^{p_h T_h}$. The probability $\tilde{\pi}_h^{(h,k)}$ that item $k$ is in the cache just before a request (which by the PASTA property is also the steady-state probability for the item to be in the cache) can therefore be expressed as

$$
\frac{\prod_{s=1}^h (1 - e^{-p_s T_s})}{\prod_{s=1}^h (1 - e^{-p_s T_s}) + e^{-p_0 T_0} (1 + \sum_{s=1}^{h-1} \prod_{s=1}^h (1 - e^{-p_s T_s}))}.
$$

(11)

Due to the nature of $h$-LRU, $T_1$ can be found from analyzing LRU, $T_2$ from 2-LRU, etc. Thus, it suffices to define a fixed point equation for $T_h$. Under the IRM model this is simply $m = \sum_{k=1}^n \tilde{\pi}_h^{(h,k)}$ due to the PASTA property. These fixed point equations can be generalized without much effort to renewal arrivals as explained in [10, Appendix B].

The following property is proven in [10, Appendix D], where we also show that $T_1 < T_2 < \ldots < T_h$ must hold to have a fixed point.

**Proposition 1.** The fixed point equation $m = \sum_{k=1}^n \tilde{\pi}_h^{(h,k)}$ has a unique solution $T_h$ which is such that $T_h > T_{h-1}$.

When $h = 2$ Equation (11) simplifies to $(1 - e^{-p_1 T_1})(1 - e^{-p_2 T_2})/(1 - e^{-p_1 T_1} + e^{-p_2 T_2})$ which coincides with the hit probability of the so-called refined model for 2-LRU presented in [13, Eqn (9)]. For $h > 2$ only an approximation that relied on an additional approximation of independence between the $h$ lists was presented in [13, see Eqn (10)]. In Figure 1 we plotted the ratio between our approximation and the one based on (10) of [13]. The results indicate that the difference grows with $h$. We show in [10, Appendix C] that it typically decreases as the popular items gain in popularity.

As (11) does not rely on the additional independence approximation, we expect that its approximation error is smaller and even tends to zero as $m$ tends to infinity. This is confirmed by simulation and we list a small set of randomly chosen examples in Table 1 to illustrate.

2) MAP arrivals: For order $d$ MAP arrivals, characterized by $(D_0^{(k)}, D_1^{(k)})$ for item $k$, we obtain a $(h+1)d$ state MC by additionally keeping track of the MAP state immediately after the requests. The transition probability matrix has the same form as $\tilde{P}_{h,k}$, we only need to replace the probabilities of the form $e^{-p_s T_s}$ by $e^{-D_0^{(k)} T_s}(-D_0^{(k)})^{-1} D_1^{(k)}$ and $1 - e^{-p_s T_s}$ by $(I - e^{-D_0^{(k)} T_s})(-D_0^{(k)})^{-1} D_1^{(k)}$. The fixed point equation for determining $T_h$ is found as

$$
m = \sum_{k=1}^n \frac{\tilde{\pi}_h^{(h,k)} + \tilde{\pi}_h^{(h,k)}}{1/\lambda_k} (I - e^{-D_0^{(k)} T_s})(-D_0^{(k)})^{-1} D_1^{(k)},
$$

(12)

where $\lambda_k$ is the request rate of item $k$ and $\tilde{\pi}_h^{(h,k)} = \pi_0^{(h,k)} \left( \prod_{s=1}^\ell (I - e^{-D_0^{(k)} T_s})(-D_0^{(k)})^{-1} D_1^{(k)} \right) \Xi_\ell$.

for $\ell = 1, \ldots, h$, where $\Xi_\ell = I$ for $\ell < h$ and $\Xi_h = (I - e^{-D_0^{(k)} T_h})(-D_0^{(k)})^{-1} D_1^{(k)}$. Finally, let $\nu^{(k)}$ be the stochastic invariant vector of $(D_0^{(k)})^{-1} D_1^{(k)}$ that is, its $d$ entries contain the probabilities to be in state 1 to $d$
immediately after an arrival. Hence, \( \pi_0^{(h,k)} \) can be computed by noting that \( \sum_{k=0}^{h} \pi_k^{(h,k)} = 1(k) \).

IV. ASYMPTOTIC EXACTNESS OF THE APPROXIMATIONS

In this section, we give evidences that the approximations presented in the previous section are asymptotically exact as the number of items tends to infinity. We first provide numerical evidence. We then show that the transient behavior of LRU(\(m\)) and \(hl\)-LRU converges to a system of ODEs. By using a change of variable, these ODE can be transformed into PDEs whose fixed points are our TTL-approximations.

A. Numerical procedure and validation

For LRU(\(m\)), the fixed point of Equation (7) can be computed by an iterative procedure that update the values \(T_i\) in a round-robin fashion. This iterative procedure is described in [10, Appendix A] and works well for up to \(h \approx 5\) lists but can be slow for a large number of lists. The computation for \(hl\)-LRU is much faster and scales linearly with the number of lists: by construction, the first \(h-1\) lists of a \(hl\)-LRU cache behave like an \((h-1)\)-LRU cache. Once \(T_{h-1}\) has been computed, the right-hand side of the fixed point equation (12) is increasing in \(T_h\), and can therefore be easily computed with a complexity that does not depend on \(h\).

1) Accuracy for LRU(\(m\)): We show that our TTL-approximation for LRU(\(m\)) is accurate by comparing the approximation with a simulation. We assume that the inter-request times of item \(k\) follow a hyperexponential distribution with rate \(z\rho_k\) in state one and \(\rho_k/z\) in state two, while the popularity distribution is a Zipf-like distribution with parameter \(\alpha\), i.e., \(\rho_k = (1/k^\alpha)/\sum_{i=1}^{m} 1/i^\alpha\). Correlation between consecutive inter-request times is introduced using the parameter \(q \in (0,1]\). More precisely, let \((D_0^{(h)}, D_1^{(h)})\) equal

\[
p_k \begin{pmatrix} -z & 0 \\ 0 & -1/z \end{pmatrix}, q \begin{pmatrix} z & 1-z/z+1 \end{pmatrix} \begin{pmatrix} 1/z \end{pmatrix} - (1-q)D_0^{(h)}.
\]

The squared coefficient of variation (SCV) of the inter-request times of item \(k\) is given by \(2(z^2 - z + 1)/z - 1\) and the lag-1 autocorrelation of inter-request times of item \(k\) is

\[
\rho_1 = (1-q) \frac{(1-z)^2}{2(1-z)^2 + z}.
\]

In other words the lag-1 autocorrelation decreases linearly in \(q\) and setting \(q = 1\) implies that the arrival process is a renewal process with hyperexponential inter-request times.

Table II compares the accuracy of the model with time consuming simulations (based on 5 runs of 2 \(\cdot 10^6\) requests). We observe a good agreement between the TTL approximation and simulation that tends to improve with the size of the system (i.e., when \(n\) increases from 100 to 1000).

2) Accuracy for \(hl\)-LRU: For the IRM model the TTL approximation was already validated by simulation in Table I. Using the same numerical examples as for LRU(\(m\)) we demonstrate the accuracy of the TTL-approximation under MAP arrivals in Table III. Simulation results are based on 5 runs containing \(2 \cdot 10^6\) requests each and are in good agreement with the TTL-approximation.

\[
\begin{array}{ccccccc}
\hline
n & q & z & method & b_0 & b_1 & b_2 \\
\hline
100 & 1 & 2 & model & 0.26898 & 0.19304 & 0.53798 \\
10 & 0.27021 & 0.19340 & 0.53639 \\
100 & 0.3712 & 0.05889 & 0.90399 \\
100 & 0.3723 & 0.06106 & 0.90171 \\
1000 & 1 & 2 & model & 0.22530 & 0.16262 & 0.61155 \\
100 & 0.22599 & 0.16256 & 0.61145 \\
10 & 0.3112 & 0.04963 & 0.91925 \\
100 & 0.3108 & 0.04969 & 0.91923 \\
1000 & 0.1 & 2 & model & 0.21603 & 0.14526 & 0.63870 \\
100 & 0.21603 & 0.14526 & 0.63881 \\
10 & 0.3006 & 0.02044 & 0.94950 \\
100 & 0.2984 & 0.02032 & 0.94985 \\
\hline
\end{array}
\]

B. Asymptotic behavior and TTL-approximation

In this subsection, we construct two systems of ODEs that approximate the transient behavior of LRU(\(m\)) and \(hl\)-LRU. These approximations become exact as the popularity of the most popular item decreases to zero:

**Theorem 1.** Let \(H_t(t)\) be the sum of the popularity of the items of list \(t\) and \(\bar{h}_t(t)\) be the corresponding ODE approximation (Equation (18) for \(hl\)-LRU and Equation (22) for LRU(\(m\))). Then: for any time \(T\), there exists a constant \(C\) such that

\[
E \left[ \sup_{t \leq T} \left| H_t(t) - \bar{h}_t(t) \right| \right] \leq C \sqrt{\max_k p_k},
\]

where \(C\) does not depend on the probabilities \(p_1, \ldots, p_m\), the cache size \(m\) or the number of items \(n\).

Our proof of this result is to use an alternative representation of the state space that allows us to use techniques from stochastic approximation. We present the main ideas in this paper while the technical details are provided in [10, Appendix E].

We associate to each item \(k\) a variable \(T_k(t)\) that is called the request time of item \(k\) at time \(t\) and an additional variable that tracks if an item appears in a list. Our approximation is given by an ordinary differential equation (ODE) on \(x_{k,h}(t)\) that is an approximation of the probability that \(T_k(t)\) is greater than one of the two deadlines.

**Table II**

<table>
<thead>
<tr>
<th>(n)</th>
<th>(q)</th>
<th>(z)</th>
<th>method</th>
<th>(b_0)</th>
<th>(b_1)</th>
<th>(b_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>2</td>
<td>model</td>
<td>0.26898</td>
<td>0.19304</td>
<td>0.53798</td>
</tr>
<tr>
<td>simul.</td>
<td>0.27021</td>
<td>0.19340</td>
<td>0.53639</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>model</td>
<td>0.3712</td>
<td>0.05889</td>
<td>0.90399</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simul.</td>
<td>0.3723</td>
<td>0.06106</td>
<td>0.90171</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>model</td>
<td>0.22530</td>
<td>0.16262</td>
<td>0.61155</td>
</tr>
<tr>
<td>simul.</td>
<td>0.22599</td>
<td>0.16256</td>
<td>0.61145</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>model</td>
<td>0.3112</td>
<td>0.04963</td>
<td>0.91925</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simul.</td>
<td>0.3108</td>
<td>0.04969</td>
<td>0.91923</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.1</td>
<td>2</td>
<td>model</td>
<td>0.21603</td>
<td>0.14526</td>
<td>0.63870</td>
</tr>
<tr>
<td>simul.</td>
<td>0.21603</td>
<td>0.14526</td>
<td>0.63881</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>model</td>
<td>0.3006</td>
<td>0.02044</td>
<td>0.94950</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simul.</td>
<td>0.2984</td>
<td>0.02032</td>
<td>0.94985</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table III**

<table>
<thead>
<tr>
<th>(n)</th>
<th>(q)</th>
<th>(z)</th>
<th>method</th>
<th>(h = 2)</th>
<th>(h = 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>2</td>
<td>model</td>
<td>0.33619</td>
<td>0.34292</td>
</tr>
<tr>
<td>simul.</td>
<td>0.53449</td>
<td>0.54150</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>model</td>
<td>0.88249</td>
<td>0.83718</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simul.</td>
<td>0.87936</td>
<td>0.83449</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>2</td>
<td>model</td>
<td>0.61028</td>
<td>0.61605</td>
</tr>
<tr>
<td>simul.</td>
<td>0.61016</td>
<td>0.61587</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>model</td>
<td>0.90103</td>
<td>0.86300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simul.</td>
<td>0.90071</td>
<td>0.86262</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.1</td>
<td>2</td>
<td>model</td>
<td>0.64744</td>
<td>0.65841</td>
</tr>
<tr>
<td>simul.</td>
<td>0.64807</td>
<td>0.65899</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>model</td>
<td>0.94935</td>
<td>0.94646</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simul.</td>
<td>0.94924</td>
<td>0.94632</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
than $b$ while appearing in a list $l$. A more natural representation would be to consider the time since the last request. Our ODE approximation would then be replaced by a partial differential equation (PDE) by replacing the ODE in $x_{k,t,b}$ by a PDE in $y_{k,t,s}$, where $y_{k,t,s}(t) = x_{k,t,t-s}(t)$. However, when working directly with the PDE, the proofs are much more complex. In each case, we show that the fixed point of the PDE corresponds to the TTL-approximation of LRU($m$) and $h$-LRU presented in Sections III-A and III-B.

To ease the presentation, we present the convergence result when the arrivals follow an IRM model, where each item $k$ has a probability $p_k$ of being requested at each time step. This proof can be adapted to the case of MAP arrivals but at the price of more complex notations. Indeed, for IRM, our system of ODEs is given by the variables $x_{k,t,b}(t)$ which are essentially an approximation of the probability for item $k$ to be in a list $l$ while having been requested between $b$ and $t$. If the arrival process of an item is modeled by a MAP with $d$ states, then our approximation would need to consider $x_{k,t,b,l}(t)$ which would approximate the probabilities for item $k$ to be in state $j$, in list $l$ and having been requested between $b$ and $t$. A detailed proof for the case of MAP arrivals is beyond the scope of this paper, both because of space constraints and for the sake of clarity of the exposition.

1) LRU: We first construct the ODE approximation for LRU. In this simpler case the proof of the validity of the Che-approximation could rely on a more direct argument that uses the closed-form expression for the steady state distribution of LRU, as in [8]. Yet, the ideas presented in this section serve to illustrate the more complex cases of $h$-LRU and LRU($m$).

The request time of an item $k$ evolves as follows:

$$\tau_k(t+1) = \left\{ \begin{array}{ll} \tau_k(t) & \text{if } k \text{ is not requested} \\ t+1 & \text{if } k \text{ is requested.} \end{array} \right. \quad (13)$$

At time 0, $\tau_k(0) = -1$ if the item is in the $r$th position in the cache and $\tau_k(0) = -(m+1)$ if the item is not in the cache.

The cache contains $m$ items. We denote $\Theta(t) = \sup \{ b : \sum_{k=1}^n 1\{\tau_k(t) \geq b\} \geq m \}$ the request time of the $m$th most recently requested item. When using LRU, an item having a request time greater or equal to $\Theta(t)$ is in the cache. Let $H(t)$ be the sum of the popularities of items in the cache:

$$H(t) = \sum_{k=1}^n p_k 1\{x_k(t) \geq \Theta(t)\}. \quad (14)$$

Our approximation of the probability for item $k$ to have a request time after $b$, is given by the following ODE (for $b < t$):

$$\dot{x}_{k,b}(t) = p_k (1 - x_{k,b}(t)). \quad (15)$$

with the initial conditions that for $t > 0$, $x_{k,t}(t) = 0$ and for $t = 0$, $x_{k,b}(0) = 1\{\tau_k(t) \geq \Theta(t)\}$. Similarly to the stochastic system, we define $\theta(t) = \sup \{ b : \sum_{k=1}^n x_{k,b}(t) \geq m \}$, which is the time for which the sum of the approximated probabilities of having items requested after $b$ is equal to $m$. The approximation of the hit ratio for LRU is then given by

$$h(t) = \sum_{k=1}^n p_k x_{k,b}(t).$$

Once these variables have been defined, the key ingredient of the proof is to use the same changes of variables as in the proof of Theorem 6 of [9], which is to consider $P_{\alpha,b}(t)$:

$$P_{\alpha,b}(t) = a^{1-\alpha} \sum_{k=1}^n (p_k)^\alpha 1\{x_k(t) \geq b\},$$

where $a := \max_{k=1}^n p_k$. These variables are defined for $\alpha \in \{0,1,\ldots\}$ and $b \in \mathbb{Z}$. The collection of variables $\{P_{\alpha,b}\}_{\alpha,b}$ describes completely the state of the system at time $t$ and lives in a set of infinite dimension.

Similarly, we define a set of functions $\rho_{\alpha,b}$ by $\rho_{\alpha,b} = a^{1-\alpha} \sum_{k=1}^n (p_k)^\alpha x_{k,b}(t)$. The functions $\rho_{\alpha,b}$ are solutions of the system of ODEs $d/dt \rho_{\alpha,b}(t) = f_{\alpha,b}(\rho)$, where

$$f_{\alpha,b}(\rho) = a^{1-\alpha} \sum_{k=1}^n (p_k)^{\alpha+1} - a \rho_{\alpha-1,1}(t).$$

The proof of the theorem, detailed in [10, Appendix E1], relies on classical results of stochastic approximation. It uses the fact that

- the function $f$ is Lipschitz-continuous
- $f$ is the $E[P_{\alpha,b}(t+1) - P_{\alpha,b}(t) | P(t)] = f_{\alpha,b}(P(t))$
- The second moment of the variation of $P(t)$ is bounded:

$$\text{E} \left[ \|P(t+1) - P(t)\|_2^2 | P \right] \leq a.$$

Note that Equation (14) can be transformed into a PDE by considering the change of variable $y_{k,s}(t) = x_{k,t-s}(t)$. The quantity $y_{k,s}(t)$ is an approximation of the probability for an item $k$ to have been requested between $t - s$ and $t$. The set of ordinary differential Equations (14) can then be naturally transformed in the following PDE:

$$\frac{\partial}{\partial t} y_{k,s}(t) = p_k (1 - y_{k,s}(t)) - \frac{\partial}{\partial s} y_{k,s}(t).$$

The fixed point $y$ of the PDE can be obtained by solving the equation $\frac{\partial}{\partial t} y = 0$. This fixed point satisfies $y_{k,s} = 1 - e^{-ps}$. For this fixed point, the quantity $T = t - \theta$ satisfies $m = \sum_{k=1}^n (1 - e^{-ps})$. This equation is the same as the Che-approximation, given by Equation (1).

2) h-LRU: The construction for h-LRU can be extended to the case of h-LRU by adding to each item $h$ variables $L_{k,\ell}(t) \in \{true, false\}$. For item $k$ and a list $l$, $L_{k,\ell}(t)$ equals true if item $k$ was present in list $l$ just after the last request of item $k$ and false otherwise. Similarly to the case of LRU, we define the quantity $\Theta_{\ell}(t)$ to be the request time of the least recently requested item that belongs to list $\ell$ at time $t$, that is,

$$\Theta_{\ell}(t) = \sup \{ b : \sum_{k=1}^n 1\{\tau_k(t) \geq b \land L_{k,\ell}(t) \geq m \} \}.$$
We then define \( \dot{x}_{k,t,b}(t) \) that is an approximation of the probability for item \( k \) to have \( \tau_k(t) \geq t \) and \( L(t) = \text{true} \).

As \( L(t) \) is always equal to true, the ODE approximation for \( x_{k,1,b}(t) \) is the same as (14). Moreover, this implies that \( \Theta_1(t) \geq \Theta_2(t) \) for \( \ell \geq 2 \). For the list \( \ell = 2 \), the approximation is obtained by considering the evolution of \( L(t) \). After a request, \( L(t+1) \) is true if \( \tau_k(t) \geq \Theta_1(t) \) or \( \tau_k(t) \geq \Theta_2(t) \) and \( L(t) = \text{true} \). Both these events occur if \( \tau_k(t) \geq \Theta_1(t) \) and \( L(t+1) = \text{true} \) as \( \Theta_1(t) \geq \Theta_2(t) \). This suggests that, if the item \( k \) is requested, then, in average \( L(t+1) \) is approximately \( x_{k,2,\theta_1(t)} + x_{k,2,\theta_2(t)} - x_{k,2,\theta_1(t)} \), which leads to the following ODE approximation for \( x_{k,2,b} \):

\[
\dot{x}_{k,2,b} = p_k(x_{k,2,\theta_1(t)} + x_{k,1,\theta_1(t)} - x_{k,2,\theta_1(t)} - x_{k,2,b}),
\]

where \( \theta_1(t) = \sup\{b : \sum_{k=1}^{n} x_{k,\ell,b}(t) \geq m_k\} \) for \( \ell \in \{1,2\} \).

The formulation for the third list and above is more complex. In Section III-B, we showed that, for the computational system, we do not necessarily have \( \Theta_1(t) \geq \Theta_2(t) \) when \( \ell \geq 2 \), which implies that the ODE approximation for \( h \)-LRU has \( 2^\ell - 1 \) terms.

Applying the reasoning of \( L(t) \) to compute \( \dot{L}_{k,\ell}(\ell \geq 3) \) involves computing the probability of \( \tau_k(t) \geq \Theta_{\ell-1}(t) \) and \( L(t) = \text{true} \) or \( \tau_k(t) \geq \Theta_2(t) \) and \( L(t) = \text{true} \).

When \( \Theta_1(t) \leq \Theta_{\ell-1}(t) \), both these events occur if \( \tau_k(t) \geq \Theta_{\ell-1}(t) \) and \( L_{k,\ell}(t) = L_{k,\ell-1}(t) = \text{true} \). This suggest that the ODE for \( x_{k,\ell,b}(t) \) involves a term \( x_{k,2,\ell-1,b}(t) \), that is an approximation for the item \( k \) to have a request time after \( \Theta_{\ell-1}(t) \) and such that \( L(t) = L_{k,\ell}(t) = \text{true} \). Note, for \( \ell = 2 \) we have \( x_{k,1,\ell-1,b}(t) = x_{k,2,\ell,b}(t) \) as \( L_{k,1}(t) \) is always true, but this does not hold for \( \ell > 2 \). This leads to:

\[
\dot{x}_{k,\ell,b} = p_k(x_{k,\ell,b}(t) + x_{k,\ell-1,\theta_{\ell-1}(t)} - x_{k,\ell-1,naxx(\theta_{\ell-1}(t),\theta_1(t))} - x_{k,\ell,b}).
\]

A similar reasoning can be applied to obtain an ODE for \( x_{\ell,b}(t) \) as a function of \( x_{\ell-1,b}(t) \), \( x_{\ell-2,b}(t) \) and \( x_{\ell-2,b}(t) \). For example, for \( \ell = 3 \) this approximation becomes

\[
\dot{x}_{k,2,3,b}(t) = x_{k,2,3,b}(t) + x_{k,2,3,b}(t) - x_{k,2,3,b}(t) - x_{k,2,3,b}(t)
\]

as \( L_{k,1}(t) \) is always true.

The hit probability for list \( \ell \) used in Theorem 1 is then

\[
h_\ell(t) = \sum_{k=1}^{n} x_{k,\ell,\theta_1(t)},
\]

where the variables \( x_{k,\ell,b} \) satisfy the above ODE.

The proof of Theorem 1 in the case of \( h \)-LRU is very similar as the one for LRU and uses the same stochastic

\[
\text{when } h = 3, \text{ the variables } \Theta_1(t) \text{ are not always ordered. For example, consider the case of four items } \{1,2,3,4\} \text{ and } m_1 = m_2 = m_3 = 3. \text{ If initially the three caches contain the three items 1,2,3. Then, after a stream of requests: 4, 3, 2, 1, the cache 1 and 3 will contain the items } \{1,2,3\} \text{ while the cache 2 will contain } \{1,2,4\}. \text{ This implies that } t - 3 = \Theta_2(t) < \Theta_3(t) = \Theta_1(t) = t - 2.
\]

The proof of Theorem 1 in the case of \( h \)-LRU is very similar as the one for LRU and uses the same stochastic

\[
\text{when } h = 3, \text{ the variables } \Theta_1(t) \text{ are not always ordered. For example, consider the case of four items } \{1,2,3,4\} \text{ and } m_1 = m_2 = m_3 = 3. \text{ If initially the three caches contain the three items 1,2,3. Then, after a stream of requests: 4, 3, 2, 1, the cache 1 and 3 will contain the items } \{1,2,3\} \text{ while the cache 2 will contain } \{1,2,4\}. \text{ This implies that } t - 3 = \Theta_2(t) < \Theta_3(t) = \Theta_1(t) = t - 2.
\]
A. Synthetic (static) workloads

For the synthetic workloads we restrict ourselves to LRU, 2-LRU and LRU(m, m). The latter two algorithms both use a cache of size m and additionally keep track of meta-data only for the m items in list 1.

Figure 2 depicts the hit probability as a function of the cache size when n = 1000, items follow a Zipf-like popularity distribution with parameter 0.8 under IRM and renewal requests (with z = 10, see Section IV-A1). Figure 3 shows the impact of having correlation between consecutive inter-request times (that is, q = 1/20 instead of q = 1 for z = 2, 10).

One of the main observations is that LRU(m, m) performs very similar to 2-LRU under IRM, renewal and MAP requests. In fact, 2-LRU performs slightly better, unless the workload is very dynamic (z = 10 and q = 1 case). Another important observation that can be drawn from comparing Figures 2 and 3 is that the hit rate of both 2-LRU and LRU(m, m) significantly improves in the presence of correlation between consecutive inter-request times (that is, when q < 1), while LRU does not. Recall that LRU(m) needs to update at most one list per hit, as opposed to h-LRU. Thus, whenever both algorithms perform alike, LRU(m) may be more attractive to use.

Figure 4 shows that the hit rate of 2-LRU and LRU(m, m) both increase with increasing lag-1 autocorrelation and more importantly that the hit probability of LRU is completely insensitive to any correlation between consecutive inter-request times. Figure 4 further indicates that the hit probability also increases with ρ1 when splitting the cache in two lists of equal size (although the gain is less pronounced). As indicated by the following theorem, whose proof is given in [10, Appendix F], the insensitivity of LRU is a general result.

Theorem 2. Assume that the items’ request processes are stationary, independent of each other and that the expected number of requests per unit time is positive and finite. Then, the hit probability of LRU only depends on the inter-arrival time distribution. In particular, it does not depend on the correlation between inter-arrival times.

This theorem complements the results of Jelenkovic and Radovanovic who showed in [12], [11] that for dependent request processes, the hit probability is asymptotically, for large cache sizes, the same as in the corresponding LRU system with i.i.d. requests. Our insensitivity result is valid not just asymptotically but requires the request processes of the various items to be independent.

B. Trace-based simulation

To perform the trace-based simulations we rely on the same 4 IR cache traces as in [4, Section 4]. In this section, we only report the result for the trace collected on Monday 18th Feb 2013. We also simulated the other traces and obtained very similar results.
The hit probability of LRU(m) with a split cache and/or virtual lists normalized by the LRU hit probability is depicted in Figure 5 as a function of the cache size \( m \). It indicates that LRU(m) is more effective than LRU, especially when the cache is small. For small caches using a virtual list is better than splitting the cache and using both a virtual list and split cache offers only a slight additional gain. While not depicted here, we should note that using more virtual lists or splitting the cache in more parts sometimes result in a hit probability below the LRU hit probability for larger caches.

Figure 6 compares h-LRU with LRU(m) using virtual lists, where the hit probability is now normalized by the hit probability of LRU(m, m) to better highlight the differences. We observe that 2-LRU differs by less than 1\% from LRU(m, m), while 5-LRU and LRU(m, m, m, m, m) differ by less than 2\%. Given that h-LRU may require an update of up to \( h \) lists, while LRU(m) requires only one update in case of a hit, LRU(m) seems preferential in this particular case.

VI. CONCLUSION

In this paper, we developed algorithms to approximate the hit probability of the cache replacement policies LRU(m) and h-LRU. These algorithms rely on an equivalence between LRU-based and TTL-based cache replacement algorithms. We showed numerically that the TTL-approximations are very accurate for moderate cache sizes and appear asymptotically exact as the cache size grows. We also provide theoretical support for this claim, by establishing a bound between the transient dynamics of both policies and a set of ODEs whose fixed-point coincides with the proposed TTL-approximation.

A possible extension of our results would be to study networks of caches in which LRU, LRU(m) or h-LRU is used in each node. Further, our TTL-approximation with MAP arrivals can be readily adapted to other policies such as FIFO(m) and RAND(m) introduced in [9]. In fact, a generalization to a network of caches would be fairly straightforward for the class of RAND(m) policies.

REFERENCES

Meeting Soft Deadlines in Single- and Multi-Server Systems

Esa Hyytiä
Department of Computer Science
University of Iceland

Rhonda Righter
Department of Industrial Engineering
and Operations Research, UC Berkeley

Jorma Virtamo
Department of Communications and Networking, Aalto University

Abstract—We consider single- and multi-server systems, where jobs have a maximum waiting time (deadline) defined, e.g., by a service level agreement. A fixed cost is associated with deadline violations and the task is to minimize the long-run cumulative costs. Job sizes (service durations) are observed upon arrival, and current queue backlogs are known. For a single FCFS server, the optimization task is to find the optimal admission policy that may reject a job upon arrival if admitting it would cause in future one or more deadlines to be violated (in expectation). For parallel FCFS servers, the policy must (i) either accept or reject a job upon arrival, and if accepted, (ii) assign it to one of the servers. We derive efficient deadline-aware policies in the MDP framework. For a single server, we obtain the optimal admission policy. For dispatching to parallel servers, we develop efficient heuristic admission and dispatching policies, whose performances are evaluated by means of numerical examples. Additionally, we give some exact closed-form results for heavy-traffic limits.

Index Terms—admission control; task assignment; deadline; QoE; parallel processing; cloud computing; non-linear cost

I. INTRODUCTION

Today’s Internet is full of services, requiring rapid and timely responses. For example, for interactive applications such as the large-scale online services provided by Google, Facebook and Amazon, subsecond response times are a common objective [1]. In particular, tails of the service time are seen as one of the most crucial performance measures [2] affecting the quality of experience (QoE), i.e., how customers perceive the service, which then eventually translates to profits (or losses). Therefore, in this paper we assume jobs have deadlines for waiting times until the start of service. If this deadline is exceeded, a fixed cost is incurred. In particular, we assume a best effort loss system where a job missing its deadline may as well be discarded without any additional cost other than the cost due to the missed deadline.

In general, the possible controls to achieve SLA goals are (i) admission control, (ii) dispatching rules, (iii) scheduling within a server / data-center, (iv) job migration, and (v) control of service rate, e.g., via number of servers or speed scaling. We consider the first two: admission control and dispatching.

We first consider a single-server system modelled as an M/G/1-FCFS queue, where jobs have a maximum waiting time \( \tau \) that should not be exceeded, or the fixed unit cost is incurred. The dynamic decision problem is to find the optimal admission policy that minimizes long run costs. Then we consider admission and dispatching to a set of parallel heterogeneous FCFS servers. The routing decision must be made upon arrival and is irrevocable, i.e., a job cannot be moved to another queue later. (Job reallocations can incur large overheads.)

Our main contributions are the following: (i) we derive new theoretical results for the M/G/1 queue subject to this non-linear cost structure (the so-called value function with respect to deadline violations) that enable efficient static and dynamic admission policies; (ii) we give exact closed-form expressions for the probability of deadline violations, the steady-state distribution, and the value function (at the heavy-traffic limit) in M/M/1; (iii) we devise an explicit method for solving the optimal admission policy for the M/G/1 queue; and (iv) we derive efficient heuristic admission and dispatching policies for systems of parallel servers. The performance of the obtained deadline-aware policies are further evaluated numerically, giving insight into this important cost structure.

The rest of the paper is organized as follows. Section II formally introduces the model and notation. In Section III, we analyze a single M/G/1 queue with respect to deadlines, and complement this with numerical examples in Section IV. In Section V, we apply our results to derive efficient dynamic dispatching policies for the original multi-server problem. Section VI concludes the paper.

A. Related work

Our work is different from past work in that our policies are dynamic, they minimize costs due to deadline violations, and they take into account the size of arriving jobs. Liu et al. [3] consider service-level-agreements (SLAs) in the context of e-commerce with a general class of SLAs comprising throughput, mean delay and (soft) deadlines. Web servers are modelled as a queuing network with generalized processor sharing (GPS) servers. The decision variables are the probabilities for static routing and the weights for the GPS scheduling. (Note that, e.g., Hadoop uses FCFS by default. This is often also the case in the context of supercomputing where concurrent multitasking may be impractical or even infeasible.) Saovapakhiran et al. [4] consider average delay SLAs in the context of cloud computing. The mean delays follow directly from Little’s theorem, which simplifies their analysis considerably.

Stidham [5] considers the admission problem. The models closest to ours are the admission control to an GI/M/1 queue and to two parallel GI/M/1 queues. However, the cost structure...
is different: each admitted job gives a fixed reward, while costs are incurred according to a convex holding cost rate function of number of jobs in (each) server. Lehoczky [6] studies a single GI/M/1 queue with deadlines in heavy-traffic. Glazebrook et al. [7] consider a system of parallel (exponential) servers with abandonments. These works, however, are different as the admission control and routing (if any) are based on the number of jobs. Li and Glazebrook [8] consider a single server with jobs that leave the system if their waiting time exceeds a random class-dependent waiting time. The scheduling problem is to minimize the number of abandonments. In contrast, we know the deadline of each job and obtain a critical-backlog policy that depends also on the (exact) size of the arriving job.

Gupta and Harchol-Balter [9] study a system where a PS server, preceded by a FCFS queue, admits at most k jobs concurrently. The aim is to minimize the mean delay. The minimization of the mean delay, or some other related linear quantity such as slowdown, has been the typical objective also for routing (dispatching) problems (without the option to reject jobs). See, for example, [10], [11], [12], [13], [14], and the references therein. For related scheduling problems see [15], [16], [17]. In contrast, [18] studies a closely related model with deadlines, but without the option to reject jobs.

II. PRELIMINARIES

A. Model and notation

We assume a service level agreement (SLA) in the form of the maximum waiting time τ. The maximum tolerated waiting time is simply referred to as the deadline. If this deadline is exceeded, a fixed cost of 1 is incurred. We also allow the policy to reject a job upon arrival. For simplicity, we assume the same unit cost for rejected jobs as for deadline violations (or higher cost for deadline violations), even though it would be possible to consider models where discarding a job incurs, e.g., a higher cost than missing a deadline. Consequently, we limit ourselves to admission policies that discard a new job unconditionally when the target deadline cannot be met.

More formally, the cost function is a step-function of waiting time in queue, W, where, if W = w,

\[ d_\tau(w) = I(w > \tau), \]

so the mean cost rate in terms of SLA violations is

\[ r = \lambda \mathbb{P}\{W > \tau \text{ or job is rejected}\}. \]

We consider FCFS queues, with Poisson arrival rate λ, and i.i.d. service times denoted by \( X_j \sim X \). Job sizes become known upon arrival, and hence, for a single server, u, the backlog in the queue, gives the relevant state information. In the multi-server setting, jobs are assigned (dispatched) to a server immediately upon arrival and the assignment is irrevocable. The service rate at server i is \( c_i \), so the service time of job j if assigned to server i is \( X_j/c_i \). We denote the offered load \( \lambda \mathbb{E}[X] \) by \( \rho \) (in the multi-server setting, \( \rho = \lambda \mathbb{E}[X]/\sum c_i \)), and \( f(x) \) is the pdf of the job size distribution, \( F(x) = \mathbb{P}[X \leq x] \) the corresponding cdf, and \( F(x) = 1 - F(x) \).

B. Value functions

The value function, well known in the context of Markov decision processes [19], [20], is defined as the expected difference in costs between a system that is initially in a given state s and a system in equilibrium,

\[ v(s) := \lim_{t \to \infty} \mathbb{E}[V_t(s) - r_t], \]

where the random variable \( V_t(s) \) denotes the costs the system incurs during \((0, t)\) when initially in state \( s \), and \( r \) is the long-run mean cost rate. The value function enables us to quantify how much better or worse initial state \( s_2 \) is than \( s_1 \) simply by computing \( v(s_2) - v(s_1) \), which is the important quantity for policy iteration. In this paper, we will derive and utilize value functions related to the M/G/1 queue subject to deadline cost structure.

III. SINGLE-SERVER ANALYSIS

We start by analyzing a single M/G/1 queue where jobs can be discarded upon arrival in order to minimize the long-run rate of deadline violations.

A. Admission policies \( \xi(u) \)

We let the random variable \( U \) denote the backlog in the queue, and consider admission policies in which a function \( \xi(u) \) defines a threshold for the maximum service time a job may have in order to be admitted to the queue in state \( U = u \), i.e., jobs of size \( x > \xi(u) \) are rejected. We will show that such a policy is optimal, and how it can be efficiently computed.

Because it is optimal to unconditionally discard jobs whose deadline \( \tau \) would be violated, we start with a basic policy that discards such jobs and accepts the rest:

Definition 1 (Basic admission policy):

\[ \xi_0(u) := \begin{cases} \infty, & u \leq \tau, \\ 0, & u > \tau. \end{cases} \]

This policy is also the individually optimal policy for arriving jobs to minimize their own costs. The M/G/1 queue with \( \xi_0(u) \) is clearly stable for any \( \lambda \geq 0 \), and the mean cost rate (blocking rate) is

\[ r := \lambda \mathbb{P}\{U > \tau\}. \]

This simple admission policy can be optimal:

Proposition 1: The basic admission policy \( \xi_0(u) \) is optimal for M/D/1 queues.

Proof: This follows trivially from the fact that all jobs are identical and therefore there is no reason to wait for a better job to arrive later (waiting actually includes a risk that no job arrives before the server idles).

In general \( \xi_0(u) \) is not the optimal admission policy (and more jobs will be rejected because of the negative externalities they would impose on future jobs if admitted). The standard approach to find the optimal policy in the MDP framework is based on studying the value function defined Section II-B. Before discussing this in detail, we first demonstrate how the steady-state distribution can be computed for an arbitrary admission policy \( \xi(u) \), with \( \xi(u) = 0 \) for all \( u > \tau \).
B. Steady-state distribution of backlog

In this section, we study how to determine the steady-state distribution of the backlog $U$ in a single $M/G/1$ queue subject to an arbitrary admission policy $\xi(u)$. This quantity then allows one to examine, e.g., the service-time distribution of the admitted and rejected jobs with the given $\xi(u)$.

To this end, we adapt the level-crossing methodology developed in [21], where the workload distribution in a finite size buffer was studied. In this system, jobs are admitted as long as they fit in the buffer, i.e., the size (service time) $X$ of the arriving job is less than the free space in the buffer, which in our context would translate to a deadline for job completion time (in contrast to waiting time).

For an arbitrary admission policy, $\xi(u)$, such that $\xi(u) = 0$ for $u > \tau$, we let $g(u)$ denote the unknown continuous density function of the steady-state distribution of $U$ for $U > 0$, and let $\pi_0 = \mathbb{P}(U = 0)$. The probability flow downwards across a test level at $u$ is simply $g(u)$. The probability flow upwards is due to jobs that arrive when the backlog is below the level $u$, $U < u$, and the service time of the job $X$ is sufficiently long to cause a jump across the level $u$, $U + X > u$, and the job is admitted, i.e., $X < \xi(u)$. This results in the Volterra integral equation of the second kind,

$$
g(u) = \lambda \left( \pi_0 Q(0,u) + \int_0^u g(v)Q(v,u)\,dv \right),$$

where $Q(v,u)$ is the probability that a job arriving in state $v < u$ is admitted and the backlog increases beyond $u$,

$$
Q(v,u) = (F(\xi(v)) - F(u-v))^+, 
$$

and where $(x)^+ = \max\{x, 0\}$.

Because $\xi(u) = 0$ for $u > \tau$, $Q(v,u) = 0$ for $v > \tau$, so the Volterra equation (3) reduces to

$$
g(u) = \lambda \left( \pi_0 Q(0,u) + \int_0^\tau g(v)Q(v,u)\,dv \right), \quad u > \tau, \quad (4)
$$

and it remains to determine $g(u)$ for $0 < u \leq \tau$.

In general, the above integral equation can be solved numerically by first setting $\pi_0 = 1$. Then for arbitrary $u$, (3) is defined in terms of $g(v)$ with $v \leq u$, and $g(u)$ can be worked out iteratively in the forward direction, and then the whole distribution, including the atom $\pi_0$, is normalized, as explained in [21].

Now that the equilibrium distribution of the backlog in the $M/G/1$ queue with an arbitrary admission policy $\xi(u)$ is available, several other interesting performance quantities can be determined. The job rejection rate is

$$
r = \lambda \left( 1 - \pi_0 F(\xi(0)) - \int_0^\tau g(u)F(\xi(u))\,du \right). \quad (5)
$$

The rejection probability is $\mathbb{P}\{X > \xi(U)\} = r / \lambda$, and the fraction of time the server is busy (carried work) is simply

$$
\rho^* = 1 - \pi_0.
$$

Let $p(x)$ denote the probability that a job with service time $x$ is admitted to the queue. From PASTA,

$$
p(x) = \pi_0 1(\xi(0) \geq x) + \int_0^x g(u)1(\xi(u) \geq x)\,du.
$$

When $\xi(u)$ is a decreasing function of $u$, the above simplifies significantly. For the special case of the $M/M/1$ queue with the basic admission policy $\xi_0(u)$, we have a closed-form solution:

**Proposition 2**: The steady-state distribution of the $M/M/1$ queue with deadline $\tau$ and the basic admission policy $\xi_0(u)$ is given by

$$
\pi_0 = \frac{\mu(\mu - \lambda)}{\mu^2 - \lambda^2e^{(\lambda - \mu)\tau}};
$$

$$
g(u) = \pi_0 \lambda e^{\lambda \min(u,\tau)} - \mu u. \quad (6)
$$

**Proof**: Substitute the above trial into (4).

**Corollary 3**: The mean cost rate in the $M/M/1$ queue with the basic admission policy $\xi_0(u)$ is

$$
r = \lambda \int_\tau^\infty g(u)\,du = \frac{\lambda^2(\mu - \lambda)}{\mu^2e^{(\lambda - \mu)\tau} - \lambda^2}. \quad (7)
$$

C. Value function for the $M/G/1$ queue

For an arbitrary $\xi(u)$ with $\xi(u) = 0$ for $u > \tau$, we obtain the following general result for the value function $v(u)$.

**Proposition 4**: The value function $v(u)$ for the $M/G/1$ queue with arbitrary admission policy $\xi(u)$ with respect to deadline violations is

$$
v(u) - v(\tau) = (\lambda - \mu)(u - \tau), \quad u > \tau, \quad (8)
$$

and, for $0 < u \leq \tau$, with $y = \xi(u)$, it satisfies

$$
v'(u) = -r + \lambda F(y) + \lambda F(y)\mathbb{E}[v(u + X) - v(u) | X \leq y]
$$

$$
= -r + \lambda F(y) + \lambda \int_0^y f(x) [v(u + x) - v(u)]\,dx \quad (9)
$$

with the boundary condition $v'(0) = 0$.

**Proof**: When $u > \tau$ no new jobs are accepted until the backlog decreases to the level $\tau$, after a deterministic time interval $u - \tau$. Thus, $v(u) - v(\tau)$ represents the difference between the mean number of jobs arriving in this interval and the costs that the system on average incurs in the same interval (without conditioning on the initial state, i.e., starting in equilibrium), yielding (8).

For $0 < u \leq \tau$, consider an indefinitely small time interval $\delta$ such that $\delta < u \leq \tau$, and let $y = \xi(u)$. Then, referring to Figure 1, we have, by definition,

$$
v(u) = (\lambda F(y) - r)\delta + \lambda F(y)\mathbb{E}[v(u + X) | X \leq y]
$$

$$
+ (1 - \lambda F(y))v(u - \delta),
$$

which leads to (9).

The boundary condition $v'(0) = 0$ follows by considering an initially empty system, $u = 0$, until the arrival of the first accepted job. The mean time to this event is $1/(\lambda F(y))$, where $y = \xi(0)$. It follows that

$$
v(0) = \frac{\lambda F(y) - r}{\lambda F(y)} + \mathbb{E}[v(X) | X \leq y].
$$

Substitution into (9) with $u = 0$ yields $v'(0) = 0$.
We also have, with \( y = \xi(\tau) \),
\[
\begin{cases}
v'(\tau^-) = (1 + \rho F(y))(\lambda - r) - \lambda F(y) \\
v'(\tau^+) = \lambda - r.
\end{cases}
\] (10)

**D. On computing \( v(u) \) for M/G/1**

Note that, given \( r \) and \( \xi(u) \), \( v'(u) \) in (9) depends only on the values of \( v(u') \) with \( u' \geq u \), so using (8), it is possible to solve the differential equation backwards from \( u = \tau \) to \( u = 0 \). The problem is that in general we do not know the mean cost rate \( r \) even for the basic admission policy \( \xi_0(u) \). One option is to solve the Volterra equation (4) and then utilize (5) to determine \( r \). Alternatively, it is also possible to estimate \( r \) by process simulation.

However, in the case of a discrete-state-space system, solving Howard’s equations would yield both the relative values (i.e., the value function apart from an unimportant additive constant) and the mean cost rate. It turns out that this is the case also in our problem when the boundary condition (9) is taken into account. When the differential equation (9) is solved (numerically) backwards from \( u = \tau \) to \( u = 0 \), e.g., using the Runge-Kutta method with a given value of \( r \), the resulting value \( v'(0) \) at the origin depends parametrically on \( r \). The value of \( r \) is then uniquely determined by the condition \( v'(0)[r] = 0 \). Generally, this is a non-linear equation and has to be solved numerically using some iterative method.

The situation is illustrated in Figure 2, where the correct \( r \) is determined for the M/M/1 queue with the basic admission policy \( \xi_0(u) \). The left graph shows three solutions for the differential equation obtained with \( r \in \{0.2, 0.25, 0.3\} \), and the right graph depicts \( v'(0) \) as a function of (trial) \( r \). We observe that the differential equation system behaves systematically and it is straightforward to determine the correct \( r \) that satisfies the boundary condition of (9). As a result, we obtain both the value function and the mean cost rate for a given \( \xi(u) \).

**E. Exact solutions with \( \xi_0(u) \) when \( \rho \to 1 \) and \( \rho \to \infty \)**

We first consider the M/M/1 queue when \( \rho \to 1 \), and then the M/G/1 queue when \( \rho \to \infty \). If jobs missing their deadlines were not discarded, these queues would become unstable. However, stability is not an issue in our case. Interestingly, it turns out that in these specific cases, we obtain exact closed-form expressions for several important quantities.

For the M/M/1 queue at the “heavy-traffic” limit where \( \rho \to 1 \), we have the following result.

**Proposition 5:** For the M/M/1 queue with \( \lambda = \mu = 1 \) and \( \tau = 2 \), the value function is initially a quadratic and then a linear function of the backlog \( u \),
\[
v(u) - v(\tau) = \begin{cases}
\frac{1 + \rho}{2 + \lambda \tau} (u - \tau), & u > \tau, \\
\frac{\lambda^2}{4 + 2 \lambda \tau} (u^2 - \tau^2), & 0 \leq u \leq \tau,
\end{cases}
\] (11)
where
\[
v(\tau) = \frac{(\lambda \tau)^2 (3 + 2 \lambda \tau) / 6 - 1 - \lambda \tau}{(2 + \lambda \tau)^2}.
\] (12)

**Proof:** The value function (11) can be shown to satisfy (8) and (9) simply by substitution, and hence it is the correct value function. For the constant term \( v(\tau) \), we utilize the identity
\[
\mathbb{E}[v(U)] = \pi_0 v(0) + \int_0^\infty g(u) v(u) \, du = 0,
\]
where \( \pi_0 \) and \( g(u) \) define the steady-state distribution of the backlog, given by (6) when \( \lambda \to \mu \). Hence,
\[
\pi_0 (v(0) - v(\tau)) + \int_0^\infty g(u) (v(u) - v(\tau)) \, du = -v(\tau),
\]
and substituting (11) into the left-hand side gives (12). ~

Even though a queue rejecting jobs when the backlog \( U > \tau \) is always stable (for any \( \lambda \geq 0 \), solving the value function numerically for \( u < \tau \) from the differential equation (9) becomes difficult even for M/M/1 when \( \lambda \) is large, as small errors in \( v(u') \) for \( u' \geq u \) can have a big impact on \( v'(u) \), and consequently, on the solution itself. Fortunately, the asymptotic case \( \lambda \to \infty \) can be deduced for general service-time distributions:

**Lemma 6:** When \( \lambda \to \infty \), the value function for M/G/1 with \( \xi_0(u) \) satisfies
\[
v(u) - v(\tau) = \frac{u - \tau}{\mathbb{E}[X]}, \quad u \geq 0.
\] (13)

**Proof:** Suppose \( u \geq \tau \). When \( \lambda \) is large, the backlog in the queue decreases only slightly below \( \tau \) before a new job arrives. Consequently, the job admission rate, \( \lambda - r \), tends to \( 1 / \mathbb{E}[X] \) as \( \lambda \to \infty \), and (8) reduces to
\[
v(u) - v(\tau) = \frac{u - \tau}{\mathbb{E}[X]}, \quad u \geq \tau.
\] (14)

Suppose next that \( u < \tau \). Let \( N \) denote the number of (practically instantaneously) admitted jobs until the backlog
exceeds $\tau$, i.e., $N$ is the smallest number (“stopping time”) such that
\[ u + X_1 + \ldots + X_N > \tau. \]

Focusing on the admitted jobs, we have
\[ v(u) = E[-N + v(u + X_1 + \ldots + X_N)]. \]
Applying (14) on the right-hand side then gives,
\[ v(u) = -E[N] + \frac{E[u - \tau + X_1 + \ldots + X_N]}{E[X]} + v(\tau). \]

For the random sum, we can apply Wald’s theorem [19],
\[ v(u) - v(\tau) = -E[N] + \frac{u - \tau}{E[X]} + \frac{E[N]E[X]}{E[X]} = \frac{u - \tau}{E[X]}, \]
and (14) holds for all $u \geq 0$.

The reference state can be chosen arbitrarily, and, e.g.,
\[ v(u) - v(0) = \frac{u}{E[X]} \tag{15} \]

Note also that with $\lambda - r = 1/E[X]$, the derivative $v'(\tau^-)$ of (10) reduces to $\lambda - r$ equaling $v'(\tau^+)$, i.e., in this limit, there is no jump in the derivative at $u = \tau$. This is obvious also from (13) and (15), which are valid for all $u \geq 0$.

**Proposition 7.** The value function for M/G/1 with $\xi_0(u)$ in the heavy-traffic limit, when $\lambda \to \infty$, is
\[ v(u) = \frac{u - \tau}{E[X]} - \frac{E[X^2]}{E[X]^2}. \tag{16} \]

**Proof:** At the heavy-traffic limit, a job with service time $X$ is admitted to the system each time and immediately after the backlog decreases below $u = \tau$. That is, jobs are admitted after i.i.d. time intervals $X$. Consequently, the residual time to next arrival has pdf
\[ f_\tau(t) = \frac{t f(t)}{E[X]}. \]
Hence, the steady-state distribution at the heavy-traffic limit is $g(u) = 0$ for $u < \tau$, and for states $u = \tau + t \geq \tau$,
\[ g(\tau + t) = \frac{t f(t)}{E[X]}, \quad t \geq 0. \]

The identity $\int_0^\infty g(u) v(u) du = 0$ (as $\pi_0 = 0$) with (13) gives
\[ v(\tau) = -\int_0^\infty t f(t) \frac{t}{E[X]} dt = \frac{E[X^2]}{E[X]^2}. \]

Substituting this into (13) completes the proof.

Trivially, when $\lambda \to 0$, no job arrives, no deadlines are violated, and therefore both $r \to 0$ and $v(u) \to 0$. Hence, for very small values of $\lambda$, the value function is practically constant zero, then at $\rho = 1$ we obtain the quadratic form (for $u < \tau$ and M/M/1), which then transforms to the straight line with slope $\mu$ as $\rho \to \infty$ (for a general service-time distribution). This is illustrated in Figure 3 with the standard M/M/1 queue.

---

**F. Quadratic approximation for $v(u)$**

Motivated by the special case of M/M/1 with $\xi_0(u)$ and $\rho = 1$, we propose a quadratic approximation for $v(u)$ when $0 \leq u \leq \tau$. This approximation can be expected to be a good match when $\rho$ is not too large. (See the previous section. When $\rho$ is large, linear approximation should be used instead.) In particular, our proposal is
\[ \hat{v}(u) - \hat{v}(\tau) = \begin{cases} (\lambda - r)(u - \tau), & u > \tau, \\ A(\rho^2 - \tau^2), & 0 \leq u \leq \tau, \end{cases} \]
where the constant $A$ can be defined in different ways. First, requiring that $\hat{v}(0) - \hat{v}(\tau) = v(0) - v(\tau)$, yields
\[ A_1 = v(\tau) - v(0). \tag{17} \]

Second, we can set $\hat{v}(\tau^-) = v'(\tau^-)$ (in addition to $\hat{v}'(u) = v'(u) = 0$) and use (10), yielding
\[ A_2 = \frac{(1 + \rho F(y))(\lambda - r) - \lambda F(y)}{2\tau}, \tag{18} \]
with $y = \xi(\tau)$. We recall that for the M/M/1 queue with $\xi_0(u)$ and $\rho = 1$, the value function is quadratic, these expressions are exact, and $A_1 = A_2$. However, in general, with an arbitrary $\xi(u)$ and an arbitrary arrival process, this is not the case. (See Figure 5, discussed later.) This approximation will be exploited in Sections IV-A and V-C.

**G. Policy iteration in a single M/G/1 queue**

Let us now consider the single M/G/1 queue, starting with using $\xi_0(u)$. Let $v(u)$ and $r$ denote the value function and mean cost rate with $\xi_0(u)$.

a) **Policy iteration:** Suppose $u \leq \tau$ when a job with service time $x$ arrives, so $\xi_0(u)$ would accept the job, but that may not be the optimal action. The first policy iteration (FPI) step rejects the job if the expected increase in future costs would be higher than the cost of discarding the job immediately, i.e., if
\[ v(u + x) - v(u) > 1. \]

As $v(u)$ is an increasing function of $u$ for $\xi_0(u)$ (and for any other reasonable admission policy) and it has a linear tail with positive slope $\lambda - r$, the above defines a new admission policy when $u \leq \tau$, $\xi_{FPI}(u) = y$, where $y$ is such that $v(u + y) - v(u) = 1$. Note that $\xi_{FPI}(u)$ is finite for every $u \leq \tau$. 

---

Fig. 3. Value functions for M/M/1 with $\tau=2$ when $\rho=\{0.5, 1, 2, 4\}$, and at the heavy-traffic limit when $\rho \to \infty$ (dashed line).
In this section, we determine the optimal admission policy for an arbitrary M/G/1 queue with respect to deadline violations. The results in Section III-C hold for an arbitrary admission rule \( \xi(u) \). The optimal admission rule is obviously also some threshold policy \( \xi_{\text{opt}}(u) \), with value function \( v_{\text{opt}}(u) \).

**Corollary 9:** For \( u \leq \tau \), and for \( y = \xi_{\text{opt}}(u) \) such that
\[
 v_{\text{opt}}(u + y) - v_{\text{opt}}(u) = 1, \tag{22}
\]
\[
 v_{\text{opt}}'(u) = \lambda F(y) - r + \lambda F(y) E[v_{\text{opt}}(u + X) - v_{\text{opt}}(u) \mid X < y]. \tag{23}
\]

The key observation is that \( y = \xi_{\text{opt}}(u) \), so \( v_{\text{opt}}'(u) \) depends solely on \( v_{\text{opt}}(u') \) with \( u' \geq u \). As we know \( v_{\text{opt}}(u) \) for \( u \geq \tau \), it is again possible to solve the differential equation backwards, eventually giving us the value function \( v_{\text{opt}}(u) \), the mean cost rate \( r_{\text{opt}} \), and the optimal admission policy \( \xi_{\text{opt}}(u) \).

As explained in Section III-D, for a fixed \( \xi(u) \) and unknown \( r \), we can either (i) solve (9) multiple times iteratively in order to determine the correct \( r \), or (ii) solve the Volterra equation (4) first that gives the correct \( r \), and then solve (9) once. For determining the optimal admission policy and the corresponding mean cost rate \( r_{\text{opt}} \), however, the only possibility is to use method (i), since \( \xi_{\text{opt}} \) is unknown as long as \( r_{\text{opt}} \) is unknown, and hence the Volterra equation cannot be used to determine \( r_{\text{opt}} \). In contrast, it is easier to use the approximations discussed in Section III-F, and they yield near-optimal solutions.

**IV. Single-Server Experiments**

In this section, we give several numerical examples with single-server systems illustrating both the theoretical results derived in the earlier section, and some interesting phenomena caused by the deadline cost structure itself.

**A. MM/1**

In Figure 5, we have depicted several value functions and the corresponding admission policies resulting from one policy iteration round for the M/M/1 queue with \( \mu = 1, \tau = 2 \), and \( \rho = 1 \) and \( \rho = 2 \). By \( F(\xi) \) we mean one policy iteration round from \( \xi \), and Optimal is the optimal admission policy. The value function corresponding to \( \xi_{\text{opt}}(u) \) is computed backwards from the admission rule. Interestingly, FPI(\( \xi_{\text{opt}} \)) and Optimal are practically equivalent.

With \( \rho = 1 \) (upper row), the two quadratic approximations for the value function of \( \xi_{\text{opt}}(u) \) are exact and not shown. However, with \( \rho = 2 \) the corresponding value function is not a quadratic function, and therefore the approximations using \( A_1 \) and \( A_2 \) indeed deviate from the exact value function. Also, \( A_2 \), defined by the derivative \( v'(\tau^-) \), yields a value function that is similar to that of the optimal policy.

**B. Steady-state distribution**

The steady-state distribution of \( U \) for the M/M/1 queue with \( \xi_{\text{opt}}(u) \) and \( \tau = 2 \) is given by (6) and depicted in Figure 6 (left) for \( \rho = 0.75, 1, 1.5 \). With \( \rho = 1 \), the steady-state distribution is flat for \( 0 \leq u \leq \tau \), whereas with \( \rho \neq 1 \) it is either exponentially decreasing (\( \rho < 1 \)) or increasing (\( \rho > 1 \)). For
such that the mean service time is $\mu = 1$ and $\sigma = 1$, while $\varphi = 1$ by more than $\mu = 1$. The optimal admission policy and the mean cost rate. Figure 7 depicts the deadline violation probability as a function of the offered load $\rho$ for these three service-time distributions with the optimal admission policies. When $\rho$ is low, the fixed service time of M/D/1 yields the lowest deadline violation rate. However, as the offered load increases, it becomes obvious that a high variance can actually be advantageous. The reason for this is that when service times vary and jobs are plenty, it is possible to save many mice by discarding a few elephants.

### C. M/G/1

Let us next study the effect of the service-time distribution on the deadline violation rate in the M/G/1 queue with the optimal admission policy. We consider three Weibull distributions, $X_i \sim \text{Weibull}(\alpha_i, \beta_i)$, $i = 1, 2, 3$, with parameters $(\alpha_i, \beta_i)$ such that the mean service time is $E[X_i] = 1$, while the variance is varied, $\sigma_i^2 \in \{0, 1, 5\}$. With $\sigma_i^2 = 0$, we obtain the M/D/1 queue, $\sigma_i^2 = 1$ corresponds to the M/M/1 queue, and $\sigma_i^2 = 5$ then represents an M/G/1 queue with a more variable service-time distribution. For Proposition 1, $\bar{\xi}_0(u)$ is optimal for the M/D/1 queue, and the mean cost rate can be obtained by solving (9) (or by means of simulation). For the other two queues, we need to solve (23) in order to determine the optimal admission policy and the mean cost rate.

Figure 7 depicts the deadline violation probability as a function of the offered load $\rho$ for these three service-time distributions with the optimal admission policies. When $\rho$ is low, the fixed service time of M/D/1 yields the lowest deadline violation rate. However, as the offered load increases, it becomes obvious that a high variance can actually be advantageous. The reason for this is that when service times vary and jobs are plenty, it is possible to save many mice by discarding a few elephants.

### V. PARALLEL SERVERS

We now use the value functions for single M/G/1 queues to develop efficient dispatching and admission rules for the parallel server system, as in [22] and [20, Section 11.5].

#### A. Model and heuristic policies

We let $s$ denote the state of the system, $s = (u_1, \ldots, u_n)$, where $u_i$ is the backlog in server $i$. The admission and routing decision for a job with size $x$ is denoted by $\alpha = \alpha(s, x)$,

$$\alpha(s, x) = \begin{cases} 0, & \text{if the job is rejected}, \\ i, & \text{if the job is routed to server } i. \end{cases}$$ (24)
The basic static dispatching policy is the Bernoulli split (RND), which assigns a job to server \(i\) with probability \(p_i\), \(i = 1, \ldots, n\), and where \(p_i\) is chosen to balance the load. Given the actual service times and backlogs are available, the commonly used dynamic heuristic dispatching policy is least-work-left (LWL). This policy assigns the new job to the queue with the shortest backlog (in time). Hence, it is the optimal myopic policy with respect to both waiting time and deadline violations. In addition to dispatching, the control policy \(\alpha\) should exercise some admission control such as the basic admission rule (1).

The multi-server setting is fundamentally more complicated than a single-server system because there are both admission and dispatching decisions to be made, and the state-space is multi-dimensional. Not surprisingly, only a few optimality results are available (even without the option to reject jobs), and these typically assume identical servers, exponentially distributed service times, and minimizing mean latency as the objective, and they do not use arrival job size information. In this paper, we give an optimality result in the multi-server setting only for the trivial case when all jobs and all servers are identical. Its proof is the same as that of Proposition 1.

**Proposition 10:** When all jobs are identical, \(x_i = x\), and all servers are identical, \(c_i = c\), then LWL-opt, choosing the queue with the shortest backlog, and rejecting jobs whose deadline would be violated, is the optimal policy.

**B. Admission control with policy iteration**

With Poisson arrivals, whenever jobs are assigned in i.i.d. fashion, then each server \(i\) also receives jobs according to a Poisson process, and the whole system decomposes into \(n\) independent M/G/1 queues. Given value functions for the individual M/G/1 queues, the value function of the whole system is

\[
v(s) = \sum_i v_i(u_i),
\]

where \(u_i\) denotes the backlog in queue \(i\). Then \(d_i(u_i) = I(u_i > \tau)\), for \(i = 1, \ldots, n\), corresponds to the so-called immediate cost for queue \(i\) (whether the deadline violation occurs or not). We let \(i = 0\) denote the action of rejecting a job, and correspondingly, we define \(d_0(u) = 1\) and \(v_0(u) = 0\). Consequently, we can carry out FPI, which gives an improved job admission and dispatching rule

\[
\alpha(s, x) = \arg \min_{i \in \{0, \ldots, n\}} d_i(u_i) + v_i(u_i + x/c_i) - v_i(u_i),
\]

where the difference in the value function corresponds to the expected increase in future costs.

**Remark 11:** Consider a system of identical servers with (uniform) RND as the dispatching policy and a common arbitrary \(\xi(u)\) as the admission rule so that the queue-specific value functions are identical convex functions. Then the admission cost \(a(u, x)\) is an increasing function of \(u\), and applying FPI gives LWL-opt. That is, jobs are routed according to LWL, and then rejected if \(x > \xi_{FPI}(u)\). In particular, starting from RND-opt yields LWL-opt.

There are two observations to be made at this point. First, each policy iteration round improves both the admission and dispatching rules. Second, starting from RND yields “only” LWL, which, though reasonable, is not the optimal dispatching policy in general, because it ignores arriving job sizes (see later numerical examples).

In the ideal case, we would carry out the second policy iteration round. Unfortunately, computing the value function for a dynamic policy such as LWL is hard, and one has to resort to more ad hoc solutions such as the so-called Lookahead policy improvement (LPI) [23], where the idea is to consider also the assignment of the next arriving job, after which the (static) basic policy takes over, i.e., the basic action is \((i, j)\) assigning the current job to queue \(i\) and the next (tentatively) to queue \(j\).

In the following two sections, we will evaluate the FPI and LPI policies and compare their performance to that of RND and LWL, augmented with different admission rules.

**C. Two identical servers**

Let us first consider a small system of \(n = 2\) identical exponential servers with \(\mu = 1\) and \(\tau = 2\). We consider two heuristic reference dispatching policies: static RND and dynamic LWL. Both dispatching policies are complemented with admission control at each server. In particular, we consider two admission policies, \(\xi_0(u)\), and the optimal one, \(\xi_{\text{opt}}(u)\) (assuming RND), and denote the corresponding complete control policies as \(\text{RND-}\xi_0, \text{RND-}\xi_{\text{opt}}, \text{LWL-}\xi_0\) and \(\text{LWL-}\xi_{\text{opt}}\), respectively. We note that LWL is a dynamic dispatching policy for which we do not know the optimal admission control, and therefore we resort to the admission policy that is optimal for RND.

Additionally, we have FPI and LPI based on the value function derived in Section III. We use the quadratic approximation \(A_2\) for \(0 \leq u < \tau\) and \(\text{RND-}\xi_{\text{opt}}\) as the starting point. We also experimented with \(\text{RND-}\xi_0\). As the policy iteration step also improves the admission rule, the results were only marginally worse than with \(\xi_{\text{opt}}\).

The numerical results are depicted in Figure 8 (left). We can see that FPI-\(\xi_{\text{opt}}\) (i.e., LWL-\(\xi_{\text{opt}}\), see Remark 11) is strong, and LPI-\(\xi_{\text{opt}}\) is even better. Moreover, LWL-\(\xi_0\) is initially good (it is optimal when \(\rho = 0\)), but becomes weak as the load approaches \(\rho = 1\) and (longer) jobs need to be rejected. With heterogeneous servers, LWL-\(\xi_{\text{opt}}\) is no longer the same as FPI-\(\xi_{\text{opt}}\), and in fact the latter turns out
to be a clearly better policy. Finally, LPI is the best control policy especially when $\rho$ is moderate or high, and large jobs should be rejected proactively.

VI. CONCLUSIONS

Our deadline-based cost structure is motivated by the need to provide fast responses in today’s large-scale cloud based services; “respond promptly or not at all”. We derived value functions and the optimal admissions policy for single-server systems. These results were complemented with expressions for the steady-state distribution of backlog and other important performance metrics (e.g., deadline violation rate, carried load), including exact closed-form results in specific (heavy-traffic) limits. We applied our single-server results to systems of parallel servers to develop efficient deadline-aware job admission and dispatching policies. Numerical examples show that our heuristics are superior to standard policies, especially with heterogeneous servers under heavy load. Finally, we note that our results are also useful for more complicated systems of parallel servers, including power-of-two type approaches, where admission and dispatching decisions are based on a small (random) subset of servers. A more detailed investigation of this research direction is left as future work.

ACKNOWLEDGEMENTS

This work was supported by the Academy of Finland in the FQ4BD and TOP-Energy projects (grant nos. 296206 and 268992).

REFERENCES

Performance Analysis of CoDel and PIE for Saturated TCP Sources

Fabian Schwarzkopf, Sebastian Veith, Michael Menth,
Chair of Communication Networks (KN), University of Tuebingen, Sand 13, 72076 Tuebingen, Germany

Abstract—In the recent years, the bufferbloat phenomenon was observed which is mainly due to oversized unmanaged buffers in the Internet. This triggered a new discussion of active queue management (AQM) algorithms in the IETF. “Controlled Delay” (CoDel) and “Proportional Integral controller Enhanced” (PIE) are considered as an alternative to “Random Early Detection” (RED). Their intention is both to take advantage of large buffers for occasional bursts and to limit queuing delays most of the time. Moreover, they are able to cope with varying bandwidth.

In this paper, we study the performance of CoDel, PIE, and CoDel-ACT, which is an effective modification of CoDel that leads to better performance than CoDel in our studies. We experiment with saturated TCP sources and a fixed-bandwidth bottleneck link and focus on the delay-limiting phase of the algorithms. We investigate the impact of configuration parameters and traffic load on link utilization and queuing delay. We study the timely evolution of queuing delays and drop patterns, and point out significant differences among the algorithms. In particular, we show that CoDel’s drop behavior changes over time and may lead to underutilization.

I. INTRODUCTION

Active queue management (AQM) mechanisms for the Internet have been discussed since the early ’90s [1] and Random Early Detection (RED) [2] has been proposed as a standard [3]. Although implemented in many devices, network operators hesitate to turn RED on because it requires careful adjustment to the specific network environment. In 2009 excessive packet delay was observed in wireless networks [4] due to oversized buffers and persistently full queues. Two years later, this phenomenon was referred to as bufferbloat [5]. It underlined the importance of managing buffers. As the community felt that RED is not sufficient to control queues, the IETF working group “Active Queue Management and Packet Scheduling” (aqm)1 was founded in 2013 and has adopted two major AQM algorithms for potential standardization. “Controlled Delay” (CoDel) [6] which was originally proposed in 2012 [7] and “Proportional Integral controller Enhanced” (PIE) [8] which was originally published in 2013 [9]. Both algorithms have the benefit that they cope with varying bandwidth as they rather control delay than queue size. They achieve that goal with two different strategies. CoDel monitors the actual queuing delay of dequeued packets; if the delay is continuously above a certain threshold for a given interval, CoDel drops packets with excess delay on dequeue. PIE first measures the departure rate on the bottleneck link and predicts the queuing delay. Then, a target delay, the estimated delay, and its recent changes contribute to the calculation of a packet loss probability based on which PIE possibly drops packets on enqueue. Thus, both approaches are significantly different. We also consider a variant of CoDel that modifies the growth of CoDel’s internal count value as well as the time to wait before dropping mode starts. We refer to this variant by CoDel-ACT (CoDel-adapted-count-and-time).

If AQMs are very aggressive, the utilization of a potential bottleneck link may suffer. Thus, there is a tradeoff between low queuing delay and high resource utilization. In this paper, we investigate the three AQM algorithms CoDel, CoDel-ACT, and PIE. We illustrate an initial burst-allowing phase and a succeeding delay-limiting phase in the presence of saturated TCP flows. We investigate the impact of the algorithms and their configuration parameters on queuing delay and utilization for the delay-limiting phase. A time-dependent analysis of the AQM algorithms reveals significant differences among their drop patterns and explains the observed phenomena. The findings contribute to a deeper understanding of the algorithms that are currently under standardization in IETF.

The rest of the paper is structured as follows. Section II reviews the AQM algorithms under study. Section III describes the simulation setup and discusses simulation results, giving statistical evidence and visual insights about the performance behavior and differences of CoDel, CoDel-ACT, and PIE. Section IV reviews related work, and Section V summarizes most important findings of our work and draws conclusions.

II. ALGORITHMS UNDER STUDY

A. CoDel

For our simulations we implemented the C++-like pseudocode that is given for CoDel in [6]. CoDel equips packets on queue with a timestamp so that it can calculate their queuing time on dequeue. The procedure dequeue() dequeues a packet and returns a pointer to it (r,p) as well as a boolean r.ok_to_drop which is true if the queuing time has continuously exceeded the delay threshold target for at least interval time. The parameters target and interval are recommended to be set to 5 ms and 100 ms, respectively, and in particular independently of the networking scenario. As CoDel has no other parameters, it can be used without parameter adaptation. The actual logic of CoDel is given in Listing 1 which is performed whenever a packet is dequeued.

1https://datatracker.ietf.org/wg/aqm/history/
CoDel uses some state variables. The boolean dropping indicates whether CoDel may drop packets if two additional conditions are also met. To drop a packet, its ok_to_drop must be true, otherwise CoDel leaves its dropping mode. CoDel controls its drop frequency by ensuring that the next packet is not dropped before time next_drop. The time between successive values of next_drop depends on the persistence of the observed congestion which is tracked by the integer count. As the count variable becomes large if the queuing delay exceeds target for long time, the time between successive values of next_drop becomes short which increases the drop rate.

Listing 1. CoDel’s dequeue algorithm.

```c
Packet* CoDelQueue::deque() {
    double now = clock(); // current time
dodequeResult r = dequeue();
    if (dropping) {
        if (!r.ok_to_drop) {
            r = drop(r.p);
        }
        while (now >= drop_next && dropping) {
            drop(r.p);
            r = dequeue();
            if (r.ok_to_drop) {
                dropping = false;
            } else {
                count = count + 1;
                drop_next = drop_next + interval / sqrt(count);
            }
        } else if (r.ok_to_drop) {
            drop(r.p);
            dropping = true;
        if (count > 2 && now-drop_next < 8*interval) {
            // count = 1;
            // count = count - 2;
        } else {
            count = count - 2;
        }
        drop_next = now + interval/sqrt(count);
    }
    return (r.p);
}
```

As CoDel's algorithm in Listing 1 is not our contribution, we leave its study to the reader. However, we discuss some observation that the reader should understand about the algorithm. CoDel enters its dropping mode only if the queuing delay exceeds target for more than interval time, and stays in dropping mode until a dequeued packet's delay falls below target. After packet loss, CoDel determines the next next_drop time, and after subsequent packet loss within a dropping phase, CoDel increments count. This leads to increasing drop rates until queuing delay decreases and CoDel leaves its dropping mode. If CoDel reenters the dropping mode, the count value is reset to 1 only if the last dropping phase was sufficiently long ago, otherwise count is set to an only slightly smaller value than before. Note that CoDel can drop several consecutive packets at once because the next value of next_drop is determined relative to the last value of next_drop and not relative to the current time unless CoDel just entered the dropping mode.

C. PIE

We implemented PIE according to the appendix of the IETF draft in [8] but corrected some obvious glitches in the pseudocode. The changes comprise the reset of the accumulated drop probability at each drop as well as prevention of a negative drop probability p by limiting its range to [0,1].

PIE updates its drop probability p every t_update time (16 ms by default) using the currently observed queuing delay D_q and the control parameter qdelay_ref (16 ms by default). On arrival, a packet is dropped with drop probability p unless the queue is obviously not congested which is expressed in [8] through appropriate conditions, e.g., if the current queuing delay is less than qdelay_ref/2. We first explain how the current queuing delay D_q is measured and then how p is adapted.

The current delay D_q is estimated by Little’s law which requires the departure rate. The latter is obtained by the following concurrent process. If the queue contains more than T_{DQ} bytes and is currently not in a measurement state, then a new departure rate measurement starts at time t_i. The measurement stops at time t_{i+1} when T_{DQ} bytes are dequeued. The current dequeue time D is calculated according to Equation (1) and smoothed to obtain the average dequeue time D_{avg} according to Equation (2). Then, the departure rate R is computed like in Equation (3) which allows to estimate

variants lead to an algorithm with suitable properties that K. Nichols has suggested in 2012. However, its benefits were not quantified at that time so that it was no longer considered.

CoDel-ACT is a modification of CoDel that addresses the fact that CoDel waits a fixed time interval before reentering the dropping mode and that CoDel sets the new value of count to count-2 so that count can continuously grow on persistent congestion. To prevent this behavior, K. Nichols suggested two modifications. The first modification requires that the queuing delay exceeds target for at least interval / sqrt(count) time instead of a fixed duration of interval before the procedure dequeue sets r.ok_to_drop to true for a packet so that CoDel switches to dropping mode. The second modification decays count when entering the dropping mode using the following code:

Listing 2. CoDel-ACT’s modification to CoDel’s dequeue algorithm addressing the *-marked lines in Listing 1.

```c
if (count > 2 && now-drop_next < 8*interval) {
    count = count - 2;
} else {
    count = count - 2;
}
```

The decay applies only if count is greater than 126. If so, the function scales count with the decay parameter 0.9844 rather than just subtracting a fixed value of 2. Thus, this modification reduces count more strongly than the current IETF variant in [6] when entering the dropping mode.
Using Equation (4), given the queue length \( L_q \). If the queue contains less than \( T_{DQ} \) bytes, the rate \( R \) cannot be updated and the last value of \( R \) is reused for the estimation of the current queuing delay \( D_q \).

\[
\Delta = t_{i+1} - t_i \tag{1}
\]

\[
\Delta_{avg} = \frac{1}{4} \cdot \Delta + \frac{3}{4} \cdot \Delta_{avg} \tag{2}
\]

\[
R = \frac{T_{DQ}}{\Delta_{avg}} \tag{3}
\]

\[
D_q = \frac{L_q}{R} \tag{4}
\]

The drop probability \( p \) is periodically updated similarly as in Equations (5) and (6) using the current queuing delay \( D_q \), the previous queuing delay \( D_{q\text{prev}} \), and the control parameter \( q_{\text{delay_ref}} \), the intervals.

The equations in [8] include in addition a case analysis of \( p \) that we omit for brevity. Equation (5) uses the two factors \( \alpha \) and \( \beta \) for which values are recommended in [8]. While CoDel updates its state variable \( \text{count} \) only in dropping mode, PIE updates its state variable \( p \) also when the estimated queuing delay \( D_q \) is below the reference value \( q_{\text{delay_ref}} \).

\[
p = p + \alpha \cdot (D_q - q_{\text{delay_ref}}) + \beta \cdot (D_q - D_{q\text{prev}}) \tag{5}
\]

\[
D_{q\text{prev}} = D_q \tag{6}
\]

The draft [8] extended the original algorithm in [9] with some modifications that were proposed by CableLabs simulations. The modifications comprise (1) a de-randomization of drop events and (2) an on/off mechanism of the algorithm. The first modification accumulates the current drop probability \( p \) at every packet enqueue using the variable \( \text{accu}_\text{prob} \). As long as \( \text{accu}_\text{prob} < 0.85 \), no packets are dropped. If \( \text{accu}_\text{prob} \) becomes \( \geq 8.5 \), then the packet is dropped in any case. If \( \text{accu}_\text{prob} \) ranges between these limits, drops are randomly performed with probability \( p \). Every drop resets \( \text{accu}_\text{prob} \). The second modification describes the inactivation of PIE with a reset of PIE’s internal variables in the absence of congestion. By default, PIE is reactivated if the queue length exceeds one third of the total size. The second modification is irrelevant for our simulations because PIE is never deactivated due to lasting congestion in our experiments.

III. Results

In this section we first describe our simulation setup. Then we study queuing delays with CoDel, CoDel-ACT, and PIE and illustrate that these algorithms exhibit in the presence of saturated TCP sources first a burst-allowing phase and then a delay-limiting phase. In this work we focus on the delay-limiting phase. We first provide evidence about queuing delay distributions depending on the traffic load. Then we study the impact of configuration parameters on queuing delay and utilization of the bottleneck link. As we observe a non-monotone dependency of CoDel’s utilization on the traffic load, we perform a time-dependent analysis to better understand the performance behavior of the AQM algorithms. To that end, we investigate how state variables, drop rates, drop patterns, and queuing delays evolve during the delay-limiting phase.

A. Simulator and Network Topology

All simulations were performed with INET 2.4.0 [10] in the OMNet++ network simulation framework 4.4.1 [11]. We used the Network Simulation Cradle 0.5.3 [12] to simulate TCP sources, which facilitates the application of real world network stacks from Linux kernels in simulation programs. All simulations are conducted with Linux kernel 2.6.29.

We simulate clients that are connected to a server over a high-bandwidth link with 1 Gb/s and a shared bottleneck link with 10 Mb/s, which results in a one-sided dumbbell topology. We configured a one-way propagation delay of 0.1 ms for the fast access link and 5 ms for the bottleneck link, which yields a minimum round-trip time (RTT) of about 10 ms.

We choose a buffer size for the bottleneck link of 250 KB, i.e., two times the bandwidth delay product at an RTT of 100 ms. We set the buffer so large to study the AQM mechanisms with negligible amount of tail drops. We investigate the presented AQM algorithms for control of the queue on the bottleneck link with varying traffic load in terms of 1, 4, 16, and 64 saturated TCP NewReno flows. The flows were randomly started within the first second of a simulation run.

If not mentioned differently, we configure CoDel and PIE with their recommended default parameters: target=5 ms, interval=100 ms, q-delay_ref=16 ms, and t-update=16 ms. We configure CoDel-ACT with the same parameters as CoDel. While we provide only figures for 10 ms RTT and TCP NewReno in this paper, we run the same experiments with TCP CUBIC and/or 100 ms RTT. Their results are qualitatively the same, but differ in quantity.

B. Illustration of Burst-Allowing and Delay-Limiting Phase

Figure 1 illustrates the queuing delay of consecutive packets at the beginning of a simulation for 1, 4, 16, and 64 concurrent TCP flows. We observe a large spike in the first 1 s – 5 s, but then queuing delay is limited to low values. This is exactly what AQMs should do: they allow that infrequent large bursts can use the available buffer, but they avoid a standing queue under persistent load. We denote these phases as burst-allowing and delay-limiting phase. They can be observed with saturated TCP sources.

With PIE, we experience initially large delays and the duration of such phases are almost independent of the traffic load. It is limited to 1.5 s in all cases. With CoDel, initially large delays increase with traffic load and so does their duration. As a result, CoDel leads to the same queuing delay as PIE for 64 TCP flows, but the duration of large delays is almost 5 s long. CoDel-ACT reveals identical queuing delays as CoDel as their behaviors do not differ within the first few drop phases. From then on, a separate curve for CoDel-ACT is visible in the figure. At the beginning of the simulation, CoDel’s and CoDel-ACT’s \( \text{count} \) value is small so that the minimum time between drops \( \text{interval}/\sqrt{\text{count}} \) is rather large and initial bursts are cut down only slowly. As CoDel’s and CoDel-ACT’s drop rate scales with \( \text{interval} \), increasing \( \text{interval} \) also extends the initial phase with large experienced delays. PIE obviously increases its drop rate more
quickly and stops phases of increased delays earlier than CoDel while allowing the use of the same buffer size. The figure also shows that the simulated process is not stationary before 5 s. Therefore, we use only data gathered after 10 s when considering averages, distributions, or quantiles.

C. Queuing Delays in the Delay-Limiting Phase

While Figure 1 illustrates the initial queuing delay, Figure 2 quantifies the long-term queuing behavior of the considered AQM algorithms for 16 concurrent TCP flows as complementary distribution function (CDF) of the queuing delay. With CoDel, about 4% of the packets experience no queuing delay. This is a hint that the bottleneck link may be underutilized. The queuing delay of most other packets is almost equally distributed between 0 ms and 20 ms. With CoDel-ACT, there are fewer packets with no queuing delay, but most packets have a queuing delay less than 10 ms. Moreover, CoDel-ACT centers the queuing delay between 4 ms and 8 ms like a Normal distribution. The same holds for PIE but with a larger mean and variance. Although PIE leads mostly to larger queuing delays than CoDel for 16 flows, the probability for large queuing delays is higher for CoDel than for PIE. As PIE drops on enqueue and the bandwidth is constant in our simulation, the queue length distribution is very similar to the queuing delay distribution. This is different with CoDel and CoDel-ACT as they perform drop on dequeue. Therefore, observed queue lengths are larger which effects that their CDFs have a bias of about 2 – 3 packets to larger values compared to the CDF of the corresponding queuing delay.

Table I shows the mean and the 99% quantile \( (Q_{99\%}) \) of the queuing delay for different traffic load and for a minimum RTT of 10 ms and 100 ms. In all cases, the mean queuing delay of CoDel and CoDel-ACT increases with more flows while the one of PIE is about 16 ms. The 99% quantiles are significantly larger. While CoDel leads to smaller queuing delay than PIE at low traffic load, it surpasses the one of PIE for many flows. In contrast, CoDel-ACT leads to the least queuing delay in all cases.

Table 1

<table>
<thead>
<tr>
<th>no. TCP flows</th>
<th>CoDel mean</th>
<th>CoDel 99%</th>
<th>CoDel-ACT mean</th>
<th>CoDel-ACT 99%</th>
<th>PIE mean</th>
<th>PIE 99%</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTT=10 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4.54</td>
<td>9.22</td>
<td>4.54</td>
<td>9.22</td>
<td>16.13</td>
<td>32.14</td>
</tr>
<tr>
<td>4</td>
<td>8.79</td>
<td>17.67</td>
<td>6.60</td>
<td>11.63</td>
<td>15.79</td>
<td>24.89</td>
</tr>
<tr>
<td>16</td>
<td>14.19</td>
<td>35.75</td>
<td>7.16</td>
<td>15.25</td>
<td>15.95</td>
<td>28.51</td>
</tr>
<tr>
<td>64</td>
<td>19.94</td>
<td>44.19</td>
<td>10.36</td>
<td>26.10</td>
<td>15.97</td>
<td>33.30</td>
</tr>
<tr>
<td>RTT=100 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.16</td>
<td>6.03</td>
<td>1.15</td>
<td>6.03</td>
<td>8.28</td>
<td>28.93</td>
</tr>
<tr>
<td>4</td>
<td>3.46</td>
<td>10.85</td>
<td>3.49</td>
<td>10.85</td>
<td>16.49</td>
<td>38.57</td>
</tr>
<tr>
<td>16</td>
<td>6.47</td>
<td>19.28</td>
<td>4.74</td>
<td>12.06</td>
<td>15.75</td>
<td>31.35</td>
</tr>
<tr>
<td>64</td>
<td>15.84</td>
<td>36.17</td>
<td>8.63</td>
<td>21.71</td>
<td>15.93</td>
<td>33.76</td>
</tr>
</tbody>
</table>

PIE is configured with \( qdelay\_ref=16 \) ms which results in fact in about 16 ms average queuing delay for all investigated load levels. Therefore, one can think of PIE’s \( qdelay\_ref \) as a parameter that leads to a certain average queue length. In contrast, CoDel and CoDel-ACT are configured with \( target=5 \) ms, but the observed average queuing delays are clearly larger for most load levels. Thus, CoDel’s \( target \) provides only relative control of the average queuing delay.

D. Impact of AQM Control Parameters

CoDel, CoDel-ACT, and PIE have two control parameters each: a delay threshold and a time scale parameter. In the following, we investigate their impact on average queuing delay and utilization in the delay-limiting phase for various traffic load. The presented results are mean values over 20 runs.

1) Impact of Delay Thresholds: CoDel and CoDel-ACT use \( target \) to decide whether a packet’s delay is considered as too long so that the algorithms switch to dropping mode after some time and possibly discard such packets. PIE uses \( qdelay\_ref \) to increase or decrease its drop probability \( p \) and a fraction of \( qdelay\_ref \) helps PIE to prevent losses
in the absence of congestion. We investigate the impact of these delay thresholds during the delay-limiting phase for various traffic load while using the default time scale parameters interval=100 ms and t_update=16 ms for CoDel, CoDel-ACT, and PIE, respectively.

Figure 3a shows the average queuing delay for the three AQM algorithms depending on the delay threshold. We observe that larger delay thresholds result in longer average queuing delays for all algorithms. With CoDel, larger traffic loads lead to larger average queuing delays that are mostly larger than target. This shows again that target provides only relative control on queuing delay. We observe similar effects for CoDel-ACT, but the number of TCP flows has less influence on the average queuing delay compared to CoDel. In contrast, PIE is able to control the queuing delay independently of the traffic load. The measured delay almost equals the configured qdelay_ref. Moreover, small values of qdelay_ref cause smaller queuing delays than CoDel’s and CoDel-ACT’s target value in the presence of many TCP flows. A target value of 5 ms for CoDel leads for 16 TCP flows to about the same queuing delay as a qdelay_ref value of 16 ms for PIE. Thus, the two recommended default values lead to comparable results in this particular setting.

Figure 3b shows the utilization for the three AQM algorithms. CoDel’s utilization is below 100% for small values of target, and ranges between 95% - 100% for its default parameter of 5 ms depending on the traffic load. CoDel-ACT improves the utilization compared to CoDel while the observed queuing delay is even shorter. The default parameter target=5 ms leads to almost 100% utilization for most traffic loads. Also PIE’s utilization suffers from a small delay threshold, but the recommended qdelay_ref=16 ms achieves 100% utilization for all investigated loads. A closer look at CoDel’s utilization reveals that it is high for 1 and 64 TCP flows, but it is low for 4, 8, and 16 TCP flows. This non-monotone order is non-intuitive, therefore, we investigate and explain that issue further in Section III-E.

2) Impact of Time Scale Parameters: CoDel’s dynamics scale with the time parameter interval. First, the dropping mode is triggered after interval time if all packets experienced too long delay within that time. Second, the minimum time within a drop phase is a dynamic fraction of interval. And third, the duration for which CoDel remembers the count state parameter of the last drop phase also depends on interval. PIE’s dynamics scale with t_update because after that period PIE regularly adjusts its drop probability. We investigate the impact of the time scale parameters during the delay-limiting phase for various traffic load while using the default delay thresholds target=5 ms and qdelay_ref=16 ms, respectively. We report findings but omit figures due to space limitations.

CoDel’s and CoDel-ACT’s queuing delay increases linearly with interval and clearly depends on traffic load. For 4 or more flows, CoDel-ACT leads to about half the queuing delay compared to CoDel. PIE causes a queuing delay of 16 ms for t_update=16 ms and all investigated traffic loads. This value is lower for smaller values of t_update but only slightly larger for larger values up to t_update=150 ms. In addition, we also observe some dependence on the traffic load in these ranges.

![Fig. 3. Impact of the delay thresholds](image-url)
The utilization of CoDel heavily depends on interval. It is between 98.5% and 100% for small values of interval around 25 ms, takes a minimum of 95% – 100% at interval=100 ms, and values between 99% and 100% at interval=200 ms. Again, 1 and 64 TCP flows lead to higher utilization than 4, 8, and 16 TCP flows. CoDel-ACT shows increasing utilization for increasing values of interval. A single TCP flow always leads to lower utilization than other traffic loads. At interval=125 ms or larger, 100% utilization is reached for all traffic loads. PIE achieves 100% utilization for t_update=16 ms or larger. Smaller values of t_update cause lower utilizations which also depend on traffic load.

E. Time-Dependent Analysis of the Delay Limiting Phase

In the remainder of this work, we study the time-dependent behavior of the AQM algorithms. We first point out that their state variables, drop rates, drop patterns, and queuing delays behave differently and depend on traffic load. We show how loss patterns and queuing delays even change over time for CoDel which also affects utilization.

1) State Variables: CoDel’s and CoDel-ACT’s state variable count and as well as PIE’s drop probability p memorize the recently experienced congestion to some degree. We investigate how they evolve over time. Figure 4 shows a time series of CoDel’s and CoDel-ACT’s count variable as well as PIE’s drop probability p for different numbers of concurrent TCP flows. The data is taken from the first 100 s of a single simulation run. CoDel’s count variable increases linearly over time with a slope depending on the traffic load. An exception is the transmission of a single TCP flow for which count remains very low. The boundless growth of the count variable is due to CoDel’s algorithm and the saturated TCP sources. The count variable increases during a drop phase with every consecutive drop. When CoDel leaves its dropping mode, the rate increase of several saturated TCP sources is large so that the queue length rises quickly again. As a result, the next dropping mode is triggered before a duration of 8*interval has past since the last next_drop instant. Therefore, CoDel does not reset its count variable to 1 so that count can increase without bounds. An exception is the experiment with a single TCP flow where it obviously takes longer until CoDel switches to dropping mode again.

CoDel-ACT’s count variable behaves differently over time. Like with CoDel, it stays low for the transmission of a single TCP flow. However, it oscillates around values 50, 300, and 1300 for higher traffic load and does not continuously grow. On the one hand, CoDel-ACT returns to dropping mode more quickly than CoDel because the time during which increased queuing delays must be observed before CoDel-ACT switches to dropping mode is interval/sqrt(count) instead of interval so that CoDel-ACT does not reset count to 1, either. On the other hand, CoDel-ACT sets count to 0.9844·count at the beginning of a new dropping mode, which is lower than count-2 for count values larger than 128. That feature affects that CoDel-ACT’s count value is bounded in contrast to the one of CoDel. Average count values depend on the traffic load because the number of TCP flows governs the overall rate increase of the traffic aggregate when connections are in congestion avoidance phase.

Fig. 4. Evolution of state variables over time.

Fig. 5. Time-dependent drop rates.

PIE’s drop probability p oscillates around values that depend on the traffic load. In the presence of a single flow, the drop probability is almost zero. At the beginning of a congestion phase, the drop probability p reaches its operating point faster than count values do and reveals an initial overshoot before returning to a stationary level.

2) Drop Rates: Figure 5 visualizes time-dependent drop rates for the three AQM algorithms. We calculated them...
by applying the time-exponentially weighted moving average (TEWMA) [13] with a memory of 500 ms to individual packet losses. All drop rates oscillate around values that depend on the traffic load. For 64 flows, CoDel takes about 20 s until its stationary drop rate is reached. CoDel-ACT reaches that level already after 6 s and PIE does so within 2 s. The slow increase of drop rates for CoDel and CoDel-ACT is caused by the rather slow increase of the count value after simulation start. PIE produces the least drop rates, followed by CoDel-ACT and CoDel with the largest drop rates. The broad curve of CoDel's drop rate reveals oscillations with higher amplitude than those of CoDel-ACT and PIE. This is due to CoDel’s extreme alternating drop and non-drop phases which are illustrated next.

3) Loss Patterns and Queuing Delays: The plots in Figure 6 show the time series of queuing delays of consecutive packets as a curve and packet losses as vertical lines. Intervals of 1 s duration after 5 s, 50 s, and 500 s simulation time are provided. As CoDel-ACT’s and PIE’s results after 5 s and 500 s almost equal those after 50 s, we omit them in the figure.

a) Comparison of Loss Patterns and Queueing Delays after 50 ms: Figure 6b compares loss patterns and delays of the three algorithms after 50 s. In case of a single flow, they all drop only a single packet after observation of too long delays. The packet loss causes a decrease in traffic rate and experienced queuing delay shortly after. The time between losses is the same for CoDel and CoDel-ACT, but it is almost 3 times longer for PIE. As a consequence, PIE loses fewer packets and achieves higher resource utilization than CoDel variants (see Figure 3b). PIE drops packets on enqueue while CoDel and CoDel-ACT drop packets on dequeue. This leads to a larger interval between packet loss and delay reduction for PIE in the figure.

We now consider multiple concurrent TCP flows. CoDel exhibits drop phases alternating with non-drop phases. The drop phases are rather short and end when the queuing delay of dequeued packets falls below target. CoDel drains a long queue within 10 ms – 60 ms by discarding packets on dequeue. The high density of consecutive vertical lines reveals very high drop rates within drop phases. They are enabled by the very large count values of about 2500, 6000, and 16250 after 50 s persistent traffic load of 4, 16, and 64 flows. They lead to a minimum time between consecutive packet losses of 2.0 ms, 1.3 ms, and 0.8 ms. As the transmission of a
CoDel causes long non-drop phases by waiting at least interval time after increased queuing delays are observed in a non-drop phase before dropping mode is triggered again. Therefore, non-drop phases are at least interval time long. The non-drop phases are longer for a few TCP flows than for many TCP flows because the overall rate needs more time to recover and to anew drive the queue into congestion. CoDel-ACT also exhibits drop phases alternating with non-drop phases. However, drop phases show lower drop rates compared to CoDel, and non-drop phases are shorter. The time between drops in a drop phase is larger than with CoDel because CoDel-ACT's count variable is around 50, 300, and 1300 for 4, 16, and 64 flows, which leads to at least 14.1 ms, 5.8 ms, and 2.8 ms between packet drops. Therefore, CoDel-ACT takes longer to sufficiently reduce the queue length which extends the drop phase compared to CoDel. The shorter non-drop phases are facilitated by the fact that CoDel-ACT must wait only interval/sqrt(count) time after observing extended queuing delays before it switches to dropping mode. They are around 18.3 ms, 5.8 ms, and 2.8 ms long, i.e., some non-drop phases are very short and can be recognized only by reduced queuing delay. Thus, CoDel-ACT restarts dropping very fast and leads to shorter delays than CoDel. The time between drop phases decreases with increasing number of TCP flows for the same reason as with CoDel. PIE drops packets continuously over time and almost randomly. However, slightly decreased and increased queuing delays correlate with slightly decreased and increased drop rates.

A comparison of resulting queuing delays shows significant differences among the algorithms. With CoDel-ACT, the queuing delay oscillates with moderate amplitude around a short value, with PIE with a moderate amplitude around a larger value, and with CoDel with a large amplitude around a load-dependent value. This explains why CoDel leads to mean queuing delays that are mostly smaller than those of PIE in Table I, but may cause significantly larger quantiles.

A closer analysis of CoDel's queuing times reveals that with a single flow, the queuing delay is always positive. With 4 and 16 flows, the queuing delay of some packets is zero, and with 64 flows, queuing delay is again mostly positive. Zero queuing delays may indicate an idle link, which causes underutilization. This explains the non-monotone dependence of the utilization on the traffic load observed in Figure 3b. The reason why a single flow avoids zero queuing delays in contrast to 4 or 16 flows is that CoDel's count does not rise for a single flow so that consecutive drops are spaced sufficiently far apart. Therefore, CoDel drops only a single packet before leaving the dropping mode instead of draining the queue.

b) CoDel's Varying Drop Behavior over Time: Figures 6a–6c reveal that CoDel's drop behavior changes over time: drop phases are long after 5 s, shorter after 50 s, and very short after 500 s. This is due to increasing count values which cause larger drop rates. We consider 16 flows. Figure 6a shows that queuing delays after 5 s simulation time become short at the end of a drop phase but stay above zero. Thus, the queue never empties and the bottleneck link is well utilized. Drop rates after 5 s are low enough that the overall traffic rate is reduced carefully so that it matches approximately the link rate at the end of the drop phase. As a result, the queue is not fully drained and underutilization does not occur. Figure 6b shows that queuing delays after 50 s sometimes fall down to zero at the end of a drop phase. This means that the queue is sometimes empty and the link may be idle, leading to underutilization. Drop rates are larger than after 5 s so that the overall traffic rate is reduced too much and falls below the link rate at the end of the drop phase. The remaining queue content may not suffice to fully fill the link until the traffic rate turns up again, and cause the link to run idle for short time. Figure 6c illustrates that drop phases after 500 s stop with larger queuing delays than after 50 s and that zero queuing delays are mostly avoided. This leads to improved utilization after 500 s. After 500 s simulation time, drop rates are so large that multiple packets are dropped at once and the queue is drained faster than a single RTT so that the traffic rate is not yet reduced at the end of the drop phase. This avoids underutilization for two reasons. First, when CoDel stops dropping, the queue holds more packets with a queuing time less than target than after 50 s. Thus, there is more remaining data. Second, the queue increases again after the end of a drop phase until the effect of reduced traffic rates becomes visible at the bottleneck link. It causes the queuing delay to first slightly increase and then decrease before increasing again.

4) Evolution of Utilization and Queuing Delay: We studied the evolution of time-dependent utilization averaged over 1 s in the presence of 16 TCP flows and report results without figures. PIE and CoDel-ACT achieve about 100% utilization during the entire simulation. In contrast, CoDel's utilization starts with 100%, falls down to 95% after 30 s simulation, and slowly increases to 99% after 500 s. We define the variant CoDel-count-n which has count set to the constant value \( n \). CoDel-count-600 leads to almost 100% utilization, CoDel-6000 to 95%, and CoDel-60000 to 99%. This confirms that utilization of CoDel depends on count in a non-monotonically way and, therefore, changes over time if count increases from small to large values.

IV. Related Work

The bufferbloat phenomenon has been reported in [5]. While some authors are rather doubtful about its prevalence and impact [14], [15], bufferbloat has been demonstrated in cellular networks [16]. The authors of [17] pointed out many sources contributing to Internet latency and countermeasures.

In [7], CoDel was suggested to control queuing delay independently of buffer size, RTTs, bottleneck bandwidth, and even under varying bottleneck bandwidth. A comparison with RED was also provided. Similar results are reported in [18] from an actual Linux testbed. AQMs may be combined with scheduling mechanisms [19]. For instance, CoDel is mostly recommended to be combined with stochastic fair queueing (SFQ) to isolate flows against each other [20].
The authors of [21] compared the performance of CoDel and RED using simulations and evaluated queuing delay and throughput with different buffer sizes. They concluded that RED is also able to control the queue at a reasonable length but does not extend the transmission time of files because it leads to fewer packet drops than CoDel. Interactions between CoDel and LEDBAT have been studied in [22]. The authors of [23] have presented a software-defined implementation of RED and CoDel in an FPGA to support 10 Gb/s links.

In [9], PIE was presented and it was shown that it is able to control delay while maintaining high utilization during different congestion levels. The work compared CoDel and PIE, and showed that CoDel is not able to control queue delay under heavy load. In [24], CableLabs simulated both CoDel and PIE in DOCSIS cable modems. They demonstrated that actively managed buffers reduce the queueing delay. Another simulation of the DOCSIS cable modems showed that CoDel has problems to adjust to a sudden change in bottleneck rate under unresponsive loads [25]. Another comparison between PIE and CoDel for DOCSIS is provided in [26].

In [27], the impact of the main parameters of CoDel and PIE is analyzed. The authors compared CoDel, PIE, and Adaptive RED (ARED) in a testbed environment at an RTT of 100 ms. They used the originally published PIE algorithm which differs from the newer one used in our work.

V. CONCLUSION

In this work we have investigated the utilization and queueing delay of the three AQM algorithms CoDel, CoDel-ACT, and PIE for saturated TCP flows and a bottleneck with constant bitrate. We first illustrated the existence of a burst-allowing phase and a delay-limiting phase in the presence of saturated TCP sources. Then, we showed that the AQM algorithms lead to different queueing delay and arrival rate distributions. While PIE is able to keep the average queueing delay at its configured qdelay_ref parameter, CoDel’s average queueing delay depends both on its target parameter and the traffic load. We investigated the impact of configuration parameters on average queueing delay and utilization. CoDel-ACT leads to higher utilization and less queueing than CoDel while PIE leads to better utilization at the expense of increased queueing delay. We performed a time-dependent analysis of the algorithms’ delay-limiting phase. We investigated how state variables count and p, and drop rates evolve over time which revealed a boundless growth of CoDel’s count variable for saturated TCP sources. An analysis of drop patterns and queueing delays showed significant differences in the operation of the three algorithms. CoDel’s drop behavior changes over time due to increasing count and leads to potential underutilization. CoDel-ACT is designed to avoid such behavior and leads to shorter queueing delays. PIE does not exhibit such behavior by design.

Our work contributes to the understanding of novel AQM algorithms currently discussed for standardization in IETF, and potentially to their improvement. We revealed some disadvantageous properties of CoDel compared to CoDel-ACT or PIE. However, the experiments were limited to a single bottleneck with constant bitrate of 10 Mb/s and saturated TCP flows. To recommend one of the three AQM algorithms, further studies are needed, in particular with different traffic models and varying bandwidths.

REFERENCES


Stochastic Upper and Lower Bounds for General Markov Fluids

Florin Ciucu  
University of Warwick

Felix Poloczek  
University of Warwick / TU Berlin

Jens Schmitt  
University of Kaiserslautern

Abstract—Promising perspectives of a hypothetical ‘Tactile Internet’, or ‘Internet at the speed of light’, whereby network latencies become imperceptible to users, have (again) triggered a broad interest to understand and mitigate Internet latencies. In this paper we revisit the queueing analysis of the versatile Markov Fluid traffic model, which was mainly investigated in the 1980-90s, yet with low accuracy. We derive upper bounds on the tail distribution of the queue size, which improve state-of-the-art results by an exponential factor $O(\kappa^{-n})$ in a special case, where $0 < \kappa < 1$ and $n$ is the number of multiplexed sources; additionally, we provide the first lower bounds. The underlying results are quite general in that they can be easily adapted to derive the delay distribution for SP, FIFO, and EDF scheduling. Our overall results rely on a powerful martingale methodology which was recently shown to be highly accurate.

I. INTRODUCTION

Latency/delay is a fundamental metric of communication and human perception. Not surprisingly, several recent studies reported a strong correlation between Internet latencies and the revenue of major online service providers, e.g., Google, Bing, or Amazon [39], [26], [42]; a typical cited argument is that an additional 100ms in latency would cost Amazon 1% of sales. Moreover, innovations in electronics, systems, and protocols, together with the apparent abundance of computing resources and network bandwidth, have recently sparked the hope for a major paradigm shift called ‘Tactile Internet’ [22], or ‘Internet at the speed of light’ [41]. A potential major benefit of consistently achieving negligible network latencies (e.g., sub-100ms or sub-10ms) would be the immediate opportunity for major innovations in exciting areas such as health-care (e.g., remote surgery), transportation (e.g., fully automatic driving), or entertaining.

The increasing and broadly recognized importance of network latency has recently materialized, at a large scale, into a dedicated workshop [2] and the funding of a major European Union FP-7 project [1]; a relevant outcome of such concerted efforts was a comprehensive survey on a taxonomy of latency sources and mitigating techniques [8]. While there has been much work on system solutions such as traffic engineering and replication strategies to reduce latencies [19], there has also been shown that achieved improvements reached a point of diminishing returns [31], particularly due to the dominating queueing delays (i.e., the time spent by packets in buffers, waiting to be forwarded).

As the development of system solutions to mitigate Internet latencies can strongly benefit from the fundamental/theoretical understanding of network delays, there is a need for the parallel development of related analytical tools. A similar moment occurred in the 1980-90s, when novel analytical/queueing tools emerged to especially cope with the case of non-renewal arrivals. To some extent, this goal was driven by the increasing prevalence of audio and video content, which, when viewed as stochastic processes, are subject to some form of statistical correlations. In fact, such a new characteristic of Internet traffic determined the questioning of the common and analytically convenient assumption of Poisson arrivals (at the packet level), which was convincingly shown to be largely misleading when improperly used [36]. Besides accounting for non-renewal arrivals, an additional goal of emerging analytical theories was to permit the queueing/delay analysis over a network path; such a problem is in itself extremely challenging in the case of non-Poisson arrivals, as attested by the state-of-the-art in the classical queueing theory.

A general queueing tool which can deal with non-renewal arrivals over a network path is the network calculus [7], [10]. Initially conceived by Cruz as a deterministic queueing theory [17], its main and radically new conceptual characteristic of deterministically bounding arrivals was extended in a probabilistic framework to account for not necessarily bounded arrivals. Given its wide modelling scope and also the ability to render (deterministic) worst-case measures for network queueing metrics such as end-to-end delays, the deterministic branch of network calculus has some notable practical applications, e.g., towards the certification of the Airbus A380 AFDX backbone [5]. The broad applicability of (deterministic) network calculus comes however at the price of providing bounds (i.e., not exact results), whose practical tightness in terms of efficient numerical algorithms remains an open issue in the network case [6] despite significant recent progress [3]. In turn, the stochastic branch of network calculus suffers from similar low accuracy issues, including the single-node case whereby the stochastic bounds can be loose by orders of magnitude in the case of non-renewal arrivals [14]; this drawback is exacerbated in the network case, especially under heavy-tailed arrivals [32].

In a stochastic setting, an alternative queueing tool is the theory of effective bandwidth [27] which can address a broad class of (non-renewal) arrival processes in a unified and conceivably quite elegant manner. Unlike stochastic network calculus, which yields results in terms of non-asymptotic bounds, effective bandwidth provides exact results but only...
in the so-called large-buffer or many-sources asymptotic regimes [34]. Unfortunately, when fitted to practical (finite) regimes, such results are however largely inaccurate and thus practically questionable [12]; a convincing explanation is provided in [40], i.e., for more variable sources than Poisson such as most Markov-modulated processes, the underlying additivity property reveals both the elegant nature of effective bandwidth but also the conceivably conservative nature of the exact results in finite regimes.

Motivated by the need for a unified queueing tool to address non-renewal arrivals, and more importantly in terms of practically accurate results, this paper revisits the queueing analysis of the versatile class of Markov Fluid traffic models. Informally, a source is a Markov Fluid if the associated arrival process is a function of a Markov process on a finite state space [28]. The seminal model is the so-called Anick/Mitra/Sondhi On-Off model, in which a source produces data (as continuous ‘fluid’) either at some positive constant or zero rates, depending on the state of a Markov process [4]; for related generalizations, which are able to model larger classes of traffic patterns (e.g., video), see [43], [28]. It is worth pointing out that fluid models are the continuous approximation of their discrete counterpart, and are essentially motivated by computational reasons such as dealing with (discrete) state space explosion or various underlying granularity and sizes (e.g., modelling a large number of small packets in a short time interval) [24].

In this paper we generalize existing stochastic upper bounds, available for the specific On-Off process [14], to the case of general Markov fluids. The obtained stochastic (upper) bound on the steady-state queue size distribution is shown to be tighter than the existing state-of-the-art result from [35] by an exponential factor \(O(e^{-\kappa})\), in the case of a superposition of \(n\) On-Off sources, where \(\kappa\) is an explicit constant with \(0 < \kappa < 1\). Additionally, we provide for the first time the matching lower bounds, which have the same analytical structure as the upper bounds. Furthermore, by leveraging the framework of the stochastic network calculus, we provide per-flow delay bounds in a scheduling scenario with either FIFO (first-in-first-out), SP (static priority), or EDF (earliest deadline first) scheduling, when the flows are Markov fluids as well.

Unlike most of alternative results from the effective bandwidth (1980-90s), e.g., [28], [16] and stochastic network calculus literature (1990s-), e.g., [10], [13], [23], our overall results employ a powerful martingale approach which was proposed by Kingman to derive bounds in GI/G/1 (renewal) queues [29]. This methodology was adopted in [20], [35], [10] to the case of non-renewal processes. The numerical tightness of the obtained upper bounds on the queue-size distribution, and their noticeable superiority to alternative effective bandwidth/stochastic network results, was recently exposed in [14], [37], who further provided per-flow queueing results under several scheduling policies with similar high accuracy. An additional practical benefit of the martingale approach is that it provides the most critical component of latency, i.e., its tail (e.g., the 95\textsuperscript{th}-percentile) and not its average; in fact, the importance of the tail to the development of the much desired and anticipated “latency tail-tolerant” systems has been convincingly exposed in a recent Google study [19].

The remainder of the paper is organized as follows: In §II we present a general tool to analyze Markov fluids in an infinite buffer queueing model. This tool is subsequently applied to derive upper and lower bounds on the queue size distribution (§III), and to the (per-flow) delay bounds in a scheduled system (§IV). In §V we provide a collateral but important result, i.e., the first analytical construction for the space-parameter in the classical effective bandwidth approximation. Finally in §VI we summarize the paper.

II. A General Queueing Result

In this section we first derive a general queueing result which can be instantiated to several scenarios, e.g., the queue size distribution for a single process or the delay distribution in the case of scheduling.

The general queueing model is depicted in Figure 1. Given a continuous time model, \(A_1(t)\) and \(A_2(t)\) are two (cumulative) Markov fluid processes served at some constant rate \(C > 0\) with corresponding departure processes \(D_1(t)\) and \(D_2(t)\), respectively. Each process \(A_k(t)\), \(k = 1, 2\), is modulated by a reversible Markov process \(Z_k(t)\) with \(n_k + 1\) states, generator \(Q_k = (q_{k,i,j})_{i,j=0,...,n_k}\), equilibrium distribution \(\pi_k = (\pi_{k,0},...,\pi_{k,n_k})\), arrival rates \(r_k = (r_{k,0},...,r_{k,n_k})\), and increment process \(a_k(s) = r_k Z_k(s)\) \(\forall s \geq 0\). We remark that if \(Z_k(t)\) was not reversible, then one could consider as input the corresponding reversed process. The reversibility property enables expressing Reich’s equation for the steady-state queue size as

\[
Q = \sup_{t \geq 0} \{A_1(t) + A_2(t) - Ct\}
\]

For each arrival process \(A_k(t)\) we consider the generalized eigenvalue problem

\[
Q_k h_k = -\gamma_k u_k h_k, \quad k = 1, 2,
\]

where \(Q_k\) are the underlying generators, \(\gamma_k\) are the eigenvalues, \(h_k\) are the right eigenvectors, \(u_k\) are diagonal matrices with \((u_k,0, u_k,1,\ldots, u_k,n_k)\) on the diagonal, and where

\[
u_k,j = y_{k,j} = C_k
\]

are the instantaneous queueing drifts for \(j = 0, 1,\ldots, n_k\). Here, \(C_1\) and \(C_2\) are positive values such that \(C_1 + C_2 = C\), and can be regarded as the per-class/flow allocated capacity. Assuming the per-class stability conditions

\[
\sum_{j=0}^{n_k} \pi_{k,j} u_{k,j} < 0, \quad k = 1, 2
\]
Lemma 5.1 from [35] guarantees the existence of real generalized eigenvalues \(-\gamma_k\) (as the ones with the biggest negative real parts) and also of the generalized eigenvectors

\[ h_k = (h_{k,0}, h_{k,1}, \ldots, h_{k,n})^T \]

with positive coordinates. Thus, \(\gamma_k > 0\) for \(k = 1, 2\).

We now present a general result which will be used throughout the paper.

**Lemma 1.** Consider the single-node queueing scenario from Figure 1 and the solutions for the generalized eigenvalue problems from Eq. (1). Then for all \(0 \leq u \leq t \) and \(\sigma \geq 0\)

\[
\mathbb{P}\left( \sup_{0 \leq s < t-u} \left\{ A_1(s, t-u) + A_2(s, t) - C(t-s) \right\} > \sigma \right) \\
\leq \inf_{0 \leq \gamma \leq \min \{\gamma_1, \gamma_2\}} \inf_{C_1+C_2=C} \lim_{\kappa \to \infty} \kappa e^{-\gamma(C_1+\sigma)} ,
\]

where

\[
\kappa = \sum_{i,j} \pi_{i,1} \pi_{2,j} \frac{h_{i,1}^2 h_{j,2}^2}{\min_{u_1+u_2 \geq 0} h_{1,i}^2 h_{2,j}^2},
\]

whereas the condition \(C_1 + C_2 = C\) is subject to the stability conditions from Eq. (2).

Let us make several observations about the two inﬁmum operators. The parameter \(\gamma\) in the outer inﬁmum reconciles the different burstitness of the two not necessarily homogeneous ﬂows \(A_1(t)\) and \(A_2(t)\), loosely expressed through the exponential decay factor. The extreme optimal value \(\gamma^* = \min \{\gamma_1, \gamma_2\}\) is attained when \(\sigma \to \infty\); in turn, a numerical optimization after \(\gamma\) is necessary in ﬁnite regimes of \(\sigma\). In turn, due to the implicit expression of \(\kappa\) in terms of the (generalized) eigenvectors from Eq. (1), which depend on \(C_1\) and \(C_2\), the values for the inner inﬁmum are subject to further numerical optimizations.

Lemma 1 generalizes a result from [14] (see Theorem 1 therein) to the case of general and not necessarily homogeneous Markov fluid processes. The theorem also generalizes a result from [35] (see Proposition 5.1 therein), restricted to \(A_2(t) = 0\), and also the seminal result from Kingman [29] to the non-renewal case. We point out that the key beneﬁt of our generalization result is that it can lend itself to per-ﬂow delay bounds in a scheduling scenario (see §IV).

**Proof.** Fix \(u \geq 0\) and \(\sigma \geq 0\). Since the two arrival processes are reversible, we can rewrite the probability from Eq. (3), by shifting the time origin, as

\[
\mathbb{P}\left( \sup_{t \geq u} \left\{ A_1(u, t) + A_2(t) - C(t) \right\} \geq \sigma \right) \\
= \mathbb{P}\left( \sup_{t \geq u} \left\{ A_1(u, t) + A_2(u, t) - C(t-u) \right\} \\
\quad + A_2(u) - C_2 u > C_1 u + \sigma \right).
\]

Let the following stopping time

\[
T := \inf \left\{ t > u : A_1(u, t) + A_2(u, t) - C(t-u) \\
\quad + A_2(u) - C_2 u > C_1 u + \sigma \right\}.
\]

In the rest of the proof we shall bound \(\mathbb{P}(T < \infty)\), which is exactly the target probability from Eq. (4).

Let \(\mathbb{P}_{i,j}\) denote the underlying probability measure conditioned on \(Z_1(u) = i\) and \(Z_2(0) = j\), for \(0 \leq i \leq n_1\) and \(0 \leq j \leq n_2\). Next we deﬁne the following two processes

\[
\tilde{M}_{1,t} := \frac{h_{1,1} Z_1(t)}{h_{1,1}} e^{-\int_0^t (Q_{1,1}(s) Z_1(s)) ds} \forall t \geq u \text{ and} \\
\tilde{M}_{2,t} := \frac{h_{2,1} Z_2(t)}{h_{2,1}} e^{-\int_0^t (Q_{2,1}(s) Z_2(s)) ds} \forall t \geq 0.
\]

\(M_1(t)\) and \(M_2(t)\) are martingales with respect to (wrt) \(\mathbb{P}_{i,j}\) and the natural ﬁltration (see [21], p. 175). Considering the solution of the generalized eigenvalue problem from Eq. (1), we can rewrite

\[
\tilde{M}_{1,t} = \frac{h_{1,1} Z_1(t)}{h_{1,1}} e^{-\int_0^t (Q_{1,1}(s) Z_1(s)) ds} \forall t \geq u \text{ and} \\
\tilde{M}_{2,t} = \frac{h_{2,1} Z_2(t)}{h_{2,1}} e^{-\int_0^t (Q_{2,1}(s) Z_2(s)) ds} \forall t \geq 0.
\]

For \(0 \leq \gamma \leq \min \{\gamma_1, \gamma_2\}\) we consider the transformations

\[
M_{k,t} = \tilde{M}_{k,t}^\gamma, \quad k = 1, 2.
\]

Denoting by \(\mathcal{F}_{k,s}\) the natural ﬁltrations of \(M_{k,t}\) we can write for \(0 \leq s \leq t\)

\[
E \left[ M_{k,t} \mid \mathcal{F}_{k,s} \right] = \frac{E \left[ M_{k,t}^\gamma \mid \mathcal{F}_{k,s} \right]}{E \left[ M_{k,s}^\gamma \right]} \leq E \left[ M_{k,t} \mid \mathcal{F}_{k,s} \right] \frac{1}{\gamma} \\
\leq \frac{M_{k,s}}{\gamma^2} = M_{k,s}.
\]

where the first line is due to Jensen’s inequality (applied to the concave function \(x \mapsto x^{\frac{1}{\gamma}}\) for \(x \geq 0\) and the second inequality is due to the martingale property of \(\tilde{M}_{k,t}\). Therefore, the new processes \(M_{k,t}\) are also martingales; we point out that their construction is motivated by the need of having the same decay rate, i.e., \(\gamma\), in the corresponding exponentials.

Next we invoke a result from [11] (stating that the product of two independent martingales is also a martingale) and the Optional Switching Theorem ([25], p. 488), and obtain that the process

\[
M_t := \begin{cases} 
M_{1,t} & , t \leq u \\
M_{2,t} & , t > u
\end{cases}
\]

is also a martingale (note that \(M_{1,u} = 1\) by deﬁnition). It can be explicitly written as

\[
M_t = \begin{cases} 
\frac{h_{2,1} Z_1(t)}{h_{2,1}} e^{(A_2(t) - C_1 t)} , t \leq u \\
\frac{h_{1,1} Z_1(t)}{h_{1,1}} e^{(A_1(u,t) + A_2(t) - C t)} , t > u
\end{cases}
\]
Referring now to the stopping time $T$ from Eq. (5), which may be unbounded, we construct the bounded stopping times $T \wedge v$ for all $v \in \mathbb{N}$. For these times, the Optional Sampling Theorem (see, e.g., [25], p. 489) yields

$$E_{i,j}[M_0] = E_{i,j}[M_{T \wedge v}] ,$$

for all $v \in \mathbb{N}$, where the expectations are taken wrt the underlying probability measures $P_{i,j}$. Moreover, from the definition of $T$ as an infimum over a set, it holds for $v \geq 0$ that

$$\left( u_1, \mathcal{E}(T) + u_2, \mathcal{E}(T) \right) I_{(T \leq v)} \geq 0 , \quad (6)$$

where $I_{\{\cdot\}}$ denotes the indicator function. Using now that $E_{i,j}[M_0] = 1$ we obtain for $v > u$

$$1 \geq E_{i,j}[M_{T \wedge v}(T \leq v)] \geq \min \left\{ \frac{h_{1,i}}{h_{1,j}}, \frac{h_{2,i}}{h_{2,j}} \right\} e^{(C_1 u + \sigma)\pi_{i,j}} (T \leq v) ,$$

where the ‘min’ operator is taken over the set \{(h_{1,i}, h_{2,i}) : u_{1,i} + u_{2,i} \geq 0\} according to Eq. (6).

Finally, by deconditioning on $i$ and $j$ (recall that $Z_1(u)$ and $Z_2(0)$ are in steady-state by construction) we obtain

$$P(T \leq v) \leq \kappa e^{-\gamma(C_1 u + \sigma)} .$$

Letting $v \rightarrow \infty$ completes the proof.

III. QUEUE SIZE DISTRIBUTION: UPPER AND LOWER BOUNDS

Here we apply Lemma 1 to derive upper bounds on the steady-state queue size distribution, and then provide the corresponding lower bounds.

A. Upper Bounds

Consider a server with constant rate $C > 0$ serving a Markov fluid process $A(t)$ which is modulated by a reversible Markov process $Z(t)$ with $n + 1$ states, generator $Q = (q_{i,j})_{i,j=0,...,n}$: equilibrium distribution $\pi = (\pi_0, \ldots, \pi_n)$, arrival rates $r = (r_0, \ldots, r_n)$, and increment process $a(s) = r_{Z(s)}$ for $s \geq 0$. The steady-state queue size is

$$Q = \sup_{t \geq 0} \{ A_1(t) + A_2(t) - Ct \} ,$$

and the steady-state virtual delay is defined via

$$\{W(t) \geq d\} = \left\{ A_1(t - d) + A_2(t - d) \geq D_1(t) \right\} \subset \left\{ \sup_{t \geq 0} \{ A_1(t) + A_2(t) - Ct \} \geq Cd \right\} .$$

Let the generalized eigenvalue problem

$$Qh = -\gamma uh , \quad (7)$$

where $u$ is a diagonal matrix with $(u_0, u_1, \ldots, u_n)$ on the diagonal, and where

$$u_j = r_j - C$$

are the instantaneous queueing drifts for $j = 0, 1, \ldots, n$. Assume further the stability condition

$$\sum_{j=0}^{n} \pi_j u_j < 0 .$$

As mentioned earlier, there exists a real generalized eigenvalue $-\gamma < 0$, and the corresponding eigenvector $h = (h_0, h_1, \ldots, h_n)^T$ with positive coordinates.

An immediate consequence of Lemma 1, instantiated with $A_2(t) := 0$ and $u := 0$, is the following:

**Corollary 2. (Queue Size Distribution: Upper Bound)**

Consider the previous queueing system with a single Markov fluid $A(t)$ served at rate $C$. Then the stationary queue size distribution $Q$ and waiting time distribution $W(t)$ satisfy for all $\sigma, t \geq 0$

$$P(Q \geq \sigma) \leq \sum_{h_0 \geq \sigma} \pi_i h_i e^{-\gamma \sigma} , \quad (8)$$

$$P(W(t) \geq d) \leq \sum_{h_0 \geq \sigma} \pi_i h_i e^{-\gamma C d} , \quad (9)$$

where $\gamma$ and $h = (h_0, h_1, \ldots, h_n)^T$ are the solutions of the generalized eigenvalue problem from Eq. (7).

Let us next compare this bound with the state-of-the-art bound from [35], i.e.,

$$P(Q \geq \sigma) \leq \sum_{h_0 \geq \sigma} \pi_i h_i e^{-\gamma \sigma} . \quad (10)$$

$$P(W(t) \geq d) \leq \sum_{h_0 \geq \sigma} \pi_i h_i e^{-\gamma C d} . \quad (11)$$

Clearly, our bound is tighter due to the additional constraint on the ‘min’ operator from Eq. (8).

To give an explicit order of the improvement, consider the classical scenario when the process $A(t)$ is a superposition of $n$ Markov-modulated On-Off processes (see Figure 2). Each sub-process is modulated by a Markov process with two states, denoted by ‘On’ and ‘Off’, and which communicate at rates $\lambda$ and $\mu$. While in the ‘On’ state, each sub-process generates ‘fluid’ at some constant rate $P$. To avoid trivial situations we assume that $nP > C$ and that the utilization factor $\rho = \frac{nP}{C}$ satisfies the stability condition $\rho < 1$.  

\begin{center}
\begin{tikzpicture}
    \node [state] (1) {1};
    \node [state] (2) [right of=1] {2};
    \node [state] (3) [right of=2] {\ldots};
    \node [state] (4) [right of=3] {n};
    \node [state] (5) [above of=1] {0};
    \node [state] (6) [below of=1] {2};
    \node [state] (7) [below of=2] {n};
    \path[->] (1) edge node {$\lambda$} (2);
    \path[->] (2) edge node {$2\lambda$} (3);
    \path[->] (3) edge node {$2\lambda$} (n);
    \path[->] (n) edge node {$\mu$} (2);
    \path[->] (n) edge node {$n\lambda$} (7);
    \path[->] (5) edge node {$\mu$} (1);
    \path[->] (1) edge node {$\lambda$} (6);
    \path[->] (6) edge node {$\mu$} (7);
\end{tikzpicture}
\end{center}

Fig. 2. A superposition of $n$ On-Off processes
A key advantage of the chosen multiplexed On-Off model is that it lends itself to an *explicit* solution for the generalized eigenvalue problem from Eq. (7), i.e.,
\[ \gamma = \frac{n(N+1)\eta(1-\rho)}{nP-C}, \]
where \( \theta = \log \frac{\lambda e^{-\mu}}{\mu - \eta} \) (see also [35]). Thus, an explicit bound in Corollary 2 is
\[ \mathbb{P}(Q \geq \sigma) \leq \kappa^\eta e^{-\gamma \sigma}, \]
where \( \kappa = \rho \left( \frac{\lambda e^{-\mu}}{\mu - \eta} \right)^{\frac{1}{\gamma}} \) and \( p = \frac{\mu}{\theta} \) is the steady-state probability for an On-Off process to be in the ‘On’ state. As it was shown in [14] that \( 0 < \kappa < 1 \), whereas the prefactor from Eqs. (10) and (11) is clearly greater than 1, it follows that the improvement of our bound from Corollary 2 relative to the one from Eq. (10) and (11) is of the order \( O(\kappa^n) \) (in the specific case of a superposition of On-Off processes).

The technical explanation for this drastic improvement can be found in the proof of Lemma 1. More specifically, the key observation resides in Eq. (6): in the current case of a single fluid \( A(t) \), the equation ‘says’ that there must be a non-negative drift \( a(Z(T)) - C \) when the stopping time \( T \) is attained; although seemingly elementary, this observation does have the reported drastic impact on the bounds. We point out that this observation is reminiscent of the works in [9], [35].

**B. Lower Bounds**

We now provide the matching lower bounds for the upper bound from Corollary 2. We consider the same scenario from \( \gamma \) III-A with a single Markov fluid \( A(t) \) served at rate \( C > 0 \).

**Corollary 3.** *(Queue Size Distribution: Lower Bound)*
The stationary queue size distribution \( Q \) and waiting time distribution \( W(t) \) satisfy for all \( \sigma, t \geq 0 \)
\[ \mathbb{P}(Q \geq \sigma) \geq \frac{\sum \pi_i h_i}{\max \{u_i | u_i \geq 0\}} e^{-\gamma \sigma}, \]
\[ \mathbb{P}(W(t) \geq \sigma) \geq \frac{\sum \pi_i h_i}{\max \{u_i | u_i \geq 0\}} e^{-\gamma C \sigma}, \]
where \( \gamma \) and \( h = (h_0, h_1, \ldots, h_N)^T \) are the solutions of the generalized eigenvalue problem from Eq. (7).

Remarkably, the only difference to the corresponding upper bound is that the factor \( \min \{u_i | u_i \geq 0\} \), from Eq. (8) is now replaced by \( \max \{u_i | u_i \geq 0\} \). In particular, in a scenario where the set \( \{u_i | u_i \geq 0\} \) only consists of a single element, upper and lower bounds coincide and hence, Eqs. (8) and (14) (as well as Eqs. (9) and (15)) provide exact results. The proof of Corollary 3 is similar to that of Lemma 1, but it uses an additional stopping time following an idea from [38] to derive bounds in the GI/G/1 (renewal) queue:

**Proof.** Fix \( \sigma \geq 0 \). The queue size distribution can be written as
\[ \mathbb{P}(Q \geq \sigma) = \mathbb{P} \left( \sup_{t \geq 0} (A(t) - Ct) \geq \sigma \right). \]

As in the proof of Lemma 1, let \( \mathbb{P}_i \) denote the underlying probability measure conditioned on \( Z(0) = i \) for \( 0 \leq i \leq n \) (recall that \( Z(t) \) is the underlying modulating Markov process of \( A(t) \)). Again, the process
\[ M_t := \frac{h_Z(t)}{h_i} e^{\int_0^t \lambda Z(s) ds} \]
is a martingale wrt \( \mathbb{P}_i \); and the natural filtration. Considering the solution of the generalized eigenvalue problem from Eq. (7), we have \( \forall t \geq 0 \)
\[ M_t = \frac{h_Z(t)}{h_i} e^{\int_0^t \lambda Z(s) ds} = \frac{h_Z(t)}{h_i} e^{\gamma(A(t)-Ct)}. \]

Let us now define the stopping time
\[ T := \inf \{ t \geq 0 \mid A(t) - Ct \geq \sigma \} \]
which is the equivalent of the one from Eq. (5), in the current context with a single fluid. Define further a second stopping time for some \( y > 0 \):
\[ T_y := \min \{ T, \inf \{ t \geq 0 \mid A(t) - Ct \leq -y \} \}, \]
(17)
as the first hitting time of the boundary of the interval \([-y, \sigma]\). Since \( T_y \) is a finite stopping time, relative to the natural filtration \( \mathcal{F}_t \), it follows from the Optional Stopping Theorem that the process \( \{M_{T_y \wedge \tau} \} \) is a martingale, which is bounded and hence uniformly integrable. Since \( M_{T_y \wedge \tau} \rightarrow M_{T_y} \) a.s. and in \( L^1 \), we can further write
\[ E_i [M_0] = E_i [M_{T_y \wedge \tau}] = E_i [M_{T_y}] \]
\[ = E_i [M_{T_y} \mid A(T_y) \geq CT_y + \sigma] \mathbb{P}_i (A(T_y) \geq CT_y + \sigma) + E_i [M_{T_y} \mid A(T_y) \leq CT_y - y] \mathbb{P}_i (A(T_y) \leq CT_y - y), \]
(18)
where the expectations are taken wrt \( \mathbb{P}_i \).

To deal with the first term we first observe that
\[ \{A(T_y) \geq CT_y + \sigma\} \Rightarrow \{T_y = T\}. \]

Because the process \( A(t) \) is continuous it follows that the ‘hitting’ condition from Eq. (17) is attained with equality, i.e.,
\[ A(T_y) = CT_y + \sigma. \]

Moreover, since \( \gamma > 0 \), we can bound the conditional expectations from Eq. (18) as follows
\[ 1 \leq E_i [M_0] \leq \sup_{u_i \geq 0} \|h\| \gamma \mathbb{P}_i (A(T_y) \geq CT_y + \sigma) + \sup_{u_i \geq 0} \|h\| \gamma \mathbb{P}_i (T < \infty) \]

Letting \( y \rightarrow \infty \) the second term vanishes and thus
\[ 1 \leq \sup_{u_i \geq 0} \|h\| \gamma \mathbb{P}_i (T < \infty). \]
By recalling that \( \{ T < \infty \} \) is the same (a.s.) with the event from Eq. (16), and finally deconditioning on \( i \) (i.e., 
\[ P(T < \infty) = \sum_{i} \pi_i P(T < \infty), \]
the proof for the queue size \( Q \) is complete.

The proof for the waiting time \( W(t) \) is entirely analogous. \( \square \)

For an explicit lower bound, consider again the case when \( A(t) \) is the superposed process from Figure 2. Denoting \( p = \frac{\lambda}{\mu} \), the equilibrium distribution \( \pi \) of \( Z(t) \) has the components \( \pi_i = \binom{n}{i} p^i (1 - p)^{n-i} \) for \( i = 0, 1, \ldots, n \). Recalling the explicit solution of the underlying generalization eigenvalue problem from Eq. (12), the prefactor of the exponential bound from Eq. (14) becomes after elementary calculations

\[
\sum_{i=0}^{n} \binom{n}{i} p^i (1 - p)^{n-i} e^{\theta(n-i)} = e^{\theta n} \left( pe^{-\theta} + 1 - p \right)^n = \rho^n,
\]
where \( \rho = \frac{n \frac{\lambda}{\mu}}{C} \) is the utilization factor. The explicit lower bound is thus

\[
P(Q \geq \sigma) \geq \rho^n e^{-\gamma \sigma},
\]
which has the same analytical structure as the corresponding upper bound from Eq. (13), except for the constant in the prefactor (i.e., \( \rho \) vs. \( \kappa \), both belonging to \( (0, 1) \)).

C. Evaluation

In order to numerically validate the bounds given in Corollaries 2 and 3, we consider the simulation scenario as in Figure 3: A 3-state Markov process alternates with rates \( \lambda_{i,j} \) between one inactive and two active states. In the active states, fluid is generated with rates \( P \) and \( 2P \), respectively; in the inactive state no fluid is generated.

Figure 4 shows the CCDF of the aggregate delay distribution for a scenario with \( n = 2, 5, 10 \) such Markov processes being multiplexed. The parameters are \( \lambda_{i,j} = 1 \) (for all \( i, j \) ), \( P = 1 \), and \( C > 0 \) is scaled such that for the link utilization holds \( \rho = 0.75 \) (Figure 4a) and \( \rho = 0.9 \) (Figure 4b), respectively.

By the symmetry of the transition rates, the Markov process is reversible. The analytical results are shown as coloured areas between the upper and lower bounds from Eqs. (9) and (15), the simulations are displayed as single points.

To gain insight into the impact of multiplexing, we refer to Figure 5: For different violation probabilities \( \varepsilon = 10^{-2}, 10^{-4}, 10^{-6} \), the delay is given in dependency of the number of multiplexed sources \( n \). Again, the link utilization is \( \rho = 0.75 \) (Figure 5a) and \( \rho = 0.9 \) (Figure 5b) and the bounds from Eqs. (9) and (15) are shown as coloured areas; in addition, the state-of-the-art (upper) bounds from Eqs. (10) and (11) are displayed as lines. One immediately sees that the beneficial effect of multiplexing (multiplexing gain) manifests itself in an exponential decay of the corresponding delay. Although this effect is also captured by the state-of-the-art bounds, they are too large by a factor of at least three times the width of the newly obtained interval, revealing their large inaccuracy.

IV. PER-FLOW DELAY DISTRIBUTION UNDER SCHEDULING

Clearly, the formulation from Lemma 1 is cumbersome, especially due to the apparently obscure parameter \( u \). It has however the key advantage that it can be broadly applied to obtain per-flow/class bounds in the general queueing scenario from Figure 1, besides the previous results on the queue size distribution. We will derive in particular bounds on the virtual delay \( W_i(t) \) of the tagged flow \( A_i(t) \), under three scheduling policies: FIFO, SP, and EDF. While the derivations resemble those from [14], we present the generalized results (i.e., holding for general Markov fluids) for the sake of completeness.

First, we describe a common step to the derivations of all the three cases. In a scheduled scenario, the departure process \( D_i(t) \) of the tagged flow has the representation

\[
D_i(t) \geq A_1 + S_1(t) := \inf_{0 \leq s \leq t} \{ A_1(s) + S_1(s, t) \},
\]
where \( S_1(t) \) is the service process, encoding information about the cross flow \( A_2(t) \) and the specific scheduling policy, and where ‘*’ is a \( (\min, +) \) convolution operator [10].

Using the equivalence of events

\[
\{ W_i(t) \geq d \} = \{ A_1(t - d) \geq D_i(t) \},
\]
we can bound the distribution of \( W_i(t) \) as

\[
P(W_i(t) \geq d) \leq P(A_1(t - d) \geq A_1 + S_1(t))\] .

The three delay bounds (i.e., for FIFO, SP, and EDF) can be then obtained by plugging in the service process for the corresponding scheduling policy, and by choosing a suitable value for the parameter \( u \) in Lemma 1.

A. FIFO

Under this policy, the fluid from the sources \( A_1(t) \) and \( A_2(t) \) is processed in the order of the respective arrival times. The service process of the tagged flow is [18]

\[
S_1(s, t) = \lceil C(t - s) - A_2(s, t - x) \rceil I_{t-x > s} \] ,

where \( I_{t-x > s} \) is the indicator function.

for some fixed $x \geq 0$, independent of $s$ and $t$. Eq. (21) continues as follows
\[
\mathbb{P}(W_1(t) \geq d) \leq \mathbb{P}\left( \sup_{0 \leq s < t-d} \{ A_1(s, t - d) - C(t - s) \} \geq 0 \right).
\]
Note that we could restrict the range of $s$ from $[0, t]$ to $(0, t-d)$, using the positivity of the $'i_d'$ operator and the monotonicity of $A_1(s, t)$. By making the choice $x := d$, it follows that
\[
\mathbb{P}(W_1(t) \geq d) \leq \mathbb{P}\left( \sup_{0 \leq s < t-d} \{ A_1(s, t - d) + A_2(s, t - d) - C(t - s) \} \geq 0 \right).
\]
Further applying Lemma 1 with $u := 0$ and $\sigma := Cd$ yields:

**Corollary 4. (Per-Flow Delay Distribution: FIFO)**

Under FIFO scheduling, the delay of flow $A_1(t)$ satisfies for all $d \geq 0$
\[
\mathbb{P}(W_1(t) \geq d) \leq \kappa e^{-\gamma Cd},
\]
where $\kappa$ and $\gamma$ are given in Lemma 1.

**B. SP**

Under this policy, fluid from $A_1(t)$ is only served when there is no unprocessed fluid from $A_2(t)$. A service process for the low-priority tagged flow is given by [10]
\[
S_1(s, t) = C(t - s) - A_2(s, t),
\]
such that Eq. (21) continues as follows:
\[
\mathbb{P}(W_1(t) \geq d) \leq \mathbb{P}\left( \sup_{0 \leq s < t-d} \{ A_1(s, t - d) - A_2(s, t - C(t - s)) \} \geq 0 \right).
\]
Recalling the arbitrary split $C_1 + C_2 = C$, Lemma 1 yields (with $u := d$ and $\sigma := 0$):

**Corollary 5. (Per-Flow Delay Distribution: SP)**

Under SP scheduling, the delay of flow $A_1(t)$ satisfies for all $d \geq 0$
\[
\mathbb{P}(W_1(t) \geq d) \leq \kappa e^{-\gamma C_1 d},
\]
where $\kappa$ and $\gamma$ are given in Lemma 1.

**C. EDF**

An EDF server associates the relative deadlines $d_1^*$ and $d_2^*$ to the fluids of $A_1(t)$ and $A_2(t)$, respectively. All fluids are served in the order of their remaining deadlines, even when they are negative. A service process for the tagged flow $A_1(t)$ is given by [33]
\[
S_1(s, t) = [C(t - s) - A_2(s, t - x + \min\{x, y\})] \cdot I_{t > s},
\]
for some $x > 0$ and where $y := d_1^* - d_2^*$.

For the sake of brevity, we only consider the case $y \geq 0$. Setting $x := d$ as for the FIFO case, Eq. (21) continues to
\[
\mathbb{P}(W_1(t) \geq d) \leq \mathbb{P}\left( \sup_{0 \leq s < t-d} \{ A_1(s, t - d) + A_2(s, t - d) - C(t - s) \} \geq 0 \right).
\]
By changing the variable $t \leftarrow t + d - \min\{d, y\}$ we get
\[
\mathbb{P}(W_1(t) \geq d) \leq \mathbb{P}\left( \sup_{0 \leq s < t-d} \{ A_1(s, t - \min\{d, y\}) + A_2(s, t) - C(t - s) \} \geq 0 \right).
\]
We can now apply Lemma 1 with $u := \min\{d, y\}$ (note that both $d$ and $y$ are positive) and $\sigma := C(d - \min\{d, y\})$, and finally obtain:

**Corollary 6. (Per-Flow Delay Distribution: EDF)**

Under EDF scheduling with $y := d_1^* - d_2^* \geq 0$, the delay of flow $A_1(t)$ satisfies for all $d \geq 0$
\[
\mathbb{P}(W_1(t) \geq d) \leq \kappa e^{-\gamma C_2 \min\{d_1^* - d_2^*, d\}} e^{-\gamma Cd},
\]
where $\kappa$ and $\gamma$ are given in Lemma 1.
V. An Analytical Construction for the Space-Parameter in the Effective Bandwidth Approximation

As a collateral result, we now give the first analytical construction for the space-parameter in the effective bandwidth approximation, which was proposed in the 1980 – 90s as a fundamental technique for traffic engineering. For an arrival process \( A(t) \), the effective bandwidth of \( A(t) \) is defined for some \( \theta > 0 \) as [27]

\[
\alpha (\theta, t) := \frac{1}{\theta} \log E \left[ e^{\theta A(t)} \right].
\]

Let

\[
\alpha_\theta := \lim_{t \to \infty} \alpha (\theta, t)
\]

and \( \theta^* \) be the solution of

\[
\alpha_\theta = C,
\]

where \( C \) is the rate at which \( A(t) \) is served at a stable queue. The effective bandwidth approximation states that the steady-state queue size distribution satisfies

\[
P(Q > \sigma) \sim \kappa e^{-\theta^* \sigma}, \tag{26}
\]

where \( \kappa \) is the asymptotic constant, \( \theta^* \) is the asymptotic decay rate, and \( f(x) \sim g(x) \) means that \( f(x)/g(x) \to 1 \) as \( x \to \infty \) (see [12], [15]).

The space-parameter of the effective bandwidth approximation is generally the parameter \( \theta \) in the expression of \( \alpha(\theta, t) \), and more particularly \( \theta^* \), which was shown to play a fundamental role in traffic engineering; unfortunately, the construction of \( \theta^* \) is based on either numerical search or simulations [15]. To the best of our knowledge, the next result provides the first analytical construction of \( \theta^* \) (in the case of Markov fluids).

**Lemma 7. (Space-Parameter Construction)** For the previous queueing scenario it holds

\[
\theta^* = \gamma, \tag{27}
\]

where \( \gamma \) was defined in Corollary 2 (as the solution of the generalized eigenvalue problem from Eq. (7)).

**Proof.** From the construction of \( \gamma \) we have that

\[
(Q + \gamma u) h = 0. \tag{28}
\]

Let the diagonal matrix \( V \) with \( (\gamma_1, \gamma_2, \ldots, \gamma_n) \) on the diagonal, and construct the matrix

\[
Q_\gamma := Q + V.
\]

Then it holds that

\[
Q_\gamma x = \alpha_\gamma \gamma_1 x \tag{29}
\]

where \( \alpha_\gamma \gamma \) is the spectral radius of \( Q_\gamma \) and \( x \) is the corresponding (positive) eigenvector (see [28]).

Let us now observe that

\[
0 = (Q_\gamma - \alpha_\gamma I) x = (Q_\gamma - C \gamma I) h.
\]

Therefore, \( C \gamma \) is an eigenvalue for the eigenvalue problem from Eq. (29) and thus

\[
\alpha_\gamma \geq C,
\]

since by construction \( \alpha_\gamma \gamma \) is the corresponding spectral radius.

To show the converse, i.e., \( \alpha_\gamma \leq C \), consider the exact asymptotic decay of the distribution of the queue occupancy (of \( A(t) \) when fed at a queue with capacity \( C \)), i.e.,

\[
\lim_{\sigma \to \infty} \frac{1}{\sigma} \log P(Q > \sigma) = -\theta^*, \tag{30}
\]

where \( \theta^* = C \) (see [27]). In other words, \( \theta^* \) is the exact asymptotic decay rate. As Corollary 2 predicts \( \gamma \) as a decay rate, in terms of an upper bound, it follows that \( \gamma \leq \theta^* \). Finally, since \( \alpha_\gamma \) is increasing in \( \theta \) (see [10], p. 241), it follows that

\[
\alpha_\gamma \leq \alpha_\theta = C,
\]

completing the proof. Alternatively, one can invoke the lower bound from Corollary 3. Note that a more direct proof can be
given by invoking the exact result from Eq. (30) along with the upper/lower bounds from Eqs. (10) and (14), respectively.

VI. CONCLUSIONS

In this paper we have advanced the queueing analysis of the versatile class of general Markov fluids. In particular, we have improved the state-of-the-art upper bounds on the queue size distribution by an exponential factor in the special case of Markov modulated On-Off sources. We have further provided the first matching lower bounds, and also the first bounds on the per-flow delay distribution under FIFO, SP, and EDF scheduling.

An attractive feature of our stochastic bounds is that they are obtained using a powerful martingale methodology, which essentially invokes Kolmogorov-Doob inequality arguments, and which was shown to render accurate stochastic bounds. An alternative and arguably more powerful technique based on integral equations was provided by Kingman [30] in the case of networks with a discrete-time setting; its extension to the non-renewal case can in principle provide clues for further improving the current upper/lower bounds.

REFERENCES

Demo: Resilient Integration of Distributed High-Performance Zones into the BelWue Network Using OpenFlow

Mark Schmidt†, Robert Finze†, Daniel Reutter∗, Michael Menth∗
∗ Chair of Communication Networks and † Zentrum für Datenverarbeitung, University of Tuebingen, Tuebingen, Germany

Abstract — The Internet service provider (ISP) BelWue interconnects higher education and research institutions in Baden-Wuerttemberg, Germany. Its 10 Gb/s backbone among university campuses has been augmented with an additional optical 100 Gb/s high-speed network called NeIF. The bwNET100G+ project sets up special high-performance zones (HPZ) at a few university campuses and seeks for cost-efficient interconnection over the NeIF and integration into the legacy infrastructure. We propose for that purpose SDN-NeIF, a software-defined network architecture leveraging OpenFlow and iBGP. The demo illustrates this concept for three university locations with a special focus on resiliency. It leverages a virtualized testbed consisting of four desktop PCs and three consumer switches. The demo validates the SDN-NeIF architecture and serves as a first proof-of-concept.

I. INTRODUCTION

The Internet service provider (ISP) BelWue [1] interconnects higher education and research institutions in Baden-Wuerttemberg, Germany. BelWue offers universities a redundant 10 Gb/s uplink to its core network and the Internet. The core network was recently expanded with a new 100 Gb/s optical platform, called NeIF (Netzwerk fuer Innovation und Forschung / network for innovation and research). A 10 x 10 Gb/s switching matrix in any university location enables the setup of 10 Gb/s optical point-to-point connections between any two locations. To leverage the increased network capacities, novel high-performance zones (HPZ) with high-performance servers and switches are installed within the bwNET100G+ project at university locations in Karlsruhe, Tuebingen, and Ulm. One objective is to seek for cost-efficient interconnection of HPZs over the NeIF and their integration into the legacy infrastructure which can be achieved by the use of SDN-capable switches instead of full routers.

In this paper we describe an architecture (SDN-NeIF) that leverages OpenFlow [2] and iBGP [3] for that purpose. We describe a demo implementing SDN-NeIF on a virtualized platform and illustrate the data flow under working and failure conditions.

The design meets some important real-world requirements. Optical 10 Gb/s links must be efficiently used as they do not suffice to support a full mesh; furthermore, some of them need to be reserved for special services. While campuses have a redundant uplink to the existing BelWue network, the NeIF typically connects neighboring locations only with a single link. In case of a link failure in the NeIF, failover mechanisms should guarantee communication with and among HPZs over the legacy network. The solution should be independent of the internal structure of the campus network and the HPZ because these structures may vary among locations. Moreover, the design must respect that border routers and border switches are administered by BelWue while other campus and HPZ infrastructure is operated by universities. We choose a software-defined networking (SDN) approach because it facilitates cost-effective IP-based forwarding using switches and allows for advanced traffic engineering and security solutions in the future.

The remainder is structured as follows. Section II reviews related approaches interconnecting “Science DMZs” and research networks. In Section III, we introduce the SDN-NeIF architecture. Section IV explains a virtualized testbed implementing SDN-NeIF and Section V describes the demo to be shown. Section VI concludes this work.

II. RELATED WORK

We briefly review other projects that provide high-speed networks for research and how they are integrated into a campus infrastructure.

ESnet Science DMZ [4], [5] proposes high-speed zones within a university that form a dedicated network for research data in addition to a general purpose network at a university campus. A high-speed DMZ is a central place for storage and computation servers which has special security policies and enforcement. Typically, the access switch for a high-speed DMZ is directly connected to the university border router. SDN facilitates the communication within a DMZ and among different DMZs of a university. The research network within a university provides a bandwidth of 100 Gb/s to the DMZs which allows to use resources across DMZs without relaying traffic through the campus network. A high-speed network among the DMZs of different universities is not mentioned.

Internet2 [6] provides US educational and research centers with a typical uplink of 10 Gb/s through an optical 100 Gb/s backbone. Its goal is to provide high-speed access for collaborative applications, distributed research experiments, grid-based data computation, and analytics. In particular, Internet2 can interconnect Science DMZs at different locations.

McCahill [7] described at an Internet2 technical meeting a bypass of the campus core network of a university for special traffic, e.g., scientific data. Different departments of a
university are connected to the campus core network via SDN-capable switches and the bypass among them is implemented with dedicated high-speed links.

SciPass [8] describes a security-enhanced Science DMZ. An intrusion detection system (IDS) is connected to an SDN-capable switch and classifies traffic as trusted and untrusted. An example for trusted traffic may be scientific data exchanged between universities. Once a flow is classified as trusted, the network is configured so that this flow is routed around potential bottlenecks and can bypass firewalls. The goal of this approach is to enable the utilization of a 100 Gb/s connection between different campuses.

GEANT2 [9] is a European research network. The GEANT project already exists for several years and its backbone has been upgraded from 100 Gb/s in 2013 to currently 500 Gb/s. Its core is based on dark fibres with an OTN switching layer and a GMPLS control plane.

### III. SDN-NeIF Architecture

We introduce the physical interconnection of the BelWue core, campus networks, and HPZs, explain the integration of the HPZs using OpenFlow and iBGP, and present failover mechanisms.

#### A. Physical Interconnection

Figure 1 shows a campus router (CR) confines the border of a campus network and connects to a BelWue border router (BR) on the premises of the university. The BR provides a 10 Gb/s uplink to the BelWue core network and access to the Internet. All HPZs together have an IPv4 /22 address space out of which each HPZ has its own /24 prefix that is administered by the corresponding university. A HPZ is attached to a BelWue border switch (BS) on the premises of the university which interconnects via the NeIF to BSs of other locations. The BS has a direct link to the CR and the BR.

![Figure 1: Topology of SDN-NeIF with three universities connected to the legacy BelWue network and the NeIF.](image)

#### B. Integration with OpenFlow and iBGP

The BS integrates the HPZ, the campus network, the BelWue network, and the NeIF. It is configured via OpenFlow by its BS controller (BSC) which is operated by BelWue. The BS connects to its BSC via a BelWue transit network attached to the BR so that OpenFlow control messages are forwarded from the BS over the BR to the BelWue core. To that end, the BS has a static route towards its BSC via the BR, and the BR announces a route towards the BS to the BSC.

Within a HPZ, devices are directly connected to the BS or indirectly via an IP gateway which is responsible for a subnet within the /24 network of the HPZ. For directly connected devices in the HPZ, the BSC installs dynamic flow entries on the BS as follows. If the BS receives a packet for a device in the HPZ which does not match any existing flow entry, the BS informs the BSC. The BSC sends an encapsulated ARP request to the BS which then broadcasts the ARP request to all of its neighbors within the local HPZ. The corresponding device replies, the BS relays the response to the BSC, and the BSC installs an appropriate forwarding rule on the BS. The dynamic installation of flow entries for devices directly attached to the BS may be optimized, e.g., by leveraging information from a local DHCP server. The BSC can install flow entries for IP prefixes for which connected gateways in the HPZ are responsible. To that end, the BSC has an iBGP speaker running and may be contacted by a gateway via iBGP to learn about it and its address space.

All BSs in the BelWue network exchange routing information via iBGP and so does a BR with the same university’s CR. The objective is that all campus networks can reach each other and that they are reachable from the Internet. In SDN-NeIF, the BS announces the /24 prefix of the local HPZ via iBGP towards the BR. As the BS is only an SDN-capable switch and not a router, it cannot speak BGP itself. Therefore, the corresponding BSC acts as proxy for its BS, i.e., the BSC maintains an iBGP session with the BR and announces the availability of the local /24 HPZ prefix via the BS.

In a similar way, the BSC maintains an iBGP session with the CR and informs the CR that the BS is the next hop for the /22 prefix of all HPZs. Therefore, traffic from a campus network to a remote HPZ is normally forwarded through the NeIF instead through the BelWue core due to a shorter AS path. In addition, the CR announces the prefix of the campus network to the BSC which then installs an appropriate forwarding rule for campus traffic on the BS.

The BSs of different locations are interconnected with multiple 10 Gb/s point-to-point connections which are aggregated to a single virtual link by the Link Aggregation Control Protocol (LACP) [10]. They constitute the core network of SDN-NeIF. To relay traffic destined to remote HPZs, the BSCs configure their BSs with flow rules using the /24 IP prefixes of the remote HPZs as match patterns and appropriate neighboring BSs as forwarding action. The same is done for the address spaces of other campus networks so that traffic originating in a HPZ is forwarded through the NeIF to destinations in remote campus networks.

#### C. Failover Mechanism

For resilience reasons, every university campus is in fact connected by two BRs with identical IP address over non-overlapping paths to the BelWue core network. Likewise, there are two CRs with identical IP address within a campus network, and a full mesh interconnection among the two BRs and CRs. The Virtual Router Redundancy Protocol (VRRP) [11] is used between the two CRs so that they act as one virtual router. This ensures connectivity even if
one fails. Similar applies to the two BRs. Thus, any CR, BR, or path towards the BelWue core network may fail without compromising the uplink of a location.

As a result, we can consider the connection between the BS and its BSC as highly reliable because the BSC is located in the BelWue core network. SDN-NeIF implements out-of-band signalling since OpenFlow control traffic is not carried over the NeIF. A potential failure in the NeIF is detected by adjacent BSs which inform their BSCs about this event.

In such a case, the BSCs reconfigure the BSs to send affected traffic for remote HPZs via the corresponding BRs from where the traffic is attracted through the BelWue core network and the BR of the destination to the remote BS.

If traffic between two HPZs is affected by a non-adjacent failure in the NeIF, the BS detecting the failure reroutes the traffic. This may be problematic since the uplink of another university is used for rerouting. Therefore, we are currently working on BSC-to-BSC communication so that a BSC can notify the other BSCs about that failure. If a BSC is informed about a failure within the NeIF, it can reconfigure its BS to reroute affected traffic. Thereby, affected traffic is rerouted through the uplink of the originating network.

IV. SDN-NeIF Testbed

The SDN-NeIF testbed implements a topology as shown in Figure 1 with three university locations (Site A, B, and C) and the BelWue site. Each university site consists of a high-performance host (HPH) in the HPZ, a campus host (CH), a CR, and a BR. The BelWue site accommodates the BSCs for the three BSs, an Internet host (IH), and a BelWue core router (BCR). The implementation of the BSCs is based on the Ryu SDN framework [12] which supports all OpenFlow versions from 1.0 up to 1.5. We chose Ryu because it is implemented in Python which allows for rapid prototyping and it is relatively lightweight compared to other SDN controllers. As the large number of physical entities is expensive and difficult to manage, we model most of them as virtual machines (VMs) and implement the BSs and auxiliary switches (AUX) on switch partitions. In the following, we map virtualized entities to hardware, discuss the hardware platform of the testbed, and explain the virtualization approaches.

A. Mapping Virtualized Entities to Hardware

Figure 2 depicts the testbed which consists of four desktop PCs, three SDN-capable switches, and one unmanaged consumer switch. CH, CR, BR, and HPH of a university location are realized as VMs on a dedicated desktop PC and the BS is realized on a partition of an SDN-capable switch. As the BSCs and the IH do not require dedicated external Ethernet interfaces and computational power, they are realized as lightweight containers instead of VMs on an additional desktop PC. They are connected via virtual network interfaces to the host which acts as BCR and is connected to the unmanaged switch. This switch represents the BelWue core network connecting the BRs and the BSCs and giving access to the Internet.

Figure 3 illustrates the organization of the VMs of a university location on a desktop PC with a quad-port NIC. Each port of this NIC is used for one VM. As some VMs require multiple interfaces, VLAN tags are used to identify their traffic which is multiplexed over a single port. The HPH is directly connected to the BS using a 1 Gb/s link. All other ports are connected to the AUX partition of the SDN-capable switch. The SDN functionality is not enabled for the AUX partition of the switch because it is only used for multiplexing and demultiplexing VLANs, and possibly for connecting them to other VLANs or physical 100 Mb/s ports. The BS is controlled over a configured VLAN (BR/BS-Ctrl) which is exclusively used for that purpose. Therefore, the BR requires one VLAN for data traffic (BR/BS-Data) and one for control traffic (BR/BS-Ctrl) towards the BS.

B. Experimental Platform

The computers hosting VMs are equipped with an Intel Core i7 CPU, 16 GB RAM, and an Intel 1350-T4 NIC with 1 Gb/s interfaces to provide sufficient computational power and virtual network interfaces for the VMs. The PC holding the containers is equipped with an Intel Core i5 CPU, 4 GB RAM, and a 100 Mb/s NIC as it does not need that much resources. The hosts, VMs, and containers run a minimal Ubuntu as operating system with additional tools for debugging and testing. The virtual routers, namely BR, CR, and the BCR modelled by the host use the Quagga [13] routing suite with the BGP module enabled.
Like already mentioned in section IV-A, we use VLAN tags to distinguish multiple virtual interfaces per NIC. The igb [14] driver of the NIC can be configured in a way that automatically adds a VLAN tag for traffic from VM to host and strips the VLAN tag for traffic from host to VM at the transition between VM to host. This way the VMs do not see any VLAN tags or even have to know about them.

C. Virtualization Platform

To use hardware-accelerated virtualization on the Intel x86 platform, some extensions are needed as the architecture itself is not virtualizable. Intel VT-x [15] enables basic hardware acceleration on that platform. VT-d [16] provides an IOMMU which allows to pass-through devices, e.g., NICs from the host to the VM. VT-C [17] comprises Virtual Machine Device Queues (VMDq) [18] which enables multiple queues and Single Root I/O Virtualization (SR-IOV) [19] which is an extension to the PCI standard. With these extensions it is possible to provide multiple virtual NICs on the host, each with its own queue per physical NIC, and to individually pass them through to a VM.

These hardware features must be supported by the hypervisor running the VMs. As hypervisor we use kvm [20], which is part of the Linux kernel, in conjunction with the qemu [21] tool. The VMs are managed with the libvirt [22] framework. Regular desktop PCs can provide these features, but in contrast to servers, the features are mostly disabled by default. Some additional kernel parameters need to be set to enable all required features and libvirt has to be configured to use a special pass-through method. As a result, we run multiple VMs per host with virtual NICs having an overall I/O performance close to the one of a dedicated physical machine.2

D. Container Platform

Linux containers (LXC) [23] provide a method for virtualization at the operating system level that enables multiple isolated Linux systems (containers) on a host. All containers share the same Linux kernel with the host. For resource limiting, process prioritization, and access control to hardware, LXC relies on Linux kernel cgroups [24]. Isolated namespaces [25], [26] allow containers to have their own process and network space within the host system. LXC and containers save computational and resource overhead compared to a hypervisor and VMs. Running containers as unpriviledged users prevents them to damage the host system and access hardware directly.

V. DEMO

The demo illustrates traffic forwarding in SDN-NeIF in failure-free scenarios and in case of link failures in the NeIF. The topology, statistics, and the flow tables of the BSs are visualized by a web app. The demo serves as validation of the architecture and a first proof-of-concept.

VI. CONCLUSION

We proposed SDN-NeIF as an SDN-based network architecture that integrates university HPZs into university campuses, the BelWue network, and the high-speed NeIF network. OpenFlow controllers act as iBGP proxies for integration into the BelWue network and reconfigure BSs in case of a failure in the NeIF to reroute traffic through the BelWue core network. The presented testbed consists of four desktop PCs, three SDN-capable switches, and one consumer switch, and virtualization techniques are used to model a multitude of testbed nodes. The testbed helped to develop SDN-NeIF and serves as a first proof-of-concept. SDN-NeIF is currently implemented as a prototype on the real NeIF platform.

REFERENCES

Demonstrating a Personalized Secure-By-Default
Bring Your Own Device Solution Based on Software Defined Networking

Steffen Gebert, Thomas Zinner, Nicholas Gray, Raphael Durner, Claas Lorenz, Stanislaw Lange

Abstract—Network virtualization is one classical use-case for Software Defined Networks (SDN). By programmatically instantiating virtual networks, traffic from one or more devices can be separated or connectivity can be established as needed. S-BYOD, which is presented in this demonstration, applies the SDN concept to Bring Your Own Device (BYOD) scenarios and offers personalized virtual networks that are set up and extended on demand. This is done once the user authenticates, activates access to additional applications, or as soon as applications scale out and involve more servers. The described proof-of-concept implementation explores, to what degree an agent-less BYOD solution, based only on SDN, can lower the attack surface by explicit user opt-ins for particular services. Further, an assessment of the number of required rules within the flow tables of switches completes this work.

I. INTRODUCTION

While SDN's means to create virtualized networks can be helpful for BYOD scenarios, where potentially insecure devices should be included in organizational IT infrastructure, none of the currently available solutions applies SDN to offer a fine-grained restriction of network access. Instead, commercial solutions either group connect all devices into a special VLAN that can be further restricted, or - in the case of more sophisticated Mobile Device Management (MDM) solutions - an agent runs on the smartphones or tablets that checks its trustworthiness.

Sardine-BYOD (S-BYOD), the solution introduced in this work, aims at solving the problem only on the network level. No modification of the user-owned device or restriction of privileges is needed. Instead, S-BYOD restricts each device to its own virtual network. After authentication, the user can explicitly enable corporate services for this specific device following the *sudo* principle of operating systems. Based on an off-by-default approach, the number of services, to which a device running a potentially malicious app, can connect is reduced. To activate access to a particular service, an authenticated user can simply use the web portal of the BYOD solution.

II. SARDINE-BYOD (S-BYOD)

The introduced BYOD solution is publicly available as open source and implemented as plugin for the ONOS controller. Its main innovation is the implementation of personalized virtual networks which are set up and adjusted on demand.

In contrast to other BYOD solutions, which usually group all devices together in a dedicated VLAN and thus do not allow to set up fine-grained policies, S-BYOD individually restricts network access for every device by the means of Software Defined Networking. Further, these virtual networks are automatically adjusted according to the user’s current requirements, i.e., when additional services are requested or when the user roams between access points, as well as when changes within the IT services infrastructure are made.

A. Building Blocks

To achieve the outlined goal of SDN-based virtualized networks for BYOD users, the setup builds up on the following components and mechanisms, which are also shown in Figure 1:

- **OpenFlow-enabled switches**: The wired network infrastructure is built using OpenFlow-enabled switches. OpenFlow [2] is a protocol that offers a standardized API for the communication between SDN controllers and OpenFlow-enabled switches. This allows to programmatically establish and modify individual virtual networks.

- **ONOS SDN controller**: ONOS is a well-known controller for software defined networks and allows to modify the forwarding tables of switches using the OpenFlow protocol. By default, ONOS includes basic network services, including reactive forwarding and DHCP and offers APIs for SDN applications running inside the controller.

- **Wireless Access Points**: Given the lack of OpenFlow-based wireless access points (APs), traditional APs are used, to which the BYOD users can connect. These APs form an
**Extended Service Set (ESS)**, meaning they all advertise the same ESSID and are connected to the fixed, OpenFlow-based network.

**Corporate Services**: Users of the wireless network need to access business applications that are running in the corporate infrastructure. Such applications can include browser-based access via HTTPS, email services using IMAP/SMTP, or other IP-based protocols, e.g., for printing. Furthermore, Internet access might need to be available to a private device in the corporate environment. It is assumed that most of these applications are running within a private cloud environment in the corporate IT infrastructure.

**Service Discovery**: As cloud-based applications scale accordingly to their usage, the virtual machines, on which an application is running, change over time, i.e., when using modern deployment techniques [3]. Hence, the IP addresses of these applications change as well. In order to allow different instances of cloud applications to connect to each other resp. to connect to their microservice instances [4], service discovery systems [5] are an essential part of modern cloud-based application architectures. This work makes use of Consul [6] as discovery service.

**Captive Portal**: Finally, a web-based captive portal is used to authenticate and authorize users. This portal has access to the corporation’s authentication services, i.e., LDAP/AD, and further uses two-factor authentication (2FA), based on the Time-Based One-Time Password Algorithm (TOTP) [7] to prevent an attacker from accessing corporate services with stolen credentials. The captive portal communicates with the BYOD controller module using RESTful APIs. In the demonstration, the captive portal is implemented using Meteor [8] as client- and server-side web framework. Figure 2 shows the portal, in which the authenticated user can explicitly enable particular applications, while every action that elevates privileges has to be authenticated using a 2FA verification.

**Fig. 2.** Screen shot of the captive portal to activate access to corporate services. Green: activated services, red: deactivated services, yellow: services to be activated after 2FA.

### B. OpenFlow Rule Setup

In order to allow basic network connectivity, redirect unauthenticated users to the captive portal, as well as to drop all unwanted traffic, the following rules are defined by the ONOS controller in order of increasing priority:

**Table-miss action drop all** $r_{miss}$: This deny all default rule drops all unmatched packets by defining an all-wildcard rule without any actions.

**ARP handling** $r_{ARP}$: All ARP packets are forwarded to the SDN controller and processed by ONOS’ proxyarp app.

**DHCP service** $r_{DHCP}$: All DHCP packets are forwarded to the SDN controller and processed by ONOS’ DHCP app.

**HTTP to controller** $r_{http_to_ctrl}$: For intercepting outgoing HTTP connections of any unauthenticated client, this low-priority rule redirects any TCP traffic to port 80 to the controller. This allows S-BYOD to intercept the HTTP connection and redirect the client to the portal web site for authentication.

**DNS server connectivity** $r_{DNS}$: As soon as a new client is detected, rules are provisioned to allow DNS traffic to the corporate DNS server. Therefore, persistent rules for each direction between a client device and the DNS server are installed throughout the path. Connectivity to the DNS server configured through DHCP is essential also for unauthenticated users, as otherwise no attempts to establish outgoing HTTP connections will be made, which is necessary for the portal redirection.

**Portal connectivity** $r_{portal}$: To enable connectivity between the client device and the captive portal via a secure HTTPS connection, the complete path is provisioned by one rule per direction between client and portal server.

**Service Connectivity** $r_{service}$: As soon as a user has successfully authenticated at the portal and enabled a particular service, these rules permit network-side access to the IP addresses of the servers offering this service, e.g., email, web-based service, or any other protocol. Given the higher priority of rules, the previously applied rules to drop packets are overridden to successfully forward packets between the client and the destination server.

**Internet Connectivity** $r_{external}$: All traffic that is routed outside of the Layer 2 network, in which clients and the accessed applications reside, require connectivity to the default gateway. Therefore, once the user enables Internet connectivity in the portal, rules that allow traffic between the MAC addresses of the client and the default gateway are installed on switches along the path. Given the gateway’s configuration to not route traffic inside the OpenFlow-managed network, access to this MAC address does not allow the user to bypass security means which prevent accessing other internal devices.

All client-specific rules are installed once the new client is connected. All service-specific rules are installed once a user enables or disables application access in the captive portal. Once the BYOD service gets notified by the controller that a host disappeared from the network, all these rules are removed from the switches.

Once the controller receives packets originating from an unauthorized user accessing a web page via the $r_{http_to_ctrl}$ redirection, it instructs the switch to rewrite the source and
destination IP and MAC addresses to the portal using a packet_out message. As the packets on the way back from the portal web server to the user also need to match the intercepted connection’s addresses, further packet_out messages also rewrite the endpoint addresses of this intercepted connection. As the portal only responds to the HTTP request with a HTTP Location redirect header to its resolvable address, no flow entries are set up in the switches for this short-lived connection.

C. Wireless Client Mobility

To ensure high user satisfaction, wireless clients need to be able to roam between wireless access points, i.e., when moving from one room or building to another. The security mechanisms and rules installed by the BYOD solution therefore have to support this use case.

As the introduced S-BYOD relies mostly on static rules for performance and scalability reasons, a roaming client needs to be detected quickly and accordant rules need to be updated as well. RFC 5227 [9] introduces procedures for mobile clients to detect the correct setup of wireless networks spanning multiple access points. This is achieved by sending an ARP packet to the previous gateway, once the client switches to the new AP. As this mechanism is implemented in most of the current operating systems, this also offers a safe way to detect the roaming device. In particular, the controller immediately notices this through the received ARP packets which originate from a different switch (port) and thus allows S-BYOD to update all flow rules and maintain connectivity for the roaming device.

III. DISCUSSION

While ease of use is one important factor for user satisfaction, security of a BYOD implementation is an important aspect as well. Therefore, different mechanisms of S-BYOD that mitigate or still allow certain known attacks will be discussed in the following. Further, an estimation of the resource requirement in number of flows in OpenFlow switches will be provided.

A. IP Address Spoofing

To allow connectivity to the portal, every device connected to the WiFi has to receive an IP address from the DHCP service. By spoofing the IP address of another, already authenticated client, an attacker could gain access to internal services. Guessing such IP address is easy, as the used IP address range is known once connected to the network. Therefore, all client-specific flow rules \( r_{\text{service}} \) and \( r_{\text{external}} \) have to also match against the client’s MAC address and thus lead to all traffic using a self-assigned IP address being dropped.

B. Wireless Client Isolation

In order to prevent communication between mobile devices and especially to prevent an attacker gaining knowledge of other (potentially authenticated) users’ MAC addresses, the access points operate in client isolation mode. By activating such a setting, no incoming traffic of wireless clients is sent directly back to the air interface, but only to the wired connection. There, the wired OpenFlow-based network takes care of separation of clients. Further implications of ARP/MAC spoofing will be discussed in Section III-D. OpenFlow-enabled WiFi-APs, if they would exist, are therefore not even needed, as long as wireless clients never need to communicate directly.

C. Estimation of OpenFlow Rules

One concern with large-scale OpenFlow deployments is the number of required forwarding rules. Therefore, the usage of flow table entries is discussed in the following. As all rules defined by S-BYOD use exact matches, i.e., MAC and IP addresses, these rules can be stored in content-addressable switch memory. In contrast, using wildcards, e.g., matching for \( \text{nw}_\text{dst}=192.168.0.\_\text{net} \), would require expensive and very limited ternary content-addressable memory (TCAM).

In real-world scenarios this is mitigated by utilizing multiple switches and thus, the client-specific rules are distributed over the switches. Then, only the switches on the path between a client and its connection endpoints receive the accordant rules. For the extreme case that all users are connected to only a single switch, the number of required rules is calculated as follows and provides a worst-case estimation:

\[
\begin{align*}
\text{n}_{\text{total}} &= \text{n}_{\text{base}} + \text{n}_{\text{client}} + \text{n}_{\text{service}} \\
\text{n}_{\text{base}} &= \left[ r_{\text{miss}} + r_{\text{ARP}} + r_{\text{DHCP}} \right] + \left[ r_{\text{discovery}} + r_{\text{HTTP}_\text{to}_\text{ctrl}} \right] \\
\text{n}_{\text{client}} &= c \cdot \left[ r_{\text{DNS}} + r_{\text{portal}} \right] \\
\text{n}_{\text{services}} &= \sum_{i=1}^{c} \sum_{j=1}^{\text{max}} n_{i,j} \cdot |r_{\text{service},j}| 
\end{align*}
\]

Figure 3 depicts this relationship for different numbers of active BYOD clients and enabled applications per user, while assuming that every service is reachable through two IP addresses.
fine-grained level, while the network infrastructure itself does not have to be changed. In contrast, the motivation behind S-BYOD is to not require changes to the devices, but instead providing a high level of security through the support of the network infrastructure.

V. CONCLUSION

The presented approach for an SDN-based BYOD implementation can be used together with an existing OpenFlow-based infrastructure. It makes use of SDN’s means to dynamically define virtual networks, which are set up and maintained on a per-user level. Using a portal web site and two-factor authentication, a safe yet comfortable way is provided to the user, to only selectively enable connectivity to the applications that are currently needed and thus lower the range of services that a potentially malicious device can access. By connecting to a service discovery system used by corporate applications, changes can be immediately reflected with updates to the flow rules in the switches and thus provide always up-to-date network access.

ACKNOWLEDGMENTS

This work has been performed in the framework of the SARDINE project and is partly funded by the BMBF (Project ID 16KIS0261). The authors alone are responsible for the content of the paper. The authors want to thank Lorenz Reinhart and Benedikt Pfaff for their programming work.

REFERENCES


Fig. 3. Amount of required rules in a single-switch setup, 2 IPs per application.

IV. RELATED WORK

Programmable BYOD Security (PBS) applies SDN/OpenFlow to mobile devices using the PBS-DROID application. This lets the Android device run an OpenFlow-controlled switch with the smartphone apps connected to it. By running on the device itself, it allows per-app policies. Compared to S-BYOD, PBS is able to work on a more...
Demonstrating Context-Aware Services in the MobilityFirst Future Internet Architecture

Francesco Bronzino, Dipankar Raychaudhuri and Ivan Seskar
WINLAB, Rutgers University, North Brunswick, NJ 08902, USA
Email: {bronzino, ray, seskar}@winlab.rutgers.edu

Abstract—As the amount of mobile devices populating the Internet keeps growing at tremendous pace, context-aware services have gained a lot of traction thanks to the wide set of potential use cases they can be applied to. Environmental sensing applications, emergency services, and location-aware messaging are just a few examples of applications that are expected to increase in popularity in the next few years.

The MobilityFirst future Internet architecture, a clean-slate Internet architecture design, provides the necessary abstractions for creating and managing context-aware services. Starting from these abstractions we design a context services framework, which is based on a set of three fundamental mechanisms: an easy way to specify context based on human understandable techniques, i.e., use of names; an architecture supported management mechanism that allows both to conveniently deploy the service and efficiently provide management capabilities; and a native delivery system that reduces the tax on the network components and on the overhead cost of deploying such applications.

In this paper, we present an emergency alert system for vehicles assisting first responders that exploits users location awareness to support quick and reliable alert messages for interested vehicles. By deploying a demo of the system on a nationwide testbed, we aim to provide better understanding of the dynamics involved in our designed framework.

I. INTRODUCTION

Context-aware systems offer entirely new opportunities for application developers and for end users by gathering context data and adapting systems behaviour accordingly [1]. Especially in combination with mobile devices these mechanisms are of high value and are used to increase usability tremendously. Context data, usually identified as external factors from the network environment, can extend across a wide variety of different fields including for example: environmental conditions, time, location, available energy, network attachment points, channel conditions, and communicating sources and destinations. Moreover, human related social states can be part of the analyzed environment, for example, if a user is in meeting, busy, or free.

The MobilityFirst (MF) future Internet architecture project represents a clean-slate Internet architecture which provides the necessary abstractions for creating and managing context-aware services. In particular, the architecture enables dynamic identification of endpoints based on context attributes through the use of named-object identifiers and global name resolution [2]. The current Internet primarily supports a primitive to send data to a specific IP address, which limits applications to cast all communication intent in those terms. This primitive is inflexible when the network location of the destination (or even the principals constituting the destination) is not known a priori. For example, several mobile or Internet-of-Things applications can benefit from context-aware primitives such as "send this message to all taxis in the Times Square area" or "request power consumption readings from devices in my living room", which are cumbersome to implement in IP. In contrast, MF enables context-aware communication primitives based on attributes more general than just the network location and dynamically associates a context identifier to its constituent principals.

Our strategy is to develop an architecture where we can name environmental contexts that change where and how messages are routed and delivered. Application services could expect from such architecture improvements that span areas including security, communication efficiency, and energy management. In order to do so, we identify three strategic mechanisms that are required to accomplish the set goals: an easy way to specify context based on human understandable techniques, i.e., use of names; an architecture supported management mechanism that allows both to conveniently deploy the service and efficiently provides management capabilities; and a native delivery system that reduces the tax on the network components and on the overhead cost of deploying such applications.

II. MOBILITYFIRST PROTOCOL STACK

The context services framework design presented in this paper is based on the MobilityFirst future Internet architecture. In order to at best understand this design, we first need to introduce the architecture network protocol stack and its core technology components, previously presented at different venues [2], [3]. The MobilityFirst architecture’s main design centers around a new name-based service layer which serves as the “narrow-waist” of the protocol stack. The name-based service layer uses the concept of “flat” globally unique identifiers (GUIDs) for network attached objects, a single abstraction which covers a broad range of communicating objects from a simple device such as a smartphone to a person, a group of devices/people, contents or even contexts. This name-based services layer makes it possible to build advanced mobility-centric services in a flexible manner while also improving security and privacy properties. Network services are defined by the source and destination GUID and a service identifier to specify the delivery mode such as multicast, anycast, multi-homing, content retrieval or context-based message delivery. A hybrid name/address based routing scheme is used for scalability, employing a Global Name Resolution Service (GNRS) to dynamically bind the GUID to a current set of network addresses (NAs). The GNRS in MobilityFirst is a logically centralized service responsible for naming, security, and augmenting network layer functionality. The clean separation

*Research supported under NSF Future Internet Architecture - Next Phase (FIA-NP) grant CNS-1345295
of identity and location of endpoints is the key to achieve seamless mobility and in achieving trustworthiness ensuring that identities and their locations can be easily verified.

Data transport in MF is achieved by transferring blocks in a segmented manner using storage-aware routers unlike the current Internets end-to-end approach using TCP/IP. A block transport protocol transports blocks, or large chunks of contiguous data, in a hop-by-hop reliable manner as opposed to traditional transport protocols like TCP that transport small packets in an end-to-end rate-controlled manner. Segmented transport generalizes hop-by-hop transport to segments or a sequence of contiguous links terminated by storage-aware routers or endpoints. Congestion control across segments follows a segment-level back pressure approach similar to Hop [4].

For evolvability, MobilityFirst incorporates a virtualized compute layer that enables novel programmable network services to be deployed rapidly into the routing fabric. However, there are two key challenges to be addressed to make this approach practical. First, the compute layer must not introduce significant overhead on the default forwarding path for legacy traffic. Second, the compute layer must ensure resource containment for each service and isolation across different services for security and accountability. The compute layer in MF relies on such an API similar in spirit to software defined networks, but goes beyond the basic use case of virtualized network control and management to offer more general "packet cloud" services in the data path.

Finally, at the core of the architecture is a name-based networking abstraction that contrasts with the name-address conflated communication interface associated with Berkeley sockets and the TCP/IP stack. All network-attached objects in the MobilityFirst architecture enjoy direct addressability through long lasting unique network names or identifiers (we use GUIDs). This new GUID-centric network service API, first presented in [5], offers network primitives for basic messaging (send, recv) and content operations (get and post) while supporting several delivery modes natively supported by the MF network such as multihoming, multicast, anycast and DTN delivery.

III. CONTEXT SERVICE

While the MobilityFirst architecture provides the necessary abstractions for creating and managing context-aware services, new mechanisms are required in order to fully exploit them. Starting from the three strategic mechanisms defined, i.e. an easy way to specify context, an architecture supported management mechanism, and a native multi-point delivery system, three fundamental technology components are exploited to design the MobilityFirst based context services framework.

Global Name Resolution Service: Recall that in MobilityFirst the GUID represents an abstract endpoint that is independent of the network topology. We leverage this independence to build contextual networking; our approach is to overload the GUIDs at many levels of the network. A GUID could thus represent many abstractions; for example, a cell-phone, a person, or a group that is defined by context. In this sense, contextual communications are similar to *-cast types networking. For example, we could overload a GUID that names sending a message to a meeting. The system would have to first translate the GUID into a list of each person in the meeting. Further translation would be needed to identify the device each person is currently using. The Global Name Resolution Service is then the first key enabling technology that allows us to define context by collecting common set of network entities under a single name defining the context.

In-network computing capabilities: To enable future extensions to the network protocol without expensive hardware replacements and disruption, MobilityFirst builds in an optional and dynamically pluggable compute plane. Examples of such need are additions of new service types, new principal types, new addressing structures, or extensions to the end-to-end security protocol. We envision service providers and network operators to be able to perform relatively simple upgrades in the form of software updates and addition/replacement of pluggable hardware modules to extend the data plane functionality. Furthermore, we also postulate that such extensibility can enable third party application service providers (via the ISPs) to deploy either service end-points or service adaptors that are both closely integrated with the delivery path and best located to improve client experience. In-network computing capabilities are perfectly suited to deploy distributed and scalable management mechanisms to collect contextual information and manage membership. This last step is fundamental to the success of the architecture; for this purpose, in-network deployed services interact with the GNRS to translate collected contextual information into context names. In order to best support the distributed nature of the in-network components, service addressing is of fundamental importance. Generally, the compute layer will be hosted at key locations, whereas network delivery mechanisms will allow to select the best located replica of the service.

Multicast delivery: Multicast protocols have been long studied in the literature and different protocols and architectures have been proposed to support this type of delivery.
mechanisms. While these mechanisms are perfectly suited for static, tree based communications a different approach might be required for contextual applications. In particular, given the highly dynamic nature that defines context services, due to their multiple evolving factors, more flexible multicast delivery mechanisms are required. As importantly, in wireless environments, there is a desire to take advantage of inherent multicast/broadcast medium to enable efficient point to multi-point delivery services. The MobilityFirst architecture solves this problem by implement a lightweight multicast delivery system that, through name grouping in the GNRS, limits per group state within network elements taking per hop decisions on multicast splitting based on Longest-common (LC) look-ahead techniques.

IV. MOBILITYFIRST BASED PROTOTYPE

In order to move towards tested based experimentation we developed a prototype that included the main components that are part of the designed architecture. Due to space constraints, in this paper we will only introduce the main features of such prototype as first presented in [3] and the new components employed in the demo. As the MobilityFirst project addresses the feasibility of building systems and networks in a clean-slate design, it requires the development of such components from scratch. The result of this efforts consists in three main tools: a GNRS implementation based on DMap’s design [6], a Click [7] based software router, and a multiprotocol stack and network API for clients. Applications and network services can be implemented as extensions of these basic elements.

Global Name Resolution Service. A GNRS implementation has been written in Java to provide a hardware and operating system agnostic implementation. The server is organized into several individual modules: network access, GUID mapping, persistent storage, and application logic. The application logic serves as a central point of coordination within the framework of the GNRS server daemon. The network access component ensures that the GNRS server is able to operate over any networking layer/technology without changes to the core code. This replaceable component currently supports IPv4 and MF routing. The GUID mapping module, relying partly on a networking implementation, enables the server to determine the remote GNRS hosts responsible for maintaining the current bindings of GUID values. Persistent storage is handled independently from the rest of the server and exposes only a very simple interface, mapping to the application messages available in the protocol. A BerkeleyDB provides both in-memory and on-disk storage for GUID bindings.

Routers. The software router is implemented as a set of routing and forwarding elements within the Click modular router. The router implements dynamic-binding using GNRS, hop-by-hop transport, and storage-aware routing as presented in Section II. It integrates a large storage, an in-memory hold buffer, to temporarily hold data blocks when destination endpoints experience short-lived disconnections or poor access connections. For dynamic in-network binding of GUID to NA, the router is closely integrated with the in-network GNRS by attaching to a local instance of the distributed service. A particular instance of this system, implements what we call a MobilityFirst access router, a router providing access connectivity to clients, supporting different access technologies (e.g., WiFi, WiMax, Ethernet). Thanks to the modular structure of Click, we are able to extend the software implementation with additional logic modules to support programmable network services. The router software also collects statistics at different layers of the protocol stack that can be reported through Click’s control interface.

Host network stack and API. The host stack has been implemented on Linux and Android platforms as a user-level process built as an event-based data pipeline. The stack is composed of a flexible end-to-end transport to provide message level reliability, the name-based network protocol including the GUID service layer, a reliable link data transport layer, and a policy-driven interface manager to handle multiple concurrent interfaces. The device-level policies allow users to manage how data is multiplexed across one or more active interfaces. The previously introduced socket API [5] is available both as C/C++ and JAVA libraries and implements the name-based service API which include the primitives send, recv, and get and a set of meta-operations available for instance to bind or attach a GUID to one or more NAs, configure transport parameters in the stack, or to request custom delivery service types such as multicast, anycast, multihoming, or in-network compute. Similarly to the router implementation, the protocol stack collects and optionally reports traffic and resource statistics to a backend data repository.

All these components have been designed with flexibility in mind trying to reduce dependency from specific systems to a minimum. The set of basic requirements necessary to run any of these elements is minimal as any x86/x64 machine (physical or virtualized) running a recent Linux distribution can host them (the development has been based on Ubuntu 12.04 LTS).

V. EMERGENCY SERVICE DEMO

We use the developed MobilityFirst prototype presented in Section IV to implement a context services framework designed around the one described in this paper. The GNRS and the native delivery mechanisms, together with the newly implemented in-network context service, enable the MobilityFirst architecture to efficiently deploy context based services improving current architectures in a variety of ways. In order to demonstrate the mechanisms involved in implementing context based services and to showcase some of its benefits, we developed and deployed a contextual application that implements an alert system for vehicles assisting first responders. This service is aimed at providing ways to quickly and reliably transmit emergency messages to group of receivers identified by the conveniency of their geographical position or by the authoritative importance in the emergency matter. To better understand this consider an example where a car accident occurs on the roads of a metropolitan area; in this case, three potential candidates emerge for providing assistance: first, defined by a very small geographical area, other close passbyers could be informed in order to provide first assistance; second, based on a larger area, emergency vehicles such as ambulances could be alerted; finally, given the importance of informing the central authorities, a police station could be included due to its relevance on emergency matters.

The system architecture deployed, as depicted in Figure 3, includes, together with the defining technologies of the MobilityFirst architecture, two core components: the in-network compute management framework and an Android
We exploit the available network host demo. Human based interactions to better understand the presented distributed service. A web interface is also developed for operations, the GNRS API is exploited to interact with the location. Moreover, to implement the required management functionalities to create and manage context based APIs (i.e. using REST) and provide the core required capabilities. Context based operations are exposed using web based APIs (i.e. using REST) and provide the distributed service. A web interface is also developed for human based interactions to better understand the presented demo.

**In-network compute management.** A Ruby based context service is implemented to enable in-network computing capabilities that provide the distributed service described in the previous section. This service is then deployed in proximity of favorably located routers by directly connecting it to the Routers API that provides access to the in-network compute capabilities. Context based operations are exposed using web based APIs (i.e. using REST) and provide the core required management functionalities to create and manage context groups and report context information (e.g. users current location). Moreover, to implement the required management operations, the GNRS API is exploited to interact with the distributed service. A web interface is also developed for human based interactions to better understand the presented demo.

**Contextual application.** We exploit the available network host stack and API to deploy the context application components. In particular, an Android smartphone application has been developed providing two groups of functionalities: first, the core alert operations that allow for quickly broadcasting the alert message to the current context members and to generate notifications for received emergency messages. Second, management operations are allowed, providing ways to interact with the service management framework and create, join and leave given contexts, by exploiting locally collected GPS coordinates.

The demo will be centered around three key operations: 1) creation and management of context groups based on geographical location (Figure 3(a)); 2) dynamic creation of routing information for multicast based delivery (Figure 3(b)); 3) evaluation of efficiency in providing the service. The demo will use the GENI nationwide testbed infrastructure [8] to deploy 3 different locations from where wireless nodes (i.e. Android phones) access via WiFi and WiMax the MobilityFirst architecture. Our current deployment spans 7 GENI sites across the US, 14 Xen VMs (2 VMs per site) each with 1 GB memory and one 2.09 GHz processor core provide us with the possibility to run one router per location and use the other node for application or services. All routers have a core-facing interface connected to a layer-2 network that connects all seven sites. This was setup using a multi-point VLAN feature provided by Internet2’s Advanced Layer-2 Service (AL2S).

In order to better understand all the dynamics involved in the demo, a visualization system has been developed using web-based technologies (e.g. javascript, Google Maps, etc.). This visualization will provide information about different components state. In particular it will show the following information: location of mobile devices and their current status (idle, sending alert, alerted by other nodes), GNRS entries that enable context management and multicast routing, traffic crossing all the nodes in the system and finally important general events spanning all the networking layers of the architecture.

**VI. CONCLUSIONS**

In this paper we presented the details of a context service framework that exploiting core components of the future Internet architecture called MobilityFirst, allows the deployment of flexible and efficient context based services and applications. To demonstrate its properties, a demo using a working prototype has been presented showcasing the core functionalities of the system, using a set of programmable and mobile nodes distributed across different sites on a US testbed. In the future we plan to extend the showcased demo components to support a variety of context services, allowing other research and industry players to exploit the available framework to further push the state of the art.

**REFERENCES**

jLISP: An Open, Modular and Extensible Java-Based LISP Implementation

Andreas Stockmayer*, Mark Schmidt*, Michael Menth*
* Chair of Communication Networks, University of Tübingen, Tübingen, Germany

Abstract—LISP is a standardized overlay protocol implementing the locator/identifier split for the Internet. Multiple extensions exist, e.g., for NAT traversal, traffic engineering, etc. Existing LISP implementations are mostly platform-specific, hard to extend or they are closed source. In this work we present a Java-based implementation for LISP which is open source, modular, and features a plugin mechanism that enables simple integration of new functionality. It provides behavior for LISP nodes, infrastructure nodes, and various extensions, it is easily portable, can be run on various operating systems and platforms, in particular on Android smartphones. The demo illustrates LISP-based communication including the extensions mentioned above. In addition, the extensibility is demonstrated by a plugin for a statistics application.

I. INTRODUCTION

The locator/identifier (Loc/ID) split separates the name of a host, its identifier, from the address of its location, the locator. The locator/identifier binding can be provided by a distributed database, the mapping system, so that traffic to a specific node can be tunneled or address-translated to its locator after lookup of its identifier. Around 2006 this concept was suggested to solve the problem of quickly increasing BGP [1] routing tables in the Internet [2] and a working group in IETF was set up to standardize the Locator/Identifier Separation Protocol (LISP) [3]. LISP by itself cannot solve this scaling issue as large-scale adoption is prerequisite. However, LISP provides an overlay network which is attractive, e.g., for traffic engineering (TE). LISP differentiates from other routing overlays through its control plane which automatically maps locators to identifiers, and the associated mapping system which supports service-specific mapping. LISP may support software-defined networking (SDN) [4], datacenter networking, service function chaining (SFC), NAT traversal, mobile networking, and others.

There are several closed-source and open-source LISP implementations. We started extending them to integrate novel functionality but discovered that they rather focus on performance than extensibility. Some of them are platform-dependent, implement only a subset of standardized features, or are no longer supported. As we feel the need for a LISP software base for research purposes, we provide jLISP as an open-source, easy to extend, and platform-independent LISP implementation that also runs on smartphones. In this paper, we give an introduction to LISP, review other implementations, explain the functionality and software architecture of jLISP and how new features can be integrated, and present a demo that illustrates jLISP in different application scenarios.

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-1. The authors alone are responsible for the content of this paper.

The rest of the paper is structured as follows: In Section II we give a brief overview of the LISP protocol. Section III reviews other LISP implementations. jLISP is presented in Section IV and Section V presents the demo. Section VI concludes this work.

II. LISP

The LISP protocol implements the Loc/ID split. Nodes have Endpoint Identifiers (EID) as names and Routing Locators (RLOCs) as globally routable addresses. Within a LISP domain, EIDs may be used for forwarding. Tunnel routers (xTRs) [3] encapsulate LISP traffic to other LISP domains after retrieval of RLOC/EID mappings from the mapping system. To make EIDs of a LISP domain reachable over the Internet, the xTR registers them at the mapping system. Figure 1 illustrates communication with LISP. A source node with EID 10.0.0.1 in LISP domain 10.0/16 sends a packet to a destination node with EID 20.0.0.1 in LISP domain 20.0/16. The packet is forwarded to its default gateway which is xTR with RLOC 172.10.0.1. The xTR sends a Map Request to the Map Resolver, the interface of the mapping system, and receives a Map Reply containing the RLOC 172.20.0.1 for EID 20.0.0.1. The xTR encapsulates the packet with a LISP header containing control information, a UDP header to port 4342, and IP header to the destination RLOC. Upon reception of the packet, the destination xTR strips these headers and forwards the packet to the destination node.

The xTR can enable interworking with the non-LISP Internet by address translation [5]. That means, it acts like a NAT for traffic leaving the LISP domain for the classic Internet. With this approach, a LISP domain is treated like a private network. In contrast, proxy xTRs (PxTR) make nodes in LISP domains globally reachable via BGP.

A LISP Mobile Node (MN) [3] is a mobile device with an EID and xTR functionality. If the MN becomes connected to a host network, it registers that address as RLOC for its EID at the mapping system to ensure global reachability.

LISP-TE allows forwarding LISP traffic over a sequence of so-called re-encapsulating tunnel routers (RTRs). To that end, the EID is mapped to the RLOC sequence of these RTRs. Upon reception of a packet, the RTR decapsulates the packet, looks up the mapping, and re-encapsulates the packet to the next hop in the LISP overlay.

LISP NAT traversal [6] makes xTRs behind a NAT reachable in the Internet, which is especially useful for MNs. A NAT traversal router (NTR) facilitates that function. If an xTR behind a NAT attempts to register at the mapping system, it receives a response with a list of NTRs. It then registers at an
The NTR adds the RLOC of the xTR to its NAT table. Furthermore, it registers the xTR's EIDs with its own RLOC at the mapping system.

The LISP canonical address format (LCAF) enables more advanced TE by extending mappings with context information. E.g., the RLOC of an EID may depend on the specific service or traffic class of the packet so that voice and data traffic is tunneled by xTRs over different paths.

LISP is currently extended with features for security and hybrid access, there are multiple use cases for the MN concept in the context of datacenters and multihoming, and the LISP control plane may be applied to other data planes.

### III. RELATED WORK

OpenLISP [7] is an early open source implementation whose data plane is implemented in the kernel of FreeBSD and its control plane in the user space of FreeBSD and Linux. The limitation of the data plane to FreeBSD [8] makes OpenLISP hard to use. Code for the kernel stack is neither easy to read nor a suitable base for fast prototyping of extensions. The control plane code for Linux is written in C, also hard to read, and documentation exists only for the usage of the program. Thus, also the control plane is hard to extend.

Open Overlay Router [9] (OOR), the successor of lisp-Mob [10], is an open source implementation that uses LISP or VXLAN [11]/GRE [12] as an overlay for SDN. Its focus is support for network function virtualization (NFV), e.g., by SFC, and integration with OpenDaylight [13]. OOR is available for Linux, Android (only on rooted devices) and OpenWRT. The extended feature set makes OOR attractive for application, but bloats the code which is written in C. This is an obstacle for developers who want to implement their own features in the code base.

Lispers.net [14] is a Python implementation aiming to implement the complete feature set of LISP. Additional non-LISP-specific functions and behaviors are added which makes lispers.net a general overlay controller that focuses on bleeding edge LISP technology. As the source code of lispers.net is proprietary, it cannot be used for own extensions.

There are several other proprietary LISP implementations. The implementations on Cisco routers and on AMV's FRITZ!Box home routers are most widely spread. Both can be used to test LISP and to connect with the beta network. However, they are not open source and cannot extended for own purposes.

### IV. jLISP ARCHITECTURE

jLISP is an easily extensible and portable open source implementation of LISP. It is implemented in Java and runs in the user space. We chose Java as programming language for platform independence. The code is object-oriented and modular which facilitates readability, reusability, and extensibility. We describe the architecture of jLISP, comment on supported features and a plugin mechanism, and report some performance results.

#### A. Architecture

jLISP is split into three different, independent modules which are realized as standalone Java packages. Figure 2 depicts these modules: the data plane, the control plane, and general networking. They are used for the implementation of xTRs, KTRs, NTRs, the mapping system, and in particular for novel LISP-based applications.

![Fig. 2: jLISP modules](image-url)
The data plane module contains classes to encapsulate payload with a LISP header and decapsulate it while extracting header information. An RTR implementation leverages the control plane, the data plane module, a datagram socket, and some additional logic. An RTR receives LISP-encapsulated traffic, reads one header field, performs Map Requests, and sends LISP-encapsulated traffic to the next overlay hop.

The general networking module contains classes to build and parse transport protocol headers as well as IPv4 and IPv6 headers. The xTR uses this module for interpreting non-LISP traffic to retrieve destination EIDs from IP packets. LCAF implements conditional forwarding and requires Layer 4 information, e.g., port numbers which can be obtained from packets with classes from this module.

A central feature of jLISP is the simple construction of objects for control plane, data plane, or general networking packets. They may be either constructed from parameters or parsed from a byte stream of a packet yielding the packet data. This serialization of packet objects into packet byte streams and deserialization of packet byte streams into packet objects enables a simple handling of packets for application programmers without dealing with low-level programming. This is a significant advantage compared to existing implementations and important for extensibility and rapid prototyping. Therefore, this framework allows to write lightweight tools and LISP applications with only little knowledge of the full source code of jLISP. The control plane module is already used in our practical networking course [15] to let students write a simple variant of the LIG [16].

jLISP is a user space program and does not require modifications of the operating system. Therefore, jLISP can be run on devices without special privileges and thereby avoids some security concerns. Technically, jLISP provides a tun device over which all raw IP traffic with EID source addresses is forwarded to the xTR application. The xTR application receives raw IP traffic from the tun device, encapsulates it, and forwards it to a datagram socket, or receives LISP traffic from a datagram socket, decapsulates it, and forwards it to the tun device. In fact, one datagram socket is used for LISP data traffic and another for LISP control traffic. As a potential modification, the tun interface may be swapped by a tap interface which offers control of Layer 2 traffic in the network. This facilitates, e.g., a simple extension of xTRs to ARP proxies to connect remote LISP domains to one VPN. The drawback is the need for parsing and constructing Layer 2 headers of the traffic which requires more computing effort. This architectural base enables porting jLISP to any platform like Android which may be used as substitute for a tun/tap device. In fact, one datagram socket is used for LISP data traffic and another for LISP control traffic. As a potential modification, the tun interface may be swapped by a tap interface which offers control of Layer 2 traffic in the network. This facilitates, e.g., a simple extension of xTRs to ARP proxies to connect remote LISP domains to one VPN. The drawback is the need for parsing and constructing Layer 2 headers of the traffic which requires more computing effort. This architectural base enables porting jLISP to any platform like Android which may be used as substitute for a tun/tap device.

B. Features

jLISP is compatible with the LISP RFCs. Based on the presented modules, jLISP provides explicit programs for all LISP components: xTRs, RTRs, NTRs, and the mapping system.

The xTR component is split into an ingress tunnel router and egress tunnel router (I/TR/ETR). The ITR receives raw traffic and LISP-encapsulates it before forwarding. The ETR receives LISP-encapsulated traffic and decapsulates it. The xTR may also re-encapsulate packets and provide RTR functionality by calling the encapsulation routine after decapsulation instead of forwarding the raw traffic.

A MN is built on the base of the xTR. The xTR is equipped with an EID on its LISP tun interface. This address is the sole prefix this xTR is responsible for and the xTR registers that EID with the mapping system whenever it receives a new RLOC which is the address of the external network interface.

The mapping system uses hash maps to store register messages for EID prefixes. They are retrieved with an algorithm for longest prefix match. Since the entire register message is saved, this structure supports new LISP register formats by design as the information is stored as opaque data. Therefore, the mapping system can store both normal EID-to-RLOC mappings and more complex EID-to-LISP-TE-path mappings containing as list of RTRs, and return them on request. The storage backend of the mapping system can be replaced by another that provides a class with a store and request method for mappings. Normally, the mapping system is filled with Map Register messages from an xTR. We also provided a non-standardized interface to allow third-party controllers to fill the mapping system with mappings and program a network. This can be used for TE experiments and integration with third-party controllers. Potential use cases are NFV and SFC applications.

jLISP also provides an NTR which is a modified xTR with a NAT and some additional logic. We provide two components: the one implementing the current Internet draft [6] and a modified version. This modification reduces the computation load on the NTR. In the current draft, the NTR interprets the Message Register message from the registering node and sends a new one to the mapping system. We proposed that a registering nodes sends an encapsulated Map Register message to the NTR, which decapsulates and forwards it to the mapping system. Furthermore, our modification especially ensures that communication behind a provider NAT still works if a flow sends packets infrequently. To that end, the client sends empty keep-alive messages to the NTR which prevents that the connection between client and NTR is deleted from the NAT table.

C. Plugins

jLISP offers a plugin mechanism to improve extensibility. For control plane traffic, jLISP provides hooks to intercept control messages before they are sent and after they are received which allows plugins to modify them. For control plane traffic, raw traffic may be intercepted before being received by the ITR or sent by the ETR, and encapsulated traffic before being sent by the ITR or received by the ETR. This enables a developer with only little knowledge of jLISP to intercept packets at any stage of the normal LISP pipeline.

D. Throughput and Load Measurements

We tested the implementation on commodity laptops on a 100 Mb/s LAN and could use the full bandwidth. The load was handled by a single core of a mobile CPU, but jLISP is able to spread it across up to 100 CPU cores if needed. This is achieved with two thread pools that are filled with up to 50 worker threads to process incoming and outgoing traffic.
V. Demo

The demonstration illustrates LISP communication between a MN behind a NAT and a node in a LISP domain using jLISP components.

We use a semi-virtualized testbed which is visualized in Figure 3. A testbed server hosts two LISP domains with nodes and xTRs realized as virtual machines (VMs). It further hosts a mapping system and an NTR. The MN runs on an Android machine. An OpenWrt [17] router with integrated NAT is used as access point (AP) and connects the smartphone with the server.

![Virtual Testbed Server](image)

Fig. 3: Testbed setup

To be able to use hardware-accelerated virtualization on the Intel x86 platform used in the testbed server, some extensions are needed as the architecture itself is not virtualizable. Intel VT-x [18] enables basic hardware acceleration on that platform. To actually use these hardware features, the hypervisor that runs the VMs has to support them. We use KVM [19] as hypervisor, which is part of the Linux kernel, with QEMU [20] as virtualizer. The server itself is set up with an Ubuntu 15.10 [21] operating system. The VMs are managed with the libvirt [22] framework as frontend. As a result, we run multiple VMs per host with a performance close to a dedicated physical machine. The connection between the VMs and the physical interface of the testbed connected to the access point is realized by an Open vSwitch [23]. The demo shows a MN exchanging traffic with a node in one of the LISP domains. The MN first attempts to register itself at the mapping system but does not receive a Map Notify message confirming it registration. Instead, the MN receives a notification that it is located behind a NAT including a list of available NTRs. Then, the alternative NAT traversal proposed in Section IV is applied. The MN sends an encapsulated Map Register message to the NTR. The NTR decapsulates it, adds an entry to its NAT table, and forwards the Map Register to the mapping system. After the MN receives a Map Notify from the mapping system, it starts sending packets to the node in the LISP domain. The first packet is sent to the NTR. The NTR faces a cache miss, requests the RLOC for the destination EID from the mapping system, and forwards the LISP-encapsulated packet to the xTR of the LISP domain. The xTR delivers the packet to the destination. The destination responds, the packet is relayed to the xTR which also faces a cache miss, requests the RLOC for the EID from the MN using the mapping system, receives the RLOC of the NTR, and forwards the packet to the NTR. After reception of the packet, the NTR consults its NAT table for the MN’s EID, and forwards the packet to the MN.

We demonstrate the extensibility, we provide a simple statistics plugin running on any node. It reports the number of sent packets to the master node while packets are exchanged. The master node aggregates and presents the data.

VI. Conclusion

We presented jLISP as a novel implementation of LISP which excels by a highly extensible architecture (modularity, object-orientation, plugin mechanism), and offers itself for rapid prototyping. jLISP is sufficiently fast and offers parallel processing if needed. It is platform-independent, runs in the user space, and supports all features of the currently standardized LISP protocol. The demo leverages jLISP and runs on a semi-virtualized testbed. It illustrates how a Mobile Node behind a NAT communicates with the help of NAT traversal with a node in a LISP domain. The NAT traversal implements an improvement of the current Internet draft. A simple statistics application illustrates jLISP’s plugin mechanism.

References


208
Network as a Service - A Demo on 5G Network Slicing

Rastin Pries*, Hans-Jochen Morper*, Nandor Galambosi†, Michael Jarschel*

* Nokia Bell Labs, Munich, Germany.
† Nokia Bell Labs, Budapest, Hungary.

Email: {rastin.pries,hans-jochen.morper,michael.jarschel}@nokia-bell-labs.com

Abstract—5G mobile networks have to support a huge number of different service types such as health, automotive, logistics, energy, and public safety. A one-size-fits-all solution like it exists today that delivers all the services to every device everywhere in the network is not a viable option. Network slicing allows to offer programmable network instances, which match the requirements of the individual use cases, subscriber types, and applications. Within a slice, network functions and elements can be instantiated according to the specific service demands such as ultra-low latency, massive IoT, or dense broadband. This demo presents 5G network slicing using the example of a health insurance provider. The health service includes the use of a smart wearable for the customer with a specific traffic pattern. A tailored-to-fit network slice enables the health provider to connect these smart devices efficiently to the cloud.

Keywords—network slicing; orchestration; NFV; VNF; cloud

I. INTRODUCTION

Currently, all different mobile devices are using one single mobile core network - regardless of their demands. While this works very well today, mobile operators will face serious problems in 5G networks due to the different demands and device types as well as the increasing number of devices [1]. For example, current mobile networks are able to handle 10s of users with a few Mbps, but are not able to handle 100,000 devices in a single cell transmitting with a few kbps. Thus, future mobile networks will have to be able to support at least the following four different demands:

- mobile broadband
- ultra-low latency
- dense broadband
- massive connectivity

Here, mobile broadband stands for the normal traffic from smartphones, tablets, etc. In contrast, autonomous driving, tele-medicine or certain types of industrial communication have high demands on the latency. Furthermore, during a mega event, e.g., concert or sports event, high bandwidth has to be provided in addition to caching servers and edge computing capabilities. Finally, Internet of Things (IoT) devices are popping up everywhere. These devices have little bandwidth and mobility requirements but the mobile network has to cope with a huge number of devices. Having just one mobile network architecture for coping with all these different demands will not suffice in future 5G networks. This is where network slicing comes into play. Using network slicing, multiple independent and dedicated network instances can be created within the same infrastructure to run services that have completely different requirements for latency, reliability, throughput, and mobility.

In the next section, we introduce network slicing in detail and present the challenges. Section III describes our 5G network slicing demo and finally, we draw conclusions and give a brief outlook on future work in Section IV.

II. NETWORK SLICING

The concept of network slicing dates back to the early 1990s, when virtual connections could be established using Asynchronous Transfer Mode (ATM). Newer approaches already define virtual network controllers, resource managers, and virtualized physical resources [2] [3]. A prerequisite for network slicing is the virtualization of the different network elements of the mobile network. Thus, network slicing gained momentum with the setup of the ETSI Network Functions Virtualization (NFV) group [4].

According to NGMN [5], a network slice, namely “5G slice”, supports the communication service of a particular connection type with a specific way of handling the C- and U-plane for this service and only provides the traffic treatment that is necessary for the use case, avoiding all other unnecessary functionality. Thus, with the combination of NFV and Software Defined Networking (SDN), the mobile network can be instantiated on demand.

Considering an IoT network slice, the slice has to support on the one hand a lot of C-plane traffic but the U-plane traffic is generally quite low and the mobility management can be simplified as IoT devices are often fixed. In addition to the optimized mobile core, the Radio Access Network (RAN) can also be optimized for the specific use case. To be able to support millions of IoT devices and to not waste the limited energy resources of the devices, the signaling overhead on the air interface can be reduced to a minimum without harming any standards and the devices can also transmit in another frequency band, e.g. narrow band [6].

Thus, network slicing is a key enabler for network operators to expand existing businesses and creating new ones. Slices can be offered to third-parties such as media, automotive, health, and public safety via a suitable API for providing Network as a Service (NaaS). Figure 1 shows the concept of network slicing starting on the
left with the assignment of devices to network slices, the service optimized architectures, and the service and network orchestrator for mapping the different service and network demands.

There are currently three main challenges for the introduction of network slicing. The first challenge is, how to assign the devices to the network slice. If one device can only be assigned to one slice, this can be handled by the IMSI. However, if each service of a device can use different network slices, the assignment has to be handled via service identifiers and signaled via the control channel. The second challenge is the isolation of the slices. This is especially challenging in the eNodeBs. Finally, the estimation of the required resources for a service to be able to guarantee certain SLAs while not wasting resources is quite ambitious.

III. DEMONSTRATION

Our showcase is a health insurance provider who requests a network slice for offering individual services for the customers. The so-called H&F provider advertises a health insurance together with a smart watch. The smart watch, as an IoT device, allows for online training with direct professional feedback to the user based on submitted vital parameters. In return, the service provider gets an overview about the health condition of his customers allowing to adjust the tariffs accordingly and can help the customer in case of an emergency.

Within the demo, we illustrate network slicing for this use case from three different angles:

- Operators perspective
- Third party service providers view, in our example the health insurance provider
- End user with the smart watch

We show how to configure and order a network slice as a third party provider. Afterwards, we illustrate from the operators point of view, how the functional elements of the slice are set up and configured using a Network Orchestrator, cf. Figure 2, implemented by Nokia Bell Labs and Nokias Cloud Application Manager (CAM) [7]. The network orchestrator is responsible for evaluating the required number of network elements based on the parameters entered in the ordering portal. In our example, these are two edge gateways, one Internet gateway, and one controller instance shown in the cloud in Figure 2. The four radio nodes are physical instances. Although it is possible to include also the radio access, we focus on the setup of the mobile core network in our demo. The screenshot of the CAM, cf. Figure 3, illustrates the setup of the virtual Internet gateway based on OpenStack. The CAM is responsible for dynamically deploying, interconnecting, monitoring, and configuring virtual network instances.

The functional elements are set up in the cloud and the network is automatically configured according to the selected setup of the health insurance provider. Finally, we illustrate the communication via a network slice with the smart watch. The end user can monitor its vital parameters and gets an individual training. The health insurance provider instead, gets detailed statistics about the network slice, including the number of active end users and network statistics. From the operators perspective, the demo illustrates how easily a network slice can be set up on demand according to the customer needs and how slicing can help to reduce the TCO.
IV. CONCLUSION

The demo illustrates that it is possible to create a 5G network slice automatically within a few minutes, with individual statistics and billing. We show that network slicing allows to create an individual programmable network architecture for thousands of use cases, subscriber types, and apps. Thus, network slicing can be used by network operators to provide Network as a Service (NaaS), which gives new opportunities for different verticals. In future, we will extend the network slicing concept to include also the radio access. We will further split the network elements into services and integrate the location awareness of these.

REFERENCES


Automated Establishment of a Secured Network for Providing a Distributed Container Cluster

Marcel Großmann, Andreas Eiermann
Faculty of Information Systems and Applied Computer Science
Otto-Friedrich University,
Bamberg, Germany
Email: marcel.grossmann@uni-bamberg.de, andreas@hypriot.com

Abstract—Modern service providers use virtualization in order to feasibly scale their applications. Their approaches exploit virtual machines to master the quality of service requirements, primarily redundancy, reliability and security. Accordingly, the services provided are hosted in data centers that fulfill the customers’ needs. The standards of entry burden small enterprises by requiring them to establish their own distributed service delivery cloud. Therefore, we developed an energy-efficient solution, Hypriot Cluster Lab (HCL), that adapts Docker technologies in order to run on ARM powered single board computers. Herein, we demonstrate how HCL works with redundancy and replication between several hosts on different locations via wide area networks. Accordingly, we use a mesh network based on virtual private LANs to enable encrypted communication over the Internet between distributed hosts. Our result presents a fault tolerant, reliable and secure extension to connect several independent hosts with HCL to achieve the QoS-requirements.

Keywords—Container Virtualization; Cluster Computing; Distributed Computing; Mesh Networking; Privacy; Reliability; Redundancy; QoS;

I. INTRODUCTION

History has demonstrated that new technologies embed themselves in new areas of application in ways that complement existing infrastructures. For example, personal computers did not disestablish mainframes, but became a standard accessory on desktops for accessing the functional features of mainframes. Nowadays, single board computers (SBC) show similar behavior as they become established within the computer market as energy-efficient and inexpensive devices. They allow vendors to build manifold appliances that are part of the Internet of Things (IoT). However, more devices in the IoT endorse unnecessary data communications that burden core networks and data centers in the cloud. Moreover, requesting services from the cloud “creates high latency to application, congestion and bottleneck to the network” [1]. As proposed by Aazam and Huh [2], fog computing is one possible solution for relocating virtualized services from the cloud to the edge network.

To empower edge nodes, SBCs are a perfect complement to relieve core networks of unnecessary traffic between IoT devices and centralized services in the cloud. Each SBC is able to aggregate, filter, and analyze data from connected IoT devices and pass only the relevant information to cloud services. Therefore, virtualized functionality of the data center is shifted to the edge nodes, thereby reducing latency times of IoT devices that request services in the cloud. Fog computing is based on the same paradigm as cloud computing, virtualization, which enables it to fluently migrate services.

Setting up SBCs at the edges of a network changes the way virtualized services are provided because they are unable to run virtual machines (VM) for services in a similar manner to the cloud. However, lightweight solutions already exist. For example, LINUX Container (LXC) behaves similar to a VM, but is challenging to use. Felter et al. [3] explain that containers are very beneficial, especially for small SBCs, because they “offer the control and isolation of VMs with the performance of bare metal” [3]. Fortunately, a more user-friendly option, Docker, was programmed three years ago.

Direct comparison of VMs and container virtualization in Figure 1 illustrate differences between the approaches. Docker containers are more lightweight compared to VMs without a guest operating system (OS), but are restricted to the functionality of the host OS. For example, a container built on Windows is not able to run a Linux host. Nevertheless, Docker packages a piece of software and its dependencies in a filesystem called image, which contains everything the application requires for execution. Each image’s filesystem consists of layers that track various operations, similar to a versioning tool. This is beneficial if several containers use the same image because all layers of this image are started once, and for each container only the difference is loaded into its container environment.

Former cluster solutions that come with security features are primarily built for services hosted on VMs. For example,
Eucalyptus [4] uses VTun interfaces to connect various cluster front-ends via a tunnel, so that VM instances are connected by bridging their virtual interfaces to the VTun one. In contrast, the educational cloud computing platform called Seattle [5] offers a lightweight virtualization approach based on sandboxes that run inside Vessels and can be installed on ARM-powered devices. However, Seattle is only programmable with a subset of the Python programming language and is therefore not an alternative for the delivery of existing services.

Upon summing up the former developments, this paper outlines Dockers’ (version \( \geq 1.10 \)) and SBCs’ influence on fog computing. We demonstrate how to establish a distributed and secured cluster setup on the Raspberry Pi (RPi) 3 Model B by distributing them at different locations. It is the basis for making vision fog computing a reality.

In Section II we describe our setup for the demonstration infrastructure. Subsection II-A focuses on the establishment of a secure connection, followed by the initialization of a Docker cluster in Subsection II-B. The security for the proposed configuration is evaluated briefly in Section III. Subsequently, Docker-specific security features are compared to this approach in Subsection III-A. Current restrictions are then shown in Subsection III-B. The paper is concluded with Section IV.

II. FOG COMPUTING WITH DOCKER

A. Initialization of a Distributed Mesh Infrastructure

Private transmissions over the Internet call for Virtual Private Networks (VPN), which use tunneling and encryption to establish a secure connection between two hosts attached to the Internet. Traditionally, VPNs are created between two endpoints and additional tunnels are added for larger networks with more sites. According to Khanvilkar and Khokhar [6], an open source Linux based VPN solution, tinc\(^1\), offers all four VPN security aspects: confidentiality, data integrity, authentication, and anti-replay. Moreover, it provides a built-in routing daemon that configures a full mesh network and eases the establishment of single-domain VPNs. Therefore, only endpoints are specified and tinc itself creates tunnels, thereby allowing an easier configuration and improved scalability [7].

First, for each tinc endpoint, several configuration files need to be generated that contain

- the name of the private network with its corresponding tinc-up and tinc-down scripts, to configure the host’s network interface,
- the configuration file of the host itself, and
- a generated RSA key pair.

To connect the mesh network, tinc configurations are exchanged and stored on each endpoint including each node’s public RSA key.

Our demonstration uses the tincregistrar\(^2\) framework to administrate nodes of a private network and to automate the initialization of the mesh, which is publicly available on

\(^1\)https://www.tinc-vpn.org/

\(^2\)https://github.com/whatever4711/tincregistrar.git

Github. It provides a script to generate the tinc configuration on each host and communicates with a Django\(^3\) application to store and distribute endpoint configurations to ease the exchange of the necessary files. The server application runs inside a Docker container and can therefore start on each Docker-enabled Linux host. The basic mechanism of the communication between the registrar and one endpoint, cf. Figure 2:

1) With POST an endpoint’s tinc configuration is sent to the server and stored there.
2) By using GET, tinc configurations are retrieved for all registered endpoints.
3) DELETE removes endpoints from registered nodes.

After announcing its own configuration to the registrar, an endpoint performs a loop until it receives the configuration from all attached nodes. Within the receipt, the tinc daemon is started and data communication is possible within the tunneled mesh network.

For our demonstration, the mesh infrastructure is setup as shown in Figure 3, where each node is connected to the Internet and reaches a tincregistrar instance without being restricted by network address translation (NAT) or a firewall. Though tinc offers session traversal utilities for NAT (STUN) protocols to circumvent restrictions, our ongoing work examines this functionality.

B. Running Hypriot Cluster Lab on the Distributed Mesh

Hypriot Cluster Lab (HCL) [8] was initially designed to be void of a configuration, while running it on a local cluster

\(^3\)https://www.djangoproject.com/
of RPis and performing all necessary steps to configure the cluster to communicate in a virtual LAN (VLAN) with its own dynamic host configuration protocol (DHCP) server. With tinc, it is unnecessary to create an additional VLAN and start a DHCP service. Thus, HCL only operates on the infrastructure generated by tinc. The reduced set of activities required for running HCL is depicted in Figure 4.

Leaving out the address configuration, HCL boots up and uses the tunnel interface directly. Thereafter, it re-configures the Docker engine to advertise the functionality of Docker Swarm⁴ and initiates a Consul⁵ container on a leader node. This node is a key-value (KV) store with a built-in DNS server that can easily discover and track the health of various services throughout an infrastructure. Subsequently, all other nodes join the existing Consul service by starting their own Consul containers. Similarly, all nodes connect with a Docker Swarm⁴ container and start another instance as a manager or replica of the cluster. Docker-Compose⁴, Consul⁵, Docker Swarm⁴ and other suitable tools can all be used to run services on the configured cluster. However, concerns may arise from the architecture of Docker images when running them on ARM machines.

III. Evaluation of Demonstration Approach

A. Existing Docker Security Features vs. Tinc

In order to communicate with Docker in a safe manner, transport layer security (TLS) can be enabled so that the daemon only allows connections from authenticated clients with a certificate signed by a certificate authority (CA). The Consul service also offers a TLS encryption of all of its network traffic. It is setup by establishing an internal CA and distributing all necessary certificates and key files among all connected Consul hosts. By enabling all provided security features, the control communication for Swarm and its necessary KV store are secured by TLS.

⁴https://docs.docker.com/
⁵https://github.com/hashicorp/consul

In conclusion, the establishment of natively integrated security features requires two adjustments and is therefore painful to organize and concerns only the control mechanisms of a Swarm. In contrast, our approach avoids provided security features and their inability to secure the inter-container communication. In fact, secured connections are established by tinc before the HCL cluster is booted up. As proof, all communications of a Swarm over a tinc connection are analyzed next. We start a HCL cluster on two nodes and reconfigure it on both nodes. Before changing the configuration of HCL, we stop it with:

VERBOSE=true cluster-lab stop

Modifying the following configuration parameter in /etc/cluster-lab/cluster.conf instructs HCL to disable VLAN and DHCP, and operate on tinc’s tun0 interface.

INTERFACE="tun0"
ENABLE_VLAN="false"
ENABLE_DHCP="false"

Additionally, the avahi configuration file /etc/avahi/avahi-daemon.conf needs to allow point-to-point discovery (allow-point-to-point=yes), and a restart of the service to activate this configuration. Then, we start HCL with:

VERBOSE=true cluster-lab start

On the first node, we execute these commands to create an overlay network called ‘wireshark’ and start a webserver named ‘web’.
docker network create -d overlay \
--ip-range=172.29.254.0/24 \
--subnet 172.29.254.0/24 wireshark

docker run -it --rm --name web -v \
/var/run/docker.sock:/var/run/docker.sock \
--net wireshark firecyberice/whalesay:web

On the second node, tcpdump is initiated in order to capture the payload of tinc on UDP port 655 and UDP port 4789. The payload is used by Docker for its Virtual Extensible LAN (VXLAN) overlay networks. Afterwards, a web call is made against the webserver on the first node. The ‘web’ can be resolved in the ‘wireshark’ overlay network by the internal Domain Name System (DNS) of Docker.

docker run -it --rm --net wireshark \
aplince:3.2 /bin/sh -c "apk add \
--update curl & & curl -I "
'http://web:5000/message/Hello'"

The captured results are:

```
#vxlan
tcpdump: listening on eth1 [...]  
0 packets captured [...]  
#tinc
tcpdump: listening on eth1 [...]  
5972 packets captured [...]  
#vxlan
tcpdump: listening on tun0 [...]  
12 packets captured [...] 
```

In conclusion, packages are traversing the tun0 device and not the eth1 device, as the traffic is already encrypted by tinc. The number of tinc packets is far greater compared to those of VXLAN on tun0. This arises because the only VXLAN traffic are the HTTP request and response, while the tinc traffic contains additional control messages for Swarm and Consul. These results affirm that our approach boots up a VPN-based mesh network between the nodes Swarm is running on and secures the control channel. Even inter-container communication traverses the Internet in an encrypted manner that is impossible by adopting Docker’s and Consul’s security mechanisms.

B. Current Restrictions

The tincregistrar from section II-A is currently proof-of-concept for how a tinc network could be initialized by using a central instance for managing the information of each node in the private network. At the moment, it does not provide the necessary security for the control channel, but it is on our future work agenda.

IV. CONCLUSION & FUTURE WORK

In our demonstration we spawned a private, secured, and geographically distributed cluster to run binned services with a secured link. This allows connection of Docker Swarm instances over the Internet with no traffic leaving the secured tinc mesh network. Moreover, we contribute open source tools to easily setup a distributed HCL in a fog environment.

In our ongoing work, we focus on securing the tincregistrar framework. Until now, it only supported unencrypted control traffic for registering and communicating with the endpoints of a tinc mesh network.

Future extensions could include frameworks that support Docker and are able to manage containers inside a cluster, such as Kubernetes or Apache Mesos.

REFERENCES


\footnote{http://kubernetes.io/}

\footnote{http://mesos.apache.org/}
Sector: TCAM Space Aware Routing on SDN

Sai Qian Zhang, Qi Zhang, Ali Tizghadam, Byungchul Park, Hadi Bannazadeh, Raouf Boutaba and Alberto Leon-Garcia

Abstract—In Software Defined Networking (SDN), fine-grained control over individual flow can be achieved by installing appropriate forwarding rules in switches and routers. This allows the network to realize a wide variety of functionalities and objectives. But at the same time, this flexibility and versatility come at the expense of (1) a huge burden on the limited Ternary Content Addressable Memory (TCAM) space, and (2) limited scalability due to the large number of forwarding rules handled by the controller. To address these limitations, we present Sector, a switch memory-aware routing scheme that reduces TCAM space usage without introducing network congestion. We consider static and dynamic versions and propose corresponding solution algorithms. Experiments show our algorithms can reduce TCAM space usage and network control traffic by 20% – 80% compared with the benchmark algorithms on different network topologies.

I. INTRODUCTION

Software Defined Networking (SDN) is an architecture that enables logically centralized control over distributed network resources. In SDN, a centralized controller makes forwarding decisions on behalf of the network forwarding elements (e.g switches and routers) using a set of policies. Based on given high level design requirements, the source and the destination node of each flow is dictated by the Endpoint Policy and the flow path is decided by the Routing Policy [1]. For example, the shortest-path routing policy asks the network to forward packets along the shortest path between two nodes. Other routing policies, such as the ones that improve resource utilization, quality of service and energy usage have also been proposed in the literature [2,3,4]. These features make SDN an attractive approach for realizing a wide variety of networking features and functionalities.

Despite its benefits, however, implementing routing policies in SDN may require fine-grained control over flows, which can place a huge burden on switch memory space. In particular, the Ternary Content Addressable Memory (TCAM) is a special type of high speed memory that can search the entire memory space within a single clock cycle. However, it is also well known to have limited capacity and high power consumption [5]. The largest memory space on a TCAM chip is far less than that of Binary Content Addressable Memory (CAM). For example, HP ProCurve 5406zl TCAM switch hardware can support 1500 OpenFlow rules. As each host requires dozens of OpenFlow rules on average, a 5406zl switch can only support at most 150 users [6]. Moreover, TCAM is also energy-hungry. It consumes 30 times as much as the consumed energy of SRAM with the equal number of entries [7]. Given that the amount of power consumption is proportional to the number of used entries in TCAM, several research studies have focused on reducing the TCAM space consumption [1,5,8] in order to improve scalability and reduce energy consumption.

Another scalability problem arise in the centralized SDN controller. For every subtle change on the network topology or routing policy, the controller must deliver the control message to each network element that implements the policy. As the average flow size in both wide-area and data center networks is small (around 20 packets per flow [6]) and the inter-arrival rate of the flows in the high-performance network is extremely high (less than 30ms [6]), a huge workload is imposed on the controller as the network size grows. Since each switch typically has limited bandwidth on the path to the controller, and moderate rate insertion time, the high workload received by the controller often causes large rule installation overhead and low flow set up rate. In the modern data center networks, 1ms additional latency for the delay-sensitive flow can be intolerable [9]. Therefore, the limited flow set up rate can dramatically hurt the overall performance and the quality of service. It is important to reduce the interaction between control plane and data plane in order to achieve better network scalability and performance.

To address the issues of switch memory space limitation and scalability, recent work proposed to control flows collectively at an aggregated level. This allows the use of prefix aggregation and wild card rules to minimize the number of stored entries [1][8]. These works have focused on compressing the entries of each individual switch, while preserving the routing policy (i.e. without changing the forwarding paths) [5]. However, we find that in large networks, typically multiple candidate paths are available for routing each individual flow while still satisfying performance and business constraints. Therefore, if we can additionally control the flow forwarding paths, we achieve substantial gain in term of TCAM space savings and controller scalability. To this end, we present Sector, a routing scheme that minimizes TCAM space consumption in SDN networks without causing network congestion Sector takes advantages of the large number of available forwarding paths and routes traffic in a way that improves network scalability and reliability. The main objectives of Sector are (1) Minimizing the switch memory space utilization given the end point connection request, and (2) Reducing control traffic by decreasing the interaction between the controller and network.

In this paper, we first introduce the TCAM space minimization problem and analyze its complexity. We then propose heuristic algorithms for both static and dynamic versions of the problem. Through experiments, we show our algorithms can reduce the TCAM space usage and network control traffic by 20% – 80% compared with the benchmark algorithms.

The rest of paper is organized as follows. Section II surveys related work. Section III provides a motivating example of the TCAM space minimization problem. Section IV presents the
problem overview. Section V and VI presents our solutions for the static version of the problem. Section VII presents our solution for the dynamic version of the problem. After presenting experimental results and testbed implementation in Section VIII and IX, we conclude the paper in Section X.

II. BACKGROUND AND RELATED WORK

OpenFlow is the most popular implementation of SDN [10]. An OpenFlow table entry can be represented by a triplet $(M, P, A)$, where $M$ is the matching field used to match the packet, $P$ is the matching precedence of the entry and $A$ is the action field which contains operations on the matched packet. The matching field usually includes source and destination IP addresses, MAC addresses and input port number. The action field includes operations such as forwarding the packet to a output port or modifying the packet header. Upon receiving a packet, the switch searches for the rule with the highest priority that matches the packet, then executes corresponding actions defined by that rule. OpenFlow also supports wildcards in the input port as well as subnet masks in IP and MAC address [10].

There are several studies on minimizing TCAM space using subnet masks and wildcard rules. However, prior work has focused on compressing the entries of a single switch, while preserving the routing policy [5]. One Big Switch [1] and Palette [8] decompose network access policies into small pieces and then distribute them using less TCAM space. Mosrur et al. [11] designs routing algorithms to distribute access policies across intermediate switches with minimum switch memory consumption in a datacenter network. Rami et al. [24] studies the effect of flow table size on the maximum number of flows supported. CacheFlow [25] develops a algorithm for placing rules in a TCAM with a limited space.

Scalability has been a key issue in SDN. DevoFlow [6] proposes a scalable SDN framework by using wildcard entries to reduce the control plane visibility on the microflows. However, it does not offer any quantitative analysis on how to use the wildcard to achieve optimal performance. DIFANE and Kandoo [12,13] propose efficient and scalable SDN frameworks which split the workload of the central controller to distributed authorized components. However, they do not address the problem of global visibility. In [14,15], the scalability issue is solved by using multiple independent controllers to consistently manage the network with minimal interaction, but they do not mention how these controllers are coordinated and the overhead brought by distributed control.

III. A MOTIVATING EXAMPLE

We provide a motivating example to demonstrate the benefit of Sector. The topology and port numbers between nodes are shown in Figure 1(a) and the end point policy is illustrated in Figure 2(a). Two source hosts with the IP address 000 and 001 send traffic to two destination hosts 10C and 10D respectively. We call a group of source and destination address a demand pair, therefore there are four demand pairs in this example. The bandwidth consumption of each demand pair equals 1 and the capacity of each link is 10. Traditional traffic engineering (e.g. ECMP) spreads the flows evenly in the network to balance network link utilization, which gives one of the feasible solutions in Figure 1(b). The OpenFlow table of each switch is shown in Figure 2(b). A total of 11 entries is installed. To set up these new flows, 11 additional control packets are sent from the controller, as at least one initial packet in each new flow is processed by the controller. In total $11 + 2 \times 4 = 19$ packets are transmitted between controller and the switches. By comparison, Sector produces the solution in Figure 1(c) and forwarding tables shown in Figure 2(c). Instead of routing the traffic of each demand pair respectively, Sector aggregates the flows and uses subnet masks to reduce the number of entries in each table. The maximum bandwidth consumption in the Sector solution is also 2, and 8 additional entries are installed on nodes A – F, which requires 8 control packets sent from controller. A total $8 + 2 \times 4 = 16$ packets are transmitted between controller and switches. This reduces TCAM space and control traffic by 27.2% and 15.8% respectively. From the above example, we draw the following conclusions: (1) If fine-grained control is not required on specific flows, TCAM space consumption and control traffic can be reduced by using subnet masks on the source and destination addresses to aggregate flow entries. (2) As the network size increases, the number of control packets to set up a flow is approximately equal to the number of entries installed in the TCAM (ignoring the initial packet of the flow send to the controller). So minimizing TCAM usage can also save control traffic indirectly. (3) Besides finding a path which minimizes TCAM consumption, the constraint on link capacity must also be guaranteed. For example, the two solutions above have the same maximum link utilization.

IV. PROBLEM OVERVIEW

The design objective of Sector is to minimize the total number of OpenFlow entries installed in all the switches. To keep the problem generic, we assign a weight $w(v)$ to each
switch \( v \) in the network, which is the cost of installing an additional rule in the switch. For the choice of \( u(v) \), in the simplest case, we can set \( u(v) = 1 \) to achieve the goal of minimizing total number of forwarding entries in the switches.

Moreover, adjusting the value of \( u(v) \) allows us to model other objectives. For instance, since power consumption of a switch is linearly proportional to the TCAM space usage [5], by setting \( u(v) \) to the average power consumption per rule for switch \( v \), we can model the problem of minimizing total energy consumption in the network. The objective is therefore to minimize the total weighted cost, given a set of demand pairs and the constraint on link resource utilization. We call this the TCAM Space Minimization Problem (TSMP).

TSMP is a rather complex problem to analyze and solve directly. To simplify our analysis, we divide TSMP into two sub problems Efficient Partitioning Problem (EPP) and Efficient Routing Problem (ERP). The EPP focuses on partitioning all the demand pairs into groups. We call these groups the routing groups. The source addresses and destination addresses in the same routing group have common prefixes. For example, the four demand pairs \((000, 100), (000, 101), (001, 100), (001, 101)\) in the Figure 2(a) form a routing group with prefix 0 \( \ast \ast \) and 1 \( \ast \ast \), where we use \( s_a, d_a \) to represent the demand pair. We can use the addresses with subnet mask \( s_a = 0 \ast \ast \) and \( d_a = 1 \ast \ast \) to represent all the source addresses and destination addresses in the routing group \( u \). When partitioning is complete, for each routing group there will be a ERP, where we route all demand pairs in that routing group to minimize total TCAM space usage.

In the next two sections we first discuss our algorithm for ERP, and then the solution for EPP, which relies on the solution algorithm for ERP to make partitioning decisions.

V. EFFICIENT ROUTING PROBLEM

The goal of ERP is to connect each demand pair for a given routing group that consume minimum weighted sum of switch memory space while satisfy the load balancing on links. Formally, we model the network as a graph \( G = (V, E) \), where each node \( v \in V \) represents an OpenFlow switch and each switch \( v \) is assigned a cost \( w(v) \) on per rule inserted. Without loss of generality, we assume each flow entry in the flow table can be represented by a 4-tuple \((s, i, d, j)\), where \( s, i, d \) constitute the matching field: \( s, d \) represent the source and destination address information such as source/destination IP/MAC address, \( i \) is the input port number of the switch where the packet comes in, \( j \) is the output port number of the switch where the packet is directed to, which constitutes the action field of the OpenFlow entry. We neglect rule priority temporally and consider it later.

Let \( U \) be the set of routing groups and \( K_u (u \in U) \) denote the set of demand pairs in \( u \). Let \( s_u \) and \( d_u \) denote the source and destination addresses of demand pair \( k \). We use a 4-tuple \((s_u, i, d_u, j)\) to represent the OpenFlow rule installed for the routing group \( u \in U \), where \( s_u \) and \( d_u \) are the source and destination addresses with the subnet masks respectively. Table I provides a quick glossary of definitions.

Let \( \pi(v) \) be the set of port numbers of switch \( v \), we make the port number equals to the label of the links that the port connects to (Figure 3). Then denote \( p(v) = \{(x, y) : x \in \pi(v), y \in \pi(v)\} \) as the set of port pairs of switch \( v \). For example \( p(A) \) in Figure 3 is \{1, 2, 4\} and \( p(A) = \{(1, 2), (2, 1), (1, 4), (4, 1), (1, 2), (2, 4), (4, 2), (1, 1), (2, 2), (4, 4)\} \). Let \( y_{ij} \in \{0, 1\} \) represent whether a 4-tuple is installed to direct the flow of \( k_u \) from input port \( i \) to output port \( j \). Let \( x_{ijk} \in \{0, 1\} \) denote whether a 4-tuple entry is installed to directed traffic of a demand pair \( k \in K_u \) from input port \( i \) to output port \( j \). Let \( \alpha(v) \) denote the total number of 4-tuples installed on switch \( v \). Our goal is to minimize the total weighted sum of rules installed in the switches

\[
\min \sum_{v \in V} w(v) \alpha(v)
\]

where \( \alpha(v) \) represents the number of 4-tuples \((s_u, i, d_u, j)\) installed in \( v \). To compute \( \alpha(v) \), note that for the same switch \( v \) and same routing groups, three conditions may happen:

1. No 4-tuple \((s_u, i, d_u, j)\) needs to be installed on \( v \). That is, \( \sum_{j \in \pi(v)} p_{s_u(i, d_u, j)} = 0 \) and therefore \( \alpha(v) = 0 \). Where \( p \) is the step function, \( p(x) = 0 \) if \( x \leq 0 \) and \( p(x) = 1 \) if \( x \geq 0 \).

2. All the flows installed on \( v \) are forwarded to one output port, i.e., \( \sum_{j \in \pi(v)} p_{s_u(i, d_u, j)} = 1 \). One entry \((s_u, i, d_u, j)\) is enough to direct the flows of \( K_u \), with \( s_u \) and \( d_u \) in the address field and wildcard in the input port field. Therefore \( \alpha(v) = 1 \).
3. All the flows installed on \(v\) are forwarded to more than one output port. That is, \(\sum_{j \in \pi(v)} \mu(\sum_{i \in \pi(v)} y_{ij}) > 1\). Therefore, the source and destination fields must be fully specified to differentiate each flow and so that the flows can be directed to corresponding output ports. Hence the total number of entries installed is \(\sum_{i \in \pi(v)} \sum_{j \in \pi(v)} \sum_{k \in K_u} x_{ijk}\), which is the number of demand pairs whose flows traverse through \(v\). This can be justified by the following example: Assume a set of rules \(\{0, 1, 10, 4\} \cup \{0, 1, 11, 3\} \cup \{0, 1, 2, 11, 6\}\) is installed on \(s1\). The flow table of \(s1\) shown in Figure 4. As the table shows, the source and destination address must be fully specified so that each flow can be identified by the intermediate switch to direct to its corresponding output port.

Combining these 3 cases, \(a(v)\) can be defined as follows:

\[
a(v) = \begin{cases} 
0 & \text{if } \sum_{j \in \pi(v)} \mu(\sum_{i \in \pi(v)} y_{ij}) = 0 \\
1 & \text{if } \sum_{j \in \pi(v)} \mu(\sum_{i \in \pi(v)} y_{ij}) = 1 \\
\sum_{i \in \pi(v)} \sum_{j \in \pi(v)} \sum_{k \in K_u} x_{ijk} & \text{if } \sum_{j \in \pi(v)} \mu(\sum_{i \in \pi(v)} y_{ij}) > 1
\end{cases}
\]

We also have to make sure that the number of rules installed in each switch does not exceed its TCAM space capacity, let \(r_v\) be the capacity of switch \(v\); we then have:

\[
a(v) \leq r_v \quad \forall v \in V \quad (2)
\]

Next, we relate \(x_{ijk}\) to \(y_{ij}\). Equation (3) ensures that 4-tuple rule \((s_u, i, d_u, j)\) is installed if any flow of demand pair \(k\) is sent from input port \(i\) to output port \(j\):

\[
\sum_{k \in K_u} x_{ijk} \leq y_{ij} \quad \forall (i, j) \in p(v), v \in V \quad (3)
\]

Next we build the path between each source host to the destination host. Let \(l_{vk}\) denote if edge \(e \in E\) is used to direct the flow of demand pair \(k\). Define \(Q_k = \{Q_k \subseteq V : s_k \in Q_k, d_k \not\in Q_k\} \forall (k \in K)\) and define \(\pi(Q_k)\), the set of edges in the cut defined by \(Q_k\), that is the set of edges in \(G\) which have ingress node in the set \(Q_k\). Then we have:

\[
\sum_{e \in \pi(Q_k)} t_{ek} \geq 1 \quad \forall k \in K_u \quad (4)
\]

By max-flow/min-cut theorem, equation (4) ensures there exists at least one path between \(s_k\) and \(d_k\) [16]. Next the following equations make sure OpenFlow entries are installed to direct the flow to each used link:

\[
l_{vk} \leq \sum_{e \in V} \sum_{(e, j) \in \pi(v)} x_{ijk} \leq 1 \quad \forall e \in E, k \in K_u \quad (5)
\]

\[
l_{ve} \leq \sum_{e \in V} \sum_{(e, j) \in \pi(v)} x_{ije} \leq 1 \quad \forall e \in E, k \in K_u \quad (6)
\]

\[
\sum_{(i, j) \in \pi(v)} x_{ijk} \leq 1 \quad \forall k \in K_u, v \in V \quad (7)
\]

Equations (5)(6)(7) ensure that if link \(e\) is used to direct the flow for \(k\), then there exists exactly one flow entry in the ingress switch of \(e\) to direct the flow of \(k\) to \(e\) and there exists one flow entry in the egress switch of \(e\) to accept the flow of \(k\) from link \(e\). Finally, we have the performance guarantee on maximum bandwidth utilization rate for all the links. Define \(B_k\) the bandwidth consumption for the demand pair \(k\), \(C_e\) the capacity of each link \(e \in E\), and define \(\beta\) the threshold on link utilization rate, we have:

\[
\sum_{k \in K} B_k \leq \beta C_e \quad e \in E \quad (8)
\]

The goal of ERP is to minimize objective function (1), subject to equations (2) – (8).

Next we analyze the complexity of ERP. Theorem 1 shows the NP-completeness and inapproximability of the ERP. Theorem 2 shows that even without the load balancing guarantee (8), or with some other performance guarantee rather than (8), the ERP is still NP-hard and \((1 - \epsilon)\ln |V|\) inapproximable for any \(\epsilon > 0\).

**Theorem 1.** ERP is NP-complete and inapproximable.

**Proof:** The proof is based on reduction from the 3-partition problem. Consider the part of hierarchical tree topology in Figure 5(a). Four source hosts inject packets to \(A, B, C, D\), and the bandwidth consumption \(B_k\) of the traffic injected on \(A, B, C, D\) are \(b_1, b_2, b_3, b_4\) respectively. The maximum usage on bandwidth \(\beta C_e\) of link \((E, H)\), \((G, H)\) and \((F, H)\) equal \(\frac{1}{4}(b_1 + b_2 + b_3 + b_4)\). To satisfy (8), the flows from the four source nodes must be partitioned into three subsets with the same total amount of bandwidth \(\frac{1}{4}(b_1 + b_2 + b_3 + b_4)\). Therefore by knowing whether the problem is feasible or not, we know whether the set of numbers \(\{b_1, b_2, b_3, b_4\}\) can be partitioned into three subsets with the equal sum of elements. Since the decision version of 3-partition is NP-complete, any polynomial-time approximation algorithm for this problem would solve the 3-partition problem in polynomial time, which is not possible unless \(P = NP\).

Following the same arguments, we can also show that TSMF is also NP-complete and inapproximable.

**Theorem 2.** Even without the link capacity constraints (i.e., equation (8)), ERP defined by \((1 - \epsilon)\ln |V|\)-approximation algorithm for any \(\epsilon > 0\), where \(|V|\) is the number of nodes in \(G\).

**Proof:** The proof is based on a reduction from the set cover problem. Consider a multi-root hierarchical tree topology in Figure 5(b), each node on layer 3 does not fully connect to every node on layer 2 due to link failure. 4 source hosts forms a routing group, each connects with the switches \(A, B, C, D\) and send traffic to the core switch \(H\). Assume \(\lambda_s\) is large and the weight of all the switches on layer 3 and layer 1 is small, the objective functions (1) is equivalent to minimize the number of entries inserted on layer 2 switches.

Since each additional switch used on layer 2 to direct the flow from \(A\) to \(D\) corresponds to an additional flow entry.
inserted on that switch, minimizing number of entries on layer 2 switches is equivalent to minimizing the number of switches used on layer 2. Define the universal set \( U = \{A, B, C, D\} \) which consists of all the layer 3 switches and assign a subset of \( U \) to each switch on layer 2, the subset for each switch on layer 2 consists of the switches on layer 3 that switch connects with. For example, the subset for \( E = \{A, C\} \) and the subset for \( F = \{B, D\} \). In order to make sure there is a path from \( A \) to \( U \) to destination \( H \), we need to ensure each switch on layer 3 connects to at least one switch on layer 2. Therefore, minimizing the number of additional flows inserted on layer 2 switches is equivalent to minimizing the number of layer 2 switches used to direct the flow, which is equivalent to minimizing the number of subsets used to cover the universal set \( U \), which is the definition of set cover problem. Since the set cover problem is NP-hard and cannot be approximated with in a factor of \((1-\epsilon)\ln n\) for any \( \epsilon > 0 \) (where \( n \) is the size of the set), the ERP is also NP-hard and \((1-\epsilon)\ln n\) inapproximable for any \( \epsilon > 0 \).

Since ERP is both NP-complete and inapproximable, we propose a simple and efficient heuristic to solve ERP. Without loss of generality, given an undirected topology \( G = (V, E) \) the graph can be made directed by replacing each undirected link \( e \) by two directed links \( e^+ \) and \( e^- \), mark both directed links \( e^i \) evolved from \( e \). We define a new directed graph \( G' = (V', E') \), and \( \delta(e^i) (e \in E') \) as the entrance switch (head) of \( e^i \) and \( \delta(e^-) (e \in E') \) as the egress switch (tail) of \( e^i \), an directed link \( e^i \) is a link from its entrance switch (head) to its egress switch (head). Define \( C_{e^i} (e \in E') \) the capacity of the link \( e^i \), which equals that of \( C_e \), where \( e \) is the undirected link from which \( e^i \) is created. We relate the cost of inserting rules on switches to the weight of the directed links of the switches. First, we provide the following definition:

**Definition 1.** Link \( e^i \) is ready for routing group \( u \) if: 1. \( \delta(e^i) \) contains a 4-tuple \((s_u, i, d_u, e^i) \); 2. \( \delta(e^-) \) contains a 4-tuple \((s_u, i, d_u, j) \).

That is, a link is ready for \( u \) if there already exists an entry on its ingress switch and egress switch to forward the flow onto this link. Next we calculate the cost of activating the links \( e^i \) on switch \( \delta(e^i) \). Let \( \ell(e) (e \in V') \) be the number of demand pairs of \( u \) that \( e \) carries after the \( e^i \) is activated. Define \( \ell(e) \) the number of egress links of \( e \) used to direct the traffic of demand pairs of \( u \) before \( e^i \) is added. Then the cost of activating this link \( e^i \), \( \text{cost}(e^i) \) is shown below:

\[
\text{cost}(e^i) = \begin{cases} 
\omega(\delta(e^i)), & \text{if } \theta_{\ell}(\delta(e^i)) = 0 \text{ or } \theta_{\ell}(\delta(e^-)) > 1 \\
\ell(e^-) - 1, & \text{if } \theta_{\ell}(\delta(e^-)) = 1
\end{cases}
\]

For each newly activated link \( e^i \), the corresponding OpenFlow rule has to be installed to the \( \delta(e^i) \) to direct the traffic. If initially no other link of \( \delta(e^i) \) is used, one OpenFlow entry \((s_u, i, d_u, n(e^i)) \) will be installed on \( \delta(e^i) \), so \( \text{cost}(e^i) = u(\delta(e^i)) \). However, if previously one egress link is activated on switch \( \delta(e^i) \), that is, initially all the flows are forwarded to single output port. To activate a new link with a new output port, we require all the flows that the switches carry to be fully specified so that they can be directed to the corresponding output ports. Hence \( \text{cost}(e^i) = (\ell(e^i) - 1) u(\delta(e^i)) \). Finally, if previously more than one link is activated on switch \( \delta(e^i) \), for each new activated egress link, a new entry \((s_u, i, d_u, n(e^i)) (e \in K_u) \) is installed to direct the flow.

An example is given in Figure 6(a): assume initially switch \( s1 \) carries two demand pairs \((0, 10)\) and \((0, 11)\) of \( u \) that have the same output port 4 \((\theta_4 = 1)\), therefore one entry is installed to route the flows as shown in Figure 6(b). Assume one more demand pair \((0, 10)\) is added and another egress link is used to direct this flow (output port is 5), then number of entries in the routing table increases by \( \ell(e^i) = 3 \). Finally the cost to activate this new link is \( 2(\ell(e^i)) \), the new flow table is shown in Figure 6(c). Algorithm 1 reuses the links which are ready by setting the weights of these links to 0. The weights of other links are updated according to \( \alpha \). If the bandwidth consumption on \( e^i \) exceeds the maximum limit \( \beta_{e^i} \), the cost of \( e^i \) is set to be infinity, \( \text{cost}(e^i) = \infty \). Finally the path can be set up by finding the shortest path between the source and the destination hosts.

We now analyze the complexity of IRA. The for loop between line 3 to 8 in IRA determines the cost for each edge \( e \in E \). In line 9, the shortest path is calculated between each \( s_k \) to \( d_k \). Therefore, the overall complexity is
Algorithm 2 Detailed Search Algorithm (DSA)

1. Set the source and destination address to the address with fully wildcard bit, set $\text{prev} = \text{curr} = \emptyset$, set $l_6 = l_4 = \eta_0$.
2. while $K = 0$ do
3. Set $\text{prev} = \infty$ and $\text{curr} = 0$
4. while $\text{curr} \leq \text{prev}$ or $\text{size}(\text{curr}) > L$ do
5. Set $u_{\text{src}}, \text{argcost}(u_{\text{src}}) = \text{FindCost}(\text{src}, l_6, 0)$
6. Set $u_{\text{dst}}, \text{argcost}(u_{\text{dst}}) = \text{FindCost}(\text{dst}, l_6, 0)$
7. Set $u_{\text{src}}, \text{argcost}(u_{\text{src}}) = \text{FindCost}(\text{src}, l_4, 1)$
8. Set $u_{\text{dst}}, \text{argcost}(u_{\text{dst}}) = \text{FindCost}(\text{dst}, l_4, 1)$
9. Select $\text{curr}$ equals to $u \in \{u_{\text{src}}, u_{\text{dst}}, u_0, u_1, u_2\}$ with the minimum $\text{argcost}(u)$, if more than one such $u$ exist or all the $\text{argcost}(u)$ equals infinity, randomly pick one.
10. Set $\text{curr} = \text{argcost}(u_{\text{curr}})$
11. if $\text{curr} > \text{prev}$ or $l_4 = l_6 = 0$, then
12. Remove all the demand pairs in $\text{prev}$ from $K$, building the path for each demand pair in $\text{curr}$ by using $IRA$.
13. Set the source and destination address to full wildcard bits. Set $\text{prev} = \text{curr}$, $\text{prev} = \text{curr}$
14. break
15. Set the binary digit on leading bit according to $u_{\text{curr}}$, update the leading bit by decreasing $l_6$ or $l_4$ by 1 according to $u_{\text{curr}}$, set $\text{prev} = \text{curr}$, $\text{prev} = \text{curr}$
16. Function $\text{FindCost}(\text{type}, l_4, d)$
17. if $\text{type} = \text{SRC}$ and $l_4 = 0$ then
18. Set the binary digit on leading bit $l$ of source address to $d$, while keeps destination address the same. Denote the routing group formed $u$.
19. if $l < \text{size}(u) \leq L$ then
20. Reset the binary digit on the leading bit $l$ of the source address to wildcard bit.
21. Return $(u, \text{IRA}(u))$
22. if $(\text{size}(u) > L)$ then
23. Return $(u, \infty)$
24. if $\text{type} = \text{DST}$ and $l_4 = 0$ then
25. Set the binary digit on the leading bit $l$ of the destination address to $d$, while keeps the source address the same. Denote the routing group formed $u$.
26. if $c < \text{size}(u) \leq L$ then
27. Reset the binary digit on the leading bit $l$ of the destination address to wildcard bit.
28. Return $(u, \text{IRA}(u))$
29. if $(\text{size}(u) > L)$ then
30. Return $(u, \infty)$
31. EndFunction

$C([K_0](|V| + |L| \log |L|))$.

VI. EFFICIENT PARTITIONING PROBLEM

After solving ERP for each routing group, we are left with the problem of partitioning $K$ demand pairs into routing groups. In this case all demand pairs can be visualized using $2^m \times 2^m$ square, where $m$ is the number of bits in the source and destination address. For example, Suppose there are 6 demand pairs $[10, 00], [11, 00], [00, 01], [00, 11], [01, 11], [01, 10]$, the corresponding square is shown in Figure 7(a). The squares representing the 6 demand pairs are coloured in blue. One of the possible partitions is shown in Figure 7(b), where the routing group $G_1$ covers the demand pairs $01, 10, 01, 11, 00, 11$, $G_2$ covers $01, 10, 10, 00, 11, 00$ and $G_3$ covers $00, 01$.

The goal of EPP is to find the routing groups so each group can be routed with the lowest cost as defined in (1). We represent each routing group by a pair of source-destination addresses with subnet mask. For example, $G_2$ in Figure 7(b) can be represented by $1_4, 00$. A drawback of aggregating flow entries is that we lose visibility into fine-grained flow characteristics, which makes elephant flow detection and rerouting harder to achieve [6]. In Sector, we use maximum routing group size $L$ to limit the maximum flow aggregation level, which allows the Sector to make a trade-off between flow visibility and TCAM space savings.

Our solution algorithm, called Detailed Search Algorithm (DSA), starts from the entire square that covers all source-destination pairs. In each iteration, it reduces the size of the routing rectangle by replacing the wildcard bit in the address with a binary digit. The output of each iteration is the routing group with the lowest average cost per demand pair in the group. Define the leading bit of an address as the leftmost wildcard bit in the address. For example, the leading bit of address $000_2$ is the third bit. If there is no wildcard bit in the address, set the leading bit to 0. Denote $\text{size}(u)$ the number of demand pairs in routing group $u$. Define $l_6$ and $l_4$ as the leading bit of source and destination address. The pseudo code of Detailed Search Algorithm is described in algorithm 2. The function $\text{IRA}(u)$ returns the minimum cost generated by $IRA$ to route all the demand pairs in $u$. The DSA algorithm works by searching the routing group $u'$ with $\text{size}(u') < L$ with the lowest average cost in a greedy fashion, and building the paths for that group with minimum cost. Afterwards, the demand pair is removed from $K$. The algorithm terminates when all the demand pairs in $K$ have been routed.

Figure 8 provides an example to illustrate DSA. Let $L$ equal 3. Initially there are 6 demand pairs. The routing group is the region circled by the red dash line, which is the whole square shown in Figure 8(a). Assume we found the routing group with minimum average cost is $[1_4, 00]$, by setting the leading bit of source address to 1, the corresponding routing group is shown in Figure 8(b). Repeat these steps until we found the routing group $1_4, 00$ shown in Figure 8(c) and 8(d). Note that further dividing this routing group will increase the average routing cost per demand pair. Then the two demand pairs in the routing group $[1_4, 00]$ will be routed by using $IRA$. DSA then removes this routing group, and repeat the process until all the demand pairs are routed. We now analyze the
complexity of DSA. The inner while loop between line 4–15 runs at most $2n$ times, since in each iteration of the inner while loop the leading bit of source address or destination address decreases by 1, the iteration will stop when all the wildcards bit in source address and destination address are filled with binary digits. For each inner while loop, the IRA is called $4$ times (line 5–8). Finally, the outer while loop (line 2–15) runs at most $|K|$ times. Therefore the complexity of DSA is $O((8mK)^2(|V| + |E| \log |E|))$. It is possible that two routing groups may overlap with each other. For the example shown in Figure 7(c), two routing groups G1 and G2 both cover the yellow square 11, 90. Assume the switch s1 carries the traffic of both routing groups, the flow of [11, 90] will satisfy the predicates for both entries, which is shown in Figure 7(d).

Therefore each entry in the switch must be assigned with a priority level. Upon receiving a packet, the switch finds the entries with a matching predicates and the highest priority level, then performs its action. One simple way to assign priorities in DSA is based on the order the routing group is generated by DSA. For example, if G1 is generated before G2, then the entry of G1 has a higher priority than that of G2 (shown in Figure 7(d)).

VII. Dynamic Scheduling of Demand Pairs

The algorithms presented in the previous sections have been focused on the static version of the problem. While they are useful for networks that have constant network demand, in reality, the demand pairs may join/leave the network dynamically. In this section we propose the dynamic algorithms to deal with this scenario.

A. Dynamic demand pairs entering

We first consider the case where a new demand pair k enters the network. Let $s_k$ and $d_k$ denote the source and destination address of $k$. We first make the following definition:

Definition 2. Let $f$ be a full address without wildcard bits we say the address $f'$ covers $f$ if $f'$ and $f$ have the same bit length and all the non-wildcard bits of $f'$ are the same as $f$.

For example, let $f = 00 * 010$ and $f' = 000$, then $f'$ covers $f$ because all the non-wildcard bits of $f'$ (the first two bits) are the same as $f$, which is 0. We next extend the definition of ready for each new demand pair $k$:

Definition 3. In a directed graph $G' = (V', E')$, link $e'$ is ready for the new demand pair $k$ if: 1. out($e'$) contains a 4-tuple $(s, i, d, v(e'))$, $i \in \pi(out(e'))$; or $(s, i, d, v(e'))$, 2. in($e'$) contains a 4-tuple $(s, n(e'), d, j)$, $j \in \pi(in(e'))$; or $(s, i, d, j)$, where $s$ covers $s_k$ and $d$ covers $d_k$.

Algorithm 3 (DANA) builds the paths for each new demand pair. The intuition behind DANA is reusing existing rules in the network. For the example shown in Figure 1(c), the routing tables are shown in Figure 2(c). Assume that there exists a new demand pair with source address/destination address 01C/101 and ingress/egress switches are A and E. Further assume that the every link has enough remaining capacity to carry the flow of this demand pair such that the switch 8 is obeyed, every switch has the same weight and enough TCAM space. One of the possible solutions is routing through the path $A, B, E, (the$ black route in Figure 9(a)), and the new routing table is shown in Figure 9(b). Three entries are added on the switch A, B, and E. DANA will generate the red route shown in Figure 9(a) and the routing table shown in Figure 9(c). By comparison, only one entry is installed on switch L, and the entries in switch A, L, and E are reused so that no additional entry is installed.

B. Dynamic demand pairs leaving

In case of a demand pair leaving the network, if the leaving renders the rule to be obsolete, this rule can be safely deleted either by the controller or idle timeout [10]. However, depending on the network traffic pattern, some unused rules can be kept for a longer time for routing future traffic flows. Details of this problem is out of the scope of the paper [17].

VIII. SIMULATIONS

A. Network Settings

We evaluated DSA on 4 different network topologies, one is a real WAN model generated by GT-ITM [18], which simulates WANs using Transit-Stub topologies. This network has 100 nodes and 127 undirected links. The other network topologies includes the Abilene (11 nodes, 13 undirected links), Fat Tree (4 pods, 4 core switch, 52 nodes and 64 undirected links) and Sprint (52 nodes, 168 undirected links). The traffic distribution for Abilene and Sprint are available in [19]. We use two models proposed in [19] Lognormal distribution ($\mu = 15.45, \delta = 0.585$), and Weibull distribution ($\alpha = 1.87 \times 10^3, \beta = 0.89$) to model the traffic distribution in the Sprint Network. And we use the Lognormal distribution ($\mu = 16.6, \delta = 1.04$) to model the traffic distribution in the Abilene Network. For the GT-ITM and Fat Tree, we use the Bimodal distribution (generated by mixture of two Gaussian Distributions) proposed in [20].

Algorithm 3 Dynamic Algorithm for New Arrivals (DANA)

1: for each new demand pair k do
2: for each link $e' \in E'$ do
3: if $e'$ is ready for k then
4: $\text{Set the cost of link } e' \text{ to } 0, \text{cost}(e') = 0$
5: if $e'$ is not ready for k then
6: Set the link cost $\text{cost}(e') = w(e')$
7: if $\text{cost}(e') \leq T_k$ or $d(out(e')) > r(out(e'))$ then
8: $\text{Set the cost of link } e' \text{ to infinity, cost}(e') = \infty$
9: Find shortest path between $s_k$ and $d_k$, if there are more than one shortest paths, randomly select one. Install the 4-tuple rules ($s_k, i, d_k, j$) along the path. Update $a(e')$
10: $\beta C_{e'} = \beta C_{e'} - L_k$

Fig. 9. Example of Flow Tables

A, B, E, (the black route in Figure 9(a)), and the new routing table is shown in Figure 9(b). Three entries are added on the switch A, B, and E. DANA will generate the red route shown in Figure 9(a) and the routing table shown in Figure 9(c). By comparison, only one entry is installed on switch L, and the entries in switch A, L, and E are reused so that no additional entry is installed.

B. Dynamic demand pairs leaving

In case of a demand pair leaving the network, if the leaving renders the rule to be obsolete, this rule can be safely deleted either by the controller or idle timeout [10]. However, depending on the network traffic pattern, some unused rules can be kept for a longer time for routing future traffic flows. Details of this problem is out of the scope of the paper [17].
The Bimodal distribution is proposed based on the observation that only a small fraction of Source-Destination pairs have large flows. Assume each switch has a capacity between 300 – 500 entries. We use the method proposed in [21] to model the link capacity, which claims that the link capacity distribution follows Zipf’s Law, and the links whose end nodes with higher degree tend to have larger link capacity. For the purpose of simulation, we set the link capacity to 39813.12Mbps (the transmission rate of optical carrier OC768) if the degrees of both endpoints of that link are larger than 3, set the link capacity to 965.28Mbps (OC192) if one endpoint has degree larger than 3 and degree of the other end point is less or equal 3, set the link capacity to 2488.32Mbps (OC48) if the degree of both endpoints is less or equal than 3.

We randomly generate demand pairs, each corresponds to a source machine and destination machine in the network. Each machine has been assigned a random type B IP address and they are connected to a switch in the network. The bandwidth consumption of the flows follows the distribution described above. Since TCAM space aware routing has not been investigated before, and there is no such a similar routing algorithm which aims to reduce the routing table size, we compare DSA with two benchmark routing schemes: ECMP and Valiant Load Balancing (VLB) which are widely used to achieve load balancing on link resources. ECMP is a routing strategy which works by equally splitting the traffic over the multiple paths with the same length (number of hops) [22]. In VLB, the flows of the same demand pair are first sent to some intermediate nodes, then forwarded to the destination [23]. After the paths are calculated by the two benchmark routing schemes, the corresponding rules (s_k, i, d_k, j) are installed to direct the flows. All the rules contain fully specified addresses s_k and d_k so that they can not be reused by the other flows.

We run each algorithm 100 times and take the average results. For the evaluation, we set the weight of each switch in (1) to 1, therefore the total cost generated by (1) equals the total number of entries installed. We compare performance of the algorithm using a metric called Traffic Saving Ratio. Assume the total amount of TCAM space consumed by DSA is T_p, and total amount of TCAM space consumed by the benchmark algorithm is T_b, then Traffic Saving Ratio (TSR) is defined as:

$$TSR = \frac{T_p - T_b}{T_b}$$

B. Evaluation of the TSR

First we evaluate the relation between the number of demand pairs and TSR. We do not limit the maximum routing group size. Table II shows the relations between the number of flows and TSR with different networks and different traffic distributions. TSR_1 is the TSR of the ECMP and TSR_2 is the TSR of the VLB. The DSA can achieve 20% – 80% saving on the TCAM space with different network topologies and traffic distributions. The saving also grows with the number of flows. This is because as the number of flows increases, more flows can be aggregated for saving TCAM space. Moreover, if we neglect the first packet of each flow which is forwarded to the controller, the TCAM space saving almost equals to the saving in the number of control traffic between the controller and the OpenFlow switches. This is because each entry in the switches requires a control packet for installation. Figure 10 to 13 show the relations between the number of flows and the actual maximum link utilization of different algorithms over different traffic distributions. When calculating the link utilization of DSA, the threshold on link utilization rate, $\beta$, is set to 0.9. The maximum link utilization rate of DSA is on average 16 – 17 percent higher than that of ECMP and VLB. Despite the higher link utilization rate of DSA, considering the huge savings on the TCAM space, we believe this is a fair trade-off.

As mentioned before, we can tune the value of L to balance the trade-off between TCAM space saving and maximum link utilization. We evaluate on the GT-TM network with 50c demand pairs. As shown in Figure 14 and 15, when L decreases from 5 to 1, the TSR decreases from 0.4107 to 0.14605, meaning less demand pairs are aggregated. At the same time, the maximum link utilization rate also decreases slightly from 0.407 to 0.752. This is because small routing group leads to fine-grained routes which in turn reduces maximum link utilization.

C. Evaluation of the DANA

We generate some demand pairs which attach to some random nodes in the network. Each demand pair has a random type B source and destination IP address, and all the demand pairs are connected by installing the rules generated by DSA. To emulate the dynamic entering of new demand pairs, we generate 50 new demand pairs and run the DANA to add the flows. We compare the performance of DANA with shortest path algorithm (SPA), which routes traffic along shortest paths. All the rules installed for SPA are fully specified addresses. TSR is defined in a similar manner as (9), with I_b and I_p means the total number of rules generated by SPA and DANA to direct the new flows. Table III shows the performance and running time of the two algorithms. As the tables shows, DANA can achieve 40% – 60% saving on TCAM space. The running time of DANA increases moderately with the

<table>
<thead>
<tr>
<th>Table II. Evaluate on all the network topologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Abilene</td>
</tr>
<tr>
<td>Sprint</td>
</tr>
<tr>
<td>GT-TM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table III. Performance and Running Time Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>Running Time</td>
</tr>
<tr>
<td>DNSA</td>
</tr>
<tr>
<td>SPA</td>
</tr>
</tbody>
</table>

223
network size. But overall running time of the algorithm is still reasonable.

IX. TESTBED DEPLOYMENT

We evaluated the functionality and implementability of the DSA in a real testbed. We built the overlay network that follows Abilene network topology by using software switch (OpenVSwitch) running on virtual machines. The OpenVSwitches communicate with each other by using the Virtual Extensible LAN. The centralized controller (Ryu) can configure the entries in the switches to build the routing paths. We built three source VMs and three destination VMs (three demand pairs), each VM is assigned with an IP address. The routing module on top of the Ryu controller takes the connection demands as the input and sends the results of DSA to the implementing module which installs the relative OpenFlow rules on the switches (Figure 16). For comparison, we also used the shortest path algorithm (Dijkstra’s Algorithm) to connect the demands pairs. For DSA, total 14 entries are installed on the switches, and the total time taken for building the path is 0.028s. For Dijkstra’s Algorithm, total 30 entries are installed on the switches with the total time 0.061s. Hence, DSA clearly saves TCAM space and path setup time.

X. CONCLUSIONS

In this paper, we propose Sector, a routing scheme to achieve savings on TCAM space in SDN without causing network congestions. We provide algorithms for this problem for both the static and dynamic scenarios. Experiments show that Sector can achieve 20% – 80% saving on TCAM space with 10% – 17% increase in maximum link utilization.

REFERENCES


[18] GT-ITM website: www.cc.gatech.edu/projects/gt-tm/


Port Based Capacity Extensions (PBCEs):
Improving SDNs Flow Table Scalability

Robert Bauer, Martina Zitterbart
Karlsruhe Institute of Technology
Karlsruhe, Germany
{robert.bauer, zitterbart}@kit.edu

Abstract—Software-defined networks (SDNs) come with great promises regarding flexible operation of networks. A key component within SDN-switches is the flow table which holds the rules that determine how data streams are handled. The flow table, however, is a scarce resource with a rather limited rule capacity. To soften this well-known hassle, we propose a novel delegation mechanism for OpenFlow-based SDNs called Port Based Capacity Extension (PBCE). PBCE provides the possibility to delegate flows from a switch with many flow table entries to another less loaded neighboring switch without breaking control plane transparency, i.e., without interfering with existing SDN applications. To do so, PBCE uses flow table aggregation based on ingress ports and a small number of special rules for both switches. In this paper, we present the PBCE delegation middleware along with a prototypical implementation and first promising performance results that demonstrate the feasibility of the approach.

Index Terms—Software Defined Networking, Flow Table Scalability, Delegation, OpenFlow

I. INTRODUCTION

Software Defined Networking (SDN) is a popular and promising technology which enables more flexible control and easier management of forwarding devices. A key component within SDN-switches is the flow table which holds the rules that determine how data streams are handled. Because the flow table of a hardware switch is often implemented with TCAM, the available rule capacity is a scarce and expensive resource. Limited rule capacity of SDN switches is a well-known and intensively studied problem (e.g., [1], [2], [3]). While the realization of fine-grained network policies may require a large rule capacity, currently available hardware often only supports between 1000 and 10000 rules. However, the load of different switches under the control of a single SDN controller may vary, i.e., there might be switches with heavily loaded flow tables and others with spare capacity. We therefore propose an easy to implement delegation mechanism called Port Based Capacity Extension (PBCE) that copes with flow table scarcity by making use of such spare capacity. The core idea of PBCE is to move OpenFlow rules from a switch A (with many flow table entries) to another less loaded switch B. Traffic is then forwarded from A to B, where the fine grained rules are installed to perform the necessary packet processing and return the traffic back to B. Roughly summarized, switch A delegates part of the flow processing to switch B. For simplicity, we will refer to switch A as delegation switch (DS) and to switch B as extension switch (ES).

Assume an example with a heavily loaded switch DS (flow table utilization at 86%) and another switch ES (14%) that is directly connected to DS. The rigid TCAM limitations of DS can severely affect network performance while ES has plenty of unused flow table entries. To utilize this existing spare capacity, we install an eviction rule in DS to forward some of the flows to ES (see Section II). Given an evicted flow F that was originally intended for output port Q on DS, we install a rule in ES that performs the fine-grained packet processing. This rule returns processed packets back to DS, but with Q as a metadata item attached to each packet (say that we translate Q into VLAN field value V). Now it suffices to add another rule to DS that forwards all packets with VLAN==V to port Q. Note that this approach requires a small number of additional rules in DS (equal to the number of ports of DS) and overwrites existing header fields like VLAN or MPLS in order to transport metadata between the two switches.

Although we focus on flow table capacity delegation throughout this paper, the general concept can be adapted to further use cases as well. Examples are:

- Use spare capacity to improve existing services like heavy hitter detection and traffic engineering. Accuracy of SDN-based monitoring for example often scales with the number of TCAM entries [4] and could directly benefit from PBCE.
- Allow incremental hardware deployment. Due to the feature-richness of the OpenFlow specification and diversity in the switch vendor market not all features are available on every forwarding device. With PBCE, potentially heterogeneous devices could be used as a resource pool to incrementally integrate required features into the networking infrastructure.
- Use it as a lightweight alternative to Network Function Virtualization and Service Function Chaining, if such solutions are unsuited, e.g., because of the general complexity, performance issues or the management and orchestration overhead.

The remainder of this paper is structured as follows: In section II we present the general architecture of PBCE. We then discuss some details of the prototypical implementation in section III and first evaluation results in section IV. Section V contains related work.
Delegated rules
Backflow rules
Monitoring and Configuration:
Delegation Decision:
Delegation Mechanism:
The PBCE decision engine uses
Two categories of monitoring data relevant
are flow rules initiated by an SDN
R
∈
of
∈
Delegation Mechanism:
inside
is
The PBCE decision engine uses
primitives:
3) Dynamically chosen eviction rules are low priority rules that match only on a specific ingress port of DS. Packets that arrive at this ingress port but do not match any other rule are forwarded to ES, i.e., the output action of the rule is set to a DPort. Because of the low priority, only new flows are affected by eviction rules. Ingress ports whose unmatched traffic is redirected to a DPort are referred to as Eviction Ports (EvPorts).
2) Delegated rules are flow rules initiated by an SDN application that are currently stored inside ES. Packets that match on a delegated rule are always sent back to the ingress port (ExPort) with forwarding metadata attached. Delegated Rules are created by the PBCE middleware.
3) Backflow rules are static rules inside DS that can read the forwarding metadata inserted at ES to forward packets on the associated egress port. If, for example, DS has n physical ports, n different backflow rules are required in order to cover all egress ports.

If these three types of rules are set up properly, packets entering DS via EvPort will follow the path depicted in Fig. 1 (blue numbering) and packet handling takes a detour via the extension switch (this is explained in more detail later). Switches can act as delegation and extension switch at the same time and multiple EvPorts can be mapped to the same DPort.

A. Components of the PBCE architecture

We now discuss the components of the PBCE architecture and the steps necessary in order to realize delegation. The architecture is shown in Fig. 2 and consists of three main building blocks:

- Monitoring and Configuration: The configuration component is required to set up the initial behavior of the middleware, e.g., selecting appropriate thresholds for the delegation mechanism. The monitoring component collects data about the current situation in the network which is required to detect bottlenecks or determine the excess capacity of extension switches.
- Delegation Decision: The PBCE decision engine uses the data collected by the monitoring component. It keeps track of active delegations and is responsible for triggering and revoking the actual delegation mechanism.
- Delegation Mechanism: The PBCE delegation mechanism controls the flow table utilization of a delegation switch and consists of two parts, port eviction and flow migration. Port eviction is responsible for traffic redirection and rule delegation (affects multiple flows linked to an EvPort). Flow migration, on the other hand, is a fine granular mechanism used to further optimize flow table utilization and the link utilization between delegation and extension switch.

1) Monitoring: Two categories of monitoring data relevant to PBCE can be distinguished. First, basic monitoring of the physical devices, which includes the current and maximum flow table utilization of the switches in the network and existing interconnections between them. The monitoring component can easily gather such information from the SDN controller or directly from the switches in a standardized way (e.g., using OpenFlow). The other category consists of extended monitoring information that is not easily available, e.g., the flow-to-ingress-port mapping (FIPM). This mapping describes the relationship between a set of flow rules R inside the flow table and the associated group of ingress ports P, i.e., packets originated from p ∈ P are matched by a rule r ∈ R. The FIPM is of particular importance to PBCE, because it can be utilized to determine the amount of flow rules that can be easily migrated in case of delegation – especially if |P| is small, which seems a reasonable assumption for many SDN scenarios. The FIPM can be derived from the global network view, e.g., by using a collector inside the SDN controller that continuously monitors Packet-In events (with the assumption that follow-up packets for a newly programmed flow will enter the switch at the same port). Note that the FIPM is an optional
The PBCE Decision Engine is a subset of the flows towards port and the amount of traffic that is selected for eviction. The third parameter, evict, is installed on DS. Because most of the treatment is taken after flows entering DS. An in-depth analysis of suitable heuristics for the decision process is beyond the scope of this paper. We do, however, present some details on the heuristic currently applied in the prototype in section III.

3) Delegation Mechanism: The delegation mechanism consists of two parts, port eviction and flow migration. Port eviction is required to evict a subset of the flows towards ES. A common way to achieve this is flow rule aggregation, i.e., install coarse-grained rules covering a set of more fine-grained rules. This approach is utilized by various scalability solutions [5], [3] and could very well be adopted by PBCE. Flow rule aggregation, however, has two major drawbacks. It requires a sophisticated and possibly hard to solve algorithm for dependency resolution (flow rules inside a single flow table with intersecting matches, see [3]). And more importantly, it is hard to establish a mapping between ingress port and the covering aggregation rule, because one aggregation rule can be associated with multiple ingress ports (which makes it difficult to inform ES about the original ingress port).

Therefore, we propose a slightly modified aggregation scheme where dynamically installed eviction rules and static backflow rules are used to redirect traffic. The eviction rule is a low-priority rule covering all packets that match solely on a specific ingress port with all the other match values wildcarded. The priority of the eviction rule is set to a value that is above the priority of the default OpenFlow table miss entry (a fully wild-carded rule that is covering all flows with minimum priority) and below the priority of the rules installed by SDN applications. Beside the eviction rules, a static set of backflow rules is installed on DS. Because most of the treatment is taken care of at ES – flow specific matching, header modifications – only a small and static amount of backflow rules is needed, more precisely a maximum of one rule for every possible egress port and a special rule for flooding.

In addition to the setup of eviction and backflow rules, the delegation mechanism has to cope with the problem of metadata transport between DS and ES. For PBCE, two metadata items are required. One metadata item is written by the eviction rule in DS and contains the original ingress port L (EvPort in Fig. 1). Without this item, an SDN application could not match on L if a flow gets evicted, because the new ingress port would be an ExPort and not L. The second – more important – metadata item is written inside ES and contains the forwarding information Q (c.f., the same variable Q in the introduction). Q is determined by the SDN application and represents the intended output action for a flow on DS. For this to work, the PBCE middleware has to intercept all Packet-In messages sent to the controller if the ingress port is an ExPort and overwrite the ingress port value that is seen by the SDN application with L. In addition, every fine-grained rule installed on ES has to be modified in a way, that the output port is changed to the ExPort (or IN_PORT in terms of OpenFlow) and Q is added to the packet for the forwarding decision in DS. The modified rules in ES are called delegated rules.

Metadata transport could be realized by overwriting existing header fields, i.e., use VLAN, MPLS, Flow Label or DSCP to transport Q and L between DS and ES. If this is not feasible – e.g., because the headers are used otherwise – PBCE would have to further encapsulate the packets and use a proprietary header to encode the metadata (we used an overwrite approach and DSCP for the evaluation presented in Section IV). Note that a single eviction rule is not sufficient because the ingress port mapping is lost and delegation would be limited to a single DPort. With the eviction scheme described here, on the other hand, it is possible to adapt the number of ports that are evicted in order to scale the amount of flow rules that are delegated towards an extension switch.

Flow migration is the second part of the delegation mechanism. Migration in this context means that existing rules in the flow table of ES are moved to the flow table of DS or vice versa. Two types of migration can be distinguished: Forward-migration and backward-migration. Forward-migration is applied to flow rules in DS, e.g., to delegate already existing rules to ES after an eviction rule was installed. This is required, because port eviction only affects new flows entering DS. Flows that already existed before the eviction rule was set up will have a higher priority and the corresponding traffic would not be redirected by only using eviction rules (which is intended). Forward-migration is done by adding an already
Because ES has plenty of excess flow table capacity (current utilization is 29%) and the link between ES and DS is not saturated, $p_0$ is selected as a single DPort. $p_{24}$ of ES thus becomes an ExPort.

After the decision process, PBCE initiates the delegation mechanism, which in turn installs eviction rules for $p_2$ and $p_4$ and seven backflow rules to cover the return traffic – one for each physical port of DS (including DPorts and EvPorts) and one to handle flooding. Note that the 852 flows pointed out earlier are not yet redirected to ES because of the low priority of the eviction rules. This is done by forward-migration, which installs altered versions of the rules in ES (forwarding metadata, output action set to ExPort) and deletes them at DS. The bottom of Fig. 3 shows the new situation in the example scenario after PBCE has performed the changes described above. The 852 flows are now stored in the flow table of ES and a small amount of eviction and backflow rules are added to the flow table of DS (labelled with 3). Flow table utilization of ES is reduced from 93% to 50%, utilization of ES is increased from 29% to 50%.

Packet forwarding and reactive flow setup is now handled as follows: If a packet arrives on an EvPort at DS and no matching rule is found in the flow table, it is forwarded using the DPort because of the eviction rule (illustrated by 4 in Fig. 3). Metadata about the ingress port (EvPort) is added to every single packet. If the packet arrives at ES and a delegated rule is present, it is directly forwarded back to the ExPort with attached forwarding metadata. Without such a rule, the default rule of ES will forward the packet to the controller, where the PBCE middleware detects a Packet-In coming from an ExPort. The new flow is then not installed as a normal rule in DS but as a delegated rule in ES. Either way, packets are sent back to DS (5), where the forwarding metadata and corresponding backflow rules are used to forward the packet to the originally intended egress port (6). Backward-migrations are applied if one of the delegated flows requires too much bandwidth on the DPort-ExPort connection (by installing a new rule for the flow in DS with higher priority and deleting the delegated rule).

### III. Implementation

The prototypical implementation of PBCE is integrated into the application execution environment of the Ryu controller [6]. This section briefly describes three important aspects of the implementation: The overall integration approach to achieve control plane transparency, the metadata transport scheme and some details on the delegation decision process.

**Control-Plane Transparency** addresses the challenge of finding a feasible way to integrate PBCE with an SDN controller in such a way, that existing SDN applications can be used without heavy modifications. We choose a middleware approach, where all internal communication between controller and SDN applications is supervised and possibly altered. More precisely, a lightweight wrapper around app_manager.RyuApp intercepts important callbacks like the packet_in handler function. This enables high flexibility, requires minimal alterations to
the core functionality of the controller and individual SDN applications only have to change their super class to the middleware wrapper. The main driving factor behind this design decision was that a middleware can easily memorize the ingress port of packets responsible for the installation of new flows which is required for an estimation of the FIPM.

**Metadata transport** is required for both, eviction rules in DS (otherwise the original ingress port would be lost) and delegated rules in ES (forwarding metadata). Because of the limited range of values (port-count of DS) the prototype uses the DSCP header field, which is sufficient for switches with a port count smaller than 63. The DSCP field is cleared inside the backflow rules.

The **delegation decision** of the prototype is primarily based on port specific flow arrival rates. We denote \( \delta(t1,t2) \) as the number of flows arrived on ingress port \( p \) between \( t1 \) and \( t2 \). The current delegation potential is then defined as \( w_i(p) = (1-k) \cdot \delta_n(t, t-1) + k \cdot \delta_n(t-1, t-n) \) where \( n \) is the number of considered samples in the near past and \( k \) is a weighting factor. We found \( n = 10 \) and \( k = 0.5 \) to be acceptable values for our scenario after a (small) series of experiments with different values for \( n \) and \( k \). For the decision process, \( w_i(p) \) \( \forall p \in Ports \setminus(DPorts \cup EvPorts) \) is determined and the port with maximum \( w_i(p) \) is chosen as EvPort. For DPort selection, the prototype selects the port with minimum link utilization. Two thresholds determine whether the delegation mechanism is triggered (executed once per decision cycle): The **delegation threshold** defines the upper limit for flow table utilization of DS, i.e., as soon as this threshold is exceeded, a new EvPort is determined. If the utilization falls below the **revocation threshold**, delegation is canceled for one EvPort. In addition, the decision process is connected to a **grace period** to avoid overregulation. This is done by incrementing a counter every time an EvPort is selected or revoked. The counter is decremented linearly and the delegation decision is delayed if the counter is greater than zero (otherwise, too many EvPorts may be selected).

**IV. EVALUATION**

We now evaluate the feasibility and performance of the prototypical PBCE implementation outlined in the previous section with a series of experiments. The key findings are:

- PBCE is able to efficiently control the average flow table utilization of a delegation switch.
- Control plane overhead in terms of CPU consumption and control traffic is dominated by monitoring and scales well with increasing delegation workloads. For smaller workloads, PBCE induces low overhead.
- Only 0.1ms of additional end-to-end delay is added for delegated flows in the data plane.

**A. Experimental Setup**

To cover a wide area of different evaluation scenarios, we first introduce a generic workload indicator called **table ratio** (TR). Given the average flow table utilization of the delegation switch \( T_{DS} \) and the extension switch \( T_{ES} \), we define TR as \( T_{DS} / T_{DS} \). Because we only allow delegated rules in ES (for simplicity of the evaluation), this parameter gives us a good indication on how much delegation work the PBCE middleware has to accomplish. For TR = 0, no flow is delegated to ES. If over-utilization occurs infrequently, the average number of rules in the flow table of ES is going to be small (compared to DS) and the value for TR is low. In case of constant and long-term over-utilization, on the other hand, the value for TR is much higher (possibly \( > 1 \), with TR = 1 indicating that – on average – half of the flows inside DS are delegated to ES).

Up to 200 instances of iperf3 with individual settings (bandwidth, flow duration, number of parallel sessions) inside a mininet environment are used for evaluation. We consider flow arrival rates between 50 and 350 flows/second and delegation thresholds in the range of 50-1600 to create various table ratios and flow table utilization patterns. The CDFs in Fig. 4 further characterize the experiments used for evaluation throughout this section. A single experiment lasts for 400 seconds and is executed in the testbed depicted in Fig. 5, which consists of three physical nodes. Each node is equipped with an Intel(R) Xeon(R) E5420 processor (2.50 GHz, 2 sockets, 4 cores/socket) and separate physical 1Gbit/s networks for control and data plane traffic. One node runs DS (Open vSwitch v2.4.0) and the mininet tool. Note that the virtual hosts are not directly attached to DS to allow flexible scale up of the traffic generation endpoints without changing the port count. ES (another Open vSwitch) is placed on a physically separated node to enable measurements with real network limitations (link capacity, latency). The Ryu controller runs a reactive SDN application that installs forwarding flows based on TCP ports. Destinations for generated traffic are selected randomly.

**B. Functional Evaluation**

For the functional evaluation, we first demonstrate that flow table capacity delegation reduces the flow table utilization of

![Fig. 4. Parameters for 5900 experiments (4759 with PBCE enabled): delegation threshold, table ratio, average flow arrival rate, average flow table utilization of DS, monitoring frequency and DS port count](image)
This section analyzes the experiments exceeding the threshold. Fig. 7 therefore shows four experiments where the amount of delegated flow rules by adapting the delegation mechanism needs to be executed and new flows may arrive in the meantime, which is especially true for scenarios with high and bursty flow arrival rates. Note that the flow table utilization of DS will always exceed the delegation threshold for a short time frame, because the delegation mechanism needs to be executed and new flows may arrive in the meantime, which is especially true for scenarios with high and bursty flow arrival rates. This effect is illustrated by the utilization peaks exceeding the delegation threshold highlighted in the upper left corner of Fig. 6 (black arrows). In order to evaluate efficiency, it is important to analyze these peaks. This is done by looking at the peak duration, i.e., the duration where the flow table utilization of DS stays above the delegation threshold after the threshold is exceeded. The CDF at the right side of Fig. 8 characterizes all peak durations for all experiments. Duration is measured in decision cycles, i.e., the minimum frequency with which the decision engine selects new EvPorts (1 second). The results show that peak utilizations above the delegation threshold can be resolved within 6 cycles in 84.6% of the cases. This behavior can be prevented in exchange for additional controller communication by explicitly deleting migrated rules in ES.

In the last part of the functional evaluation, we show that PBCE is capable of providing effective countermeasures for flow table over-utilization for various scenarios. We therefore analyze the flow table utilization of DS for all experiments where PBCE is enabled. The results in Fig. 8 (left plot) show that the prototypical implementation is able to keep the maximum average utilization (i.e., the maximum value for average utilization measured within all experiments for the given threshold, printed as a red line with square markers) below the intended value independently of delegation workloads and flow arrival rates. Note that the flow table utilization of DS will always exceed the delegation threshold for a short time frame, because the delegation mechanism needs to be executed and new flows may arrive in the meantime, which is especially true for scenarios with high and bursty flow arrival rates. This effect is illustrated by the utilization peaks exceeding the delegation threshold highlighted in the upper left corner of Fig. 6 (black arrows). In order to evaluate efficiency, it is important to analyze these peaks. This is done by looking at the peak duration, i.e., the duration where the flow table utilization of DS stays above the delegation threshold after the threshold is exceeded. The CDF at the right side of Fig. 8 characterizes all peak durations for all experiments. Duration is measured in decision cycles, i.e., the minimum frequency with which the decision engine selects new EvPorts (1 second). The results show that peak utilizations above the delegation threshold can be resolved within 6 cycles in 84.6% of the cases. Less than 2% of the cases require 10 or more cycles to push the flow table utilization back below the delegation threshold.

C. Control Plane Implications

1) Computational Overhead: This section analyzes the overhead related to CPU consumption. We show that PBCE scales well with different delegation workloads and computational overhead is below 5% for smaller table ratios. To
calculate the overhead of a PBCE-enabled experiment, we determine the CPU time (seconds) of the Ryu controller process for the total runtime of that experiment and compare it to an appropriate baseline experiment. The baseline experiment has (approximately) the same average flow arrival rate but PBCE and all monitoring is disabled, except for aggregate flow table statistics. Note that we cannot calculate the flow table utilization of the baseline experiment without aggregate flow table statistics. Because the monitoring frequency has a great influence on CPU consumption and control traffic, we consider experiments with two different frequencies $m=3$ and $m=6$ (i.e., gathering of individual flow statistics every $m$ seconds). The results in the top left of Fig. 9 show that the amount of CPU time increases linearly with the average flow arrival rate and higher delegation workloads require more CPU time. The plot in the top right of the same figure shows CPU consumption for table ratios between 0 and 14, divided into four different flow arrival rate classes $c1$-$c4$ where $c1$ represents experiments with an average arrival rate of 50-80 flows/sec ($c2=80$-130 flows/sec, $c3=130$-180 flows/sec and $c4=180$-250 flows/sec). While higher flow arrival rates lead to higher CPU consumption, the differences in CPU consumption for increasing table ratios inside one flow arrival rate class are quite small and follow a linear trend.

A more detailed CPU overhead analysis can be seen in Fig. 10. The CDFs in the top contain the deviation of CPU consumption between PBCE-enabled and baseline experiments. Overhead is expressed as a percentage, grouped by table ratio TR and monitoring frequency $m$. The results show that low delegation workloads induce low computational overhead, which seems reasonable because PBCE is inactive most of the time. For $m=6$ and $TR < 0.1$ the overhead is below 2.98% for 95% of the experiments and below 1.53% for 80% of the experiments. Higher workloads and higher monitoring frequency increase the computational overhead, e.g., for $m=3$ and TR between 0.5 and 1, overhead is below 17.25% for 95% of the experiments. Extreme delegation workloads with table ratios greater than 5 induce more, but still acceptable overhead (27.85% for $m=3$ and 18.75% for $m=6$ in 99% of the cases).

2) Control Traffic Overhead: Similar to computational overhead, the overhead in terms of control traffic is low for smaller table ratios. We use the same methodology and input data and determine the total amount of exchanged control traffic between SDN controller and switches for PBCE-enabled and baseline experiments (in MBytes). Control traffic is partitioned into receiving direction (RX) and transmitting direction (TX). RX consists of the control traffic sent from the switches and received by the controller and is dominated by monitoring results, e.g., aggregated and detailed flow statistics that are queried periodically. TX covers only the control traffic sent from the controller and consists of OpenFlow messages to install and delete rules (and monitoring queries). The two plots in the bottom of Fig. 9 show that especially the overhead for the receiving direction (Ctrl RX) increases with higher table ratios. For high flow arrival rates and high delegation workloads, we see 15 MBytes of additional received control traffic compared to less than 3 MBytes of additional traffic in the transmitting direction. This can be explained with the considerable amount of monitoring data required by PBCE, which is disabled for baseline experiments.

The detailed overhead analysis for control traffic is shown in the bottom of Fig. 10. For $m=6$ and TR < 0.1, control traffic overhead is below 5.03% for receiving direction and below 0.9% for transmitting direction (95% of the experiments). While the overhead for the transmitting direction stays low in case of higher delegation workloads (always less than 10%, even for $m=3$ and TR > 5), the overhead for the receiving direction increases significantly and reaches 86.91%
for extreme table ratios and high monitoring frequency. Note that PBCE can utilize already existing sources for monitoring data to reduce the overhead in the receiving direction, e.g., routinely executed monitoring necessary to establish the global network view of SDN.

3) Monitoring Tradeoff: We now show that the monitoring frequency is closely linked to computational overhead and the amount of traffic that must be redirected to ES. We therefore execute another experiment with static table ratio (TR = 1) and average flow arrival rate (100 flows/second) and gradually reduce the monitoring frequency. The results in Fig. 11 show that the computational overhead is significantly lower if the frequency is reduced. Less monitoring, however, affects traffic optimizations on top of the physical link between ES and DS because high bandwidth flows are not detected quickly enough. This opens up the field for various tradeoffs and optimizations, e.g., by integrating approaches like PayLess [7] or OpenSample [8].

D. Data Path Implications

Finally, we show that the redirection to a neighboring extension switch does not severely degrade the performance inside the data path. Therefore an experiment was conducted that examines the additional delay induced by delegation. The Distributed Internet Traffic Generator (D-ITG, [9]) is used in this experiment to provide accurate statistics with packet level granularity. The detailed setup is as follows: 300 consecutive D-ITG flows (TCP, constant packet rate of 30Mbit/s) are created between two virtual hosts h1 and h2 where every D-ITG flow lasts for one second before the next one is started. After ten seconds, the port that connects h1 and h2 is selected as an EvPort and the delegation is revoked after another ten seconds. This continues periodically until all 300 flows are processed. Fig. 12 shows the average delay for the 300 D-ITG flows starting with the first flow at index 0. Without delegation, the average end-to-end delay is measured as approximately 0.058 milliseconds, which is reasonable, because forwarding is handled locally inside the Open vSwitch (OVS). This delay is increased to a value between 0.15 and 0.2 milliseconds if delegation is enabled. Note that, in the latter case, packets are transmitted via a physical link to the OVS on a remote server and then back via the same link to DS (cf. test setup in Fig. 5). 0.1ms of additional delay seems acceptable for many scenarios. We compared the results with tests on a physical SDN switch, where the delay is even smaller (less than 0.13ms of total end-to-end delay for delegated flows on a Brocade ICX 6610), because the forwarding is done in hardware and not in software.

![Fig. 9. CPU consumption and control traffic divided in receiving (RX) and transmitting (TX) direction for all PBCE-enabled experiments for flow arrival rates between 100 and 300 flows/second. In addition, the plot in the top right shows the CPU consumption for 1795 experiments with various table ratios grouped by four average flow arrival rate classes from c1=50-80 flows/sec (blue line in the bottom) to c4=180-250 flows/sec (red line in the top).](image)

![Fig. 10. CPU and control traffic overhead for the experiments shown in Fig. 9 grouped by table ratio (TR) and monitoring frequency (m). Control traffic overhead is subdivided in receiving (RX) and transmitting (TX) direction.](image)

![Fig. 11. Tradeoff between monitoring overhead and utilization of delegation ports (100 flows/second, TR=1).](image)
V. RELATED WORK

Using flow aggregation or flow table decomposition to achieve scalability for software defined environments is a well-known research area [1], [2], [10], [11], [12]. Within this scope, DIFANE and CacheFlow are two prominent examples closely related to our solution. In DIFANE [5], all switches are equipped with coarse-grained, low priority partition rules that can encapsulate and redirect incoming traffic to authority switches where the appropriate packet handling is provided. Because the partition rules cover the whole flowspace, packets never have to leave the data plane and powerful partitioning and rule caching is required to assure efficiency. While DIFANE tries to optimize the network as a whole, PBCE explicitly focuses on localized scalability improvements and can be deployed without hardware modifications. CacheFlow [3], on the other hand, presents a scalability solution where large flow tables of software switches are utilized as co-located extension devices. Because it also wants to preserve existing OpenFlow semantics, CacheFlow has to deal with similar problems and we therefore borrowed some of their ideas (e.g., the general idea of the ingress-port-tagging mechanism for backflow rules). We see the main difference between CacheFlow and our solution as twofold: We want to provide greater flexibility by not relying solely on specialized extension devices but by exploiting excess capacity of the existing network infrastructure. And, secondly, our port-based eviction scheme could be a valid alternative to the CacheFlow algorithm (for future work, we plan to implement and compare both approaches). Similar to CacheFlow, [13] also utilizes software switches to improve performance but focuses on faster flow table entry installation. Various other SDN (control plane) scalability approaches exist [1], [14], [15] but—to the best of our knowledge—capacity delegation as it is described here was not yet intensively studied.

VI. CONCLUSIONS AND FUTURE WORK

This paper introduces a novel architecture for port based capacity extensions (PBCEs). We propose a port based rule aggregation scheme to redirect traffic to extension switches and present an easy to implement delegation mechanism for OpenFlow-based SDNs where excess capacity of neighboring switches is utilized to dynamically improve scalability of the flow table. First evaluation results with a prototypical implementation show that the PBCE middleware can efficiently and reliably control the flow table utilization of a delegation switch. Control plane overhead is low for smaller delegation workloads (well below 5%) and only 0.1ms of additional delay is added to the data path (for delegated flows). Important areas for future work include architectural extensions to support light-weight service chaining, delegation of other resources (e.g., optional OpenFlow features or monitoring capabilities) and an in-depth analysis of efficient delegation heuristics. We also plan to evaluate the approach using more complex and realistic topologies and compare the efficiency of PBCE with existing approaches like CacheFlow.

REFERENCES

Performance Modeling of Softwarized Network Functions Using Discrete-Time Analysis

Steffen Gebert, Thomas Zinner, Stanislav Lange, Christian Schwartz, Phuoc Tran-Gia
University of Würzburg, Institute of Computer Science
{steffen.gebert,zinner,stanislav.lange,christian.schwartz,trangia}@informatik.uni-wuerzburg.de

Abstract—The softwarization of networks promises cost savings and better scalability of network functions by moving functionality from specialized devices into commercial off-the-shelf hardware. Generalized computing hardware offers many degrees for adjustment and tuning, which can affect performance and resource utilization. One of these adjustments are the interrupt moderation techniques implemented by modern network interface cards and operating systems. Using these, an administrator can optimize either lower latencies or lower CPU overhead for processing of network traffic. In this work, an analytical model that allows computing relevant performance metrics like the packet processing time and the packet loss for generic virtualized network functions running on commodity hardware is developed. The applicability of the model is shown by comparing its outcome with measurements conducted in a local testbed featuring a VNF that acts as an LTE Serving Gateway (SGW). Based on this model, impact factors like the average packet interarrival time, the interarrival time distribution, and the duration of the interrupt aggregation interval are studied.

Keywords—Discrete-Time Analysis, Performance Modeling, NFV, VNF, Queueing Theory.

I. INTRODUCTION

The trend towards softwarization of networks, especially using Software Defined Networking (SDN) and Network Functions Virtualization (NFV), promises more flexibility and innovation for networks. Network functions running on commercial off-the-shelf (COTS) hardware have many appealing advantages such as easy scale-up or scale-down of computing resources as well as scale-out or scale-in of virtual machines among the available physical hardware. Further, faster release cycles compared to hardware devices are promised.

This high flexibility, however, comes at the expense of performance [4], [13], i.e., a lower packet throughput and longer processing delays of softwarized solutions compared to hardware-based implementations. The usage of particular network functions, for instance within network function chains, however, has stringent performance requirements. Firstly, enough function instances have to be available to handle the corresponding traffic. Secondly, the overall processing delay of a network function should be minimized, particularly in case of large forwarding graphs where such delays sum up.

Forwarding packets from the Network Interface Card (NIC) via the kernel space to a specific network function requires an interrupt to inform the Central Processing Unit (CPU) about the packet arrival. Such an interrupt is costly and prevents normal CPU execution for a couple of μs. To cope with high packet rates of today’s links with rates of almost 2 million packets per second (Gigabit Ethernet) to around 20 million packets per second (10 GE), optimization techniques are required to prevent CPU livelocks. Such techniques, e.g., interrupt moderation or interrupt coalescence, are provided by operating systems and hardware components, i.e., NICs. In order to tune COTS hardware for their particular use case, these techniques allow to adjust the trade-off between the number of interrupts and therewith the overall throughput rate, and the corresponding processing delays. For that, incoming packets are aggregated over a certain time span, resulting in a single transfer of all accumulated packets in a batch from the NIC to the kernel space and then to the corresponding network function. This batch-style packet processing is also used by advanced packet processing mechanisms. Cisco’s Vector Packet Processing (VPP) [2] uses Intel DPDK [5] to first apply busy polling of incoming packets from the NIC. Afterwards, it processes the packet headers of the aggregated packets as vectors, i.e., it processes equal headers on a protocol basis (Ethernet, IPv4, IPv6, ARP, etc.) in parallel rather than processing the complete stack step-by-step for each packet. To understand the impact of performance-relevant parameters on these metrics and in order to allow an adequate dimensioning and a proper performance prediction, appropriate performance models are required.

The contribution of this paper is a discrete-time model for Virtualized Network Functions (VNFs) running in software on commodity hardware. The presented model takes into account interrupt moderation, a technique used by current operating systems and server hardware to reduce the overall number of interrupts. Based on an exemplary network function, a mobile network Serving Gateway (SGW), we determine an empirical service time distribution. We illustrate the applicability of the model by comparing it to measurements for a fixed aggregation interval and varying interarrival times. After that, the impact of different interarrival times, interarrival distributions, and aggregation interval durations on the processing times and the packet loss are presented. The proposed model also allows computing distributions, i.e., mean values, standard deviations, as well as quantiles of the delay distributions.

The remainder of this work is structured as follows: Background information as well as related work is introduced in Section II. The steps involved in processing packets in a x86 system are described in Section III, before an abstract model is introduced in Section IV. After its applicability based on measurements is shown in Section V, exemplary evaluations of the packet processing time and packet loss behavior under different settings are presented in Section VI. Finally, Section VII draws conclusions and outlines future work.
II. BACKGROUND & RELATED WORK

This section discusses related work with respect to the performance of softwareized network functions and corresponding optimization mechanisms. Afterward, interrupt moderation techniques are discussed.

A. Performance of Packet Processing in Software

Applications processing network traffic send and receive data packets through functions provided by the operating system kernel. Accordingly, packets traverse a complex chain of forwarding steps between the NIC, the kernel, and the software application resulting in a specific delay overhead.

One major contributor to these delays are copy operations between the memory of the kernel space and the user space. To reduce this overhead, multiple techniques and frameworks that enable a faster processing of packets in software have been introduced. These approaches, e.g., Netmap [12], ClickOS [10], Intel DPDK [5], or VPP [2] bypass the kernel completely during packet reception, use shared memory buffers to avoid additional copy operations, process packets in batches, or replace the entire network stack. Accordingly, these mechanisms usually speed up specific parts of the stack. An extensive measurement study on the performance of several of the aforementioned mechanisms in case of packet forwarding is conducted in [1].

However, the abovementioned studies have several drawbacks. First, the focus on simple network functions like pure packet forwarding obscures the influence of the processing time spent in the user space on the total processing time. This component, however, might account for the majority of the total processing time. Second, measurements are conducted for very specific use cases and cannot be generalized in order to obtain a holistic evaluation of the proposed mechanisms. Finally, it is impossible to determine the feasibility of an approach without identifying its key performance indicators. Therefore, a model for analyzing the packet processing performance on COTS hardware is required. In addition to providing the capability to derive key performance indicators, model parameters can be tuned in order to represent different acceleration techniques and quantify their effects in the context of different use cases.

Based on such evaluations, it could be decided, which technique offers a good trade-off between complexity of implementation and speedup for a specific network function. As seen in [6], operating modes of network functions exist, in which the overhead of packet handling, and therefore the speedup gained by techniques like DPDK, is negligible.

The model developed in this work is a first step towards a model of packet processing in commodity hardware running a general purpose operating system.

B. Interrupt Moderation

In particular, the previously listed frameworks also help to avoid livelocks [8] that result from the CPU being effectively busy with interrupt handling instead of executing the program that processes incoming data. In order to avoid such livelocks and to reduce the overhead of packet processing in a server, several approaches that apply interrupt moderation have been introduced on operating system side as well as in networking hardware.

The networking stack (New API, short NAPI [8]) in the Linux kernel disables interrupt handling for interrupts related to receiving packets, once the first packet is processed. Followed by that, the NIC queue is polled in assumption that multiple packets arrived in a burst. After a certain number of packets have been processed, or a timeout occurs, interrupts are re-enabled and the process restarts once the next packet arrives.

Hardware-based implementations are offered in many server network adapters. The actual feature set varies between different chipsets. For receive as well as transmit directions, the NIC can hold back interrupts until either a pre-configured number of packets is received or sent, or until a pre-configured time since the first packet starting the batch passed by. Further options allow to define a threshold to differentiate between a low and a high traffic load and to specify options for both of these conditions. Finally, some NICs offer adaptive modes, in which they change their behavior based on the current receive rate.

The effects of interrupt moderation and the reduction of end-to-end delay has been subject of several studies already. The influence on passive and active network measurements is investigated in [11]. By identifying packet bursts, effects of interrupt moderation can be considered when running capacity and delay measurements using commodity NICs. A similar methodology is applied in our work in order to estimate the processing time within the application.

By increasing the interrupt rate, more context switches occur in the CPU, when switching between interrupt handling and data processing. Every context switch comes at a certain cost, especially when code and data are evicted from the CPU caches. [14] estimates a time of 3-4.5 μs for a pure context switch without any computation on a multi-core system and 1.3-1.9 μs when the processes are pinned to one CPU core. When the CPU’s cache lines are not filled, experimental results show context switch delays of 2.2-2.9 μs, when the process is pinned to a specific core and a simple program that writes memory pages is used. With virtualization, the time for context switches is reported to be increased 2.5-3-fold. As a rule of thumb, the author estimates 30 μs for a context switch in real-world scenarios. In contrast, the delays seen in the following are mostly based on one single program being executed, resulting in much lower overhead, as the contents of CPU caches are usually not evicted by code or data of other applications.

Latencies of network communication between two servers are studied in detail in [7]. The authors investigate the contributing factors to latencies in Ethernet-based TCP/IP connections and try to achieve a minimal end-to-end latency. Using a modified Linux kernel, the authors make use of nanosecond-precision timers offered by CPUs to break down the packet transmit and receive latencies for a 1 Gbps and a 10 Gbps Ethernet NIC. Based on measurements and estimations, this study indicates a total receive latency of 7.747 μs for a 1 Gbps card. The main contributors (more than 1 μs) are Interrupt cause register read requirement, SoftIRQ, Wakeup application to process socket information, as well as the example application identifying and acknowledging the received data (ACK the pong received by the remote sender).
III. SYSTEM DESCRIPTION

In order to understand the process of packet processing within a Linux system, an abstracted description is provided in the following. This process, which starts with receiving a packet on the wire and ends with the processed packet being sent over the wire, is also depicted in Figure 1.

Read from media: The network interface card reads data from the transmission media by interpreting electrical or optical signals within the MAC layer and transforms it into packets.

Store in receive queue: These packets are saved into a receive queue implemented in hardware inside the NIC. Multiple such queues can exist and, based on hashing, packets can be distributed among these queues.

Trigger interrupt: In the most simple case, the NIC triggers an interrupt signal to notify the CPU about the arrival after every received packet. Interrupt moderation techniques, which are under study in this work, aim at reducing the number of interrupts by processing multiple packets at once. Depending on the capabilities and configuration of the network card, this batch processing mechanism can be triggered by a timeout, by accumulating a specified amount of received packets, or a combination of both. Some NICs also offer adaptive modes, which adjust timers and batch sizes according to the current packet rate.

Read packet from NIC: As soon as the interrupt is sent, the CPU stops other work in order to load and execute the interrupt service routine of the NIC driver. This code then fetches the batch of packets from the network card. This process, which results in a context switch of the CPU, is rather costly as CPU registers first need to load new code and data. Additionally, this also purges other applications’ code/data and thus introduces overhead. This overhead caused by an interrupt can also lead to livelocks, if all CPU time is spent with interrupt handling. It can be reduced by avoiding interrupts for every single packet at the cost of additional delay.

Store packet in buffer: The packet data is stored in a buffer in RAM, until an application requests them for processing. The size of this buffer is limited to a fixed number of bytes\(^1\). If the application cannot catch up with reading packets, the kernel drops packets. The process of copying packet data from kernel space to user space takes additional time per packet.

Process packet in application: While the application processes the packet, it blocks the CPU.

Send packet: After processing, the packet traverses the same way backwards, until it is finally sent to media. The NIC informs the operating system about this by means of another interrupt.

IV. MODEL

A. Abstract Server Model and Performance Metrics

The queuing model used for the performance analysis of the system outlined in Section III is depicted in Figure 2. It is a generalization of the clocked approach introduced by Manfield et al. [9]. The generation of packets follows an arbitrary distribution \(A\). The packets are stored in a peripheral queue which is assumed to have infinite size. Incoming packets are transferred in a batch to the central queue after a time interval \(\tau\) initiated by the first packet after a batch transfer. The inner queue is then modeled as a \(G1^{\infty}/GI/1 - L\) system and evaluated by means of discrete-time analysis. Distributions of the batch sizes and burst interarrival times are derived in the next subsection.

B. Model of the Peripheral Queue (NIC)

In the peripheral queue, which represents the network interface card, packets are aggregated. The resulting batch is then forwarded to the central queue, which represents the CPU/software.

For the remainder of this work, we use the following notation to distinguish between random variables (RVs), their distributions, and their distribution functions. A random variable is represented by an uppercase letter, e.g., \(X\). The distribution of \(X\) is denoted by \(x(k)\) and is defined as

\[
x(k) = P(X = k), \quad -\infty < k < \infty.
\]

Furthermore, the distribution function of \(X\) is written as \(X(k)\) and is defined as

\[
X(k) = \sum_{i=0}^{k} x(i), \quad -\infty < k < \infty.
\]

Finally, \(E[X]\) denotes the mean of \(X\) and \(*\) refers to the discrete convolution operation, i.e.,

\[
a_3(k) = a_1(k) * a_2(k) = \sum_{j=-\infty}^{\infty} a_1(k - j) \cdot a_2(j).
\]

---

\(^1\)in Linux, `net.core.rmem_max = 131071 bytes`
The following distributions are used for modeling the peripheral queue:
- \( a(k) \): distribution of the packet interarrival time.
- \( r_a(k) \): distribution of the packet recurrence time.
- \( \tau(k) \): distribution of the duration of the aggregation interval.
- \( w_u(k) \): distribution of unfinished work in the system before the arrival of the \( n \)-th batch.
- \( \alpha(k) \): distribution of the interrupt processing delay.
- \( s(k) \): distribution of the interarrival time between batches.
- \( x(k) \): distribution of the batch size.
- \( f_j(k) \): distribution of the time between the start of an aggregation interval and the arrival of the \( j \)-th packet. Since the aggregation interval starts with the arrival of a packet, this time equals the sum of \( j \) interarrival times. The corresponding random variable is referred to as \( F_j \).
- \( w_i(k) \): distribution of the waiting time of the \( i \)-th packet in the peripheral queue.

The first packet arriving after a burst transfer initiates a new aggregation interval. All packets arriving in this time frame are transferred to the inner queue at the end of this interval. Based on the work in [15] and [3], the batch size distribution \( x(k) \) can be computed as follows:

\[
x(k) = \tau(0)\delta(k) + \sum_{m=1}^{\infty} \tau(m) \sum_{i=0}^{m-1} \left( f(k)(i) - f(k+1)(i) \right), k = 0, 1, \ldots \tag{1}
\]

The equation allows calculating the number of arrival events in an arbitrarily distributed time interval. The special case, in which no arrivals are observed in an interval of length 0, is covered by the first term. The function \( \delta \) is defined in Equation 2.

For the remaining interval lengths, the law of total probability is used in the second term in order to calculate the conditional probability \( x(k|m) \). It can be derived from the relationship shown in Equation 3.

\[
\delta(k) = \begin{cases} 
1 & k = 0 \\
0 & \text{otherwise} \end{cases} \tag{2}
\]

\[
x(k|m) = P \left( F(k) < m \leq F(k+1) \right) \\
= P \left( F(k) < m \right) - P \left( F(k+1) < m \right) \\
= \sum_{i=0}^{m-1} \left( f(k)(i) - f(k+1)(i) \right), m > 0 \tag{3}
\]

Since the first packet after a transfer initiates the next aggregation interval, the batch interarrival time \( s \) can be calculated as the sum of the recurrence time of one packet, i.e., \( r_a \), and the duration of the aggregation interval \( \tau \):

\[
s(k) = r_a(k) \ast \tau(k) \tag{4}
\]

Since the first packet in a batch triggers the timeout, the waiting time of consecutive packets is reduced. In particular, the waiting time of the \( i \)-th packet in the peripheral depends on the arrivals of the \( i-1 \) packets before it. Hence, the distribution of its waiting time can be computed as follows:

\[
w_i(k) = \pi_0 \left( \tau(k) \ast a(-k) \ast \cdots \ast a(-k) \right) \tag{5}
\]

C. Model of the Central Queue (CPU/software)

We model the inner queue as a \( GI[X]/GI/1 - L \) queue, i.e., a system with batch arrivals and bounded delay. The waiting time of packets is limited to a maximum value of \( L \), i.e., customers who arrive and would have to wait longer than \( L - 1 \) are rejected. Our analysis extends the work presented in [16] by introducing batch arrivals. A similar notation, as presented in the following, is used:

- \( u_{n,b_i}(k) \): distribution of unfinished work in the system before the arrival of the \( i \)-th packet of the \( n \)-th batch.
- \( B_{n,i} \): RV for the service time of the \( i \)-th packet of the \( n \)-th batch.
- \( p_s \): average blocking probability per packet.
- \( \pi_0(\cdot) \): sweep operator which sums the probability mass of negative unfinished work in the system and appends it to the state for an empty system.

\[
\pi_0(x(k)) = \begin{cases} 
x(k) & k > 0 \\
\sum_{i=-\infty}^{0} x(i) & k = 0 \\
0 & k < 0 \end{cases}
\]

- \( \sigma^m(\cdot) \): operator which truncates the upper part of a probability distribution function.

\[
\sigma^m(x(k)) = \begin{cases} 
x(k) & k \leq m \\
0 & k > m \end{cases}
\]

- \( \sigma_m(\cdot) \): operator which truncates the lower part of a probability distribution function.

\[
\sigma_m(x(k)) = \begin{cases} 
0 & k < m \\
x(k) & k \geq m \end{cases}
\]

The development of the batch arrival process is illustrated in Figure 3. Observing the packets of the \( n \)-th batch arrival, the \( i \)-th packet of the burst is accepted if the current unfinished work in the system is less than \( L - 1 \). In case the packet is accepted, the unfinished work is increased by the amount of work \( B_{n,i} \) that is required to process the packet. Otherwise, the packet as well as the remaining packets of the current batch are rejected.

The following recursive relationship can be used in order to compute the amount of unfinished work in the system:

\[
u_{n,b_i}(k) = \pi_0 \left( \tau(k) \ast a(-k) \ast \cdots \ast a(-k) \right) \tag{5}
\]

\[
u_{n,b_{i+1}}(k) = \sigma^{L-1} \left[ \pi_0(\cdot) \right] + \sigma_L [u_{n,b_i}(k)] \tag{6}
\]

\[
u_{n,b_i}(k) = \sigma^{L-1} \left[ \pi_0(\cdot) \right] + \sigma_L [u_{n,b_i}(k)] \tag{7}
\]
Hence, the remaining unfinished work in the system at the arrival of the next batch can be computed as:

\[
u_{n+1}(k) = \pi_0 \left( \sum_{i=1}^{\infty} x(i) \cdot u_{n,b_i}(k) \right) \ast s_n(-k)
\]  

(8)

Using these equations, an algorithm for calculating the workload prior to the \( i \)-th arrival can be derived. The algorithm can be used for both stationary and non-stationary traffic conditions. Under stationary conditions, the index \( n \) and \( (n + 1) \) in these equations can be suppressed, cf. Equation 9. Furthermore, we assume that the packet service time is independent of a packet’s position within the batch. Hence, the RV \( B_n \) refers to the service time for packets in the \( n \)-th batch. Similarly to Equation 9, the index \( n \) can also be suppressed under stationary conditions, resulting in RV \( B \).

\[
u(k) = \lim_{n \to \infty} \nu_n(k) \\
\nu_b(k) = \lim_{n \to \infty} \nu_{n,b}(k)
\]  

(9)

The computational diagram of the system is depicted in Figure 4. Depending on the batch size \( X \), the unfinished work after a batch arrival can be determined by following the corresponding path through the diagram. Each of the \( X \) phases in such a path represents the relationship from Equation 7. Finally, the batch interarrival time \( s_n \) is taken into account and the \( \pi_0 \) sweep operator is used in order to ensure that a proper probability distribution is returned.

It is also possible to quantify the load \( \rho \) of the central queue. This is achieved by calculating the ratio between the amount of work that arrives within a given time interval and the amount of work that is processed in this interval. In particular, we observe that the amount of work that arrives within a batch interarrival time depends on the batch size and the packet service time (cf. Equation 10). Note that both the batch size and the batch interarrival time are affected by the packet interarrival time (cf. Equations 1 and 4).

\[
\rho = \frac{E[X] \cdot E[B]}{E[S]}
\]  

(10)

Finally, the packet loss probability in statistical equilibrium can be computed as follows:

\[
p_b = \sum_{i=1}^{\infty} \left( \frac{1}{i} \cdot x(i) \cdot \sum_{j=L}^{\infty} u_{b_j}(j) \right)
\]  

(11)

Depending on the batch size and the amount of unfinished work added by each packet within the batch, the blocking probability for the latter packets within the batch increases.

### D. Combined Model

Using the two models described in Section IV-B and Section IV-C, it is possible to determine the distribution of the total processing time. It is comprised of the waiting time in the peripheral queue, the waiting time in the central queue, and the service time in the latter. The waiting time in the central queue can be calculated from the unfinished work in the system and a packet’s position in its batch. Hence, the following equation can be used to calculate the distribution of the total processing time of the \( i \)-th packet in a batch, \( d_i \):

\[
d_i(k) = \nu_i(k) \ast \nu(k) \ast b(k) \ast \cdots \ast b(k) \quad i \text{ times}
\]  

(12)

Consequently, the distribution of the total processing time for all packets can be determined via conditional probabilities:

\[
d(k) = \sum_{i=1}^{\infty} P(X = i) \cdot d_i(k) = \sum_{i=1}^{\infty} x(i) \cdot d_i(k)
\]  

(13)

### V. Applicability of the Proposed Model

In order to assess the goodness of fit of the introduced model, measurements are conducted in a test bed and compared with the model’s predictions. In the following, the components of this test bed are described alongside the methodology for accurately measuring the CPU processing times as well as the results of the comparison.
A. Testbed Setup

The testbed setup is depicted in Figure 5. The SGW [6] application runs on the Device Under Test (DUT), a server\(^2\) running a recent Linux version\(^3\) equipped with a four-port NIC. Similar to [6], GPRS Tunneling Protocol (GTP) traffic is generated using a hardware traffic generator\(^4\). In order to evaluate per-packet processing times, wiretaps that duplicate all traffic are placed between the traffic generator and the receiving NIC of the DUT, as well as between the emitting NIC of the server and the traffic sink, which is again the traffic generator. The wiretaps are connected to a hardware capture card\(^5\), which provides nanosecond precision timestamping of received traffic.

The processing time of the server is measured by calculating the time between a packet’s arrival at the first wiretap and its arrival at the second wiretap. The packets at the two wiretaps are matched based on a unique 40 byte signature that the traffic generator adds to every packet. In order to verify that the traffic generator emits packets at equidistant times and at the correct rate, the interarrival times seen at the first capture card are inspected.

The delay, how long the NIC buffers incoming packets, is adjusted using the \texttt{ethtool} command. In this context, \texttt{<N>} represents the number of the NIC and \texttt{<T>} reflects \(\tau\), i.e., the number of microseconds to wait after the first incoming packet:

\begin{verbatim}
# ethtool -C eth<N> rx-usecs <T>
\end{verbatim}

B. Estimating CPU Processing Time

In order to determine the processing time of the application code at a per-packet granularity, measurements using \texttt{tcpdump} are conducted. The time, when \texttt{tcpdump} captures a packet is on the kernel level, right after the interrupt is handled in incoming direction (from NIC 1), i.e., before the packet is copied by the kernel driver code to NIC 2 (cf. Figure 1).

One such exemplary measurement displaying the time between a packet’s arrival at the receiving and the sending side is shown in Figure 6(a). The two batches of packets each show an increasing processing time, as the first packet is processed first by the CPU and the last (10th) packet is processed after all others in this batch. Therefore, the difference in the processing delay between consecutive packets equals \(B_{a,i}\), the waiting and processing time in the application.

Given the application used in our experiments, a prototypical VNF implementation of a mobile network Service Gateway, the measurements result in a distribution of processing times with a mean of 8.336 \(\mu s\). This empirical distribution is used in the following after capping it at the 90\% quantile (16 \(\mu s\)) to remove outliers, resulting in a mean of 7.25 \(\mu s\). This distribution is shown as the red curve in Figure 6(b) and was picked as a representative from multiple measurements. The gray CDFs, as well as the corresponding means (dashed lines) show the CPU processing times of other measurements and highlight the variations between the different runs.

C. Comparison of Model Predictions and Measurements

In order to demonstrate the applicability of the proposed model, we compare its results with measurements. For that, we conduct five independent measurement runs for constant interarrival times between 5 and 12 \(\mu s\). Each measurement run lasts one minute, and the aggregation interval is set to \(\tau = 200 \mu s\).

The size of the central queue, denoted by \(L\), corresponds to 5,200 \(\mu s\) of unfinished work. Based on the measured mean service time at the CPU \(E[B] = 8.336 \mu s\), the inner queue size of \(L_{\text{byte}} = 131.071\) byte as defined by the operating system, and the packet payload of 210 byte, \(L\) computes as follows:

\[ L = E[B] \cdot \frac{L_{\text{byte}}}{210} = 5200 \mu s \]

Based on the measurements, we compute the mean processing times and the corresponding confidence intervals on a 95\% confidence level, as well as the packet loss probability. Additionally, we compute the mean processing times and the packet loss probability using the analytical model. As service time distribution, we take the empirically measured service

\[\text{delay between consecutive packets equals } B_{a,i}, \text{ the waiting and processing time in the application.}\]

\[\text{Given the application used in our experiments, a prototypical VNF implementation of a mobile network Service Gateway, the measurements result in a distribution of processing times with a mean of 8.336 } \mu s. \text{ This empirical distribution is used in the following after capping it at the 90\% quantile (16 } \mu s) \text{ to remove outliers, resulting in a mean of 7.25 } \mu s. \text{ This distribution is shown as the red curve in Figure 6(b) and was picked as a representative from multiple measurements. The gray CDFs, as well as the corresponding means (dashed lines) show the CPU processing times of other measurements and highlight the variations between the different runs.}\]

\[\text{C. Comparison of Model Predictions and Measurements}\]

\[\text{In order to demonstrate the applicability of the proposed model, we compare its results with measurements. For that, we conduct five independent measurement runs for constant interarrival times between 5 and 12 } \mu s. \text{ Each measurement run lasts one minute, and the aggregation interval is set to } \tau = 200 \mu s.\]

\[\text{The size of the central queue, denoted by } L, \text{ corresponds to 5,200 } \mu s\text{ of unfinished work. Based on the measured mean service time at the CPU } E[B] = 8.336 \mu s, \text{ the inner queue size of } L_{\text{byte}} = 131.071\text{ byte as defined by the operating system, and the packet payload of 210 byte, } L\text{ computes as follows: }\]

\[ L = E[B] \cdot \frac{L_{\text{byte}}}{210} = 5200 \mu s \]

Based on the measurements, we compute the mean processing times and the corresponding confidence intervals on a 95\% confidence level, as well as the packet loss probability. Additionally, we compute the mean processing times and the packet loss probability using the analytical model. As service time distribution, we take the empirically measured service delay between consecutive packets equals \(B_{a,i}\), the waiting and processing time in the application.

\[\text{Given the application used in our experiments, a prototypical VNF implementation of a mobile network Service Gateway, the measurements result in a distribution of processing times with a mean of 8.336 } \mu s. \text{ This empirical distribution is used in the following after capping it at the 90\% quantile (16 } \mu s) \text{ to remove outliers, resulting in a mean of 7.25 } \mu s. \text{ This distribution is shown as the red curve in Figure 6(b) and was picked as a representative from multiple measurements. The gray CDFs, as well as the corresponding means (dashed lines) show the CPU processing times of other measurements and highlight the variations between the different runs.}\]

\[\text{C. Comparison of Model Predictions and Measurements}\]

\[\text{In order to demonstrate the applicability of the proposed model, we compare its results with measurements. For that, we conduct five independent measurement runs for constant interarrival times between 5 and 12 } \mu s. \text{ Each measurement run lasts one minute, and the aggregation interval is set to } \tau = 200 \mu s.\]

\[\text{The size of the central queue, denoted by } L, \text{ corresponds to 5,200 } \mu s\text{ of unfinished work. Based on the measured mean service time at the CPU } E[B] = 8.336 \mu s, \text{ the inner queue size of } L_{\text{byte}} = 131.071\text{ byte as defined by the operating system, and the packet payload of 210 byte, } L\text{ computes as follows: }\]

\[ L = E[B] \cdot \frac{L_{\text{byte}}}{210} = 5200 \mu s \]

Based on the measurements, we compute the mean processing times and the corresponding confidence intervals on a 95\% confidence level, as well as the packet loss probability. Additionally, we compute the mean processing times and the packet loss probability using the analytical model. As service time distribution, we take the empirically measured service

\[\text{delay between consecutive packets equals } B_{a,i}, \text{ the waiting and processing time in the application.}\]
time from Figure 6. As interrupt overhead, we use $\sigma = 4 \mu s$, which is based on the values reported in [7].

Figure 7(a) shows a comparison of the measurements and values obtained from the analytical model. Error bars denote the 95% confidence intervals from five measurement runs. The bars indicate the mean processing time per packet according to the model.

For packet interarrival times below 8 $\mu s$, the error bars overlap with the mean values from the model, indicating the applicability of the model. For larger interarrival times, only a slight difference is observed between the model’s prediction and the measurements. A possible explanation for this phenomenon is the high degree of variance regarding the empirical function service times shown in Figure 6(b).

In an analogous fashion, Figure 7(b) shows the applicability of the model w.r.t. to the packet loss rate. Except in the case of $E[A] = 7 \mu s$, the error bars overlap with the values from the model. Based on the huge error bar seen in the previous figure, this interarrival time roughly corresponds to the maximum rate that the server can handle and the first occurrence of packet loss can be observed. For larger interarrival times, the model and measurements both indicate zero packet loss.

The occurrence of packet loss for an average interarrival time below 8 $\mu s$ is consistent with the definition of the system load $\rho$ in Equation 10 and the mean service time at the CPU of 7.25 $\mu s$ that is obtained after removing outliers. Since the average batch size $E[X]$ can be determined by means of $\tau$ and $E[A]$, and the recurrence time in the context of very low interarrival times is negligible, the system load can be approximated as follows:

$$\rho = \frac{E[X] \cdot 7.25}{E[S]} = \frac{\tau \cdot 7.25}{E[A](E[R_a] + \tau)} \approx \frac{7.25}{E[A]} \quad (14)$$

Hence, in the context of mean interarrival times below 7.25 $\mu s$, the system load is larger than 1, and packet loss occurs. Furthermore, the actual load is slightly higher due to the fact that the same CPU core also handles the interrupts that are caused by outgoing packets. This amount to roughly 20,000 IRQs per second in our scenarios.

VI. EVALUATION

In this section, we investigate the behavior of the packet processing server based on the introduced model. In this context, we focus on the total processing time $D$ and the packet loss probability $p_b$. The influence of the mean packet interarrival time $E[A]$ and the length of the aggregation interval $\tau$ are studied. At first, coarse-grained analyses of the resulting mean processing times and packet loss ratios for different interarrival time distributions and aggregation interval lengths are presented. Afterwards, we investigate the impact of these two influence factors for a particular packet interarrival time distribution on the distribution of processing times.

A. Impact of the Arrival Process

The sensitivity of the modeled system to different distributions of the packet interarrival time $A$ is studied based on four different distributions, namely deterministic (det), Poisson (pois), geometric (geo), and negative binomial (nbin). While for det, pois, and geo, the distributions are characterized solely by $E[A]$, the parameters $p$ and $r$ of nbin are adjusted so that $\sigma = \mu$ holds true. This ensures a constant coefficient of variation equal to 1.

1) Impact on Mean Processing Times: Figure 8 presents the mean packet processing time $D$ that results from different combinations of the distribution of packet interarrival time and its mean. While the x-axis displays the mean packet interarrival time, the y-axis indicates the average packet processing time. Additionally, line colors represent different values of the aggregation interval length $\tau$ and line styles correspond to the four distribution types.

In most cases, the curve shape is composed of three phases. First, small packet interarrival times result in high processing times that stem from long waiting times in the central queue. As soon as the average interarrival time exceeds $\tau$, in most cases, each batch is comprised of only one packet. As this packet initiated a new aggregation interval, it has to wait until the timer ends after $\tau$. Because of the low rate, the unfinished work at the central queue (the CPU) is low or oftentimes zero, resulting in immediate processing of the packet. Since in this case the processing time in the central queue is relatively low compared to the waiting time in the peripheral queue, the total processing time is mostly influenced by $\tau$. For interarrival times that follow a deterministic or a Poisson distribution, most aggregation intervals contain exactly one packet, resulting in processing times that are slightly higher than $\tau$. In contrast, the negative binomial and geometric distributions lead to bursts.
of packet arrivals that result in lower mean processing times. After the first packet of a batch starts the aggregation interval, consecutive packets still arrive within the interval and thus, have a lower waiting time in the peripheral queue.

For interarrival times that are lower than \( \tau \), but do not lead to queuing at the central queue, expected batch sizes for all distributions are larger than one. Therefore, the mean waiting time in the peripheral queue decreases and thus, the mean overall processing time \( E[D] \) also decreases.

Although the figure might suggest that decreasing \( \tau \), i.e., reducing the interrupt moderation, leads to lower processing times, this is only true until reaching a break-even point. Then, the overhead per packet caused by interrupt handling and context switches accounts for the majority of CPU time.

2) Impact on Packet Loss: As described previously, the processing time increases with the number of packets per second, because packets experience a waiting time at the central queue. As this queue is limited by \( L \) (cf. Section III), packet loss occurs once this limit is exceeded as described in Equation 11. In the following, the impact of the mean and distribution of interarrival times on the packet loss probability is evaluated.

Figure 9 depicts the packet loss probability for the four different distributions depending on different mean processing times and lengths of the aggregation interval. It can be observed that the Poisson distributed interarrival times result in the highest packet loss ratio when the system operates at a high load. The assumption behind applying interrupt moderation techniques is a certain burstiness of traffic. Hence, the packet loss ratio is up to 8\% lower for nbin than for geometrically distributed arrivals in the case of \( \tau = 200 \mu s \) depicted in Figure 9(a). Due to the higher degree of burstiness of the former, longer idle times after \( \tau \) finished occur and thus fewer interrupts are triggered.

As described in Section V, the CPU load exceeds 1 when \( E[A] \) falls below 7.25 \( \mu s \) (cf. Equation 14). This fits with the observed packet loss at \( E[A] \leq 7 \mu s \) for all distributions. In case of nbin, packet loss occurs even at \( E[A] = 8 \mu s \) due to the higher burstiness of the traffic.

However, it is questionable whether this system can be operated in overload conditions with a packet loss ratio of more than 5\%, which occurs for interarrival times of 7 \( \mu s \) and less, corresponding to more than 142,857 packets per second. Thus, the lower rate with \( E[A] = 8 \mu s \), when no packet loss occurs for all distributions except nbin, is more interesting. The reason for this behavior is, again, its burstiness and higher variation, resulting in short overload situations that lead to packet loss. In contrast, the other distributions result in more equally spaced arrivals.

For the largest aggregation interval of 500\( \mu s \), this effect is visible even for higher values of \( E[A] \). Caused by the higher expected number of packets per batch \( \tau \cdot E[A] \), the probability that packets are dropped in the central queue is increased, resulting in a higher packet loss ratio.

B. Processing Time Distributions for Varying Interarrival Times

In addition to studying the influence of the arrival process on the mean processing time, we also investigate its effect on the distribution of the processing time. Figure 10 shows the CDFs of the processing time \( D \) given an aggregation interval of \( \tau = 100 \mu s \) combined with different arrival processes and values for the mean interarrival time \( E[A] \).

For the lowest mean interarrival time of 4\( \mu s \) shown in Figure 10(a), i.e., the scenario with the highest system load, the highest processing times are observed. Furthermore, the distribution of processing times in this scenario has a low variance and similar values independent of the arrival process. This can be explained by the combination of the very high load and the fact that the system drops packets that encounter a full queue. In contrast, the distribution of the processing time in the context of \( E[A] = 8 \mu s \) differs significantly across different distributions of the interarrival time. On the one hand, the relatively stable det and pois distributions result in a narrow range of processing times which is significantly lower than for \( E[A] = 4 \mu s \). On the other hand, the higher degree of variation of the geo and nbin distributions result in a larger variety of batch sizes which, in turn, yield wide intervals of different processing times.

A further decrease of the processing times is observed for the medium interarrival times seen in Figure 10(b). In these scenarios, the distributions resulting from the nbin and geo distributions are closer to each other and begin to converge. This phenomenon can be explained by the evolution of the two arrival processes. For higher values of \( E[A] \), the coefficient of variation of geo approaches 1, i.e., that of the nbin distribution used in this work. Simultaneously, the \( r \) parameter of the nbin distribution approaches 1. Since the geometric distribution is a special case of the negative binomial distribution with \( r = 1 \), the aforementioned convergence can be explained.

Finally, Figure 10(c) displays the processing time distributions in case of \( E[A] = 30 \mu s \) and \( E[A] = 100 \mu s \), respectively. When the mean interarrival time equals the aggregation interval \( \tau \), only size 1 batches are processed in case of a deterministic arrival process. In combination with the fact that arrivals initiate the aggregation intervals, the processing time is dominated by the waiting time in the peripheral queue. For \( E[A] = 30 \mu s \), batches consist of four packets, hence the distribution consists of four segments with similar shapes corresponding to a packet’s position within a batch. The processing time distributions that result from a geometric and a negative binomial distribution converge further when \( E[A] \) is increased and overlap in case of \( E[A] = 100 \mu s \). Processing times resulting from interarrival times that follow a Poisson distribution are lower and closer to those of det rather than geo and nbin which have a significantly higher degree of variation.
VII. CONCLUSION

NFV has many appealing advantages such as easy scale-up or scale-down of computing resources as well as scale-out or scale-in of virtual machines among the available physical hardware. This high flexibility, however, comes at the expense of performance, i.e., a lower packet throughput and longer processing delays. To understand the impact of performance-relevant parameters on these metrics, and in order to allow an adequate dimensioning and a proper performance prediction, appropriate performance models are required.

The contribution of this paper is an analytical model for virtualized network functions running in software on commodity hardware. The model takes into account interrupt moderation, a technique used by current operating systems and server hardware to reduce the overall number of interrupts. Based on an exemplary network function, a mobile network serving gateway, we determine an empirical service time distribution. We illustrate the applicability of the model by comparing it to measurements obtained from a tested deployment of a VNF for a fixed aggregation interval and varying interarrival times. After that, the impact of different interarrival times, interarrival distributions, and aggregation interval durations on the processing times and the packet loss is presented. The proposed model also allows the computation of distributions, i.e., mean values, standard deviations, and also quantiles of the delay distributions can be computed. Therefore, the presented method can be used by administrators to ensure an appropriate operation of network functions based on their needs.

The model itself may be generalized to take into account acceleration techniques like Intel’s DPDK or Cisco’s VPP. This allows comparing heterogeneous network function implementations and selecting the appropriate technique for a specific use case. Furthermore, economic trade-offs between operational metrics and corresponding costs can be investigated.

ACKNOWLEDGMENT

This work has been performed in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1), and it is partly funded by the German BMBF (Project ID 16KIS0474). The authors alone are responsible for the content of the paper.

REFERENCES

Cache the Queues: Caching and Forwarding in ICN
From a Congestion Control Perspective

Dinh Nguyen, Kohei Sugiyama, and Atsushi Tagami
KDDI R&D Laboratories, Japan
{nguyen, ko-sugiyama, tagami}@kddilabs.jp

Abstract—Caching and multipath forwarding are essential ingredients of the Information-Centric Networking (ICN) architecture resulting from ubiquitous content storage and content-based node-by-node forwarding nature of ICN. Much is yet to see how they jointly act towards improving user’s quality of experience. To this end, we formulate in this paper a unified problem of caching and multipath forwarding as a network optimization problem to maximize user satisfaction which is expressed by their utility function. The formulation allows us to see caching and multipath forwarding in ICN from a congestion control perspective and to reinforce the advantage of a class of congestion-aware caching where in-network caches are used to absorb network congestion and to enhance user’s satisfaction. In that context, multipath forwarding plays the role of directing content delivery to less congested paths where network capacity is abundant or where requested content has been stored by in-network caches. We evaluate such a congestion control-based, coupled caching and multipath forwarding approach in simulations. The result confirms the advantage of our approach compared with existing ones.

I. INTRODUCTION

Ubiquitous in-network caching adopted in the new information-centric networking (ICN) architectures [1] is re-viving research interest in caching and calling for new approaches to fully integrate it into network operations. A large body of research in ICN has been devoted to address the question of cache management, i.e. cache insertion and cache eviction policies, whose main objective is to increase cache hit and reduce server/network load. Recent work begins to move the focus on the latter, not less essential part of the problem: finding the way to fully integrate caching into network functionality.

As having been observed in the literature [2], [3], [4], without proper integration, caching could do more harm than benefit. That is especially true in ICN due to its ubiquity and the interference it causes to other network operations. One evidence is the difficulty of congestion control at the end users when chunk-based in-network caching on the content delivery paths makes conclusion on whether congestion is happening or not ambiguous [5], [6]. In-network caching, moreover, can alter fairness characteristic of the network as a result of working together with AIMD congestion control: bandwidth is allocated in favor of users of popular content and in discrimination against ones who request for less popular content [7]. In case of content streaming, caching can even make it harder for video players to select the right video bitrate [8], [9].

Another important factor in ICN cache management is how to cooperate the network of caches. General speaking, cooperative caching approaches can deliver better performance compared to uncooperative ones with the cost of signaling overhead among caching nodes and/or between caching and forwarding. It is therefore desirable to limit out-of-band signaling to the minimum to reduce implementation complexity, and at the same time, retain an effective level of cooperativeness among individual caches.

In this paper, we devise a caching mechanism which works towards both integration and cooperation issues mentioned above. Our goal is to cooperate caching and multipath forwarding in order to maximize user’s satisfaction. We view caching as a facility for absorbing network congestion, and as a result, increasing user’s quality of experience. To this end, we formulate a unified problem of caching, multipath forwarding, and congestion control with the objective to maximize user utility. Under the formulation, caching and multipath forwarding work in concert, coordinated by the congestion feedback signals, to minimize the congestion incurred to users and to maximize their satisfaction expressed by their utility functions.

A. Background on ICN Caching and Multipath Forwarding

Major efforts have been made following this line of research dealing with other aspects of the combined caching and multipath forwarding problem, different from the one we consider in this paper. The virtual interest packet (VIP) framework [10] employs back-pressure forwarding and popularity caching to maximize the service rate region of a network. Its objective is, therefore, to stabilize the network at high content request rates other than concerning with individual user experience. In fact, back-pressure algorithms in general are known to result in long delay for low-rate requests as it gives absolute preference to high-pressure, popular requests [11], [12]. Furthermore, one matter here is the high-speed signaling among neighboring routers and the speed of cache management and routing decisions in the time scales of milliseconds and microseconds, which places a heavy processing load on content routers. Our proposed approach avoids this overhead by exploiting the available congestion feedback signal for the purpose of caching and multipath forwarding.

Coupling of caching and forwarding has also been advocated in [13] where in-network caching is interestingly shown to have significant benefit, not by itself but jointly with forwarding. The discussion, however, tights to an ideal nearest
replica routing (iNRR) which sends requests to the nearest node where content exists and a leave-copy-down (LCD) caching policy. iNRR requires out-of-band content discovery which is essentially broadcast in nature. The model considered therein does not account for link bandwidth constraints, thus ignoring network congestion. One of the first joint multipath forwarding and congestion control optimizations in ICN [6], on the other hand, presumes a static caching scenario where content placement in caches is preset and does not change (modeled by a mapping function) and solves an optimization problem of multipath multi-source congestion control and forwarding.

In contrast to existing work, we look into the combined problem of caching and multipath forwarding from a different, congestion-centric angle. Motivated by a recent work on congestion caching [14] and promising results reported in our previous work on caching using congestion price feedback [15], we fully develop in this paper a unified framework of caching, multipath forwarding, and congestion control which serves to unite caching and multipath forwarding for the sake of user performance. As we will show in the main parts, congestion feedback helps both (1) coordinate individual caches and (2) couple caching and multipath forwarding with the goal to minimize congestion incurred to content users.

B. Contributions

1) We formulate a unified optimization problem of congestion control, multipath forwarding, and caching in ICN (Section III). To the best of our knowledge, these triple problems have never been combined so far. Such coupling allows us to use congestion feedback for the purpose of caching and multipath forwarding, eliminating the need for out-of-band signaling. 

2) We illustrate the effectiveness of the coupled caching and multipath forwarding coordinated by the congestion control feedback in the network (Section IV).

3) We implement and evaluate the coupled delay-based caching and multipath forwarding scheme in realistic networks by simulations to show its advantage over existing schemes (Sections V and VI).

Compared to our previous work [15], which reported the effectiveness of congestion price as a caching criterion, in this paper we shift our focus to the interaction of caching and multipath forwarding over congestion feedback with new formulation, analysis, and evaluation results explicitly carried out in multipath settings.

II. SYSTEM MODEL

We start with a general CCN/NDN system model as a realization of ICN. We assume a typical CCN system [16] of which users are equipped with a congestion control algorithm regulating the flow of requests for chunks of their interested content. Intermediate routers can route requests for chunks of a content object, each to one of the (multiple) faces associated with the content name. To support multipath forwarding, FIB table should allow to associate multiple faces to one content name. Routing in ICN is an active research area (see [17], [18] for examples of current approaches). In this paper, we assume routing information is pre-computed and stored in FIB of all nodes by an independent routing mechanism.

In addition to multipath forwarding, each router has a cache store controlled by a cache management by which content objects (as a whole) are chosen for storing in or purging from cache. All content objects are cacheable. We assume uncoordinated decentralized cache management.

A user requests for a content object at a consumer node which generates and sends requests for chunks of the content. An intermediate router receives requests and forwards them using the set of faces associated with the content name in its FIB table. Request forwarding is made per chunk request.¹ When receiving a packet for delivery from an ingress face, a router immediately forwards the packet to the appropriate egress face. Each egress face has a FIFO sending queue for keeping packets in transmission.

III. PROBLEM FORMULATION

We model the network as a graph $G = \{V, E\}$ where $V$ is the set of nodes and $E$ is the set of directed links. A directed link $i,j$ is from node $i$ to node $j$. The graph is assumed to be connected. Let $N$ be the set of all content retrievals. A flow, i.e. content retrieval, $n \in N$ originates from a node where the requester resides and requests for content $k \in K$ where $K$ is the set of all content objects. Each flow $n$ has a utility function $U_n(\cdot)$ over the flow rate. For elastic traffic such as file download, the utility function is increasing, strict concave, and twice differentiable. Inelastic traffic such as video streaming, on the other hand, can have a non-concave sigmoidal utility function.² The notations in Table I are used in our development. Notation $n$ is used for a flow, i.e. user, and $k$ for a content object.

A. The Unified Problem

We formulate a combined problem of congestion control, multipath forwarding, and caching as a utility maximization problem (Problem I). The objective is to maximize the total utility of all users (1), with capacity constraints on links (2), and flow balance constraints at intermediate nodes (3). In multipath setting, flow rate to a user is the total rate over all incoming links. The inequality in flow balance constraint (3) addresses the case when content is cached at intermediate nodes. If a node hosts or caches a content object, the hosted or cached content is replied from that node without data coming from upstream nodes (4). The last constraint (5) corresponds to cache capacity constraint at each node where $z_i^k$ is a binary variable indicating content $k$ is cached at node $i$ ($z_i^k = 1$) or not ($z_i^k = 0$) and $B_i$ is the cache capacity of node $i$.

¹For simplicity, we assume no Interest aggregation in the problem formulation and analysis but does account for such aggregation in simulations.

²In case of mix elastic and inelastic traffic, multipath forwarding and congestion control should be handle in a more delicate manner to ensure convergence, which requires a new class of algorithms, e.g. [19], for the problems of multipath forwarding and congestion control presented in Sec.III-B. Congestion caching (Sec.III-C), however, works with both types of traffic.
Table I
Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Set of flows (users)</td>
</tr>
<tr>
<td>V, E</td>
<td>Set of nodes, links</td>
</tr>
<tr>
<td>K</td>
<td>Set of content objects</td>
</tr>
<tr>
<td>c_{i,j}</td>
<td>Capacity of link i, j</td>
</tr>
<tr>
<td>x_{i,j}^n (x_{j,i}^n)</td>
<td>Receiving rate (requesting rate) of flow n over link i, j (j, i), (x_{i,j}^n ≥ 0)</td>
</tr>
<tr>
<td>x_{i,j,k} (x_{j,i,k})</td>
<td>Receiving rate (requesting rate) of content k over link i, j (j, i)</td>
</tr>
<tr>
<td>z_k</td>
<td>Caching variable in {0,1} indicating content k is cached at router i or not</td>
</tr>
<tr>
<td>k(n)</td>
<td>The content requested by user n</td>
</tr>
<tr>
<td>U_n(·)</td>
<td>Utility function of flow n</td>
</tr>
<tr>
<td>L(n)/L_i(n)</td>
<td>The set of links from the source of content k(n) to user n or to router i</td>
</tr>
<tr>
<td>L^- (i)</td>
<td>The set of links over which content k is delivered to node i</td>
</tr>
<tr>
<td>b_k</td>
<td>The size of content k</td>
</tr>
</tbody>
</table>

Problem 1 (System).

\[
\max_{x \geq 0} \sum_{n \in N} U_n \left( \sum_{\ell,n \in L(n)} x_{\ell,n}^n \right) \quad (1)
\]

s.t.

\[
\sum_{n \in L(n)} x_{\ell,i,j}^n \leq c_{i,j} \quad \forall i,j \in E \quad (2)
\]

\[
\sum_{i,j \in L(n)} x_{i,j}^n \leq \sum_{i,j \in L(n)} x_{i,j}^n \quad \forall i \in V, \forall n \in N \quad (3)
\]

\[
\sum_{\ell,i \in L^- (i)} \sum_{n,k(n) = k} x_{i,j}^n = 0 \quad \forall i \in V, \forall k \in K ; z_k^i = 1 \quad (4)
\]

\[
\sum_{k \in K} b_k z_k^i \leq B_i \quad \forall i \in V. \quad (5)
\]

Note that unlike multipath forwarding in conventional network which mostly is carried out at the transport layer by end users in the form of aggregating multiple transport connections, forwarding in CCN is node-by-node: Intermediate nodes make decision on which is the next hop to forward a particular request for a content chunk. And also, what makes CCN different from conventional networks is that data go back on the reverse path over which requests come. Congestion control and multipath forwarding are, thus, performed on the request sending direction while caching is on the data receiving direction.

Since caching is assumed to work in a much slow time scale than multipath forwarding and congestion control, we subsequently decompose the problem into two sub-problems: (1) congestion control and multipath forwarding (fixing z = \{z_k^i\}), and (2) congestion caching management (when x = \{x_{i,j}^n\} has reached stable values).

B. Congestion Control and Multipath Forwarding

By fixing cache variables \{z_k^i\}, Problem 1 becomes a multipath congestion control and forwarding problem to maximize user utility as follows.

Problem 2 (Congestion Control & Multipath Forwarding).

\[
\max_{x \geq 0} \sum_{n \in N} U_n \left( \sum_{\ell,n \in L(n)} x_{\ell,n}^n \right) \quad \text{s.t.} \ (2), (3).
\]

This is essentially a multi-source multipath congestion control and node-based multipath forwarding problem similar to the one constructed in [20], [21] which however does not consider multi-source and caching, and the one in [6] where caching is fixed, i.e. content is stored in multiple predefined locations. By converting to its dual, the problem can be broken into sub-problems which are separately solved by each user and each router in a distributed manner as shown in [20], [21]. In short, users adjust their request rates \(x_n\) to maximize their benefit, i.e. the utility minus congestion cost.

Problem 3 (User).

\[
\max \left( U_n \left( \sum_{\ell,n \in L(n)} x_{\ell,n}^n \right) - \sum_{i,j \in L(n)} \lambda_{i,j} x_{i,j}^n \right) \quad (6)
\]

where \(\lambda_{i,j}\) can be interpreted as the congestion price for each data unit traveling over link i, j. The second term in (6), \(\sum_{i,j \in L(n)} \lambda_{i,j} x_{i,j}^n\), is thus the total congestion cost incurred to user n. This problem is solved by the congestion control algorithm at end-users in reaction to congestion feedback [22], e.g. using queuing delay-based TCP Vegas or loss-based TCP New Reno.

In multipath-enabled conventional networks, the problem carried out at intermediate routers is to minimize congestion cost to each destination by splitting traffic among multiple paths to that destination. At equilibrium, congestion prices over paths that carry traffic settle at the same minimum. ICN routers, however, do not know the source and destination of a flow, but only know the content being delivered. The multipath forwarding problem in ICN thus becomes minimizing the cost to retrieve each content object from upstream sources or caches. Let us look at the multipath forwarding problem solved by router i.

Problem 4 (Multipath Forwarding).

\[
\min \ x \quad q_k^i = \mu_k^i \sum_{\ell,i \in L^- (i)} x_{\ell,i}^k \quad (7)
\]

\[
= \sum_{\ell,i \in L^- (i)} \mu_k^i x_{\ell,i}^k. \quad (8)
\]

Here \(q_k^i\) is the congestion cost and \(\mu_k^i\) is the congestion price to retrieve content k at router i. Also, content delivery rate over a link \(x_{\ell,i}^k\) is the aggregate of all flows which request for the content \(x_{\ell,i}^k = \sum_{n, k(n)=k} x_{\ell,n}^n\). Denote \(x_{\ell,j,i}^k\) as the delivery rate of content k to router i over link \(\ell, j, k\) congestion cost of content k at router i is the total cost on all upstream links over which content k is delivered to router i:

\[
q_i^k = \sum_{\ell,j \in L^- (i)} \lambda_{\ell,j} x_{\ell,j,i}^k. \quad (9)
\]
As of (8), for each content \( k \), router \( i \) minimizes congestion cost retrieving the content by controlling the amount of traffic \( x^k_{\ell,i} \), it receives from each upstream neighbor \( \ell \). To do that, router \( i \) chooses the optimal request rate \( \hat{x}^k_{\ell,i} \) it sends to neighboring router \( \ell \) to minimize the cost (that is, splitting the requests it receives among the forwarding faces in FIB table). This process starts from upstream routers near the source or caches and carries on until it reaches the users of the content. The result is that, by means of multipath forwarding, congestion cost is minimized for all users of the content. One elegant solution to this multipath forwarding problem is to let each router \( i \) adjust forwarding weights to its next hops in order to minimize the congestion cost \( q^k_i \) by a gradient algorithm [20].

C. Congestion Caching Management Problem

Caching is, in essence, to replicate and move the source of a content closer to its users. The impact of moving contents closer to users is twofold. First, requesters of the cached content can enjoy higher delivery throughput since the congestion price of the content at the cache, i.e. \( \mu^k_i \), becomes zero. What is more, the load on upstream links \( L^{-}_i (k) \) from the source to router \( i \) where a content object is cached decreases, allowing other users sharing the links to increase their own rates and utilities. Assuming a time-scale separation that allows congestion control and multipath forwarding to settle down, the congestion caching management problem is to figure caching variables \( \{z^k_i\} \) in order to maximize total user utility:

**Problem 5 (Cache Management),**

\[
\max_z U(z) \triangleq \max_{x \geq 0} \sum_{n \in N} U_n \left( \sum_{\ell, n \in L(n)} x^k_{\ell,n} \right)
\]

s.t. (5).

This content placement problem however is combinatorial in nature and requires global knowledge of user utility which is impractical in ICN given the huge number of content requests. Our goal here is to find a congestion-guided cache management which maximizes \( U(z) \) as much as possible without additional coordination and knowledge sharing among routers.

The benefit of caching content \( k \) at router \( i \) is the saving in congestion cost \( q^k_i \) associated with the content delivery from upstream nodes as expressed in (9). Given that routers know the congestion cost retrieving content \( q^k_i \) by accumulating the cost on the right hand side of (9), and denote \( y^k_i = \sum_{\ell, i \in L^{-}_i (k)} x^k_{\ell,i} \) as the total retrieving rate of content \( k \) at router \( i \), we are now able to formulate the following cache management problem to maximize saving in congestion cost.

**Problem 6 (Congestion-Cost Caching),**

\[
\max_z \Phi(z) \triangleq \max_{z} \sum_{i \in V} \sum_{k \in K} \mu^k_i y^k_i z^k_i
\]

s.t. (5).

This saving reflects itself as a reduction in congestion cost to the users, i.e. the second term in (6). Lower congestion cost, e.g. loss probability or queuing delay, in reality means a feedback signal to let the congestion control at the user increase its rate and intermediate routers to divert more traffic to the cheap paths with cached content. The users, as a result, enjoy higher equilibrium rates which maximize their benefit by solving Problem 3. In that sense, caching can be considered as a facility to reduce congestion and to move the system to a new equilibrium where total utility of all users is increased. Eqs. (7) and (10) summarize the two actions of forwarding and caching made by content routers which are coupled by the congestion cost (9) to maximize the total utility of all users.

Problem 6 can be greedily solved by letting each router store the most expensive content in terms of aggregate congestion cost over a predefined period of time \( T \) whose implementation is given in Algo.3. Our conjecture is that such congestion-cost caching is optimal in reducing system-wide congestion which we leave for future work for its proof. In the next section, we make a few important observations on this class of caching.

IV. FEATURES OF CONGESTION-COST CACHING AND MULTIPATH FORWARDING

Cache network analysis is generally hard and is an active research area with works analyzing common caching algorithms such as LRU, FIFO, RANDOM (see e.g. [23], [24] and references therein). Its main difficulty lies in the fact that arrival processes to intermediate caches cannot be accurately characterized. Congestion caching analysis is even more challenging for the reason that caching decisions at a node do not only depend on decisions made at other nodes but are also tied to the congestion level of the network. In this section, we instead provide some important observations on congestion caching and multipath forwarding to illustrate their behaviors under delay-based congestion control. As congestion caching works by accumulating congestion feedback signals over a comparably long period of time which allows congestion control and multipath forwarding to stabilize, it is appropriate to make an fluid approximation of the system. Such a fluid approximation simplifies the analysis and enable us to reach meaningful results.

We assume congestion control ensures fairness of bandwidth allocation. Considering delay-based congestion control such as TCP Vegas and FAST TCP, rates are allocated according to logarithm utility functions \( U_n(x_n) = \alpha_n \log(x_n) \) and the congestion cost can be interpreted as the total queue occupancy at all network links which a given flow occupies [25]. At equilibrium, queue occupancy allocated to flow \( n \) is equal to \( \alpha_n \) and utility derivative balances with the congestion price. That is, \( \alpha_n \mu_n \) where \( \mu_n \) is the mean queuing delay incurred to user \( n \) and \( x_n \) is the equilibrium rate.

A. Standalone Caches

We first consider standalone caches which in practice can be deployed in the form of caches at the network edge. We
extend the static LFU caching analysis in [26] with a concept of queue hit.

Assume there are $K$ content objects, w.l.o.g. sorting in descending order of congestion cost, i.e. queue occupancy in case of delay-based congestion control, $q_1 \geq q_2 \geq \ldots \geq q_K$. Request rates are $r_1, r_2, \ldots, r_K$ and the congestion cost per request are $\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_K$ respectively. For simplicity of expression, in this section we assume same-size content objects and the cache size $m$ is in number of objects. When a content object is found in cache, the network is freed from carrying the cost, i.e. queue occupancy, due to this content, thus we have a queue hit. Let $X_i$ be a random variable denoting the index of content requested by $i$-th request, and $Y_{k,i}$ be the indicator function of whether content $k$ is in the cache at $i$-th request. The queue hit $H_\pi$ of a caching policy $\pi$ is thus

$$H_\pi = \lim_{t \to \infty} \frac{1}{t} \sum_{i=1}^{t} \sum_{k=1}^{K} E[1(X_i = k)Y_{k,i}]\hat{q}_k. \quad (11)$$

With an assumption that $\{Y_{k,i}\}_i$ is stationary and $E[1(Y_{k,i})] = p_k(\pi)$ is the probability that content $k$ is in cache following caching policy $\pi$, we have

$$H_\pi = \sum_{k=1}^{K} r_k \hat{q}_k p_k(\pi)$$

$$= \sum_{k=1}^{K} \hat{q}_k p_k(\pi)$$

$$\leq \sum_{k=1}^{K} \sum_{k=1}^{m} q_k \quad (12)$$

The last inequality comes from the fact that $\sum_{k=1}^{K} p_k(\pi) = m$ and can be proved using a similar technique as in [26]. By construction, queue hit rate of congestion cost caching which stores content with highest congestion cost is $H_{\text{cost}} = \sum_{k=1}^{K} q_k$. Eq.(12) is thus showing that in case of standalone caches, congestion-cost caching is optimal in relieving queue occupancy inside the network.

B. Coordinating Caches by Congestion Cost

Consider the topology in Fig. 1 which represents the case when multiple content objects are delivered along a path. Requests arrive at node 0 with intensity $r_1, r_2, \ldots, r_K$. There is cross traffic arriving at intermediate routers 1, 2, ..., $N-1$ which is modeled by intensity $r_{k,i}$, for content $k$ at router $i$ ($i \geq 1$).

Under delay-based congestion control, congestion cost of a content object observed at a router becomes the total residual queue occupancy which all flows carrying a content object take from the network. The residual queue occupancy of content $k$ at router $i$ can be expressed as

$$q_i^k = \left( r_k + \sum_{l=1}^{i} r_{k,l} \right) \sum_{j=i+1}^{N-1} \lambda_{j+1,j} \quad (13)$$

where $\lambda_{j+1,j}$ is the queuing delay on link $j+1-j$. As all content experiences the same queuing delay $\sum_{j=i}^{N-1} \lambda_{j+1,j}$ along the same path, (13) tells us that at each router congestion-cost caching works the same way as LFU which caches content with highest request rate $r_{k,1} = r_k + \sum_{l=1}^{i} r_{k,l}$. The difference however lies in the way individual caches are coordinated by congestion feedback which does not exist in LFU.

Suppose content $k$ is cached at router $i$ due to high request rate $r_{k,1}$. As a result of caching decision at router $i$, the congestion cost feedback to all downstream routers is reduced. Such lower congestion cost is, in a sense, a coordinating signal which makes the content less likely to be cached downstream. The reduction in congestion cost of content $k$ perceived at downstream router $j$ ($j < i$) is

$$\Delta \lambda_i^k = \hat{r}_{k,j} \sum_{l=i}^{N-1} \lambda_{l,1,l} \quad (14)$$

This distinctive feature of downstream coordination by congestion feedback makes congestion-cost caching more efficient in utilizing cache storage since content is not likely cached again downstream unless it experiences significant congestion along the downstream path.

C. Interaction of Caching and Multipath Forwarding

Multipath topology can be interpreted as overlaid single paths as illustrated in Fig.2. Denote $\mu_{a,b}^k = \sum_{l=1}^{N} \lambda_{a+1,b}^l$, i.e. the total queuing delay from the source to node $a$ on path $a, a \in \{1, 2\}$. As a result of delay-based multipath forwarding, traffic carrying content $k$ is split at router $i$ in such a way that queuing delay is the same over the two paths, i.e., $\mu_{a,b}^k = \mu_{a,b}^k$. Consider the case content $k$ is cached on one path, say at node $\ell_1$ on path 1, e.g. due to high total request rate $\hat{r}_{k,\ell}$. The effect on node $i$ due to such caching decision is that queuing delay retrieving content $k$ over path 1 decreases by $\Delta \mu_{a,b}^k = \sum_{l=1}^{N} \lambda_{a+1,b}^l$. In reaction to this change, node $i$ forwards more requests for content $k$ over path 1 so as to balance queuing delay on the two paths. By equalizing queue

3Suppose both paths carry traffic at equilibrium.
occupancy of content $k$ before caching happens at node $t_1$ with the queue occupancy after caching, the ratio of rate increase can be shown to be approximately

$$w^k_t \propto \frac{\mu_{t+1}^k}{\mu_{t+1}^k - \Delta \mu_{t+1}^k}. \quad (15)$$

Eq. (15) shows the interaction between caching and multipath forwarding in this representative scenario: Regarding the content being cached, caching reduces queuing delay by $\Delta \mu_{t+1}^k$ over the path, and multipath forwarding takes advantage of this reduction by increasing content delivery by a factor of $w^k_t > 1$ to the path where the content is cached.

V. IMPLEMENTATION

A. Delay-based Congestion Control and Multipath Forwarding Implementation

A delay-based congestion control similar to TCP Vegas [27] is deployed at ICN users (Algorithm 1) which consists of two parts: slow-start (lines 4–9) and congestion avoidance (lines 10–14). We note that TCP Vegas works on queuing delay feedback and keeps congestion occupancy of each flow within a preset range $[Q_{\text{min}}, Q_{\text{max}}]$ (lines 12 and 14). Since data packets in ICN can be replied from the source and multiple in-network caches, it is inaccurate measuring queuing delay by subtracting propagation delay $RTT_{\text{min}}$ from RTT as originally proposed in TCP Vegas. Multipath forwarding, furthermore, makes this measurement impossible due to variation of $RTT_{\text{min}}$ among different paths. We thus use explicit queuing delay feedback for congestion control. Accumulated queuing delay is stored in an additional field inside the data packet and updated by routers on the delivery path from the source or in-network caches to the end-user.

One important feature which needs emphasizing is that unlike loss-based congestion control, the delay-based congestion control can work more smoothly with multipath forwarding as the congestion cost, i.e. queue occupancy, can be summed up over individual paths. Loss-based congestion control however incorrectly perceives loss on any one path as congestion signal over individual paths. Loss-based congestion control can work more smoothly with multipath forwarding as the congestion cost, i.e. queue occupancy, can be summed up over individual paths. Loss-based congestion control however incorrectly perceives loss on any one path as congestion signal over individual paths.

For multipath forwarding, to keep the implementation simple, routers use deficit round-robin to split requests among available faces using the inverse of queuing delay as the forwarding weight of each face (Algorithm 2).4

B. Congestion-Cost Caching Implementation

We implement a caching algorithm to solve Problem 6 by which each router stores content objects with highest congestion costs. The cache management algorithm is given in Algorithm 3 which utilizes explicit queuing delay feedback in data packets of each content object denoted as $p(D_k^i)$. Intermediate routers need to accumulate the congestion cost $q^k$ of each content object $k$, i.e. the total congestion cost of all data packets of that content, over a predefined period of time $T$. $T$ is chosen much longer than RTT for the cost to

4Without anticipation, multipath forwarding is shown to have oscillation in [20]. However, the oscillation does not effect caching behavior since caches accumulate congestion cost over long-time periods.
The congestion cost of a given content object is computed by

\[ T \]  
To keep track of change in congestion cost after a content

Within \([T]\) Back-pressure multipath forwarding used

Multipath forwarding using queuing de-

Cache most popular content [10]

AND CONGESTION CONTROL USED IN SIMULA TIONS

Cache content with highest data

Volume (ignoring congestion cost)

150

Cache content with highest conges-

tion cost (Algorithm 3).

Inverse of RTT as forwarding weight [28]

2500

3000

100

1500

200

MULTIPA TH FORW ARDING

2000

50

500

in multiple short intervals

can be reduced by letting routers measure the cost periodically

Overhead collecting congestion cost for the purpose of caching

• The congestion cost of a given content object is computed by

accumulating the congestion price fed back by data packets of the content object for a preset time duration \( T \).

• To keep track of change in congestion cost after a content object has been cached, congestion cost of a content object is increased proportionally to the (byte) hit of the content. Overhead collecting congestion cost for the purpose of caching can be reduced by letting routers measure the cost periodically in multiple short intervals \( T_0 \) within \( T \) (\( T_0 << T \)).

VI. SIMULATION EVALUATION

We create an event-driven C++ simulator based on CCN architecture [16] for evaluating the performance of our congestion-based coupled caching and multipath forwarding in comparison with several combinations of well-known caching algorithms and multipath forwarding strategies (Table II).

A. Caching Behaviors

We first verify the behavior of congestion-cost caching in a bottlenecked topology (Fig.3) with two cascaded bottleneck links of 3Gbps in the middle of the topology. The topology mimics typical content delivery over Internet where there are multiple bottlenecks on the delivery paths. There are three repositories at node 6, 7, and 9 hosting content prefix 1, 2, and 3 respectively, each consists of 2000 content objects (50MB in size). A 2GB cache store is placed at node 2. Poisson request arrivals (Zipf 0.8, intensity 3 request/s) from access network 3, 4, and 1 request for content of prefix 1, 2, and 3 respectively. Simulations last for 3000 seconds.

Completion time distribution by prefix is reported in Fig. 4. Compared with popularity caching (POP) and LCE, congestion-cost caching (COST) has superior performance with shortest completion time across all prefixes. Notably, the proposed cache management (solid lines) maintains similar completion time distribution among all content prefixes. This is a result of selectively caching high congestion cost content, i.e., prefix 3, which significantly reduces network congestion and shortens delivery time of all prefixes.

To see this selective caching behavior and how it reacts to change in congestion cost, we additionally run a second set of simulations in which request rates for content of prefix 3 doubles at time \( t=1500s \) (Fig.5). When request rate increases at \( t=1500s \), more cache storage is used to store content of prefix 3 as the congestion cost associated with this prefix increases with the request rate. Ratio of cache used to store content of prefix 3 raises from about 0.65 to 0.8 which benefits users who request for this content prefix since it can be directly retrieved from the cache and relieves congestion on the two bottlenecks. A similar trend is observed with POP and LCE but the ratio is not as much and as consistent as with congestion-cost caching.

B. Interaction of Caching and Multipath Forwarding

We use a two-level multipath network (Fig.6) which models a simplification of CDN networks for simulations. There are two repositories of prefix 1 and 2 at node 8 and 9 respectively, each hosts 2000 contents (Zipf 0.8). Routes to prefixes 1 and 2 and request intensity from access networks are given in the

<table>
<thead>
<tr>
<th>Cache management &amp; Congestion control</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost: Congestion-cost caching</td>
<td>Cache content with highest congestion cost (Algorithm 3).</td>
</tr>
<tr>
<td>LCE: Leave-Copy-Everywhere</td>
<td>Cache all, LRU eviction</td>
</tr>
<tr>
<td>POP/Popular: Popularity caching</td>
<td>Cache most popular content [10]</td>
</tr>
<tr>
<td>Vol: Highest data volume</td>
<td>Cache content with highest data volume (ignoring congestion cost)</td>
</tr>
<tr>
<td>Vegas</td>
<td>TCP Vegas-like delay-based congestion control (Algorithm 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multipath forwarding</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queuedelay</td>
<td>Multipath forwarding using queuing delay feedback</td>
</tr>
<tr>
<td>VIP: Back-pressure forwarding</td>
<td>Back-pressure multipath forwarding used in [10] (chunk-based)</td>
</tr>
<tr>
<td>PI: Pending interest forwarding</td>
<td>Inverse of number of pending requests as forwarding weight [6]</td>
</tr>
<tr>
<td>RTT: RTT forwarding</td>
<td>Inverse of RTT as forwarding weight [28]</td>
</tr>
<tr>
<td>Singlepath: Shortest path forwarding</td>
<td>Multipath forwarding is disabled</td>
</tr>
</tbody>
</table>
Also here we can see almost all cache
value or 1000 seconds (x 10 runs)
8MB = 0 cache-forwarding-congestion
10000 objects
achieves Back-pressure [10], and DRR [29] (default)
Zipf, As a metric for user
Poisson 50MB / 0.2MB
cache-forwarding cost are compared against combinations of existing
Other simulation parameters are given in Table III.
following the max-flow from the sources to a given router.
weights. The widest links have bandwidth of 5Gbps and
C. Performance in Real ISP Topology
Next, we run simulations in a real, large-scale network
Link bandwidths are set inversely proportionally to the link
weights. The widest links have bandwidth of 5Gbps and
weight of 1. Multipath routing is pre-configured in the network
following the max-flow from the sources to a given router.
Other simulation parameters are given in Table III.

The proposed caching and multipath forwarding using congestion cost are compared against combinations of existing

5Mean ratio in the last 1500 seconds is reported; the first 1500-second period is considered as cache warm-up and discarded.

Starting from equal split without caching (cache size=0),
as cache size at router 6 increases, router 3 forwards more
traffic over the path to router 6 and less traffic to the path
to router 5 since the former path is cheaper in terms of congestion price (Fig. 7 bottom). Also here we can see almost all cache space is used to store content of prefix 1 which is much more expensive to retrieve from the source due to limited link bandwidth of 1.5Gbps on link 8→6 (compared with 3Gbps of prefix 2 over link 9→6). Delay-based multipath forwarding at router 3 does a very good job of balancing queuing delay between the two paths to prefix 1 (Fig. 7 top), which helps direct more traffic via router 6’s cache when the cache size increases and congestion price, i.e. queuing delay, decreases on that path. This verifies the coordinating feature among caching and multipath forwarding mentioned in Sec.IV-C.

C. Performance in Real ISP Topology
Next, we run simulations in a real, large-scale network
Link bandwidths are set inversely proportionally to the link
weights. The widest links have bandwidth of 5Gbps and
weight of 1. Multipath routing is pre-configured in the network
following the max-flow from the sources to a given router. Other simulation parameters are given in Table III.

The proposed caching and multipath forwarding using congestion cost are compared against combinations of existing

5Mean ratio in the last 1500 seconds is reported; the first 1500-second period is considered as cache warm-up and discarded.

Starting from equal split without caching (cache size=0),
as cache size at router 6 increases, router 3 forwards more
traffic over the path to router 6 and less traffic to the path
to router 5 since the former path is cheaper in terms of congestion price (Fig. 7 bottom). Also here we can see almost all cache space is used to store content of prefix 1 which is much more expensive to retrieve from the source due to limited link bandwidth of 1.5Gbps on link 8→6 (compared with 3Gbps of prefix 2 over link 9→6). Delay-based multipath forwarding at router 3 does a very good job of balancing queuing delay between the two paths to prefix 1 (Fig. 7 top), which helps direct more traffic via router 6’s cache when the cache size increases and congestion price, i.e. queuing delay, decreases on that path. This verifies the coordinating feature among caching and multipath forwarding mentioned in Sec.IV-C.

C. Performance in Real ISP Topology
Next, we run simulations in a real, large-scale network
Link bandwidths are set inversely proportionally to the link
weights. The widest links have bandwidth of 5Gbps and
weight of 1. Multipath routing is pre-configured in the network
following the max-flow from the sources to a given router. Other simulation parameters are given in Table III.

The proposed caching and multipath forwarding using congestion cost are compared against combinations of existing

5Mean ratio in the last 1500 seconds is reported; the first 1500-second period is considered as cache warm-up and discarded.

Starting from equal split without caching (cache size=0),
as cache size at router 6 increases, router 3 forwards more
traffic over the path to router 6 and less traffic to the path
to router 5 since the former path is cheaper in terms of congestion price (Fig. 7 bottom). Also here we can see almost all cache space is used to store content of prefix 1 which is much more expensive to retrieve from the source due to limited link bandwidth of 1.5Gbps on link 8→6 (compared with 3Gbps of prefix 2 over link 9→6). Delay-based multipath forwarding at router 3 does a very good job of balancing queuing delay between the two paths to prefix 1 (Fig. 7 top), which helps direct more traffic via router 6’s cache when the cache size increases and congestion price, i.e. queuing delay, decreases on that path. This verifies the coordinating feature among caching and multipath forwarding mentioned in Sec.IV-C.

C. Performance in Real ISP Topology
Next, we run simulations in a real, large-scale network
Link bandwidths are set inversely proportionally to the link
weights. The widest links have bandwidth of 5Gbps and
weight of 1. Multipath routing is pre-configured in the network
following the max-flow from the sources to a given router. Other simulation parameters are given in Table III.

The proposed caching and multipath forwarding using congestion cost are compared against combinations of existing
substantially reduce network load, i.e. queue occupancy at bottlenecked links, and as a result, speeds up content delivery. If caching is done based on the volume of data without considering congestion price, i.e. Vol-queuedelay, completion time is consistently longer compared with Cost-queuedelay which considers both data volume and the congestion price as in (9). Completely replacing congestion-cost caching with LCE, i.e. LCE-queuedelay, has even worse performance. All other uncoupled combinations of caching and multipath forwarding (Popular-VIP, LCE-RIT, and LCE-PI) achieve not as good performance as the proposed Cost-queuedelay. Although Popular-VIP performs comparably well compared to Cost-queuedelay, its performance deteriorates with high arrival rates as it suffers higher loss rate due to queue overflow (Fig. 10).

Hit rate, i.e. the ratio of hits to total number of requests, is reported in Fig.8. As expected, Cost-queuedelay request hit rate is not quite high compared to existing methods since it does not totally prefer high request rate content.

2) Queue Occupancy and Loss Rate: Fig.11 reports queue occupancy as an indicator of the congestion level of the network. Caching and multipath forwarding play a key role in reducing congestion inside the network. The coupled congestion-cost caching with queuing delay forwarding Cost-queuedelay guarantees the lowest queue occupancy among all the schemes at high request rates. As before, replacing congestion-cost caching with LCE (LCE-queuedelay) or volume-based caching (Vol-queuedelay) results in higher queue occupancy compared with Cost-queuedelay. This result highlights the advantage of congestion-cost caching and multipath forwarding coupled by delay-based congestion control to deal with high network load and absorb congestion.

As a direct result of low queue occupancy, loss rate, the ratio of the number of dropped packets due to queue overflow to the number of successfully received packets, is always lowest using the proposed Cost-queuedelay (Fig. 10). Baseline configuration (LCE-singlepath) cannot sustain even the lowest request rate when loss rate has already exploded.

VII. CONCLUDING REMARKS

We have demonstrated the advantages of a joint scheme of caching, multipath forwarding, and congestion control in accelerating content delivery and reducing network congestion. Tightly coupled with congestion control, multipath forwarding and caching can deliver superior performance to users. Multipath forwarding directs traffic to cheaper, less congested paths while caching itself moves expensive, costly-to-retrieve content nearer to users who are requesting it. Here we do not claim either multipath forwarding or caching is more important for the objective of maximizing user performance. It is in fact the joint effect of both on top of congestion control which maximizes user satisfaction. Congestion control feedback provides the implicit coordination between caching and multipath forwarding to achieve such a desired effect. For future work, we plan to implement the proposed scheme using random early marking, reduce the overhead of congestion price feedback, and deploy the scheme in real-world applications.

REFERENCES

Optimizing Time to Exhaustion in Service Providers Using Information-Centric Networking

Ali Shariat∗†, Ali Tizghadam∗, Alberto Leon-García∗
∗University of Toronto, Canada
†ali.shariatmadari@utoronto.ca

Abstract—Exponential traffic growth due to the increasing popularity of Over-The-Top Video services has put service providers under much pressure. By promoting in-network caching, Information-Centric Networking (ICN) is a promising paradigm to answer current challenges in the service provider’s domain. This paper reports on a cache placement strategy for service providers to delay the onset of congestion (time-to-exhaustion) to the extent possible in order to optimize their capital expenditure for their limited capacity planning budget. We show that even a limited deployment of ICN provides a substantial increase in the time-to-exhaustion of the network and a decrease in the number of links with high utilization.

I. INTRODUCTION

Current Internet is a product of four decades of evolution. Today, Internet traffic is rapidly growing due to Over-The-Top (OTT) and Video-on-Demand (VoD) services such as Netflix and YouTube. Video traffic is now consuming most of the bandwidth on the Internet. A more detailed analysis shows that Netflix (31.6%) and YouTube (18.7%) combined, account for over 50% of downstream traffic in fixed access [1]. This growth is changing the architecture of the Internet. The content creators are exploiting the economies of scale and using Content Delivery Networks (CDN) to transfer this huge traffic, which exacerbates the change. CDNs were introduced to overcome the limitations of traditional Web caching systems by deploying several caches throughout the globe and populating these caches with the popular content during the off-peak traffic hours. Some content providers are very keen to work with Service Providers (SP) to provide these caches. For example, Netflix Open Connect program is rapidly expanding its coverage by offering to install and maintain the caches in the SP’s network.

The SPs usually place CDN caches at the Internet Exchange peering points that connect to the core of the SP’s network. Therefore, the CDN architecture is optimized to deliver the content until it reaches the last mile of its path, but it does not solve the inefficient use of the SPs network infrastructure. Even when all of the users are requesting the same content, the network will find and forward the request to the nearest CDN cache. This method has the benefit that SPs have full control over the configuration and placement of the caches. Placing a transparent web cache in different topologies is studied in [5]. The authors model line and ring networks and experimentally study a single web cache case.

Information-Centric Networking [3], [6] is a clean slate networking paradigm that tries to solve current networking problems by replacing the host-to-host communication model and introduces concepts such as Naming, Name-based routing and In-network caching. Named-Data Networking (NDN) [7] is a fully-fledged ICN architecture. In NDN, content is moved and cached between neighbors and when there is an interest for a content, the network will strategically find and forward the interest towards the content store. There is a wealth of literature on cache deployment in the context of ICN [8], [9], [10] some with contradictory results. The authors in [11] provide a mathematical model of the cache miss probability

The rest of this paper is organized as follows. Section II reviews previous works on network design and cache placement. In Section III, the content delivery problem in a service provider is investigated. Further, details of our analytical model are discussed in Section IV. Simulation results and validation are provided in Section V.

II. RELATED WORKS

Content Delivery Networks provide multi-server to multi-client paradigm for everyone. CDNs forward the requests from clients towards the best server. This process heavily relies on DNS, which raises several issues [4]. To better serve the users, CDNs are putting their own caches inside the network of service providers. This architecture is a win-win situation for both the content providers and network operators. For example, Netflix Open Connect program is rapidly expanding its coverage, since they offer to install and maintain the caches for free in the SP’s network. Deploying transparent cache servers directly in the network is another method that is employed by the SPs. This method has the benefit that SPs have full control over the configuration and placement of the caches. Placing a transparent web cache in different topologies is studied in [5]. The authors model line and ring networks and experimentally study a single web cache case.

Information-Centric Networking [3], [6] is a clean slate networking paradigm that tries to solve current networking problems by replacing the host-to-host communication model and introduces concepts such as Naming, Name-based routing and In-network caching. Named-Data Networking (NDN) [7] is a fully-fledged ICN architecture. In NDN, content is moved and cached between neighbors and when there is an interest for a content, the network will strategically find and forward the interest towards the content store. There is a wealth of literature on cache deployment in the context of ICN [8], [9], [10] some with contradictory results. The authors in [11] provide a mathematical model of the cache miss probability
of a single caching system and extend it to a network of caches. In [10], the authors use different centrality metrics for sizing storage in a content-centric networks, but couldn’t find an incentive for heterogeneous caching. The authors in [12] solve a budget constrained caching problem in Content-centric networking context and note that topology has a significant impact on the optimal cache placement. They have considered hop counts as the base metric for optimizing cache placement. Reference [2] studies the evolution of CDN and its challenges and shows how ICN paradigm can help overcoming them.

III. PROBLEM DEFINITION

A. Network of a Service Provider

A service provider network (Fig. 1) usually consists of multiple layers. The core of the network transfers the highest volume of data from various aggregation sites between sources and destinations. Content servers are usually connected to the core through an Internet exchange peering point. The next level is the Aggregation level and is a concentration point of multiple distribution centers, which themselves may be connected to smaller distribution edge centers. The final layer is called Access layer and is directly connected to the consumers. For example, an access layer of a wireless service provider contains several cellular antennas.

Now consider the subscribers that request an OTT video content. For every request for a content, a new TCP connection is created between the content source and the consumer’s machine. Even when all of the users are requesting for the same content, that content is transmitted over the network multiple times. Note that the source of this content may be either controlled by the operator itself or come from a VoD content server owned by a 3rd party CDN. If the content source is a live stream from outside of the network, operators are faced with an even bigger challenge than for VoD content. Live stream content is watched by many consumers at the same time and the operator does not have any control over the content if it is originated from outside of the network. As one can see, this structure is not scalable and is a clear waste of underlying resources.

B. Time-to-exhaustion

In a network with increasing demand, such as service providers, congestion is inevitable. For example, assume that the demand is increasing by 5% every month. Fig. 2 shows such a scenario. If the current demand in the network is 100Gbps and the network can handle a maximum of 400Gbps, the current infrastructure will keep up with the traffic for the next 27 months. To maintain a congestion-free network, while the demand is increasing, a service provider has to invest in its infrastructure.

For a service provider, serving content from a local cache or peering point incurs different costs. In addition, increasing link capacity costs much more than increasing caching storage. At the same time, serving more content to users means more revenue. This demand increase will eventually exhaust the network at some point in the future (Fig. 2) unless the network onset of congestion threshold is increased. The network onset of congestion of is when the capacity of some link in the network is exceeded (congestion) and depends on different factors such as network topology, routing and link capacities.

The problem that SPs are facing is how to plan their future network to accommodate the constant increasing of the demand, to provide a congestion free network and to minimize the costs. Another challenge is that the SP already has an established network. This investment can keep the network congestion-free for a limited time. Therefore, TTE becomes crucial for network capacity planning since it affects the amount and the timing of investment in the infrastructure. With a limited budget, the SP must choose how to plan the additional capacity, where to put the caches and what content should to cached. We aim to maximize the time-to-exhaustion, considering a limited budget, by placing caches in the best locations. We will show that using NDN will outperform optimal cache placement and routing in CDN and will prolong network-to-exhaustion of the network.

IV. PROBLEM FORMULATION

We model our network as a directed graph $G(V, E)$ with the set of nodes $V$ and links $E$. $U$ is the notation for the set of nodes that have a demand for contents. $P$ indicates the set of nodes that can satisfy demands for contents, e.g. Internet exchange points. The set of nodes that are candidates for caching contents is noted by $C$. 
A. Demands and Storage Budget

To find the TTE of the network, we will model the network for one time epoch. We assume that within this time epoch, the demands are known and fixed, but the location of the caches, the cached content and content routing are not. We also assume a limited storage budget, $B$, is available for capacity planning of all the caches in the network. For each budget value, we solve a series of feasibility problems by increasing the demands following a pre-known pattern. A feasibility problem does not have any objective and will only find a feasible solution to the problem. By increasing the demand, the network will become congested, which means the problem will become infeasible. When the problem becomes infeasible, i.e. the network is saturated, we have found the TTE of the network. Final solution of the model provides a content caching policy and traffic routing that maximizes the TTE of the network.

1) Demands: We denote the demand at each node $i$ for content $k$ by $\alpha_i^k$. The demand at each node also depends on whether the node caches any content or not ($h_i^k$). In other words, the traffic of populating a cache is also a demand. Therefore, total demand at node $i$ can be written as:

$$\beta_i^k = \alpha_i^k + h_i^k \quad \forall i \in C \cup U$$  \hspace{1cm} (1)$$

Note that $\beta_i^k$ is the number of the requests for content $k$, not the size of the demand. The size of the demand is $L_k \beta_i^k$ where $L_k$ is the size of content $k$.

2) Storage Budget: Each cache in the network is assigned a part of the storage budget, denoted by $S_i$. Therefore, the budget constraint can be written as Eq (2).

$$\sum_i S_i \leq B \quad \forall i \in C$$  \hspace{1cm} (2)$$

Let $p_i$ be the binary variable that decides if node $i$ is a cache and $h_i^k$ the binary variable that shows if content $k$ is cached at node $i$. For each cache, total cached objects can not exceed the size of the storage of that cache as written in Eq (3).

$$\sum_k L_k h_i^k \leq p_i S_i \quad \forall i \in C$$  \hspace{1cm} (3)$$

Also, total number of caches placed in the network can be limited by an upper limit, $M$, as written in Eq (4).

$$\sum_i p_i \leq M \quad \forall i \in C$$  \hspace{1cm} (4)$$

To have homogeneous caching, we may also add a constraint that enforces all $S_i$ to be equal.

3) Cache Replacement Policy and Routing: Caching policy provided by the solution will maximize the TTE of the network. As mentioned earlier, $h_i^k$ is the binary variable that shows if content $k$ is cached at node $i$. Solving the model for two different time epochs with different demands will result in different $h_i^k$. The difference between $h_i^k$ for different demands will be the cache replacement policy of node $i$. Adopting a certain caching replacement policy, such as Least Recently Used (LRU) or Least Frequently Used (LFU), will reduce the TTE of the network.

The solution also provides the content routing policy for the network. Adopting a routing protocol such as shortest-path will reduce the TTE of the network. We study this effect in the result section.

B. Content Delivery Networks

In service providers, transparent caching is done by putting one or more caches in the network and re-routing the requests towards them. SPs may also host the content sources of their own or from third parties. To model this, we define a multi-commodity flow problem.

1) Flow Conservation: The flow conservation at node $s$ for content $k$ can be written as Eq (5).

$$\sum_{j \in \Gamma_s^{-}} f_{j,k}^{s,d} - \sum_{j \in \Gamma_s^{+}} f_{j,k}^{s,d} = \gamma_{s}^{k,d} - L_k \beta_s^k \delta(s-d) \quad \forall s, d \in \mathbb{V}$$  \hspace{1cm} (5)$$

We denote $f_{j,k}^{s,d}$ as the flow for content $k$ on link $(i,j)$ going to node $d$ and $\gamma_{s}^{k,d}$ for the flow for content $k$ from node $s$ to node $d$. The left-hand side of Eq (5) is the difference between total egress ($\Gamma_s^{+}$) and ingress ($\Gamma_s^{-}$) flows for content $k$ at node $s$ that is destined for node $d$.

The right-hand side of Eq (5) is the total flow that is originated at node $s$ towards node $d$ for content $k$ minus the demand at node $s$ for content $k$. $\delta(i)$ is the Kronecker delta function, it is equal to 1 when $i$ is zero, otherwise it is zero. Therefore, $L_k \beta_s^k$ in Eq (5) will only have any effect when $s$ and $d$ are the same node. In other words, the ingress and egress

<table>
<thead>
<tr>
<th>TABLE I: Notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{V}$</td>
</tr>
<tr>
<td>$\mathbb{E}$</td>
</tr>
<tr>
<td>$\mathbb{G}(\mathbb{V}, \mathbb{E})$</td>
</tr>
<tr>
<td>$\mathbb{P}$</td>
</tr>
<tr>
<td>$\mathbb{U}$</td>
</tr>
<tr>
<td>$\mathbb{C}$</td>
</tr>
<tr>
<td>$L_k$</td>
</tr>
<tr>
<td>$\alpha_i^k$</td>
</tr>
<tr>
<td>$\Gamma_i^{-}$</td>
</tr>
<tr>
<td>$v_{i,k}^*$</td>
</tr>
<tr>
<td>$B$</td>
</tr>
<tr>
<td>$\gamma_{i,j}$</td>
</tr>
<tr>
<td>$\delta(i)$</td>
</tr>
<tr>
<td>$M$</td>
</tr>
<tr>
<td>$\phi_{i,j}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Common Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_i$</td>
</tr>
<tr>
<td>$p_i$</td>
</tr>
<tr>
<td>$B_k$</td>
</tr>
<tr>
<td>$\beta_i^k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CDN Specific Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_{s}^{k,d}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NDN Specific Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma_i^{-}$</td>
</tr>
</tbody>
</table>
flow destined to node $d$ at any node other than $d$ is equal to the traffic produced at that node for node $d$. When $s$ and $d$ are equal and ingress traffic into node $d$ will be equal to the demand at node $d$. Therefore, considering the fact that node $d$ does not send traffic to itself (i.e., $f_{d,j}^k = 0, \forall j$) and $\gamma_{d}^{kd} = 0$), Eq (5) will be reduced to

$$\sum_{j \in \gamma_d} f_{j,d}^k = L_k \beta_d^k$$

2) Cache Population Traffic: The cache population traffic is satisfied by peering points. Therefore, the total demand originated at the core ($P$) of the network, must be bigger than the size of the cached content (Eq (6)).

$$\sum_{s \in P} \gamma_s^{ki} \geq L_k h_i^k \quad \forall i \in C$$

(6)

3) I/O and Link Capacity Limits: A node can only become a source of the flow for a content request when it is a cache and it has the content cached. $I(i \in C)$ in Eq (7) is equal to 1, if only node $i$ is a cache candidate. $h_i^k$ will be equal to 1 when the content $k$ is cached at node $i$. $r_i^k$ is the rate that each node can read contents from its cache storage and put them on the wire. It is the limitation of the node’s hard disks.

$$\sum_{d} a_{i,d}^k \leq I(i \in C) r_i^k L_k h_i^k$$

(7)

We also write the link capacity constraint as Eq (8).

$$\sum_{k,d} f_{i,j}^k \leq c_{i,j}$$

(8)

The complete feasibility problem that models a CDN in the network of a service provider is:

solve subject to

$$\sum_{j \in \Gamma_d} f_{j,d}^k = \sum_{j \in \Gamma_d} f_{j,s}^k = \gamma_s^{kd} - L_k \beta_s^k \delta(s-d) \quad \forall s, d$$

$$\sum_{s \in P} \gamma_s^{ki} \geq L_k h_i^k \quad \forall i \in C$$

$$\sum_{d} a_{i,d}^k \leq I(i \in C) r_i^k L_k h_i^k$$

$$\sum_{k,d} f_{i,j}^k \leq c_{i,j}$$

$$\beta_s^k = a_s^k + h_s^k \quad \forall i \in \mathbb{V} \setminus \mathbb{P}$$

$$\sum_{i} S_i p_i \leq B$$

$$\sum_{k} L_k h_i^k \leq p_i S_i$$

4) Shortest-path routing: Routing in the network of the service providers is usually based on shortest-path routing. To study the effects of shortest-path routing, we add a routing constraint to our model. Shortest-path routing is modeled using the shortest-path betweenness centrality of each link.

Betweenness centrality (BC) is one of the centrality metrics in graphs [13]. Betweenness centrality measures the degree to which a node or a link is needed when connecting other nodes along paths. Shortest-path betweenness centrality of the link $(i,j)$ with respect to the source node $s$ and the destination node $d$, denoted as $\phi_{s,j}^k$, is defined as the proportion of the number of shortest paths from node $s$ to $d$ that passes through link $(i,j)$. Therefore, the average traffic for content $k$ that passes through link $(i,j)$ from source $s$ to destination $d$ can be written as $\phi_{s,j}^k / \gamma_s^{kd}$. To model shortest path routing we can add Eq (9) to the model. Eq (9) will have the link $(i,j)$ to not transfer any traffic more than its share, if the routing is done using shortest-path.

$$f_{s,j}^k \leq \sum_{s \in \mathbb{V}} \phi_{s,j}^k \gamma_s^{kd}$$

(9)

C. Named-Data Networking

1) Interest Forwarding: To model NDN, we will find the locations that potentially can satisfy more interest in contents. This notion of interest here is more of the nature of content popularity in a node, similar to the virtual interest packets studied in [14], and is different from the Interest packet in NDN paradigm. We denote $f_{s,j}^k$ as the rate that interest for content $k$ is forwarded on link $(i,j)$. Since NDN is a point-to-point protocol we do not have flows from sources to destinations, but potential interests that move around the network until they are satisfied.

Suppose node $s$ has some interest in content $k$ ($\beta_s^k$). Therefore, the egress interests ($\sum_{j \in \Gamma_s} f_{s,j}^k$) from node $s$ is increased by $\beta_s^k$. This is written as an inequality in Eq (10).

$$\sum_{j \in \Gamma_s} f_{s,j}^k - \sum_{j \in \Gamma_s} f_{j,s}^k \leq \beta_s^k$$

(10)

Now consider a node that has a content cached in its content store and can satisfy interest for that content and remove the interest from the network. Each node also has an I/O limit for reading its content store that limits the rate interests are satisfied. Otherwise, the interest will be forwarded towards other nodes in the network. Therefore, a node can at most satisfy the interests by the rate that is bounded by its I/O limit, as written in Eq (11).

$$\sum_{j \in \Gamma_s} f_{s,j}^k - \sum_{j \in \Gamma_s} f_{j,s}^k + I(i \in C) r_i^k h_i^k \geq \beta_s^k$$

(11)

Consider the scenario that node $s$ is not caching content $k$. Therefore, Eq (10) and Eq (11) will be reduced to an equality and will enforce that node $s$ forwards all of its ingress and local interests. However, if node $s$ caches content $k$, the ingress interests can be satisfied by an amount bounded by the hardware limitations of node $s$. Finding the movement of this potential interest in the network can be used to find the best place to cache the content.

2) Link capacity constraint: The next step is to model the link capacity constraint. In NDN, Data packets follow the reverse path of the Interest packet to reach the destination. Therefore,
sending an interest over the link \((i, j)\) will result in the data sent back over the link \((j, i)\). We can use this to write link capacity constraint as Eq (12),

\[
\sum_k L_k f_{i,j}^k \leq c_{j,i}
\] (12)

Including Eq (1), Eq (2), Eq (3), the complete feasibility problem for NDN is:

\[
\text{solve subject to}
\sum_{j \in \Gamma_i^+} f_{i,j}^k - \sum_{j \in \Gamma_i^-} f_{j,i}^k \leq \beta_k^i
\]

\[
\sum_{j \in \Gamma_i^+} f_{i,j}^k - \sum_{j \in \Gamma_i^-} f_{j,i}^k + \sum_{i \in C} r_i^k h_i^k \geq \beta_k^i
\]

\[
\sum_k L_k f_{i,j}^k \leq c_{j,i}
\]

\[
\beta_i^k = \alpha_i^k + h_i^k \quad \forall i \in \mathcal{V} \setminus \mathcal{P}
\]

\[
\sum_i S_i p_i \leq B
\]

\[
\sum_k L_k h_k^k \leq p_i S_i
\]

V. Results

We evaluated the numerical result of our model using multiple network topologies. Fig. 3 is one of the Rocketfuel networks [15]. Fig. 4 is a Dorogovtsev-Goltsev-Mendes (DGM) topology and Fig. 5 is a tree network. The Rocketfuel topology is simplified by removing the leaf nodes from the original network and consolidating the demands from the removed nodes into their parent nodes [16]. The simplified network has 50 nodes and 194 directed links. These three topologies are comparable in size. The number of users is 25 nodes in Rocketfuel and 27 nodes in DGM and tree topologies. We consider one peering point for each network, and the rest of nodes are cache candidates nodes. At each node, the demand for each content follows a Zipf distribution with \(\alpha = 2\). We assumed all the users have the same demand, and it is increasing by 5% every month everywhere. This increase is based on the current observation of OTT demand increase. As mentioned earlier, for each budget point we solve a series of feasibility problem and increase the demand until the network is saturated. We also simulated the back-pressure algorithm in [14] to compare with the performance of our model.

A. Time-to-Exhaustion of different topologies

To evaluate the performance of the CDN method, we find the TTE of the network by putting at most four caches. The assigned storage budget is equally divided between these nodes, assuming they are all using similar hardware. In other words, we use homogeneous caching. For example, in Rocketfuel network (Fig. 3), Nodes 5, 10, 12 and 14, are selected for caching, and respectively, in DGM topology, Nodes 2, 3, 4, 5 and tree topology, Nodes 2, 3 and 4 are selected for caching. It is also worth noting that in the tree topology only three nodes are selected for caching, since adding more caches has not increased the TTE further. To evaluate the performance of NDN, we will enable caching in all of the candidate nodes. Therefore, storage budget will be equally divided between more nodes, and each node can cache less number of objects.

Fig. 6, Fig. 7 and Fig. 8 show the TTE in different topologies while using CDN model, NDN model and NDN simulation using back pressure algorithm. We assumed that there is demand for 2000 objects, divided into 100 popularity groups, each with the size of 1Mb and all of the links in the network have the capacity of 1Gb/s. As mentioned above, we had placed at most four caches in CDN while all of the nodes can cache in NDN scenarios. Note that in all of the topologies,
NDN-Simulation using back-pressure closely follows the our NDN-model.

At very low storage budget, CDN and NDN had a similar TTE, because most of the content is provided by the peering point, and that will become the bottleneck of the network. This means network onset of congestion will be similar for both NDN and CDN scenarios. Different topologies have different TTE for very low storage budget. The TTE depends on the onset of congestion, and the onset of congestion depends on the topology of the network. TTE is lowest for the tree topology and the highest for the DGM topology. This observation is also in agreement with the reciprocal of network criticality of each topology [17].

By increasing the caching storage, TTE is also increased. As mentioned earlier, storage budget is equally divided between all caches. Therefore, the increase in the total storage budget will increase the TTE. As the number of caches increases, each cache will receive a smaller portion of the budget. Therefore, when there is not enough additional storage available to each cache, there will be no change in the number of cached contents, and the TTE will not change either. This minimum increase in storage depends on the number of caches in the network. In NDN, the steps are larger since there are more caches and a greater increase in the total storage budget is required to cache more contents. In CDN, the steps are smaller since there are only four caches and a smaller amount of increase in storage budget, compared to NDN, will result in more cached contents. However, the height of the steps decreases with increase of the budget, because caching begins losing its effect. There is also a limit on the maximum TTE of each topology, after which even caching does not help anymore. This TTE is the maximum that a network can reach with the help of caching. Similar to the low budget TTE, the maximum TTE also depends on the topology of network.

Furthermore, in low storage budget, there is little difference in TTE between using CDN and NDN. Because of homogeneous caching, sometimes CDN even performs better. However, in all of the topologies, the network that uses CDN is saturated in much lower storage budget compared to NDN. This better performance is the direct result of the NDN paradigm. In NDN, due to its in-network caching and point-to-point nature, each cached content is sent over the links only once. However, in CDN each content is sent multiple times. This waste of link capacity shows itself by having the network saturated much sooner. There is a huge difference in maximum TTE between using CDN or NDN in each topology. In Rocketfuel, using CDN will saturate the network after 47 months. But using NDN, the network can be operational until 77 months. Similarly, DGM with CDN is operational for 74 months and with NDN for 82 months. Tree topology with CDN is operational for 27 months and with NDN for 67 months.
B. Limited NDN Deployment

To see how much of the difference in TTE between CDN and NDN comes from the number of caches in the network, we can limit the number of caches in NDN to four as well. Fig. 9 shows that even with four caches, content delivery using NDN outperforms the CDN design. We have also considered a non-practical case that every node in the CDN can also cache contents. This case is just for the comparison and in practice cannot be implemented due to the nature of CDN. One could say that one of the reasons behind the NDN proposal is the impossibility of in-network caching in TCP/IP. However, even if all of the nodes in the CDN had the caching capability, the network will saturate similar to the case that there are four caches in the network. In addition, limited NDN deployment has better TTE for low budget than full NDN deployment. This suggests limiting the number of NDN caches when the storage budget is low.

We can also look at link utilization in the network. Fig. 12 shows the percentage of links with various percentage of utilization during network congestion. Using CDN, more than 60% of the links will have a link utilization of more than 90%. In contrast, NDN scenario, even with limited deployment, has less than 20% of the links with high utilization. Using NDN, has resulted in a network that more than 40% of the links have link utilization of less than 10%. This difference in link utilization means that if CDN is used to increase the TTE of the network, we have to increase the capacity of most of the
most by 3 months. 

is employed, TTE in the Rocketfuel network is increase at 

efit to the heterogeneity. However, there is some difference in 

symmetry in tree and DGM topologies would imply little ben-

model will assign each cache a different storage capacity while 

TTE when NDN is used. In using heterogeneous caching, the 

E. Heterogeneous Caching 

The NDN does not have this problem since its strategy layer 

network the TTE will be reduced by more than 10 month. 

enforcing shortest-path routing for CDN in the Rocketfuel 

in practice routing is not optimal. As shown in Fig. 11, by 

TTE and therefore optimizes the routing of data. However, 

D. Routing Protocol Effect in CDN 

As mention above, our modeling tries to maximize the 

and lower number of links with high utilization. We studied 

different parameters that affect the performance of content 

delivery in the service provider’s network and validated our 

model by simulation. We also demonstrated that heterogeneous 

caching does not provide a substantial performance benefit 

over homogeneous caching.

VI. Conclusion 

In this paper, we propose in-network caching strategy for 

service providers in order to control the time-to-exhaustion for 

their backbone capacities. We propose that the service provider 

uses Named-Data Networking (NDN) for video delivery in its 

network. Even a limited deployment of NDN provides 

a substantial increase in time-to-exhaustion of the network 

and lower number of links with high utilization. We studied 

different parameters that affect the performance of content 

delivery in the service provider’s network and validated our 

model by simulation. We also demonstrated that heterogeneous 

caching does not provide a substantial performance benefit 

over homogeneous caching.

C. I/O Speed Effect 

One of the parameters we have considered in our modeling 

is the I/O limit of each cache. The I/O limit depends on 

hardware design of the cache. Fig. 10 shows the effect of this 

parameter. To better see the difference the I/O speed makes, 

we have increased the capacity of all of the links to limit the 

effect of congestion. As shown in Fig. 10, as the I/O limit 

increases from 10Gb/s to 100Gb/s, TTE also increases. But 

it must be said that having low link capacity will greatly diminish 

the improvement gained by having a hardware with higher I/O 

limit.

D. Routing Protocol Effect in CDN 

As mention above, our modeling tries to maximize the 

TTE and therefore optimizes the routing of data. However, 

in practice routing is not optimal. As shown in Fig. 11, by 

enforcing shortest-path routing for CDN in the Rocketfuel 

network the TTE will be reduced by more than 10 month. 

The NDN does not have this problem since its strategy layer 

can employ an optimal routing algorithm.

E. Heterogeneous Caching 

Fig. 13 shows the effect of heterogeneous caching on the 

TTE when NDN is used. In using heterogeneous caching, the 

model will assign each cache a different storage capacity while 

satisfying total storage budget constraint. It is expected that the 

symmetry in tree and DGM topologies would imply little ben-

efit to the heterogeneity. However, there is some difference in 

Rocketfuel which is less symmetric. If heterogeneous caching 

is employed, TTE in the Rocketfuel network is increase at 

most by 3 months.

Fig. 13: Heterogeneous vs Homogeneous caching storage in 

NDN 

links. But using NDN will result in a much fewer bottlenecks, 

which makes capacity planning much easier and cheaper.

VI. Conclusion 

In this paper, we propose in-network caching strategy for 

service providers in order to control the time-to-exhaustion for 

their backbone capacities. We propose that the service provider 

uses Named-Data Networking (NDN) for video delivery in its 

network. Even a limited deployment of NDN provides 

a substantial increase in time-to-exhaustion of the network 

and lower number of links with high utilization. We studied 

different parameters that affect the performance of content 

delivery in the service provider’s network and validated our 

model by simulation. We also demonstrated that heterogeneous 

caching does not provide a substantial performance benefit 

over homogeneous caching.

REFERENCES 


"From Content Delivery Today to Information Centric Networking," 


and R. L. Braynard, “Networking Named Content,” in Proceedings of the 

5th International Conference on Emerging Networking Experiments 

and Technologies, ser. CoNEXT ’09. New York, NY, USA: ACM, 

2009, pp. 1–12.


the responsiveness of DNS-based network control,” in Proceedings of the 

4th ACM SIGCOMM conference on Internet measurement. ACM, 


S. Shenker, and I. Stoica, “A Data-Oriented (and beyond) Network 


[7] L. Zhang, A. Ananayev, J. Burke, V. Jacobson, k. claffy, P. Crowley, 

C. Papadopoulos, L. Wang, and B. Zhang, “Named data networking,” 


[8] G. Tyson, S. Kauwe, S. Miles, Y. El-khatib, A. Manthe, and A. Tawel, 

“Throughput analysis of caching in content-centric networks,” in Com-

puter Communications and Networks (ICCCN), 2012 21st International 


centric networking: implications for protocol design and public policy,” 

Communications Magazine, IEEE, vol. 50, no. 12, pp. 18–26, December 

2012.


Topological Information,” in Computer Communications Workshops 


280–285.


transfer in content-centric networking,” in Teletraffic Congress (ITC), 


allocation for content-centric networking,” in Network Protocols (ICNP), 


tance distance in communication networks,” Network, IEEE, vol. 24, 

no. 6, pp. 10–16, November 2010.


for joint dynamic forwarding and caching in named data networks,” 


weights using end-to-end measurements,” in Proceedings of the 2nd 

ACM SIGCOMM Workshop on Internet Measurement, ser. IMW ’02. 


Binary Opinion Dynamics with Biased Agents and Agents with Different Degrees of Stubbornness

Arpan Mukhopadhyay  
Bell Labs - INRIA, Paris, France  
Email: arpan.mukhopadhyay@inria.fr

Ravi R. Mazumdar  
Electrical and Computer Engineering  
University of Waterloo, Canada,  
Email: mazum@uwaterloo.ca

Rahul Roy  
Indian Statistical Institute, Delhi  
Email: rahul@isid.ac.in

Abstract—In this paper, we investigate the impact of random interactions between agents in a social network on the diffusion of opinions in the network. Opinion of each agent is assumed to be a binary variable and each agent is assumed to be able to interact with any other agent in the network. This models scenarios where every agent in the network has to choose from two available options and the size of the neighborhood of each agent is an increasing function of the total number agents in the network. It is assumed that each agent updates its opinion at random instants upon interacting with other randomly sampled agents. We consider two simple rules of interaction: (1) the voter rule in which the updating agent simply copies the opinion of another randomly sampled agent; (2) the majority rule, in which the updating agent samples multiple agents and adopts the majority opinion among the sampled agents and the agent itself. Under each rule, we consider two different scenarios which have not been considered in the literature thus far: (1) where the agents are ‘biased’ towards one of the opinions, (2) where different agents have different degrees of stubbornness. We show that the presence of biased agents reduces the consensus time for the voter rule exponentially as compared to the case where the agents are unbiased. For the majority rule model with biased agents, we show that the network reaches consensus on a particular opinion with high probability only when the initial fraction of agents having that opinion is above a certain threshold. For the majority rule model with stubborn agents, we observe metastability where the network switches back and forth between stable states spending long intervals in each state.

I. INTRODUCTION

With the widespread use of online social networking, opinions of individuals are constantly being shaped by social interactions. Understanding how individual opinions are affected by social interactions and what global opinion structure emerges from such interactions are important in many contexts such as economics, politics and psychology. Mathematical models of social interaction treat the opinion of each individual in a social network as a variable taking values in a discrete or continuous subset of the Euclidean space. Although this may seem too reductive to capture the complexity of choices made by real individuals, in everyday situations, individuals in a network are often faced with only a limited number of choices (often as few as two) concerning a specific issue, e.g., pro-/anti-government, Windows/Linux, Democrat/Republican, etc. Thus, a vast body of literature treats opinions of individuals as binary variables taking values in the set \{0, 1\}.

The interactions between agents in a social network are generally modeled using simple rules that capture the essential features of individuals in a society such as their tendency to mimic their neighbors or to conform with the majority opinion in local neighborhoods. One of the models, extensively analyzed in the literature, is the voter model \cite{1}–\cite{3} or the voter rule, where an agent randomly samples one of its neighbors at an instant when it decides to update its opinion. The updating agent then adopts the opinion of the sampled neighbor. This simple rule captures the tendency of an individual to mimic other individuals in the society. Because of its simplicity, the rule has been analyzed under a variety of network topologies \cite{4} that assume connectedness of the underlying graph.\(^1\) It is known that, under the voter rule, any connected network converges to a consensus, where all individuals adopt the same opinion. It is of interest to determine the probability with which consensus is reached on a specific opinion and the time it takes for the network to converge to the consensus state.

Another rule studied in this context is the majority rule model \cite{5}–\cite{7}. In it, instead of sampling a single individual, an updating agent consults multiple individuals while performing the update and adopts the choice of the majority of the sampled neighbors. This rule captures the tendency of the individuals to conform with the majority opinion in their local neighborhoods. In a fully connected network, the majority rule also leads to a consensus among agents. However, the rate at which consensus is reached is faster than that under the voter rule.

In all the prior works on the voter models and the majority rule models, it is assumed that an agent’s decision to update its opinion does not depend on the current opinion of the agent. It is also assumed that all agents in the network have the same propensity to change their opinions. However, in a real scenario an agent may be ‘biased’ towards a specific opinion in the sense that if it holds its ‘preferred’ opinion, then the probability with which it updates its opinion is small. We may also encounter situations where some of the agents update their choices less frequently than others (irrespective of their current opinions). In this paper, we focus on these two scenarios.

\(^1\)Connectedness implies that every individual is connected to every other individual either directly or via immediate neighbors.
A. Related literature

There is a rich and growing literature that studies diffusion of technologies and opinions in large social networks in both Bayesian and non-Bayesian settings. The voter models and the majority rule models fall under the non-Bayesian setting. One of the first models in the non-Bayesian setting was studied by DeGroot [8], where the agents were assumed to update their opinions (assumed to be continuous variables within a certain range) synchronously by averaging the opinions of their neighbors. This is equivalent to the synchronous average consensus algorithms considered in [9] and thus can be analyzed using similar techniques. The ‘voter model’ with binary opinions was first studied independently in [1] and [2]. It was assumed that an agent simply copies the opinion of a randomly sampled neighbor at an instant of update. Due to its simplicity, the voter model soon became popular and was analyzed under a variety of network topologies, e.g., finite integer lattices in different dimensions [3], [10], heterogeneous graphs [11], Erdos-Renyi random graphs and random geometric graphs [4] etc. In [12], [13], the voter model was studied under the presence of stubborn individuals who do not update their opinions. In such a scenario, the network cannot reach a consensus because of the presence of stubborn agents having both opinions. Using coalescing random walk techniques the average opinion in the network and the variance of opinions were computed at steady state. A model where the agents have continuous values of opinion in the interval [0, 1] and the updates occur iteratively based on the minimization of a cost function that take into account an agent’s past opinion was considered in [14].

The majority rule model was first introduced in [15], where it was assumed that, at every iteration, groups of random sizes are formed by the agents. Within each group, the majority opinion is adopted by all the agents. Under this rule, it was shown that consensus is achieved on a particular opinion with high probability only if the initial fraction of agents having that opinion is more than a certain critical value. Furthermore, the time to reach consensus was shown to scale as logarithm of the network size (number of agents). Similar models with fixed (odd) group size were considered in [5], [6]. It was shown that for finite dimensional integer lattices the consensus time grows as a power law in the number of agents in the network.

A deterministic version of the majority rule model, where an agent, instead of randomly sampling some of its neighbors, adopts the majority opinion among all its neighbors, is considered in [16]–[19]. In such models, given the graph structure of the network, the opinions of the agents at any time is a deterministic function of the initial opinions of the agents. The interest there is to find out the the initial distribution of opinions for which the network converges to some specific absorbing state. In social networks, where the neighborhood of each agent is large, such majority rule dynamics involves complex computation by each updating agent at each update instant. Our interest in this paper is on scenarios where the agents are mobile and do not have any fixed neighborhoods. We therefore consider a randomized version of the majority rule.

B. Contributions

In this paper, we study binary opinion dynamics under the voter model and the majority rule model. Under each model, we consider the following two scenarios: 1) where the agents are ‘biased’ towards a specific opinion. 2) where different agents have different propensities to change their opinions or different degrees of stubbornness. We make the following contributions

1) For the voter model with biased agents, we derive a closed form expression of the probability with which consensus is reached on the ‘preferred’ opinion. It is observed that this probability increases rapidly to 1 as the number of agents in the network grows. This is unlike the case with unbiased agents, where the probability to reach consensus on a particular opinion remains constant for all network sizes. Using mean field techniques, we derive an estimate of the average time taken for the network to reach consensus. It is observed that the mean consensus time grows as logarithm of the network size. This is in contrast to the case with unbiased agents, where the mean consensus time grows linearly with the number of agents.

2) For the voter model with differently stubborn agents, we show that the probability of reaching consensus on a particular state is independent of the network size. We also show that the time to reach consensus grows linearly with the total number of agents.

3) For the majority rule model with biased agents, we derive a closed form expression for the probability with which consensus is achieved on the preferred opinion. It is observed that, unlike the voter model, consensus is achieved on the preferred opinion (with high probability) only if the initial fraction of agents having that opinion is above a certain threshold. This threshold is determined from the mean field analysis of the model. An estimate of the mean consensus time is also found from the mean field model. It suggests that the mean consensus time grows as logarithm of the number of agents in the network.

4) Finally, we consider the majority rule model when there are ‘stubborn’ agents in the network. The stubborn agents are assumed to have fixed opinions at all times. Therefore, in this case consensus can never be reached. We analyze the equilibrium distribution of opinions among the non-stubborn agents using mean field techniques. Depending on the system parameters, the mean field is shown to have either multiple stable equilibrium points or a unique stable equilibrium point within the range of interest. As the system size grows, the equilibrium distribution of opinions among non-stubborn agents is shown to converge to a mixture of Dirac measures concentrated on the equilibrium points of the mean field. This suggests a metastable behavior of the system where the system moves back and forth between stable configurations, spending long intervals in each configuration. The conditions for metastability are obtained in terms of the system parameters.
The rest of the paper is organized as follows. In Section II, we introduce the voter model. In Subsections II-A and II-B, we analyze the voter model with ‘biased’ agents and agents having different degrees of stubbornness, respectively. Section III introduces the majority rule model. In Subsections III-A and III-B, we analyze the majority rule model with ‘biased’ and ‘stubborn’ agents, respectively. Finally, the paper is concluded in Section IV.

II. THE VOTER MODELS

Let us consider a network consisting of $N$ social agents, where each agent can communicate with every other agent. The results derived in this paper also holds for cases where size of the neighborhood of each agent is $O(N)$. Opinion of each agent is assumed to be a binary variable taking values in the set $\{0, 1\}$. Initially, every agent adopts one of the two opinions. The agents then consider updating their opinions at points of independent unit rate Poisson processes associated with themselves. At a point of the Poisson process associated with itself, an agent either updates its opinion or retains its past opinion. In case the agent decides to update its opinion, it samples an agent uniformly at random (with replacement) from the network and adopts the opinion of the sampled agent.

Below we consider two different scenarios: (1) where the agents are ‘biased towards a specific opinion, and (2) where the agents have different propensities to change their past opinions.

A. The voter model with biased agents

We first consider the case where the agents are ‘biased’ towards one of the two opinions. Without loss of generality, we assume that all agents in the network prefer opinion $\{1\}$ to opinion $\{0\}$. This is modeled as follows: Each agent with opinion $i \in \{0, 1\}$ updates its opinion at a point of the unit rate Poisson process associated with itself with probability $q_i$ and retains its opinion with probability $p_i = 1 - q_i$. This is equivalent to an agent with opinion $i$ updating its opinion at all points of a Poisson process with rate $q_i$. In case the agent decides to update its opinion, the update occurs following the voter rule discussed in the beginning of this section. We assume $q_0 > q_1$ ($p_1 > p_0$) to imply that the agents having opinion $\{0\}$ update their opinions more frequently than the agents having opinion $\{1\}$. In the above sense, the agents are biased towards opinion $\{1\}$.

Clearly, in this case, the network gets absorbed (in a finite time) in a state where all the agents adopt the same opinion. This is referred to as the consensus state. Our interest is to find out the probability with which consensus is achieved on the preferred opinion $\{1\}$ starting from a state where a fixed proportion of agents are in state $\{1\}$. This probability is referred to as the exit probability of the network. We also intend to characterize the mean time to reach the consensus state.

The case $q_1 = q_0 = 1$ is referred to as the voter model with unbiased agents, which has been analyzed in [1], [2]. It is known that for unbiased agents the probability with which consensus is reached on a particular opinion starting from a state where $\alpha$ fraction of agents have that particular opinion is simply equal to $\alpha$, which is independent of $N$. Furthermore, the expected time to reach consensus for large $N$ is known to be $Nh(\alpha)$, where $h(\alpha) = -\alpha \ln(\alpha) + (1 - \alpha) \ln(1 - \alpha)$.

We now proceed to characterize these quantities for the voter model with biased agents.

Let $X^{(N)}(t)$ denote the number of agents with opinion $\{1\}$ at time $t \geq 0$. Clearly, $X^{(N)}(\cdot)$ is a Markov process on state space $\{0, 1, \ldots, N\}$, with possible jumps at the points of a rate $N$ Poisson process. This rate $N$ process is referred to as the global clock. All states, except the states 0 and $N$, form an open communicating class; the states 0 and $N$ are the absorbing states. Therefore, with probability 1, the process gets absorbed in one of the absorbing states in a finite time.

Proposition 1: The probability $E_N(n)$ with which the process $X^{(N)}(\cdot)$ gets absorbed in state $N$ starting with state $n \in \{1, 2, \ldots, N\}$ is given by $E_N(n) = \frac{\alpha^n}{n}$, where $r = q_0/q_1 < 1$ and $E_N(0) = 0$.

Proof: Given that the process $X^{(N)}(\cdot)$ is in state $k$ at one point of the global clock, it transits to state $k+1$ at the next point of the global clock only if one of the agents having opinion $\{0\}$ updates its opinion to opinion $\{1\}$. The probability with which any one of the $N-k$ agents having opinion $\{0\}$ decides to update its opinion is given by $q_0 \times (N-k)/N$. The probability with which the updating agent samples an agent with opinion $\{1\}$ is given by $\alpha/k$. Hence, the total probability with which the process $X^{(N)}(\cdot)$ transits from the state $k$ to the state $k+1$ is given by $p(k \to k+1) = \frac{\alpha(N-k)+q_0}{N}$. Similarly, the probability of transition from the state $k$ to the state $k-1$ is given by $p(k \to k-1) = \frac{\alpha(k-1)+q_1}{N}$. Therefore, the probability with which no transition occurs between two consecutive points of the global clock is $p(k \to k) = 1 - p(k \to k+1) - p(k \to k-1)$. Since $X^{(N)}(\cdot)$ is Markov, $E_N(n)$ must satisfy the following recursive relationship $E_N(n) = p(n \to n+1)E_N(n+1) + p(n \to n-1)E_N(n-1) + p(n \to n)E_N(n)$, which can be solved (using the transition rates given above) to yield the desired expression.

In terms of the initial fraction $\alpha = n/N$ of agents having opinion $\{1\}$, the exit probability derived above can be expressed as $E_N(\alpha) = \frac{1}{1 - q_0/q_1}$. Clearly, for $q_0 > q_1$, we have $r < 1$. Hence, as $N$ increases the exit probability rapidly increases to 1 for all $\alpha$. This is in contrast to the case with unbiased agents ($q_0 = q_1 = 1$) where the exit probability remains constant at $\alpha$ for all values of $N$.

We now characterize the mean time $T_N(\alpha)$ to reach the consensus state starting from $\alpha$ fraction of agents having opinion $\{1\}$. To do so, we consider the empirical measure process $x^{(N)}(\cdot) = X^{(N)}(\cdot)/N$, which describes the evolution of the fraction of agents with opinion $\{1\}$. The process $x^{(N)}(\cdot)$ jumps from the state $x$ to the state $x+1/N$ when one of the $N(1-x)$ agents having opinion $\{0\}$ updates (with probability $q_0$) its opinion by interacting with an agent.

In the large $N$ limit sampling with or without replacement does not make any difference.
with opinion \{1\}. Since the agents update their opinions at points of independent unit rate Poisson processes, the rate at which one of the \(N(1-x)\) agents having opinion \(\{0\}\) decides to update its opinion is \(N(1-x)q_0\). The probability with which the updating agent interacts with an agent with opinion \(\{1\}\) is \(x\). Hence, the total rate of transition from \(x\) to \(x+1/N\) is given by \(r(x \to x+1/N) = q_0 N x (1 - x)\). Similarly, the rate of transition from \(x\) to \(x-1/N\) is given by \\
r(x \to x-1/N) = q_1 N (1-x)\). From the above transition rates it can be easily seen that the generator of the process \(x^{(N)}(\cdot)\) converges uniformly as \(N \to \infty\) to the generator of the deterministic process \(x(\cdot)\) which is the unique solution of the following differential equation

\[
x(t) = (q_0 - q_1) x(t)(1 - x(t)).
\]  

Thus, by the classical results of Kurtz [20] we have that if \(x^{(N)}(0) \Rightarrow x(0)\) as \(N \to \infty\), then \(x^{(N)}(\cdot) \Rightarrow x(\cdot)\) as \(N \to \infty\), where \(\Rightarrow\) denotes weak convergence. In other words, for large \(N\), the process \(x^{(N)}(\cdot)\) can be approximated by the deterministic process \(x(\cdot)\) which is called the mean field limit of the system.

Since \(q_0 > q_1\) and \(x(t) \in [0, 1]\) for all \(t \geq 0\), we have from (1) that \(x(t) \geq 0\) for all \(t \geq 0\). Hence, \(x(t) \to 1\) as \(t \to \infty\). The mean consensus time \(\bar{T}_N(\alpha)\) for large \(N\) can therefore be approximated by the time taken by the process \(x(t)\) to reach the state \(1-1/N\) (which corresponds to the situation where all the agents except one agent have opinion \(\{1\}\)) starting with \(x(0) = \alpha\). Hence, by solving (1), we obtain

\[
\bar{T}_N(\alpha) = \frac{1}{q_0 - q_1} \ln(N - 1) - \frac{1}{q_0 - q_1} \ln \left( \frac{\alpha}{1 - \alpha} \right) = O \left( \frac{1}{|q_0 - q_1|} \ln(N - 1) \right)
\]  

(2) Clearly, the mean consensus time scales as \(O(\ln N)\). This is in contrast to the voter model with unbiased agents where the mean consensus time is known to increase linearly with the network size \(N\). Thus, in the case with biased agents, the network reaches the consensus state exponentially faster than that in the case with unbiased agents.

**Numerical Results:** In Figure 1, we plot the exit probability for both biased \((q_0 > q_1)\) and unbiased \((q_0 = q_1)\) cases as functions of the number of agents \(N\) for \(\alpha = 0.2\). For the biased case, we have chosen \(q_0 = 1, q_1 = 0.5\). We observe that in the biased case the exit probability rapidly increases to 1 with the increase in \(N\). This is in contrast to the unbiased case, where the exit probability remains constant at \(\alpha\) for all \(N\).

In Figure 2, we plot the mean consensus time \(\bar{T}_N(\alpha)\) for both the biased and unbiased cases as functions of \(N\) for \(\alpha = 0.4\). We observe that, in the biased case, the consensus state is reached in a time exponentially smaller than that in the unbiased case. This is because the bias of the agents towards one of the opinions drives the system to consensus much faster.

B. The voter model with agents having different degrees of stubbornness

We now consider the case where different agents have different propensities to change their opinions. We note that the case where some agents never update states was studied in [13]. To distinguish our work from this work we consider the following model: Each agent in the network is assumed to belong to one of the two disjoint sets \(\mathcal{S}\) and \(\mathcal{R}\). We denote by \(\gamma_S\) and \(\gamma_R = 1 - \gamma_S\) the fractions of agents belonging to the sets \(\mathcal{S}\) and \(\mathcal{R}\), respectively. Each agent belonging to the set \(\mathcal{S}\) \((\mathcal{R})\) updates its opinion with probability \(q_S\) \((q_R)\) at a point of the unit rate Poisson process associated with itself and retains its opinion with probability \(p_S = 1 - q_S\) \((p_R = 1 - q_R)\). The updates occur according to the voter rule, discussed in the beginning of this section. The probabilities \(q_S\) and \(q_R\) determine the degree of ‘stubbornness’ of agents belonging to the sets \(\mathcal{S}\) and \(\mathcal{R}\), respectively. We set \(q_S < q_R\) to imply that the agents belonging to the set \(\mathcal{S}\) update their opinions less frequently than the agents belonging to the set \(\mathcal{R}\).

The evolution of the network can be described by a two dimensional Markov process \(X^{(N)}(\cdot) = (X_S^{(N)}(\cdot), X_R^{(N)}(\cdot))\), where \(X_S^{(N)}(t)\) and \(X_R^{(N)}(t)\) denote the numbers of agents with opinion \(\{1\}\) in sets \(\mathcal{S}\) and \(\mathcal{R}\), respectively, at time \(t\). Let \((m, n)\) be the state of the process at some instant. The process transits to state \((m + 1, n)\) when one of the \(N\gamma_S - m\) agents with opinion \(\{0\}\) in the set \(\mathcal{S}\) updates its opinion by interacting
with an agent with opinion \( \{1\} \). The rate at which any one of the \( N'y \) agents having opinion \( \{0\} \) in set \( S \) decides to update its opinion is \( \{N'y - m\}q_S \). The probability with which the updating agents samples an agent with opinion \( \{1\} \) from the entire network is \( m/n \). Hence, the total rate of this transition is given by

\[
r((m, n) \to (m + 1, n)) = \frac{(N'y_m - m)(m + n)}{N}q_S
\]  

(3)

The rates of other possible transitions are similarly given by

\[
r((m, n) \to (m, n + 1)) = \frac{(N'y_m - n)(m + n)}{N}q_R
\]  

(4)

\[
r((m, n) \to (m - 1, n)) = \frac{m(N - m - n)}{N}q_S
\]  

(5)

\[
r((m, n) \to (m - 1, n)) = \frac{n(N - m - n)}{N}q_R
\]  

(6)

**Proposition 2:** Let \( E_N(\alpha_S, \alpha_R) \) denote the probability with which the network with \( N \) agents reaches a consensus state with all agents having opinion \( \{1\} \) starting with \( \alpha_S \) (resp. \( \alpha_R \)) fraction of agents of the set \( S \) (resp. \( R \)) having opinion \( \{1\} \). Then

\[
E_N(\alpha_S, \alpha_R) = \sum_{x} q_S^x q_R^{N - x}.
\]  

(7)

**Proof:** Let \( F_t = \sigma(X_N(t); 0 \leq s \leq t) \) denote the history of the process \( X_N(t) \) up to time \( t \geq 0 \). Consider the process \( X_N(t)/q_S + X_N(t)/q_R \). Using the transition rates of the process \( X_N(t) \) it is easy to see that the conditional drift of the process from time \( t \) to time \( t + h \) is given by

\[
E \left[ \frac{X_N(t + h)}{q_S} + \frac{X_N(t + h)}{q_R} \right] = \left( r((m, n) \to (m + 1, n)) - r((m, n) \to (m, n + 1)) \right) q_S + \left( r((m, n) \to (m - 1, n)) - r((m, n) \to (m, n - 1)) \right) q_R + o(h) = o(h)
\]  

(8)

Thus, the process \( X_N(t)/q_S + X_N(t)/q_R \) is a martingale. Let \( T \) denote the random time the process \( X_N(t) \) hits the consensus state. Clearly, \( T \) is an \( F_T \) stopping time. Hence, using optional sampling theorem we have

\[
E \left[ \frac{X_N(T)}{q_S} + \frac{X_N(T)}{q_R} \right] = E \left[ \frac{X_N(0)}{q_S} + \frac{X_N(0)}{q_R} \right]
\]  

(9)

The left hand side of the above equation can be written as

\[
E \left[ \frac{X_N(T)}{q_S} + \frac{X_N(T)}{q_R} \right] = \frac{N'y_S}{N} \alpha_S + \frac{N'y_R}{N} \alpha_R
\]  

(10)

\[
E_N(\alpha_S, \alpha_R) = \frac{N'y_S}{N} \alpha_S + \frac{N'y_R}{N} \alpha_R
\]  

(11)

which simplifies to (7).

**Remark 1:** From (7) we see that the exit probability does not depend on the number of agents \( N \). We also observe that for \( \alpha_S = \alpha_R = \alpha \), the exit probability is given by \( E_N(\alpha, \alpha) = \alpha \), which is also independent of \( q_S \) and \( q_R \).

The mean time \( T_N(\alpha_S, \alpha_R) \) to reach consensus starting with \( \alpha_S \) (resp. \( \alpha_R \)) fraction of agents of the set \( S \) (resp. \( R \)) having opinion \( \{1\} \) can be computed using the first step analysis of the empirical measure process \( x_N(t) = \frac{X_N(t)}{N'y_S} \). The process \( x_N(t) \) changes its state only at points of a rate \( N \) Poisson point process, referred to as the global clock. The probability \( p(x, y) \rightarrow (x + 1/N'y_S, y) \) with which the process transits from the state \( (x, y) \) at one point of the global clock to the state \( (x + 1/N'y_S, y) \) at the next point of the global clock is given by

\[
p( (x, y) \to (x + 1/N'y_S, y) ) = \gamma_S(1 - x)(\gamma_Sx + \gamma_Ry)q_S.
\]  

(12)

Similarly, the probabilities for the other possible transitions are given by

\[
p( (x, y) \to (x, y + 1/N'y_R) ) = \gamma_R(1 - y)(\gamma_Sx + \gamma_Ry)q_R
\]  

(13)

\[
p( (x, y) \to (x - 1/N'y_S, y) ) = \gamma_Sx(1 - \gamma_Sx - \gamma_Ry)q_S
\]  

(14)

\[
p( (x, y) \to (x, y - 1/N'y_R) ) = \gamma_Ry(1 - \gamma_Sx - \gamma_Ry)q_R
\]  

(15)

Since the process \( x_N(t) \) is Markov and the average gap between two points of the global clock is \( 1/N \), we have the following recursive relation

\[
E \left[ \frac{X_N(T)}{q_S} + X_N(T)/q_R \right] = \left( \frac{N'y_S}{N} + \frac{N'y_R}{N} \right) E_N(\alpha_S, \alpha_R) + 0 \times (1 - E_N(\alpha_S, \alpha_R)) \]  

(10)
\[ T_N(x, y) = p \left( (x, y) \to \left(x + \frac{1}{N\gamma_S} y\right)\right) \]
\[ \times \left( T_N \left( x + \frac{1}{N\gamma_S} y + \frac{1}{N} \right) \right) \]
\[ + p \left( (x, y) \to \left(x + \frac{1}{N\gamma_R} y\right) \right) \left( T_N \left( x + \frac{1}{N\gamma_R} y + \frac{1}{N} \right) \right) \]
\[ + p \left( (x, y) \to \left(x - \frac{1}{N\gamma_S} y\right) \right) \left( T_N \left( x - \frac{1}{N\gamma_S} y + \frac{1}{N} \right) \right) \]
\[ + p \left( (x, y) \to \left(x - \frac{1}{N\gamma_R} y\right) \right) \left( T_N \left( x - \frac{1}{N\gamma_R} y + \frac{1}{N} \right) \right) \]
\[ + p \left( (x, y) \to (x, y) \right) \left( T_N \left( x, y + \frac{1}{N} \right) \right) \] \hspace{1cm} (16)

Now using (12), (13), (14), (15) and the Taylor series expansion of \( T_N(\cdot, \cdot) \) of second order around the point \((x, y)\) we have that for large \(N\)

\[ \gamma_{RS}(y - x) \frac{\partial^2 T(x, y)}{\partial x^2} \]
\[ + \frac{qs((\gamma_S + 1)x + \gamma_R y - 2x(\gamma_S x + \gamma_R y))}{2N\gamma_S} \frac{\partial^2 T(x, y)}{\partial x^2} \]
\[ + \frac{\gamma_S q_p(x - y)}{2N\gamma_S} \frac{\partial^2 T(x, y)}{\partial y^2} \]
\[ + \frac{q_R(\gamma_S x + (\gamma_R + 1)y - 2y(\gamma_S x + \gamma_R y))}{2N\gamma_R} \frac{\partial^2 T(x, y)}{\partial y^2} = -1 \]

with boundary condition \( T_N(0, 0) = T_N(1, 1) = 0 \). An approximate solution of the above partial differential equation is given as by

\[ T_N(x, y) = N \left( \frac{\gamma_S}{qs} + \frac{\gamma_R}{q_R} \right) \]
\[ \times \left( \frac{\gamma_{SR} + \gamma_{RS}}{\gamma_{SR} + \gamma_{RS}} x + \frac{\gamma_{QR} + \gamma_{RQ}}{\gamma_{SR} + \gamma_{RS}} y \right) \] \hspace{1cm} (18)

where \( h(z) = -(z \ln z + (1 - z) \ln(1 - z)) \). The approximation is obtained by noting that the above solution is exact for the cases \( \gamma_S = 1, \gamma_R = 0 \) and \( \gamma_S = 0, \gamma_R = 1 \). Moreover, putting the solution in (17) we see that the terms containing first order partial derivatives vanish and the terms containing the second order partial derivatives simplify approximately to \(-1\).

**Numerical results:** To numerically investigate how consensus time varies with the system size \(N\), in Figure 3 we plot the mean consensus time of 1000 independent runs of a network with the following parameters: \(q_S = 0.3, q_R = 1, \alpha_S = \alpha_R = 0.8, \gamma_S = \gamma_R = 0.5\). We observe that the mean consensus time grows linearly with \(N\). In the figure, we have also plotted the mean consensus time obtained using (18). We observe a close match between the simulation result and the approximate result which suggests that the approximation provided in (18) is accurate.

III. The Majority Rule Models

In this section, we consider models where an agent, instead of interacting with a single agent, interacts with multiple agents at an update instant. As before, we assume that the agents in the network consider updating their opinions at points of independent, unit rate Poisson point processes. At a point of the Poisson process associated with itself, an agent either retains its opinion or updates it. If the agent decides to update its opinion, then it interacts with \(2K(K \geq 1)\) agents sampled uniformly at random (with replacement) from the network and adopts the opinion held by the majority of the \(2K + 1\) agents which includes the \(2K\) sampled agents and the updating agent itself. To simplify analysis, we focus on the \(K = 1\) case, where each agent interacts with two agents at its update instant. The analysis is similar for \(K > 1\) and simulations suggest that for large \(N\) varying \(K\) does not change the equilibrium properties of the network significantly.

As in the case of voter models, the decision of an agent to update its opinion is assumed to depend either i) on the current opinion held by the agent or ii) on the propensity of the agent to change opinions. Below we consider these two scenarios separately.

We note that in the majority rule model discussed above, only one agent updates its state, at each time step, by interacting with a group of randomly sampled neighbors. This is different from the previous models studied in the literature [5], [6], [15], where all members of a group of interacting agents were assumed to update their opinions simultaneously.

**A. The majority rule model with biased agents**

As in Section II-A, we first consider the case where the agents are ‘biased’ towards one of the two opinions. More specifically, we assume that an agent with opinion \(i \in \{0, 1\}\) updates its opinion with probability \(q_i\), at a point of the unit rate Poisson process associated with itself. The agent retains its opinion with probability \(p_i = 1 - q_i\). In case the agent decides to update its opinion, the update occurs according to the majority rule discussed in the beginning of this section. We assume \(q_0 > q_1\) to imply that agents with opinion \(\{0\}\) update their opinions more frequently than agents with opinion \(\{1\}\).
Proposition 3: The probability $E_N(n)$ with which the process $X^{(N)}(\cdot)$ gets absorbed in state $N$ starting from state $n \in \{1, 2, \ldots, N\}$ is given by $E_N(n) = \frac{1}{q_0 + q_1} \left(\frac{1}{\kappa_q} \ln(N(1 - \kappa_q) - 1) \right)$, where $r = q_1/q_0 < 1$ and $E_N(0) = 0$.

Proof: The proof is similar to the proof of Proposition 1 and uses the one-step analysis of the Markov chain $X^{(N)}(\cdot)$. We omit the proof due to space constraints.

We now proceed to characterize the mean consensus time of the process by analyzing the mean field limit of the empirical measure process $X^{(N)}(\cdot)/N$. As discussed for the voter model with biased agents, it can be easily verified that in this case the mean field limit $x(\cdot)$ of the process $x^{(N)}(\cdot)$ is given by

$$x(t) = (q_0 + q_1)x(t)(1 - x(t))(x(t) - \kappa_q),$$

where $\kappa_q = q_1/(q_0 + q_1)$.

From (19), it follows that the process $x(\cdot)$ has three equilibrium points at 0, 1, and $\kappa_q$, respectively. We now characterize the stability of these equilibrium points in the sense of the following definition:

Definition An equilibrium point $x_e \in [0, 1]$ of the process $x(\cdot)$ is called stable if there exists an empty set $S \subseteq [0, 1]$ containing $x_e$ but $S \neq \{x_e\}$ such that for all $x(0) \in S$ we have $x(t) \rightarrow x_e$ as $t \rightarrow \infty$. Similarly, for $x(0) \in [0, \kappa_q)$ we have $x(t) \rightarrow 0$ as $t \rightarrow \infty$. Therefore, 0 and 1 are the stable equilibrium points of the process $x(\cdot)$, and $\kappa_q$ is an unstable equilibrium point.

If for some $t \geq 0$ we have $x(t) > \kappa_q$, then (19) implies $x(t) \geq 0$. Hence, for $x(0) \in (\kappa_q, 1)$ we have $x(t) \rightarrow 1$ as $t \rightarrow \infty$. Similarly, for $x(0) \in (0, \kappa_q)$ we have $x(t) \rightarrow 0$ as $t \rightarrow \infty$. Therefore, 0 and 1 are the stable equilibrium points of the process $x(\cdot)$, and $\kappa_q$ is an unstable equilibrium point.

If $x^{(N)}(0) = \alpha = \kappa_q + \epsilon (\epsilon > 0)$, then, for large $N$, with high probability the process $x^{(N)}(\cdot)$ reaches the state 1 in a finite time. This is because for large $N$ the path of the process $x^{(N)}(\cdot)$ is close to that of $x(\cdot)$ with high probability (by the mean field convergence result) and we have already shown that with $x(0) = \alpha > \kappa_q$, $x(t) \rightarrow 1$ as $t \rightarrow \infty$. Therefore, the mean consensus time for large $N$ and $\alpha = \kappa_q + \epsilon$ can be approximated as the time taken by the process $x(\cdot)$ to reach the state 1 $- 1/N$ from the state $\alpha = \kappa_q + \epsilon$. We denote the approximate mean consensus time by $\bar{T}_N(\kappa_q + \epsilon)$. Now, solving (19) with the above limits we obtain

$$\bar{T}_N(\kappa_q + \epsilon) = \frac{1}{q_0 + q_1}\left[\frac{1}{\kappa_q(1 - \kappa_q)} \ln(N(1 - \kappa_q) - 1) - \frac{1}{\kappa_q} \ln(N - 1) - \frac{1}{\kappa_q} \ln(1 - \kappa_q) - \frac{1}{\kappa_q - \kappa_q} \ln(1 - \kappa_q - \epsilon) \right].$$

The expression for $\bar{T}_N(\alpha)$, for $\alpha = \kappa_q - \epsilon$, can be obtained similarly. From the above expressions, it is clear that the mean consensus time scales as $O(\ln N)$.

**Numerical Results:** In Figure 4, we plot the exit probability $E_N(\alpha)$ as a function of the number of agents $N$. Parameters: $q_0 = 1, q_1 = 0.6$.

![Fig. 4. Majority rule with biased agents: Exit probability $E_N(\alpha)$ as a function of the number of agents $N$. Parameters: $q_0 = 1, q_1 = 0.6$.](image-url)

B. The majority rule with stubborn agents

We now consider the majority rule model with agents having different propensities to change their opinions. Specifically, we assume that some of the agents in the network never update their opinions. We call these agents as the **stubborn** agents. The other agents, referred to as the **non-stubborn** agents, are assumed to update their opinions at all points of the Poisson processes associated with themselves. The updates occur according to the majority rule discussed in the beginning of this section. We denote by $\gamma_i, i \in \{0, 1\}$, the fraction of agents in network who are stubborn and have opinion $i$ at all times. Thus, $(1 - \gamma_0 - \gamma_1)$ is fraction of non-stubborn agents in the network.

The presence of stubborn agents prevents the network from reaching a consensus state. This is because at all times there are at least $N\gamma_0$ stubborn agents having opinion $\{0\}$ and $N\gamma_1$ stubborn agents having opinion $\{1\}$. Furthermore, since each non-stubborn agent may interact with some stubborn agents at every update instant, it is always possible for the non-stubborn agent to change its opinion. Below we characterize the equilibrium fraction of non-stubborn agents having opinion $\{1\}$ in the network for large $N$ using mean field techniques.

Let $x^{(N)}(t)$ denote the fraction of non-stubborn agents having opinion $\{1\}$ at time $t \geq 0$. Clearly, $x^{(N)}(\cdot)$ is a Markov process with possible jumps at the points of a rate $N(1 - \gamma_0 - \gamma_1)$ Poisson process. The process $x^{(N)}(\cdot)$ jumps from the state $x$ to the state $x + 1/N(1 - \gamma_0 - \gamma_1)$ when one of the non-stubborn agents having opinion $\{0\}$ becomes

---

267
active (which happens with rate $N(1-\gamma_0-\gamma_1)(1-x)$) and samples two agents with opinion $\{1\}$. The probability of sampling an agent having opinion $\{1\}$ from the entire network is $(1-\gamma_0-\gamma_1)x + \gamma_1$. Hence, the total rate at which the process transits from state $x$ to the state $x + 1/N(1-\gamma_0-\gamma_1)$ is given by $r\left(\begin{array}{c} x \\ x + 1/N(1-\gamma_0-\gamma_1) \end{array}\right) = N(1-\gamma_0-\gamma_1)(1-x)(1-\gamma_0-\gamma_1)x + \gamma_1)^2$. Similarly, the rate of the other possible transition is given by $r\left(\begin{array}{c} x \\ x + 1/N(1-\gamma_0-\gamma_1) \end{array}\right) = N(1-\gamma_0-\gamma_1)(1-x)(1-x) + \gamma_1)^2$. Using the same line of arguments as discussed for the voter model with biased agents, it can be shown from above transition rates that the process $x^{(N)}(\cdot)$ converges weakly to the mean field limit $x(\cdot)$ which satisfies the following differential equation

$$
\dot{x}(t) = (1-x(t))(1-\gamma_0-\gamma_1)x(t) + \gamma_1)^2 - x(t) \times [(1-\gamma_0-\gamma_1)(1-x(t)) + \gamma_1)^2].
$$

We now study the equilibrium distribution $\pi_N$ of the process $x^{(N)}(\cdot)$ for large $N$ via the equilibrium points of the mean field $x(\cdot)$.

From (21) we see that $\dot{x}(t)$ is a cubic polynomial in $x(t)$. Hence, the process $x(\cdot)$ can have at most three equilibrium points in $[0,1]$. We first characterize the stability of these equilibrium points according to Definition III-A.

**Proposition 4:** The process $x(\cdot)$ defined by (21) has at least one equilibrium point in $(0,1)$. Furthermore, the number of stable equilibrium points of $x(\cdot)$ in $(0,1)$ is either two or one. If there exists only one equilibrium point of $x(\cdot)$ in $(0,1)$, then the equilibrium point must be globally stable (attractive).

**Proof:** Define $f(x) = (1-x)(1-\gamma_0-\gamma_1)x + \gamma_1)^2 - x(1-\gamma_0-\gamma_1)(1-x) + \gamma_1)^2$. Clearly, $f(0) = \gamma_1^2 > 0$ and $f(1) = -\gamma_0^2 < 0$. Hence, there exists at least one root of $f(x) = 0$ in $(0,1)$. This proves the existence of an equilibrium point of $x(\cdot)$ in $(0,1)$.

Since $f(x)$ is a cubic polynomial and $f(0)f(1) < 0$, either all three roots of $f(x) = 0$ lie in $(0,1)$ or exactly one root of $f(x) = 0$ lies in $(0,1)$. Let the three (possibly complex and non-distinct) roots of $f(x) = 0$ be denoted by $r_1, r_2, r_3$, respectively. By expanding $f(x)$ we see that the coefficient of the cubic term is $-2(1-\gamma_0-\gamma_1)^2$. Hence, $f(x)$ can be written as

$$
f(x) = -2(1-\gamma_0-\gamma_1)^2(x-r_1)(x-r_2)(x-r_3).
$$

We first consider the case when $0 < r_1, r_2, r_3 < 1$ and not all of them are equal. Let us suppose, without loss of generality, that the roots are arranged in the increasing order, i.e., $0 < r_1 < r_2 < r_3$ or $0 < r_1 < r_3 < r_2 < 1$. From (22) and (23), it is clear that, if $x(t) > r_2$ and $x(t) > r_3$, then $\dot{x}(t) < 0$. Similarly, if $x(t) > r_2$ and $x(t) < r_3$, then $\dot{x}(t) > 0$. Hence, if $x(0) > r_2$ then $x(t) \to r_2$ as $t \to \infty$. Using similar arguments we have that for $x(0) < r_2$, $x(t) \to r_2$ as $t \to \infty$. Hence, $r_1, r_3$ are the stable equilibrium points of $x(\cdot)$. This proves that there exist at most two stable equilibrium points of the mean field $x(\cdot)$.

Now suppose that there exists only one equilibrium point of $x(\cdot)$ in $(0,1)$. This is possible either i) if there exists exactly one real root of $f(x) = 0$ in $(0,1)$, or ii) if all the roots of $f(x) = 0$ are equal and lie in $(0,1)$. Let $r_1$ be a root of $f(x) = 0$ in $(0,1)$. Now by expanding $f(x)$ from (22), we see that the product of the roots must be $\gamma_1^2/(2(1-\gamma_0-\gamma_1)^2) > 0$. This implies that the other roots, $r_2$ and $r_3$, must satisfy one of the following conditions: 1) $r_2, r_3 > 0$, 2) $r_2, r_3 < 0$, 3) $r_2, r_3$ are complex conjugates, 4) $r_2 = r_3 = r_1$.

In all the above cases, we have that $(x-r_2)(x-r_3) \geq 0$ for all $x \in [0,1]$ with equality if and only if $x = r_1 = r_2 = r_3$. Hence, from (22) and (21), it is easy to see that $\dot{x}(t) > 0$ when $0 \leq x(t) < r_1$ and $\dot{x}(t) < 0$ when $1 \geq x(t) > r_1$. This implies that $x(t) \to r_1$ for all $x(0) \in [0,1]$. In other words, $r_1$ is globally stable.

Hence, depending on the values of $\gamma_0$ and $\gamma_1$ there may exist of multiple stable equilibrium points of the mean field $x(\cdot)$. However, for every finite $N$, the process $x^{(N)}(\cdot)$ has a unique stationary distribution $\pi_N$ (since it is irreducible on a finite state space). In the next result, we establish that any limit point of the sequence of stationary probability distributions $(\pi_N)_N$ is a convex combination of the Dirac measures concentrated on the equilibrium points of the mean field $x(\cdot)$ in $[0,1]$.

**Theorem 1:** Any limit point of the sequence of probability measures $(\pi_N)_N$ is a convex combination of the Dirac measures concentrated on the equilibrium points of $x(\cdot)$ in $[0,1]$. In particular, if there exists a unique equilibrium point $r$ of $x(\cdot)$ in $[0,1]$ then $\pi_N \Rightarrow \delta_r$, where $\delta_r$ denotes the Dirac measure concentrated at the point $r$.

**Proof:** We first note that since the sequence of probability measures $(\pi_N)_N$ is defined on the compact space $[0,1]$, it must be tight. Hence, Prokhorov’s theorem implies that $(\pi_N)_N$ is relatively compact. Let $r$ be any limit point of the sequence $(\pi_N)_N$. Then by the mean field convergence result we know that $\pi$ must be an invariant distribution of the maps $\alpha \mapsto x(t,\alpha)$ for all $t \geq 0$, i.e., $\int \varphi(x(t,\alpha))d\pi(\alpha) = \int \varphi(\alpha)d\pi(\alpha)$, for all $t \geq 0$, and all continuous (and hence bounded) functions $\varphi : [0,1] \to \mathbb{R}$. In the above, $x(t,\alpha)$ denotes the process $x(\cdot)$ started at $x(0) = \alpha$. Hence we have

$$
\int \varphi(\alpha)d\pi(\alpha) = \lim_{t \to \infty} \int \varphi(x(t,\alpha))d\pi(\alpha)
$$

(23)

$$
= \int \varphi(t \to \infty) x(t,\alpha) d\pi(\alpha)
$$

(24)

The second equality follows from the first by the Dominated convergence theorem and the continuity of $\varphi$. Now, let $r_1, r_2, r_3$ denote the three equilibrium points of the mean field $x(\cdot)$. Hence, by Proposition 4 we have that for each $\alpha \in [0,1]$, $\varphi(t \to \infty) x(t,\alpha) = \varphi(r_1)I_{N_{r_1}}(\alpha) + \varphi(r_2)I_{N_{r_2}}(\alpha) + \varphi(r_3)I_{N_{r_3}}(\alpha)$, where for $i = 1,2,3$, $N_i, \in [0,1]$ denotes the set for which if $x(0) \in N_i$, then $x(t) \to r_i$ as $t \to \infty$, and $I$ denotes the indicator function. Hence, by (24) we have that for all continuous functions $\varphi : [0,1] \to \mathbb{R}$

268
This proves that $\pi$ must be of the form $\pi = c_1 \delta_{x_1} + c_2 \delta_{x_2} + c_3 \delta_{x_3}$, where $c_1, c_2, c_3 \in [0, 1]$ are such that $c_1 + c_2 + c_3 = 1$. This completes the proof.

Thus, according to the above theorem, if there exists a unique equilibrium point of the process $x(\cdot)$ in $[0,1]$, then the sequence of stationary distributions $(\pi_N)_{N \in \mathbb{N}}$ concentrates on that equilibrium point as $N \to \infty$. In other words, for large $N$, the fraction of non-stubborn agents having opinion 1 (at equilibrium) will approximately be equal to the unique equilibrium point of the mean field.

If there exist multiple equilibrium points of the process $x(\cdot)$ then the convergence $x^{(N)}(\cdot) \Rightarrow x(\cdot)$ implies that at steady state the process $x^{(N)}(\cdot)$ spends intervals near the region corresponding to one of the stable equilibrium points of $x(\cdot)$. Then due to some rare events, it reaches, via the unstable equilibrium point, to a region corresponding to the other stable equilibrium point of $x(\cdot)$. This fluctuation repeats giving the process $x^{(N)}(\cdot)$ a unique stationary distribution. This behavior is formally known as metastability.

To demonstrate metastability, we simulate a network with $N = 100$ agents and $\gamma_0 = \gamma_1 = 0.2$. For the above parameters, the mean field $x(\cdot)$ has two stable equilibrium points at $0.327322$ and $0.872678$. In Figure 5, we show the sample path of the process $x^{(N)}(\cdot)$. We see that at steady state the process switches back and forth between regions corresponding to the stable equilibrium points of $x(\cdot)$. This provides numerical evidence of the metastable behavior of the finite system.

IV. Conclusion

In this paper, we analyzed the voter models the majority rule based model of social interaction under the presence of biased and differently stubborn agents. We observed that for the voter model, the presence of biased agents, reduces the mean consensus time exponentially in comparison to the voter model with unbiased agents. For the majority rule model with biased agents, we saw that the network reaches the consensus state with all agents adopting the preferred opinion only if the initial fraction of agents having the preferred opinion is more than a certain threshold value. Finally, we have seen that for the majority rule model with stubborn agents the network exhibits metastability, where it fluctuates between multiple stable configuration, spending long intervals in each configuration.

REFERENCES


LiveTalk: A Framework for Collaborative Browser-Based Replicated-Computation Applications

Matthew Hemmings
University of Victoria
mhemming@uvic.ca

Daniel Ingalls
Y Combinator Research
danhingalls@gmail.com

Robert Krahn
Y Combinator Research
robert.krahn@cdglabs.org

Rick McGeer
US Ignite
rick.mcgeer@us-ignite.org

Glenn Ricart
US Ignite
glenn.ricart@us-ignite.org

Marko Röder
Y Combinator Research
marko.roeder@cdglabs.org

Ulrike Stege
University of Victoria
ustege@cs.uvic.ca

Abstract—In this paper we describe LiveTalk, a framework for Collaborative Browser-based Replicated-Computation applications. LiveTalk permits multiple users separated across the wide area to interact with separate copies of a single application, sharing a single virtual workspace, using very little network bandwidth. LiveTalk features an integrated, browser-based programming environment with native graphics and live evaluation, an integrated, pluggable web server, and a simple messaging service that serves to coordinate activity on shared application sessions, and provides for multiple, mutually-isolated sessions. The first use case for LiveTalk are collaborative big-data visualizations running on thin-client devices such as cellular phones, tablets, and netbooks. These applications form part of a new class of application where the distributed Cloud is leveraged to provide low latency, and high-bandwidth access to geographically disparate users while maintaining the feel of immediacy associated with local computation. The primary motivation of this work is to permit low latency, collaborative applications to be built quickly and easily, while requiring no setup for use by the end-user.

I. INTRODUCTION AND MOTIVATION

Cloud-based applications have a substantial number of advantages over fat-client-based desktop applications, one reason many applications have migrated to the web and the Cloud in recent decades. Application state is resident, not on a desktop, but in the Cloud, and is generally immune from local failures such as disk crashes, loss or theft of client equipment, and so on; users can access the application from anywhere there is network connectivity, using any device, including smartphones, tablets, and netbooks; the application runs in a controlled, known environment, eliminating many porting difficulties. For enterprises, thin clients and servers are easier to maintain than fat desktop PCs. Routine Information Technology tasks – data backup, etc. – are handled by the Cloud provider automatically. There is no need for on-site support or maintenance.

Collaboration is inherent in Cloud applications. An excellent example is the Google suite of productivity applications: Google Docs, Google Sheets, and Google Slides. At first glance, these are mere clones of a standard desktop suite, such as Microsoft Office or Open Office. However, a major feature is collaborative editing. Document co-authors can see, in real time, who else is editing the document and the changes that each one is making, which radically reduces the standard revision cycle of modify-then-email.

Cloud platforms and operating systems tend to be more secure than desktop systems, largely due to their heritage – most descend from the more robust and more secure Unix platform. Further, it is far easier to recover a Cloud platform from malware or attack: one simply deletes the infected virtual machine and restores to a pristine, pre-infected image.

Moreover, Cloud-based applications have now become possible, largely due to the HTML5 platform. The web has completed its evolution from being a document-exchange system to a full-fledged programming platform, with every feature available on a traditional desktop system. This includes a rich graphics platform, including 3D graphics.

There are three limitations that have slowed the uptake of Cloud-based applications:

- The need for a ubiquitous Internet connection, and in many cases a very high-speed connection.
- The remote nature of current “Classic Cloud” platforms, where the closest Cloud node is hundreds or thousands of kilometers distant, dramatically reduces the responsiveness of Cloud-based applications. Many applications are “fat data” systems – they require large data transfers between the Cloud node and the web client. For this reason, data-heavy applications such as Geographic Information Systems rely on local fat-client implementations, in order to minimize latency in response to a user request.
- Bandwidth costs, particularly for long-distance transmissions, are not dropping as rapidly as computation costs. What matters is the ratio between computation and communications costs, since the former represents our ability to capture, generate, and process data, and the latter our ability to transmit it. This ratio has dropped by two orders of magnitude over the past 20 years[12], which means that networks are becoming increasingly over-stretched. On the side of the application consumer, this disparity appears as per-month bandwidth caps; on the side of the application provider, as direct bandwidth charges; and on the side of the network-provider, as an exponentially increasing demand for bandwidth.
- Privacy and regulatory concerns often dictate that
application data must be kept in specific political jurisdictions. For example, many universities in the United States have outsourced their office applications and email to Google, using locally-branded versions of GMail and Google Apps for Enterprise. In contrast, universities in the Canadian province of British Columbia are required by local law to keep student data in Canada.

The need for an Internet connection is immutable. However, the rise of the distributed Cloud eliminates the other limitations, by bringing the Cloud closer to the user. This increases responsiveness and bandwidth to the user dramatically, effectively to the responsiveness of a program resident to a desktop PC, and eliminates regulatory and jurisdictional questions because application data is now resident in the user’s jurisdiction. Moreover, bandwidth costs are primarily long-haul costs on level-2 and level-3 providers; local bandwidth is cheap and plentiful.

This has led to the development of a number of application-specific Edge Clouds. The most familiar of these are content-distribution networks such as Akamai, Coral, CoDeeN, and the Google Global Cache. However, sufficiently popular single applications develop application-specific Edge Clouds interconnected by private networks. For example, the League of Legends game is the leader in e-sports, with the world championship filling 20,000-seat basketball and hockey arenas. Fair competition requires that the game server be under the control of League of Legends, and be equidistant from, and close to, the competing parties – no home-server advantage, and with guaranteed bandwidth to the players (League of Legends is a classic fat-client game, so in practice bandwidth requirements are relatively modest; but quality-of-service is vital). For this reason, Riot Games, the maker and operator of League of Legends, operates its own Edge Cloud of League of Legends servers with a private network between them. Riot’s explanation for this is:

“Currently, ISPs focus primarily on moving large volumes of data in seconds or minutes, which is good for buffered applications like YouTube or Netflix but not so good for real-time games, which need to move very small amounts of data in milliseconds. On top of that, your internet connection might bounce all over the country instead of running directly to where it needs to go, which can impact your network quality and ping whether the game server is across the country or right down the street.

This is why we’re in the process of creating our own direct network for League traffic and working with ISPs across the US and Canada to connect players to this network.”[33]

For this reason, there have been a number of implementations of Distributed Edge Clouds over the past decade, from PlanetLab[32], to GENI[8], [29], SAVI[22], and G-Lab[31]. These can be made very efficient[6], [7], [4], and in fact a downloadable Edge Cloud which can run in any available VM has been proposed[5]. These are all Platform-as-a-Service (PaaS) Distributed Clouds, which offer some variant of Linux Virtual Machines or Docker Containers as the execution environment.

Platform-as-a-Service environments are common programming environments instantiated on top of Infrastructure-as-a-Service (IaaS) platforms. These offer both higher-level programming abstractions and more facilities for the application developer and greater efficiency in the use of the underlying infrastructure. Commercial examples include Heroku[30], AWS Elastic Beanstalk[43], and Google App Engine[36]. Though some features of commercial PaaS facilities are not relevant for a distributed Cloud environment (e.g., automated scaling), the productivity enhancements certainly are. LiveTalk is a PaaS environment tuned to delivering sophisticated Cloud applications in a web browser. We describe LiveTalk here.

The remainder of this paper is organized as follows. In Section II we describe the architecture and initial implementation of LiveTalk. In Section III we discuss the inter-instance messaging system used to coordinate activities in LiveTalk. In Section IV we describe two example applications that we have built on the LiveTalk framework. In Section V, we discuss conclusions and future work.

II. THE LIVETALK ARCHITECTURE

A typical PaaS environment is a web framework with an automated deployment capability. To a developer, it looks very much like programming Ruby on Rails[40], Django[13] or Node.js[35], with the various deployment scripts and server plumbing taken care of. Client support is generally limited to a thin overlay on HTML, usually a template engine such as Jinja2[17] or Jade[16]. The major features of such frameworks is that database access and manipulation is done transparently through an API, so there appears to be little difference between database manipulation and manipulating a program’s data structure, and REST requests are transparently translated into method calls.

LiveTalk differs from most PaaS environments in that it presents a much higher-level client abstraction than raw HTML5. The foundational layer of LiveTalk is the Lively Web[23], [19], [39]. The client abstraction presented by Lively is a graphical user interface based on the Morphic[26] graphical interface, pioneered in Self[42] and used in Squeak[15] and Scratch[25]. Though the Lively UI is rendered using HTML5, this is very much a low-level rendering tool. The actual objects created and manipulated in a Lively Web page (called a “world”) are Morphic objects. Morphic is a user interface framework that supports composable graphical objects, along with the machinery required to display and animate these objects, handle user inputs, and manage underlying system resources such as HTML tags, CSS styles, fonts, and color maps. A primary goal of Morphic is to make it easy to construct and edit interactive graphical objects, both programmatically and...
by direct manipulation.

The easiest way to think about Lively is to consider it a JavaScript-based implementation of Squeak, its immediate predecessor. From Squeak it inherits its fundamental UI and live-evaluation programming semantics (any piece of text in a Lively world can be evaluated as a JavaScript expression); since it is implemented in JavaScript and runs in any browser, any JavaScript library or HTML widget can be incorporated in a Lively world.

Lively comes with an integrated editor, Ace[1], an integrated Wiki for version control, an integrated WebDAV[10] environment for server-side file storage, an integrated SQLite[2] database with client- and server-side APIs, an integrated messaging system which uses socket.io[20] as an underlying transport layer, and an integrated, pluggable Node.js server which both acts to serve the Lively pages themselves and can be used to add server-side code. And just like the client, the server can be programmed through an editor on a Lively page.

The Node.js pluggable server with an integrated file store (WebDAV) and SQLite database would, by themselves, make Lively a competitive web framework. However, the client-side Morphic-based programming environment, integrated Wiki, and messaging system are unequalled in other web development environments. Of particular note is that the WebDAV, database, and messaging systems all have substantial client-side APIs, so for many applications server-side programming is entirely unnecessary; indeed, we have built two-player games with no server-side programming at all.

The architecture for LiveTalk is shown in Figure 1. It should be noted that most of the features of the LiveTalk system are present in its Lively Web base; the contribution for a distributed environment is an enhanced messaging system to support privacy and isolation in collaborative applications and its integration with the GENI Experiment Engine (and, eventually, the PlanetIgnite) Distributed IaaS platform.

Developers writing a LiveTalk application will typically encapsulate the client-facing portion of the system in Lively worlds (Morphic applications written in JavaScript and encapsulated on a web page) that are stored in the Wiki-based application repository. In many cases, they will write a Node.js subserver as well, or (to avoid fate-sharing) write a separate data server which can be deployed alongside the LiveTalk server, in a companion container. They may also interact directly with the embedded server-side SQLite database through the provided API and/or use the integrated WebDAV client-side API to manipulate server-side files.

III. THE LIVE TALK MESSAGING SYSTEM

The LiveTalk messaging system is based on the Lively Web’s integrated Lively2Lively message protocol. The goals of Lively2Lively were to permit:

- Pure message based communication between Lively worlds,
- A transparent network abstraction, with simple put/get semantics on messages,
- Graceful handling of churn and parties leaving and entering systems, and
- Automated handling of messages.

The Lively2Lively system ensures reliable delivery and participant tracking in a worldwide setting through a hierarchy of session trackers, using either a client-side or a server-side API; most applications use the client-side API. Each session starts with a unique, randomly chosen UUID that registers with a session tracker on the Lively server that served the world. Sessions can use the local API to discover other sessions, by URL, user, or UUID, to send messages and receive messages.

The wire protocol for a Lively2Lively message is a JSON object, which must have the fields action (a string) and target (a UUID), which specifies, respectively, the subject and receiver of the message. Optional fields specify the sender, a unique ID for the message, an identifier of a message which prompted this message as a response, if any, and a data field, which is a JSON object containing any message data.

A session registers to handle messages of for a specific action with the registerActions method in the Lively2Lively session tracker. registerActions takes a JavaScript dictionary as an argument, where the entry messageName: function(msg, session) indicates that the function on the right-hand side is used to handle messages with the action messageName.

Message routing is handled seamlessly by the Lively servers in a manner similar to SMTP[18]. The server’s session tracker determines if it is the designated recipient of the request. If so, it delivers the message. If the recipient is a session tracked by this server, it delivers to that session, otherwise it passes it to a peer for delivery.

Lively2Lively offers the base message delivery layer for LiveTalk, but further functionality is required for a distributed application platform. We see the architecture of a distributed messaging system as similar to the stacked
services in the Internet, where each layer offers functionality building on the layer below. An architectural diagram showing some of the services appears in Figure 2.

Lively2Lively forms the base layer of the messaging system: discovery and reliable transmission, including – in the future – scalability features like multicast and publish/subscribe. The LiveTalk messaging system focuses on the next layers above the discovery and transmission layer: attaching actions to objects and offering protection and privacy for message users and groups.

We begin with the first problem: attaching messages to objects and choosing which recipient on a page is the intended recipient of a message. The choice for the first question is what is the appropriate granularity of a message participant? The choices are: Lively worlds, Lively2Lively sessions, individual morphs, or a new, abstract entity.

Worlds and Lively2Lively sessions are too coarse for the appropriate interface. A world may have multiple morphs participating in various different applications, and the odds of a name conflict are quite high. Since morphs are the general unit in Lively, using a Morph is tempting. However, conversation participants do not need to be tied to a specific graphical object, a single graphical object can have multiple messaging interfaces, and, perhaps most of all, there is no compelling reason to do it. One must remember the First Commandment of Modularity: “Thou Shalt not Overload a Concept Without an Excellent Reason”.

Conversations Our choice is therefore a new object, the Conversation. We follow the JavaScript convention: a Conversation is simply an object, which defines a set of messages to which a Conversation object responds. A Conversation also has a name, which defines the set of messages to which it responds.

The Conversation resembles nothing so much as a Java Interface, and faces the same basic constraint: the name of a Conversation must be unique across applications. And thus we chose the Java solution: to use a reversed URL, followed by the user id of the Conversation developer, followed by the name of the Conversation (chosen arbitrarily by the developer). Dots are used to separate the fields, and developers can choose a hierarchical namespace for their own Conversations if they prefer. An example Conversation name is org.lively-web.www.matt.visualizer, or org.lively-web.www.matt.visualizer.pollution.

The Conversation name defines a set of messages, so each Conversation has a method: messages which returns the names of the messages to which a participant in the Conversation responds. The full name of a message is the name of the Conversation followed by the name of the message, separated (as always) by a dot. For example, the showData message of the conversation

1Like the other commandments of information technology (“Real Operating Systems end in X”; “If It doesn’t install with apt-get, walk away”; “Design the API first, then the GUI”; “Java? Ewww...”) the punishment for disobeying this Commandment is death by a million bugs

Groups The unit of protection is the Group. A group is simply a name, an owner, a set of receivers, a key, and a Conversation. Every user – owner or receiver – is a username and the according UUID. A group’s owner decides on admissions to the group.

To join a group, an object calls LiveTalk.joinGroup(conversationName, groupName, messageMapping), where messageMapping is an object of the form messageName: function(message), declaring the message handlers for each message in a Conversation. The return value of joinGroup is a Group object, or None if the joining the group failed. The message to join a group is delivered to the group owner (person or agent) which returns the appropriate response.

The key is optional and is used for group security: to prevent non-members from sending messages to the group. When members send a message to the group, they cryptographically signs it using the key. On receipt, the LiveTalk session manager reads the signature, and, if the message is signed appropriately, invoke the registered message handler with the message as an argument.

The resulting, typical information and message flow is shown in Figure 3. The flow is depicted by dashed arrows and data requests and transfers by solid arrows; the numbers on the arrows represent the time sequence of message flows. One participant in a LiveTalk-enabled application performs an action which manipulates the shared visual space. This will typically, but not always, involve fetching data from the server. The LiveTalk application will simultaneously issue a data request to the server and issue a message to the other participants describing the manipulation of the shared space. When the data request is serviced, the local screen is redrawn in response to the request. Once the message is received by a remote participant, the remote handler will make a duplicate data request to its local server and fetch the data.

The message to the other participants will often route through the network of servers participating in the Conversation; however, this is not a requirement of the protocol. Other implementations, including direct peer-to-peer or routing through a messaging substrate such as a Dis-
tributed Hash Tables\cite{28} are possible.

The typical flow above dictates a common design pattern for LiveTalk applications, which is shown in Figure 4. A user action is serviced by an action handler method, which will typically do two things: the first is bundle up a data structure describing the user request and send it off as a message to the group, and the second is to invoke the action the user expects. Upon message receipt, the message handler interprets the message, then invokes the same action on the remote system. To most of the substrate of the application, there is no difference between a user action and an incoming message: only the message-handling/user-facing layer can see a difference.

The result is that inter-user latencies are only visible in two cases: the first is if two users attempt near-simultaneous (roughly, \(< 1\) round trip delay time) and the second is if there is out-of-band communication which has less latency than the messaging substrate.

IV. EXAMPLE LIVE TALK APPLICATIONS

This section will discuss two collaborative visualizations built in LiveTalk and deployed across the GENI Experiment Engine and SAVI Infrastructures.

A. Pollution Visualization

This visualization, pictured in Figure 5, is constructed of a type where there is a large data set stored locally. The specific dataset is the concentration of 2.5-micron particulate data (PM2.5) on a worldwide 10-km grid\cite{21}. Coarser grids of 25 km, 50 km, and 100 km are obtained from the original grid and stored alongside it. The database stores a series of triples containing the longitude, latitude, and scalar value. A snapshot exists for each month from 1997 to 2015, resulting in a database 11 GB in size. Each transaction requires fetching up to 30,000 points from the server, about an 800 KB transfer. The scalar values are displayed as color-coded rectangles on a map.

The data for each interaction is so large that it must be fetched from the local server. See below for an analysis. Collaborative interaction is done using the messaging system described above. The data and message flow follow the flows in Figure 3, and the general architecture of the client system follows that shown in Figure 4. A user interaction with the system selects a map bounding box, desired resolution (10, 25, 50, or 100 km), an opacity for the rectangles, a zoom level, a month, a year, and a maximum value, which scales the color coding. This forms the basis for a bounding box request to the database and sets drawing parameters for the application.

In addition to the data request (not shown) the user interaction generates a message sent to all other Conversation participants giving the relevant data to make a local database request and draw the results on the screen. A sample message is shown in Listing 1

```
Listing 1. Pollution Message

{
  "action":"changeMap",
  "data": {
    "user":"MattH",
    "zoom": "2.2",
    "opacity": "0.5",
    "resolution":10,
    "center": {
      "lon":388.64,
      "lat":23.76
    },
    "maxVal": "40",
    "month":9,
    "year":2004,
    "monthUpdate":true
  },
  "sender":"client−session:..",
  "target":"client−session:..",
  "messageId":"client−msg:..",
  "messageIndex":17
}
```

The wrapping surrounding the action message is the Lively2Lively protocol information, containing who sent the message, and what the message index was on the server. When a user does any manipulation of the map, such as moving the map, zooming in or out, changing the maximum pollution density, changing the date being viewed or the opacity of the points being drawn, a message containing all this information is sent to all other members of the collaborative group in the changeMap action. The registered handler for changeMap is executed on each listener.

1) Analysis: This application was built to meet the metric of having an action be received, looked up and rendered in 150ms, as this has been established as the
Table I  
LATENCY AND BANDWIDTH ESTIMATES

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Latency</th>
<th>Estimated Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Campus</td>
<td>1 ms</td>
<td>1 Gb/s</td>
</tr>
<tr>
<td>City</td>
<td>5 ms</td>
<td>1 Gb/s</td>
</tr>
<tr>
<td>Continent</td>
<td>50 ms</td>
<td>100 Mb/s</td>
</tr>
<tr>
<td>World</td>
<td>250 ms</td>
<td>100 Mb/s</td>
</tr>
</tbody>
</table>

amount of time before users begin to turn away [37], [38], [41]. The Polymaps mapping library was chosen as the basic rendering application, after a series of tests of various mapping libraries demonstrated that this was the highest-performing browser-based mapping library. It was determined that Polymaps [34] was able to render 30,000 points in 100ms, giving a total of 50 ms to send the request to the server, read the data from the database, and send the results over the wire. A special-purpose QuadTree database, described below, was written to fetch the data in 20 ms. A number of network scenarios were calculated, and the results shown in Table I.

“Campus” assumes a server on the same campus where the viewer is located and gigabit bandwidth present. “City” assumes a server within approximately 100 km, but not sharing campus internet with the viewer. “Continent” assumes that there are one or two servers per continent with the viewer connecting to the one with the lower latency, similar in style to Amazon’s AWS service. Finally, “World” assumes that there is a single server for global use and that everyone who sends a data request is sending it to the same one. 30,000 points was chosen as the target transaction size, since this suffices to display about a quarter-continent at 10 km resolution and the world at 100-km resolution; the application automatically chooses the finest resolution for a bounding box that will fit into 30,000 points. The wire protocol has about 27 bytes/point, so a size of 30,000 points gives an 810 KB transfer, or a 6.48 Mbit transfer. 1500-byte packets were assumed, with accelerated slowstart and an initial window size of 15 KB.

Computational analysis demonstrated the only feasible scenarios were “Campus” and “City”, indicating that each user must have a server within 100 km; since a user could be anywhere, this implies that servers must be everywhere. To approximate this on a continent scale, the GENI and SAVI infrastructures were used. The combination of the GENI and SAVI infrastructures, and LiveTalk result in a desktop-application-like application on a thin client, collaboratively, anywhere.

2) Pollution Quadtree Server: A special-purpose QuadTree server was used to rapidly fetch the data points. Edge servers are resource-poor, and therefore in-memory databases are not possible for a dataset of this size. Further, and geo database creates index sets which are too large for the available disk, so a special-purpose server was written to fetch the dataset from an on-disk quad tree structure, where each leaf cell was a separate file.

To avoid fate-sharing, the quadtree server was not incorporated into the LiveTalk system but was run as a standalone Flask[11] server in a separate container in the same virtual machine as the LiveTalk server.

B. NBA Shot Chart Visualization

The second application constructed on this framework is an example of visualizing sports statistics. In the National Basketball Association (NBA), one measure of determining efficacy of a player is to look at their made and missed shots. With a high enough sample set, it becomes apparent that certain players have hot zones where they shoot more effectively than others. The hot zones and shot statistics of players are of great interest to basketball aficionados among both the media and the general public.

To serve this interest, beginning in the 2013-14 season, the NBA installed SportVU [24] cameras, 6 per arena, in the catwalks. These cameras each record an image every 40 milliseconds, feed this data into specialized software and then made available to various applications both in and out of the arena. This data is available live during the game and is also available for past games.

An application to visualize this data is shown in Figure 6. It has varying ways of narrowing down the selections available, but in the final box, a selection of a player’s name will display on the chart above their made and missed shots, color-coded. This data is queried live from the NBA stats API [3], and is fetched from the Akamai content distribution network. Technically, it would be feasible to replicate the database at each visualizer site, and in future we will request permission from the NBA to replicate their data for this purpose.

The data returned from a query is a JSON object containing all the shot detail for a requested player. In Listing 2, an example of the shot detail available and what information is contained for a shot. Each shot object has a collection of properties, only three are used for the shot visualizer: x, y, and whether the shot was made or missed. The x, y co-ordinate pairs are fed to the visualization chart.
along with the made or missed flag and each is rendered on the shot chart as a scatter plot.

C. Experiments and Results

In this application, the messages sent, as shown in Listing 3, are extremely small. The action of clicking a list box item sends the name of the box clicked and the index of the item to all collaborators. Each of them separately sends a query to the NBA statistics API for their own data and renders it appropriately. Since the NBA has cloud hosted their statistics using Akamai, the localization is handled for us (15 measured GENI sites gave an average speed of 0.083s for request, processing on the server side and download of the data), so for this application, there is only a thin client for distribution to the nearest site to the user, without the large set of data. The visualization in question, using a JavaScript library called Data-Driven Documents (d3) [9], is able to render 600 points in 9 ms, or 67 points/ms, so with a remaining 70ms before hitting the desired threshold, we could conceivably render 4690 shots, more than enough to handle a season for any player on record. At scale, of course, to minimize impact to the NBA statistics servers, the data set could be created and manipulated in the same was as the Pollution Visualizer in Section 5.

Listing 2. Example Shot Data

```javascript
{
    "Shot Chart Detail": "0021500013",
    21,
    1495,
    "Tim Duncan",
    1610612759,
    "San Antonio Spurs",
    1,
    9,
    59,
    "Made Shot",
    "Hook Shot",
    "2PT Field Goal",
    "Restricted Area",
    "Center(C)"
}
```

Listing 3. Shot Chart Message

```javascript
{"action":"setList",
"data":
{"listname":"List2",
 idx":4",
"sender":"client-session:...",
"target":"client-session:...",
"messageId":"client-msg:...",
"messageIndex":4}
```

V. CONCLUSIONS AND FUTURE WORK

The LiveTalk System is the first distributed Platform-as-a-Service Cloud, running on the GENI Experiment Engine. It can be extended wherever a Docker VM can be instantiated. It incorporates a pluggable server, integrated client-side Smalltalk-like development system, an integrated messaging framework, database, and server-side file storage. Its unique strength is collaborative visualization applications across the wide area, or, as one observer described it, "Google Docs for Visualization". LiveTalk is promising but much remains to be done. We have deliberately excluded consistency from the current messaging framework, believing it to be an additional service that some applications will want and others not; it is not free. We intend to explore strategies from continuous interactive media[27] or distributed virtual worlds[14].

We further intend to offer an automated distribution service independent of the existing GEE Ansible-based distribution services, and integrate with a wide-area file system when one becomes available.

All of the software used in this paper is open-source, freely distributable, and stored in public repositories on github.com. The Web front ends for the various applications are available on a number of Lively Web servers; direct references can be found by contacting the authors.

ACKNOWLEDGEMENTS

We are grateful to our colleagues at the University of Victoria, US Ignite and the Hasso-Plattner Institute for support, many conversations, and much help. Dr. David Lary of the University of Texas, Dallas, provided the pollution data set and worked with us on that application. Andy Bavier, Niky Riga, Aki Nakao, Brecht Vermuelen, Max Ott, Hadi Bannazadeh, and Andi Bergen were very helpful with deployments of the early visualizer stages. This work was partially supported by the GENI Project Office and by MITACS.

REFERENCES


IntegraTag: a Framework for High-Fidelity Web Client Measurement

Charles Thomas∗
cthomas@comscore.com

Jeff Kline∗
jkline@comscore.com

Paul Barford†
pbarford@comscore.com

∗comScore

†University of Wisconsin

Abstract—Collecting information from web clients without explicit input from users is important in a variety of contexts including content customization, experience personalization, accounting and online advertising. A standard approach for gathering web client telemetry is through deployment of Javascript instrumentation that is placed either directly on web pages or through third-party “tags” that are referenced in web pages. In this paper we present a design study of web client measurement methods. The objective of our work is to enhance understanding of best practices in web client measurement toward the goal of developing future tags that are reliable, robust and efficient. We begin by conducting a detailed examination of Javascript instrumentation collected from five well known third party services. Our analysis shows that these code-bases have diverse capabilities and return a broad range of client characteristics. Next, we describe a web client measurement framework and an implementation that we call IntegraTag, which enables us to examine details of performance, accuracy and reliability through live deployments. We use IntegraTag to conduct case studies of tag behavior on a single website which resulted in over 500K page-loads, and on a publisher network which resulted in over 150M page-loads. We establish a lower-bound on the tag’s reporting fidelity using a Bernoulli trial. We report on the wide range of client characteristics returned by IntegraTag, as well as its performance and robustness.

I. INTRODUCTION

Web clients are defined broadly by a variety of features including: (i) the device type (e.g., desktop, smartphone, tablet, etc.), (ii) the software configuration running on the device (e.g., operating system, browser, plugins, etc.), (iii) the user of the device (to the extent where it can be inferred), and (iv) the environment of use (e.g., location, time of day, etc.).

Accurate identification and reporting of web client characteristics is important in a variety of contexts. Web publishers can tune content and user experience based on this information. For example, the manner in which a page renders can depend on device type, the physical orientation of the device and browser configuration. Similarly, marketers and advertisers use a variety of web client characteristics to aid in the selection of ads that are delivered to clients. Web tracking and accounting services such as Google Analytics [1] gather and report web client characteristics to aid publishers to understand their audience. Finally, technically savvy fraudsters aim to profit by deceiving reporting mechanisms [25]. This impacts both marketers and publishers. Accurate reporting is critical for fraud detection applications.

A standard approach for automated collection of web client data (i.e., telemetry) is through Javascript instrumentation, which is generally referred to as a “tag”. Javascript tags employ several common techniques (e.g., attaching 1×1 tracking pixels to the document, or asynchronous AJAX-style server calls) to report their measurements to the appropriate parties. When the instrumentation is deployed directly by a publisher it is referred to as a first-party deployment. Tags that are coupled with advertisements or otherwise embedded in a web page by someone other than the page’s owner are called third-party deployments. Web client telemetry is frequently collected by third parties such as advertisers and data aggregators like Oracle/BlueKai [4] who work in partnership with publishers.

The primary challenges in accurate collection of web client telemetry stem from the diversity of devices, browsers and security policies that are in current use. Device and browser combination often impact how Javascript executes, what methods one may call and the values that are returned. This complicates development and debugging of broadly-compatible instrumentation. Further, placement-specific security restrictions, such as same-origin policies or HTML5 sandboxing limit the actions that a Javascript tag may successfully execute. Finally, JavaScript tags rarely allow a user to provide feedback to the tag developer. When a Javascript tag fails to properly execute, both user and developer may be unaware of the failure. This highlights the need to integrate a robust “tag health” monitoring system into the tag’s functionality.

In this paper, we present a design study of tag-based methods for collecting web client data. The goal of our work is to elucidate issues that are central to development of tags that are flexible, robust and efficient. Our focus is on Javascript-based, third-party tags but our findings are applicable to first-party and non-Javascript instrumentation (e.g., some tags are deployed in Flash).

We begin by presenting an “insiders view” of Javascript instrumentation that is delivered by five different widely deployed third-party tags. Our objective is to assess current best practices in web client measurement. Although gathering the code-bases is trivial (Javascript is transmitted in plain text), understanding the code’s function and purpose can be a time-intensive exercise. The code is often deliberately obfuscated to thwart reverse-engineering. Due to the rise in blacklist-based ad-blocking, there is also incentive for some deployments to host their code on little-known domain names. JavaScript
code may also be mangled by minimization tools. The same code-base can execute along different paths depending on the deployment scenario and the particular state of the client that executes it. Through careful manual analysis, our results show that third-party tags are designed to return a wide variety of web client telemetry on aspects such as browser configuration, screen configuration, web page content, user actions and environmental variables to name a few.

We complement our static analysis of web tags with experiments in live deployments that are aimed at understanding basic issues of tag behavior and client diversity. To conduct these experiments we developed IntegraTag. The high-level design objective of IntegraTag is to assess and diagnose tag compatibility, reliability and performance as well as have the ability to return information about a wide range of web client features in live deployments. We implemented IntegraTag in Javascript using a design that enables features to be flexibly enabled for different diagnostics and performance tests.

We conducted a case study using IntegraTag in two live deployments. The first deployment ran the tag alongside online advertisements that are delivered to a specific live web site over the period of a week. During that period, IntegraTag was activated over half a million times by between 5,000 and 20,000 unique users per day (as identified by cookies placed in accordance with do not track). The second deployment ran on a broader selection of online ads that appeared on 152 different websites resulting in over 150M activations of the tag over a period of four days.

We configured IntegraTag to request a ping pixel on 1% of page loads. For greater reliability, the ping request was initiated prior to executing more complex tag features. Additionally, the pixel request only reported information that can be gathered without causing fatal errors. This included a placement identifier (e.g. domain name or a URL), a unique user identifier, a session identifier and a cache-breaker (a string of random text appended to a URL to prevent a browser from caching the URL). Since the ping was designed to be loaded first and have minimal functionality, we believed ping volume would be at least 1% of the total volume measured by the more complex environment-probing code. Surprisingly, the number of observed pings was significantly below the number that theory predicted. The discrepancy sets a lower bound on the fidelity of our measurements.

The second aspect of our case study considered the diverse characteristics of web clients. Specifically, we report on the results of data collected by IntegraTag configured to collect 47 different client characteristics. Our results highlight the diversity of client characteristics that are likely be of interest to third party tagging entities. We provide examples of specific characteristics that were gathered to highlight the diversity of client platforms. We examine the impact of IntegraTag on page load times. We find that its 32K code-base typically completes its browser probing activities in under 50ms. This illustrates that rich telemetry can be acquired and reported in a manner that does not interfere with overall user experience, which is a key concern for publishers and by extension third party measurement entities.

Finally, an important consideration in web client instrumentation is user privacy, which has been discussed in the popular press (e.g., [23]) and research literature (e.g., [12], [17], [15]). We do not address the issue of privacy specifically in this paper. It goes without saying that all major entities that deploy tags are concerned with legal compliance and user perception. By providing the insiders view and reporting on the details of methodology for web client instrumentation, we believe that our work helps to elucidate that conversation about privacy and could lead to new methods that strike a more widely accepted balance between privacy and functionality. We also believe that our work provides a roadmap for further investigation of these important issues.

The remainder of this paper is organized as follows. In Section II we provide an overview of tags. In Section III, we report on results of a study of a selection of third-party instrumentation that is broadly deployed on web pages. In Section IV, we describe the design and implementation of IntegraTag. The results of our deployment of IntegraTag on live web pages are reported in Section V. We discuss prior studies that inform our work in Section VI. We summarize, conclude and discuss future work in Section VII.

II. WEB TAG BASICS

We begin with a general discussion about practical aspects of instrumenting web pages with JavaScript tags. While the underlying technologies are well-known and the individual techniques straightforward, challenges arise from non-standard browser behaviors, deployment errors, unexpected user behaviors and bugs in the instrumentation itself. We limit our attention to JavaScript-based tags with brief comments on other techniques towards the end.

Information reported by JavaScript tags is currently the major tool by which online media companies (publishers, brands, agencies) measure their audiences. Since this code regularly executes alongside content that has significant monetary value, tags are typically designed in accordance with the principle that they must reliably execute without interfering with a user’s overall experience. A poor implementation risks shortening mobile device battery life, interfering with a publisher’s content or consuming a large volume of web traffic.

A. Tag Deployment

Tags are deployed when a web publisher or a brand that is running an ad campaign desires to measure some feature of their traffic that they are unable to derive from their own logs. They often address this by partnering with a third-party telemetry service. This service generates a customized block of code or a URL that points to a pixel/image or an HTTP 204 (No Content) resource hosted on one of its servers or in a CDN. Both types of requests are commonly called "pixel requests". The publisher adds this block of code to all pages that it serves. Ideally, each web browser request issued to the publisher also results in a subsequent request to the telemetry service.
Tag deployments often require careful placement within the page HTML structure. Some tags are designed to be loaded as soon as possible, others must execute as late as possible. Still other tags need to be embedded as either a sibling or child of particular DOM element to function correctly. Incorrect deployment may cause telemetry errors or complete failure in tag execution. Usually a person is required to do this, often in consultation with the telemetry company’s support staff. The scope and complexity of tag deployments on large websites has led to development of tag management systems such as Google Tag Manager 1.

Coarse statistics about a publisher’s server traffic such as hourly or daily volume are sometimes made available from the publisher’s logs. This type of information is often sufficient to identify deployment errors. But a more detailed study of a publisher’s logs, however, is rare. This is due in part to the lack of a well-defined key to perform a JOIN across different data sets. Even if all parties are setting cookies and the web client is storing them, cross-domain communication is typically prohibited. As a result, cookie syncing is usually not possible unless a dedicated process has been undertaken. User privacy is also a major concern. This is especially relevant where account logins are required or the site concerns personal, financial or medical information.

B. Data Types

The information that is transmitted in JavaScript tag-based telemetry is diverse. When third party services are used, telemetry includes an account identifier. For ad campaigns, the characteristics of an audience’s response to individual ad creatives (the image or video that a user sees as the advertisement) or placements is often of interest. As a result, ad creative identifiers are often included in ad campaign telemetry.

Sometimes individual impressions and page views are associated with a unique identifier. This is useful, if say a publisher wishes to measure dwell time on a page. This could, in principle, be measured with a monolithic pixel request that fires after a set time interval or when the page unloads. But for this approach to work, one requires consistent behavior associated with several unrelated page events including page unload, window close, tab close, process quit and page reload.

An alternative design relies on multiple pixel requests. This requires establishing the notion of an atomic event, e.g., a page view, a widget load or an impression and associating this event a key. Subsequent processing then must perform a JOIN on the several streams of data.

The context in which the tag executes can change how the tag behaves and strongly affects what it may measure. For example, a tag that executes on the top-level frame of the window can often collect a range of attributes and make observations about user actions. In contrast, a tag that lives within a sandboxed iframe will typically have access to relatively little information about user behavior. This has a direct impact on the kind of reporting or analysis that is possible on a deployment.

C. Data Reporting Methods

Most Javascript tag data is reported via query strings. To be more precise, telemetry is reported via HTTP GET requests to a 1x1 “tracking pixel”. The basic idea is that the Javascript from the tag runs and actively gathers as much information as necessary about the client browser and the viewing environment. This information is packaged (in some cases distilled, aggregated, and/or encoded) as a set of parameters and associated values. At this point, the Javascript requests an image from the a the tagging entity’s server. The image request usually returns a 1x1 pixel image which is attached to the document where the tag was placed. Telemetry is encoded and transmitted in the URL used to request the image. For example:

```
http://foo.com/p?parm1=val1&parm2=val2&parm3=val3...
```

where "p" is the name of the image file being requested. The server ignores the list of telemetry parameters (the string after the "?" character) and simply returns the image file to the browser. However, the entire request, including the parameters, is recorded in the server logs along with a date/timestamp, the User Agent string, the IP address of the client that made the request, and several other bits of information about the transaction. In this way, the telemetry is available for subsequent processing.

If the information in the telemetry can be conveyed in under about 2000 2 characters then this method is sufficient. But ad serving commonly involves many different entities working to serve a single impression. It is not uncommon to include the full referrer URL and additional query string parameters in requests, so it can happen that browsers fail to send the request or requests are truncated due to excessively long (but otherwise legitimate) query strings.

To work around any URL length upper bounds, an alternative technique is to include telemetry as part of an HTTP POST event. Indeed, two of the tags we examined (and the IntegraTag itself) return telemetry using this POST method. POST requests have several features which make them attractive for returning telemetry: (i) POST requests are “memoryless” in the sense that they are not cached and they do not remain in the browser history, (ii) POST requests always attempt to travel to the server where they will be logged, and (iii) POST requests have no restrictions on data length. Item (iii) is important since it’s trivial to generate telemetry strings which are so long they will potentially be truncated or rejected by either the browser or the server from which the pixel is being requested.

This technique was occasionally used to return raw HTML code of the page that hosted the tag. This raises privacy and security concerns. Additionally, if the message body is too

---

1https://www.google.com/analytics/tag-manager/

2The character limit for a URL in Microsoft browsers is 2083 characters. See https://support.microsoft.com/en-us/kb/208427.
large, it will interfere with user experience and also potentially incur large serving costs. Nevertheless, this method has proved invaluable to diagnose sanity check violations and to inform investigations involving fraud where other, less-invasive techniques are uninformative.

A modest commercial deployment may ingest daily volume of between 100 million and 1 billion kilobyte records and have a total daily volume of up to 1 terabyte of raw data. Much of the data that forms the basis of the present paper came from such an environment. Data processing typically is managed with MapReduce frameworks, SQL-based solutions and other tools. A common hardware configuration is either based at commercial data centers or with custom-built solutions.

D. Other Techniques

Mobile application telemetry is commonly implemented as a combination of compiled code and JavaScript. The compiled code may be Java or Objective C and the binaries often have permission to read device attributes that JavaScript alone cannot access. In particular, it is common to report a device’s IDFA (for iOS-based devices) or AAID (for Android devices) in commercial web traffic.

Other types of identifiers that attempt to associate each record with an individual browser or device have also been reported. Although these non-standard approaches such as browser fingerprinting, flash super-cookies and ISP cookie injection are known to have been employed, all have been met with resistance in part because users are either unable to opt-out of these programs or were enrolled without consent.

We close this section by referencing Table 1. This table will be discussed in greater length below but we introduce it here since it effectively highlights the diversity of tag functionality that exists among commercial tags.

III. CHARACTERISTICS OF CANONICAL WEB TAGS

We begin our study of instrumentation for user data telemetry by examining tags and associated JavaScript from several well-known third party collection entities. Our selection of the set of tags for this study is arbitrary and represents a small subset of all of the tracking entities that are active in the web today. We focused on this set of tags due to their mix of broad deployment, the set of attributes measured which relate to browser settings and our ability to convert the code into a tractable form. Based on our experience, we believe that the general characteristics of this set of tags is representative of what can be found more broadly in the web today.

Tag discovery is a straightforward task: simply crawl a set of pages where one thinks tags may be deployed, and then scan the HTML source for links to the tag source. Indeed, this process can be automated using web crawling methods such as those defined in [24], [5], though a manual crawl is also an effective discovery method of widely-deployed tags. To keep the JavaScript code as compact as possible, many of the tags that we found have run the final source code through a compressor (e.g., Minify JS [2]), which strips out whitespace and minimizes variable and function names. For example, a call to

\[ \text{getSmartStringHash}(	ext{mime_array}, \text{encoding_type}) \]

might become simply \( a(x,y) \) after compression. The result is not easily readable by a human. There are tools available which can streamline the process of analyzing a compressed tag (e.g., JSNice [13]). However, it is much less effort to start with a uncompressed version of the tag.

Once a tag is identified and downloaded, analysis of the Javascript instrumentation can be performed. While Javascript can be examined directly, instrumentation code from different entities uses a variety of conventions and paradigms to implement different functions. Some tags use obfuscation or even encryption \(^4\), which we hypothesize is done to reduce the human-readability of the code, obfuscate business partnerships or possibly to escape notice from automated analysis tools. Another challenge is that some companies deploy multiple tags, each tag designed for a different task. For example, a tag for presenting survey options might look similar to, and might deployed almost identically to, a tag from the same company that performs analytic tasks. Thus, identifying the general objectives of tagging entities can require examination of a variety of code bases.

For the purposes of this study, we narrowed our focus to the functions that gathered client metrics, and did not attempt to document every function that the third party tags performed. Many tags, for example, have varied (and in some cases, redundant) methods of communicating results back to their data-collection servers. Some tags, in addition to their analytic capabilities, also serve advertisements or other data content to the page on which they are deployed. We leave detailed examination of those functions for future work.

We selected five third-party tags for our study. We refer to the tags by the major function of the entity that has deployed them: Data Aggregator, Search Engine, News Service, Ratings Provider and Ad Metrics Service. In each case, the tags consist of a small piece of Javascript code that calls for the larger body of Javascript instrumentation that is the focus of our analysis. The instrumentation code bases that we examined varied between 15KB and 72KB.

Our analysis highlights common and distinct functionality in each of the tags that we examined. Space limitations preclude detailed descriptions of all functions that we found in the tags so we group functions into major categories as can be seen in Table 1. The table shows that most of the tags attempt to acquire information about the local context in which the code is executed. Examples include the user’s screen size, browser type, installed major plugins (e.g., Javascript and Flash), and the local timezone.

Our analysis also reveals varying degrees of analytical depth with regard to browser, screen, window, component, document, component, document.

\(^3\)The names of the companies listed in this table and the details of their code’s functionality are obfuscated based on advice from council.

\(^4\)The Ad Metrics Services tag applies rot13 encryption to one of its internal data structures. An unrelated video tag (not discussed here) employs the crypto.js tag to use strong encryption as part of video ad delivery.
and user action metrics. Some of these metrics are acquired simply. For example, once it is determined to exist, one can inspect the built in attribute `navigator.language` in most browsers to obtain a string describing the default language of the browser:

```javascript
if (typeof navigator !== "undefined" &&
    typeof navigator.language !== "undefined") {
    value = navigator.language;
}
```

Some metrics are much more challenging to acquire. For example, determining if and where the user has clicked on the page involves calls to determine the browser type, calls to install various browser-specific event listeners, a cascade of methods that each try to obtain the click information (each failing to the next method if unsuccessful), and calls to remove the event listeners when the analysis is finished.

Each third-party tag examined is tailored for the specific needs of the entity which created it. Thus, the presence or absence of any particular function does not imply any kind of deficiency in the instrumentation. For example, the Ad Metrics Service tag seems to have been designed to be paired with a specific ad. As a result much of its functionality involves monitoring interactions with the ad it is shepherding. The tag from the News Service, on the other hand, appears to have functions to facilitate eCommerce; it tracks and reports attributes like order number, tax, shipping, SKU, and promotion codes. The tag from the Data Aggregator, as another example, can do HTML parsing of the page on which the tag is deployed in order to gather contextual information and keywords from the page.

IV. DESIGNING A WEB TAG FRAMEWORK

The design goal for IntegraTag is an implementation that is flexible, robust and efficient. Flexible recognizes diverse measurement needs and refers to the ability to easily install new or modified data gathering capability in the tag. Robust recognizes the diversity of client configurations and refers to the ability to report any failures that may occur prior to completing execution. This is critical in assessing the veracity of returned telemetry. Efficiency refers to limiting the processing impact on client systems through careful functional implementation.

Since the tag functionality is nontrivial and the integrity of the telemetry is essential to the accuracy of analysis models that rely on the data, we sought to establish ground truth of traffic volume by firing a "ping" pixel with probability $p^5$. The ping is designed as the minimum unit of instrumentation so as to avoid platform compatibility issues. It does not call any methods which could potentially result in errors; it simply returns the placement identifier (e.g. the domain or URL) and some other minimal diagnostic information. Given the set of pings, it ought to be possible to estimate the overall expected volume of the data. This was a unique feature among the tags included in our survey.

The IntegraTag was designed to gather a wide range of user and environmental information. This includes some information which is typically regarded as redundant or irrelevant to serving ads. As one example, a web client’s underlying OS and platform can often be derived from the User Agent. We measured this attribute as reported by Javascript to identify web clients who modified the default User Agent string. A subset of the information returned is highlighted in Figure 1.

IV. DESIGNING A WEB TAG FRAMEWORK

The design goal for IntegraTag is an implementation that is flexible, robust and efficient. Flexible recognizes diverse measurement needs and refers to the ability to easily install new or modified data gathering capability in the tag. Robust recognizes the diversity of client configurations and refers to the ability to report any failures that may occur prior to completing execution. This is critical in assessing the veracity of returned telemetry. Efficiency refers to limiting the processing impact on client systems through careful functional implementation.

Since the tag functionality is nontrivial and the integrity of the telemetry is essential to the accuracy of analysis models that rely on the data, we sought to establish ground truth of traffic volume by firing a "ping" pixel with probability $p^5$. The ping is designed as the minimum unit of instrumentation so as to avoid platform compatibility issues. It does not call any methods which could potentially result in errors; it simply returns the placement identifier (e.g. the domain or URL) and some other minimal diagnostic information. Given the set of pings, it ought to be possible to estimate the overall expected volume of the data. This was a unique feature among the tags included in our survey.

The IntegraTag was designed to gather a wide range of user and environmental information. This includes some information which is typically regarded as redundant or irrelevant to serving ads. As one example, a web client’s underlying OS and platform can often be derived from the User Agent. We measured this attribute as reported by Javascript to identify web clients who modified the default User Agent string. A subset of the information returned is highlighted in Figure 1.

IV. DESIGNING A WEB TAG FRAMEWORK

The design goal for IntegraTag is an implementation that is flexible, robust and efficient. Flexible recognizes diverse measurement needs and refers to the ability to easily install new or modified data gathering capability in the tag. Robust recognizes the diversity of client configurations and refers to the ability to report any failures that may occur prior to completing execution. This is critical in assessing the veracity of returned telemetry. Efficiency refers to limiting the processing impact on client systems through careful functional implementation.

Since the tag functionality is nontrivial and the integrity of the telemetry is essential to the accuracy of analysis models that rely on the data, we sought to establish ground truth of traffic volume by firing a "ping" pixel with probability $p^5$. The ping is designed as the minimum unit of instrumentation so as to avoid platform compatibility issues. It does not call any methods which could potentially result in errors; it simply returns the placement identifier (e.g. the domain or URL) and some other minimal diagnostic information. Given the set of pings, it ought to be possible to estimate the overall expected volume of the data. This was a unique feature among the tags included in our survey.

The IntegraTag was designed to gather a wide range of user and environmental information. This includes some information which is typically regarded as redundant or irrelevant to serving ads. As one example, a web client’s underlying OS and platform can often be derived from the User Agent. We measured this attribute as reported by Javascript to identify web clients who modified the default User Agent string. A subset of the information returned is highlighted in Figure 1.

IV. DESIGNING A WEB TAG FRAMEWORK

The design goal for IntegraTag is an implementation that is flexible, robust and efficient. Flexible recognizes diverse measurement needs and refers to the ability to easily install new or modified data gathering capability in the tag. Robust recognizes the diversity of client configurations and refers to the ability to report any failures that may occur prior to completing execution. This is critical in assessing the veracity of returned telemetry. Efficiency refers to limiting the processing impact on client systems through careful functional implementation.

Since the tag functionality is nontrivial and the integrity of the telemetry is essential to the accuracy of analysis models that rely on the data, we sought to establish ground truth of traffic volume by firing a "ping" pixel with probability $p^5$. The ping is designed as the minimum unit of instrumentation so as to avoid platform compatibility issues. It does not call any methods which could potentially result in errors; it simply returns the placement identifier (e.g. the domain or URL) and some other minimal diagnostic information. Given the set of pings, it ought to be possible to estimate the overall expected volume of the data. This was a unique feature among the tags included in our survey.

The IntegraTag was designed to gather a wide range of user and environmental information. This includes some information which is typically regarded as redundant or irrelevant to serving ads. As one example, a web client’s underlying OS and platform can often be derived from the User Agent. We measured this attribute as reported by Javascript to identify web clients who modified the default User Agent string. A subset of the information returned is highlighted in Figure 1.

IV. DESIGNING A WEB TAG FRAMEWORK

The design goal for IntegraTag is an implementation that is flexible, robust and efficient. Flexible recognizes diverse measurement needs and refers to the ability to easily install new or modified data gathering capability in the tag. Robust recogni...
different device configurations. Complete failure of the tag can result if the code tries to carelessly call functions which are not supported by a specific browser version. The IntegraTag is built to be robust to browser compatibility issues: when possible, we probe to ascertain the existence of environmental parameters prior to querying for that information. Each probe is isolated to avoid affecting subsequent probes.

The tag also has the ability to report back to our servers if it was unable to complete one of its functions. If an error occurs at any of forty-seven separate checkpoints during code operation, an error report is generated. If a nonfatal error is encountered, the code will continue to run. It is possible that a single ad impression could generate many error reports. Using the error reports, we have been able to refine specific functions that were shown to be problematic, and reduce the number of errors that are generated by tag operations to less than 1% of impressions. Going forward we can monitor these errors and continue to enhance IntegraTag to more successfully examine the web ecosystem as it changes over time.

We typically reported at least two types of pixel each time the tag executed. The reporting mechanism is compartmentalized so that different types of data are reported independent of each other. Environment parameters (e.g. user settings, browser attributes, installed plugins, etc) are gathered at tag launch and are reported immediately via a (environment) pixel. User actions (e.g. mouse and keyboard activity, page dwell time, etc) are gathered over a period of time and reported after some delay via a (user) pixel. Failure of the environment tag does not halt firing of the user pixel. Likewise, ping and error telemetry are isolated from all other types of reporting and fired as soon as possible.

IntegraTag is implemented in Javascript. It is written in a highly modular fashion that enables sections of code to be easily enabled/disabled depending on the intended use case.

V. CASE STUDIES FROM INTEGRA TAG DEPLOYMENT

We deployed the IntegraTag on ad campaigns that ran on over 150 different websites and collected data during two tests lasting four days and a week, respectively. During each test we recorded impressions, errors and pings along with associated client data. Anomalies in ping and error data triggered several investigations which are summarized below. We start with some illustrative examples to highlight the richness and diversity of client data that was returned by IntegraTag.

The client data is unusually diverse in two ways. First, as already noted, the tag collects a wide variety of browser attributes and environmental settings. Second, for many attributes, the tag does not attempt to normalize or modify the browser response. Instead, the tag simply reports the literal responses of the browser. The result of this is a sometimes surprising range of values. Reporting literal values incurred little extra cost in transmission, storage or processing yet afforded great flexibility for exploratory projects.

- **Table I**

<table>
<thead>
<tr>
<th>Day</th>
<th>Users</th>
<th>MIME Confgs.</th>
<th>Browsers</th>
<th>Screen Confgs.</th>
<th>Time Format</th>
<th>Platforms</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/30/2014</td>
<td>20,646</td>
<td>4,695</td>
<td>3,577</td>
<td>1,652</td>
<td>142</td>
<td>22</td>
<td>39</td>
</tr>
<tr>
<td>5/1/2014</td>
<td>12,880</td>
<td>2,375</td>
<td>2,343</td>
<td>1,145</td>
<td>103</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>5/2/2014</td>
<td>14,879</td>
<td>2,788</td>
<td>2,615</td>
<td>1,343</td>
<td>103</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>5/3/2014</td>
<td>6,312</td>
<td>1,191</td>
<td>1,309</td>
<td>731</td>
<td>88</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>5/4/2014</td>
<td>5,958</td>
<td>1,102</td>
<td>1,266</td>
<td>715</td>
<td>77</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>5/5/2014</td>
<td>18,392</td>
<td>2,074</td>
<td>2,262</td>
<td>1,154</td>
<td>76</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>5/6/2014</td>
<td>20,567</td>
<td>2,303</td>
<td>2,484</td>
<td>1,237</td>
<td>86</td>
<td>19</td>
<td>19</td>
</tr>
</tbody>
</table>

- **Table II**

<table>
<thead>
<tr>
<th>Top 10 Language Settings</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-US</td>
<td>317,112</td>
</tr>
<tr>
<td>en-US</td>
<td>138,588</td>
</tr>
<tr>
<td>en</td>
<td>91,143</td>
</tr>
<tr>
<td>en</td>
<td>5,679</td>
</tr>
<tr>
<td>en-GB</td>
<td>379</td>
</tr>
<tr>
<td>en-US</td>
<td>125</td>
</tr>
<tr>
<td>es-es</td>
<td>88</td>
</tr>
<tr>
<td>en-gb</td>
<td>43</td>
</tr>
<tr>
<td>en-CA</td>
<td>43</td>
</tr>
<tr>
<td>es-419</td>
<td>41</td>
</tr>
</tbody>
</table>

Table I shows some of the breadth of attributes probed and the wide range of values observed across the seven-day testing period. Even at the level of detail in the table, we are able to derive some insight about the population characteristics: the user population on Wednesday 4/30 is nearly the same as the user population on Tuesday 5/6. But by all other measures, the Wednesday population was significantly more diverse. Since the deployed code base and the placement on the site were held fixed during this period, this feature’s cause is external to the tag deployment scenario. Based on experience, features similar to this have been traced to several different causes:

1) A special interest story attracted a small but unusually diverse set of users
2) The web site owner changed its traffic generation partners during this period
3) A browser update occurred during this time period that had both wide reach and altered our tag’s functionality

A full investigation into the underlying cause is an expensive endeavor and typically only occurs in cases where fraud, a bug in the code-base or a deployment error is a concern.
Table II displays the most frequently observed results of the browser attribute, `navigator.language`. The table shows seven different ways browsers reported a user preference for English.

In the test deployment, the ping pixel was set to fire at a rate of 1%. Thus, we expected to model the pings as a Bernoulli process with success rate $p = 0.01$. Since the ping was designed to fire more reliably than the environment pixel (which we register as an `impression`), we expected the number of observed pings to be larger than $p$ times the number of impressions. Table III shows that the observed ping volume is consistently less than 1%. A simple calculation suggests that the observed discrepancy is either an extraordinarily rare event or, more likely in our view, it is an artifact of our measurement apparatus. This simple model establishes a lower-bound on our tag’s fidelity.

We have localized the cause of some of these issues to a small number of sites, some of which generate significantly more impressions than pings (e.g., simply by running the tag code repeatedly). Previous experiences with similar site-specific anomalies identified server configuration settings as the cause. But the cause of this ping anomaly is unknown. This result highlights the unexpected and complex behaviors that can occur in the web/advertising eco-system.

The vast majority (over 99%) of error pixels that were generated during our tests (less than 1% of total impressions) were either site-specific or browser-specific. A small number of sites were responsible for the majority of site-specific errors. Three major causes were identified, each due to an action attempted by the tag. The actions are: 1) modify the local document, usually by appending an invisible tracking pixel to the DOM tree, 2) violate a security restriction such as a cross-domain request from within an iframe or 3) violation of a self-imposed limit such as dynamically creating a string whose length exceeds a fixed number of characters. Since multiple error messages may be dispatched on a single page-load, we receive a large number of these notifications.

Browser-specific errors were typically straightforward to identify. For example, users with older versions of Internet Explorer can generate errors because their browsers don’t support query functions that are available in newer browsers.

Finally, we examined the execution time of IntegraTag. Publishers and advertisers are sensitive about performance impact of tags since poor performance can degrade user experience and potentially limit the telemetry returned. For instances where IntegraTag ran to completion (i.e., all expected telemetry was returned), the average response time was less than 50ms.

### VI. Related Work

Gathering web client telemetry through various means has a long history that can be traced back to studies in the mid 1990’s. One of the first examples was work by Cunha et al., which deployed instrumentation in Mosaic browsers to collect client browsing data [7].

One of the most common forms of web client instrumentation are tools designed to be installed directly on web browsers. These are typically used to monitor a variety of client and network characteristics and often employed to help debug performance problems. Commercial examples of such tools include Firebug [19], neatest [11] and Compuware AJAX Edition [6], which are all Javascript tools focused on assessing page performance characteristics (e.g., latency and throughput). Similarly, Dhawan et al. developed the Fathom tool as a Javascript-based, Firefox extension to enable performance measurements from remote clients [8]. Netalyzr is Java app that runs in a client browser and provides a detailed analysis of network connectivity [14]. Another form of instrumentation are systems designed as independent platforms for web client measurement. An excellent example of this is the Webpagetest platform, which provides detailed client performance reports from a selection of browser types and dedicated hosts distributed around the world [26]. While these tools and systems all inform our work, to the best of our knowledge, ours is the first paper to analyze details of third party, tag-based web client instrumentation, which is perhaps the most widely used form of web client measurement in the Internet today.

Web tags have been examined in prior studies that focus on user privacy. The term “web bug” has been used in association with 1x1 tracking pixels, and the term "tracker" is commonly associated with third party code that collects user information. In [16], Martin et al. consider the prevalence of web bugs on popular web sites and discuss implications for privacy. Two relatively recent surveys of issues surrounding user tracking and privacy are provided in [17], [15]. Both of these studies highlight the tussle between privacy and online advertising that benefits greatly from user information. Concerns over user tracking have resulted in development of the Ghostery plugin for the Firefox browser, which enables users to see who is tracking them [20]. Many other such mechanisms are available. A taxonomy of different trackers including first and third party as well as social media trackers is described in [24], [10],[18]. By simulating users (based on AOL query search logs), the authors are able to project tracking prevalence and they propose methods to protect user privacy. Similarly, the authors in [12] examine privacy issues related to online advertising activities, and develop a client plugin called Privad [21] that provides targeted ads to users in a privacy preserving fashion. Several recent studies have also considered alternative

---

**Table III**

<table>
<thead>
<tr>
<th>Day</th>
<th>Impressions</th>
<th>Ping count</th>
<th>Ping/Imp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/2/2014</td>
<td>43,896,149</td>
<td>433,905</td>
<td>0.988%</td>
</tr>
<tr>
<td>4/3/2014</td>
<td>35,095,942</td>
<td>344,691</td>
<td>0.982%</td>
</tr>
<tr>
<td>4/4/2014</td>
<td>37,806,814</td>
<td>371,028</td>
<td>0.981%</td>
</tr>
<tr>
<td>4/5/2014</td>
<td>46,884,618</td>
<td>463,714</td>
<td>0.995%</td>
</tr>
</tbody>
</table>

\[\sqrt{np(1-p)} \approx 629\] Our observed discrepancy is about 10 times this figure – and in the wrong direction.

---

Under the Bernoulli model, a set of $n = 40M$ impressions that had pings fire at a rate of $p = 1\%$ results in an expected 400K pings with a standard deviation of $\sqrt{np(1-p)} \approx 629$. Our observed discrepancy is about 10 times this figure – and in the wrong direction.
methods of user tracking. For example, studies by Eckersley, and Acar et al. report on the utility of browser fingerprinting, canvas fingerprinting and evercookies as a means for user tracking [9], [3]. These studies have important implications for privacy but do not address the general issues of client measurement that are the focus of our work. Finally, the work by Nikiforakis et al. examines the trust relationships between web sites and remote Javascript library providers (including Javascript that is downloaded by tags) [22]. That study complements our by highlighting vulnerabilities that can emerge due to reliance on remote libraries.

VII. SUMMARY AND CONCLUSIONS

In this paper we present a study of instrumentation for collecting web client data. Our particular perspective is from third parties that use Javascript instrumentation, which is delivered primarily via embedded tags on web pages. We begin by presenting results from an examination of five instrumentation code-bases from large third party data collection entities. Our examination is focused on understanding the objectives for and methods of user information tracking. Our analysis reveals a wide variety of functions and features in the different code-bases. We find that the instrumentation is focused on characteristics such as browser configuration, screen configuration, web page content, user actions and environmental variables. We also find that each code-base has unique capabilities such as HTML parsing, online commerce tracking, and the ability to be paired with a specific advertisement for a more in-depth analysis of the interactions with that ad.

To understand dynamics of tag execution and the diversity of client characteristics we developed IntegraTag. IntegraTag is a Javascript framework designed to run efficiently, to enable flexible transmission of telemetry and to enable detailed diagnostics of tag behavior. We use IntegraTag to conduct a case study of tag behavior through live deployments. We report on the diversity of characteristics observed in the client population and on unexpected behaviors the highlight the difficulties in web client data gathering. The discrepancy between observation and theory of a Bernoulli trial establishes a lower-bound on IntegraTag’s reporting fidelity.

In on-going work we continue refine the capability of IntegraTag to return telemetry in a reliable and efficient fashion. We also plan to pursue development of methods for accomplishing the goals of web client data collection in a privacy preserving fashion.

ACKNOWLEDGMENTS

The authors would like to thank Kevin Springborn for his input on early versions of this work.

REFERENCES

CLUE: Clustering for Mining Web URLs
Andrea Morichetta, Enrico Bocchi, Hassan Metwalley, Marco Mellia
Politecnico di Torino
name.surname@polito.it

Abstract—The Internet has witnessed the proliferation of applications and services that rely on HTTP as application protocol. Users play games, read emails, watch videos, chat and access web pages using their PC, which in turn downloads tens or hundreds of URLs to fetch all the objects needed to display the requested content. As result, billions of URLs are observed in the network. When monitoring the traffic, thus, it is becoming more and more important to have methodologies and tools that allow one to dig into this data and extract useful information.

In this paper, we present CLUE, Clustering for URL Exploration, a methodology that leverages clustering algorithms, i.e., unsupervised techniques developed in the data mining field to extract knowledge from passive observation of URLs carried by the network. This is a challenging problem given the unstructured format of URLs, which, being strings, call for specialized approaches. Inspired by text-mining algorithms, we introduce the concept of URL-distance and use it to compose clusters of URLs using the well-known DBSCAN algorithm.

Experiments on actual datasets show encouraging results. Well-separated and consistent clusters emerge and allow us to identify, e.g., malicious traffic, advertising services, and third-party tracking systems. In a nutshell, our clustering algorithm offers the means to get insights on the data carried by the network, with applications in the security or privacy protection fields.

I. INTRODUCTION

The web has become the most popular application of the modern Internet. Originally born to access hypertext, nowadays it is used to watch videos, play games, read emails, chat online, etc. Web pages have become much more complex, with dynamic elements offering personalized views, and are now rich of multimedia contents and web applications that run inside the browser. HTTP is the de-facto standard application-layer protocol [1], allowing the browser to retrieve the hundreds of objects composing a page with a simple request-response mechanism. Billions of objects are available on the web, each of them being identified by a Uniform Resource Locator (URL). Static URLs directly point to an object, e.g., portions of text or an image file like http://acme.com/index.html, but more and more frequently URLs encode queries that servers process to return a dynamic result. For instance, a Google search, a click on “like” buttons, or the images served by an advertisement platform are typical examples of dynamic URLs, e.g., http://acme.com/s/?key=like.

Given the amount of URLs that are retrieved to fulfill ordinary browsing activities, monitoring and understanding the dynamics of the network is not an easy task. Due to the volumes of today’s Internet and the complexity of its architecture, where resources are retrieved from remote servers, cloud data-centers or Content Delivery Networks (CDN), the work of network and security analysts requires advanced tools to effectively dig into such huge amount of raw data. Moreover, in many scenarios it is required to extract knowledge from it, e.g., for investigating an incident, or to extract signatures for Intrusion Detection Systems (IDS).

In this paper, we focus on the problem of automatically analyzing web traffic leveraging URLs. We design an unsupervised methodology that groups URLs in clusters according to a similarity metric. We call it CLUE, Clustering for URL Exploration. The goal is twofold. First, we reduce the number of items the analyst has to visualize and process, from hundred thousands of single URLs to few hundreds of clusters. Second, we target the identification of automatically generated URLs, e.g., URLs generated by advertisement platforms, polymorphic malware, or wiki-like systems.

Previous studies faced the analysis of URLs or entire web pages typically with a specific goal in mind, e.g., detecting duplicated pages, improving page rank, or pinpointing phishing websites — see Sec. II for a thorough discussion of related work. Our work differs from previous approaches as we aim at offering data exploration tools not tailored to a specific goal. We assume the system is fed by a set of URLs collected by passively observing HTTP requests produced by hosts in a live network. We specifically do not want to identify clients or keep track of their past navigation history, thus preserving users’ privacy. In addition, we want to avoid the overhead introduced by web crawling techniques.

Ingenuity is required to design such a system. Unsupervised machine learning approaches like clustering algorithms have gained popularity. Clustering [2], [3] is defined as the task of grouping samples according to their similarity. Close samples are placed in the same cluster, while samples belonging to two clusters are far apart. Similarity is classically measured as the distance between two samples in a metric space, where the triangular inequality holds. The definition of a distance is unfortunately not trivial for URLs, which are indeed textual strings. We solve this problem proposing the URL-distance, a modification of the Levenshtein distance, which specifically takes into account key URL characteristics, i.e., string length and character frequencies which are different than in regular text strings. Next, we use DBSCAN, a well-known density-based clustering algorithm. As result, URLs are grouped into well-separated and cohesive clusters.

We assess CLUE performance considering a dataset of URLs accessed by ordinary users through PCs, tablets and smartphones. Half of the users are infected by a well-known polymorphic malicious software called TidServ [4], whose traffic is identified by an IDS. We use this labeled dataset as
ground-truth to tune parameters so that clusters of malicious URLs emerge. We then run the clustering algorithm on a larger dataset where no infected hosts are considered to show the potential of the approach. Results are encouraging: CLUE is able to pinpoint clusters of URLs generated by video streaming or advertisement services, and by malware families or third party tracking systems. A simple manual investigation is sufficient for the analyst to tie clusters to their category and augment the understanding of some phenomena. This strengthens the potential of CLUE to support the mining of URLs and of web traffic with applications to security and privacy protection fields. In a nutshell CLUE offers the analyst the chance to mine web traffic presenting clues about traffic and services.

This paper is organized as follows. We first discuss related works in Sec. II. Methodology is presented in Sec. III, where the CLUE is detailed. The dataset used for the experiments is described in Sec. IV, while results are discussed in Sec. V. At last, Sec. VI summarizes findings and comments on further potential improvements.

II. RELATED WORK

Several papers in the literature aim at identifying similar web pages or URLs. Each work is targeting different problems or is tied to a specific application, with custom techniques being designed. The large majority of works look for structural features that help in distinguishing different classes of websites to consequently group or classify them. Such features can be referred (i) to the URL of a web page, here intended as sequence of characters; or (ii) to the payload of the page, consisting of its layout, formatting, and syntactical properties.

A group of previous works aims at clustering web pages directly using the text they contain. Such approach requires the complete retrieval of the page, and typically expensive text-processing algorithms. A notable example of clustering applied to web content is [5]. Authors propose a methodology to quantify the syntactic similarity between generic text files through the computation of resemblance and containment features. They apply such technique to 30 M documents retrieved from the web and run clustering algorithms on top. A similar and more recent approach is presented in [6], while [7] stresses the importance of algorithmic design to achieve high scalability of clustering algorithms.

In the context of web page clustering for specific applications, the authors of [8] apply clustering algorithms to disambiguate between people’s name on the Web. They use a set of features coming both from the page content and from the URL. They split the URL into multiple components (e.g., domain name, path, parameters) and extract properties that have to be recombined together, making the whole process a thorough but expensive technique.

Authors of [9] give more importance to URLs rather than page content in the process of clustering websites. They propose a technique based on Minimum Description Length [10], which is applied to URLs. Additional features derived from content and structural properties are used only at a later, more fine-grained, clustering stage. [11] and [12] present rule-mining techniques applied on URLs only. The former is aimed at detecting web pages duplicates, while the latter presents an automated tool to explicitly detect malicious connections to Command and Control (C&C) servers through URL patterns. Only the detection of C&C is targeted.

Cantina [13] targets the automatic identification of phishing websites by analyzing URLs with text mining approaches. Authors of [14] target the same problem leveraging the Levenshtein distance to detect spelling mistakes that lead to phishing sites, while [15] applies clustering algorithms to identify spam campaigns from URLs posted on Facebook walls. Considering page ranking, [16] proposes a URL-based methodology to automatically spot “qualified” links, i.e., those implying merit of the targeted page, and noisy ones, e.g., advertisement and promotional links, that do not confer authority to a page.

Finally, authors of [17] and [18] propose web pages classification techniques solely based on websites URLs. Despite the goal of “classifying” a web resource goes beyond the scope of our work, both papers bring readers’ attention to scalability issues and time requirements in case the actual content of a page has to be fetched, and to the feasibility of content analysis.
when the semantics of a web page lie into images or graphical works and thus textual analysis is not applicable.

All previous proposals leverage some particular features in the structure of URLs they target and devise specific solutions to reach the goal. Our goal is instead to employ general-purpose data mining approaches to investigate URL structures and group together those URLs that look similar. In this respect, we aim at helping the analyst by sensibly reducing the amount of elements to analyze. We offer her the chance to check few hundreds of consistent URL groups, instead of several thousands of single URLs. CLUE is an exploration tool to dig inside the web.

III. CLUE SYSTEM DESCRIPTION

We aim at designing a completely unsupervised methodology that can support the work of a network or security analyst in extracting knowledge from the URLs the network carries. In this section, we first provide a description of the tools needed to identify and extract URLs from the network traffic. As second, we highlight the need to summarize the distance between URLs in a numeric fashion, considering several distance measures and their behavior in our field of application. Lastly, we guide readers through DBSCAN, a density-based clustering algorithm. Fig. 1 shows the overview of the CLUE architecture. It depicts the three macro processing stages and the components belonging to each of them. A detailed description of each step is proposed in the following.

A. URL Extraction

An URL is a reference to a resource which embeds two essential pieces of information: (i) The mechanism to retrieve the resource, i.e., the network protocol, and (ii) the location of the resource, i.e., the hosting server and the path to obtain it. In the web scenario, URLs point to hypertext, i.e., pages with text, images, and multimedia content. Being HTTP the de-facto standard application protocol [1], URLs now are used as identifiers to retrieve any type of content, from simple self-contained pages to rich personalized websites full of resources served by third party platforms, e.g., advertisements, video, social network plugins, etc.

The network carries billions of URLs and the understanding of the content being served through them is a complex task. URL parsing is nowadays much more complicated than in the past: the same resource can be retrieved from multiple nodes, e.g., any server replica in a CDN. Moreover, additional parameters are commonly used to fetch a specific object when multiple options are available, e.g., google.com/logo.png?xy=640x480 or google.co.uk/logo.png?xy=1024x768 may refer to the same resource. At last, automatic URLs are commonly found in Content Management Systems such as WiKi pages, in polymorphic malware, or in advertisement platforms. How to find similarities and offer the network or security analyst a scalable means to understand how the network is used is thus a challenging problem.

In the path from raw network traffic to URL clusters, the first step performed by CLUE is the extraction of URLs as they are requested by users. This can be done using logs readily available from proxy or firewall systems, or by extracting URLs directly from packets. In this work, we assume the latter case: a passive network probe is located on a link where it processes packets in real time. The probe extracts URLs and dumps them in batches for later post-processing. To do so, network flows carrying HTTP traffic have to be detected. Deep packet inspection techniques are employed for such purpose, i.e., to identify strings that match the syntax of HTTP GET or POST requests. When a HTTP request is found, the URL there contained is logged in a plain text file. When a batch of URLs is formed, it is finally possible to step ahead towards the computation of the distance among URLs. This task does not have real-time constraints and can be scheduled when the data collection from the network is complete or on demand too.

B. URL Distance Evaluation

The concept of distance refers to a specific class of dissimilarity measures that aim at quantifying numerically the degree to which two points are far away [19]. A dissimilarity measure can be called distance if it meets three key properties that characterize a measure as metric: positivity, symmetry, and triangle inequality, respectively defined as

\[ d(x_1, x_2) \geq 0, \quad d(x_1, x_2) = d(x_2, x_1), \quad d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3) \quad \forall x_1, x_2, x_3. \]

The fulfillment of these properties is mandatory when dissimilarity measures are used on top of which data-mining techniques, clustering included, run. In our case, we look for a distance metric to compute the dissimilarity of strings. Distance measures suitable for application to textual strings take the name of “string metrics” or “string distance functions”. The adoption of such metrics is popular in the field of text-mining but also in all the problems where it is required to compare groups of elements for which one has no a-priori knowledge or understanding. Textual distance metrics therefore represent a convenient and viable way to compactly represent in numbers the dissimilarity among strings.

We focus on a particular class of distance metrics, the edit-distance based functions [20]. As the name suggests, the distance between two given strings $s_1$ and $s_2$ is intended as the minimum number of steps required to convert the string $s_1$ into $s_2$. Edit-distance functions have been used to target the analysis of free text where strings are well-formed words from a dictionary, with a defined grammatical syntax and with well-understood constraints.

The most popular technique is the Levenshtein distance [21] $d_{LVS}(s_1, s_2)$ that assigns a unitary cost for all editing operations, i.e., insert, remove, or replace one character. It computes an absolute distance between pairs of strings that is at most equal to the length of the longer string. This makes the Levenshtein distance inconvenient when comparing a short URL against a long one, as URL length possibly spans from few to hundreds of characters.
The Levenshtein distance $d_{LV S}(|s_1|, |s_2|)$ is defined as:

$$d_{LV S}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \left( \frac{d_{LV S}(i-1, j) + 1}{|s_1|}, \frac{d_{LV S}(i, j-1) + 1}{|s_2|}, \frac{d_{LV S}(i-1, j-1) + I(s_1 \neq s_2)}{|s_1| + |s_2| - 1} \right) & \text{otherwise}. \end{cases}$$

where $i$ and $j$ are respectively the lengths of $s_1$ and $s_2$, i.e., $|s_1|$ and $|s_2|$, respectively, $d_{LV S}(i, j)$ is the distance between the first $i$ characters of $s_1$ and the first $j$ characters of $s_2$, and $I$ is the indicator function, namely equal to 0 when $s_{1i} = s_{2j}$.

A different approach is taken by the Jaro distance. In this case, the distance function considers the number and the order of common characters between two strings. Let $m$ be the number of matching characters, and $t$ be half the number of transpositions. The Jaro distance $d_{JRO}(s_1, s_2)$ is defined as:

$$d_{JRO} = \begin{cases} 1 & \text{if } m = 0, \\ \frac{1}{3} \left[ \frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m - t}{m} \right] & \text{otherwise}. \end{cases}$$

Given the peculiarity of URLs, whose length may vary widely and which may include random substrings, we propose a custom modification of the Levenshtein distance, $d_{LV S2}$. Specifically, we count the total number of insertions and deletions, but we weight replacement by a factor of two. The rationale is that a replacement corresponds to one combined operation of deletion and insertion. We also explicitly consider the string length, and normalize the results in a $[0, 1]$ range:

$$d_{URL}(s_1, s_2) = 1 - \frac{|s_1| + |s_2| - d_{LV S2}(s_1, s_2)}{|s_1| + |s_2|}$$

This leads to a bounded distance metric, and specifically $d_{URL} = 0$ if $s_1 = s_2$, while $d_{URL} = 1$ if the two strings are completely different.

To give the intuition of the different results achievable, consider a simple example. Let $s_1$ be “google.com” and $s_2$ be “1goggle.com”. We now compute the numerical value provided by each of the considered distance functions. The Levenshtein distance $d_{LV S}(s_1, s_2) = 2$, accounting for one insertion (“1”) and one replacement operation (“o” → “g”). For $d_{JRO}$, the number of matches $m$ is 9 (g,o,g,l,e,.c.o,m), and the number of transpositions $t$ is 0. Thus $d_{JRO} = 0.094$. Finally, $d_{URL} = 0.143$ since we have one insertion, weighted 1, and one replacement, weighted 2.

We now run a simple experiment to raise awareness on the importance of choosing an adequate distance function. We consider all the URLs found in our dataset (see Sec. IV for details) that have been generated by TidServ, a polymorphic malware. We then compute the distance between any pair of URLs $(u_1, u_2)$ according to the different definitions of $d(u_1, u_2)$ reported above. Fig. 2 shows the Cumulative Distribution Function (CDF) of the measured distances for $d_{LV S}$, $d_{JRO}$, and $d_{URL}$, respectively.

Given our goal is to cluster elements that are “close” one to the other, we prefer to have distances concentrated in ranges. A pair of similar elements should exhibit a small distance, while a pair of different elements should exhibit a very large distance. $d_{LV S}$ shows three groups in its CDF, suggesting for potential clusters. However, $d_{LV S}$ support is not bounded in a given range (in our experiments, it spans in the $[0:250]$ range), since no normalization is entailed. This makes the comparison mostly driven by string lengths, i.e., any two short strings will be much more similar than any two long strings. $d_{JRO}$ instead results in a nearly-continuous shape, showing no clear steps that would help in separating close from far away pairs. $d_{URL}$ satisfies the intuition of having distance ranges, as it clearly shows three modes in the CDF. Moreover, its support is bounded in the $[0:1]$ range, normalizing the distance with respect to the length of the two considered strings.

In our methodology, we compute the distance by submitting the entire URL as a single string made by hostname and path. We reached this decision after performing some experiments, not reported here for brevity, where we tested several definitions of $d_{URL}$ by considering hostname and path separately, and then by composing a single metric via linear combinations. However, blending hostname and path distance values resulted not straight forward and did not lead to better results.
C. Clustering

Clustering algorithms are unsupervised machine learning approaches that aim at grouping together points according to similarity metrics [2], [3]. They offer exploratory means to analyze raw data, sensibly reducing the number of elements to be analyzed from hundred thousand individual points to few hundreds clusters. Elements that are grouped in the same cluster share common features and, as such, can be analyzed as a single entity. The homogeneity of items in a cluster allows the analyst to naturally extract knowledge about the elements themselves. Clustering algorithms are thus good candidates to process URL distances and group together those URLs that show a low distance value.

Among clustering algorithms, density-based approaches define clusters as the set of elements that form areas of higher density than the remainder of the data set. Such techniques have several notable and useful advantages: (i) they do not require any knowledge on the final number of clusters in advance (one of the major weaknesses of centroid based clustering approaches, as k-means); (ii) they can find arbitrarily shaped clusters; (iii) they include the notion of outliers, which are left unclustered as noise. We rely on DBSCAN [2], one of the most popular density-based algorithms.

To better illustrate how density-based clustering algorithms work, consider a set of points in a sample space to be clustered. Let \( d(x_1, x_2) \) be the distance between two points \( x_1 \) and \( x_2 \). Consider now the sphere of radius \( E \) centered in \( x_1 \). If at least \( \minPoints \) are within distance \( E \) from \( x_1 \), the point \( x_1 \) is classified as “core point”. Formally, a given point \( x_1 \) is a core point if at least \( \minPoints \) are within distance \( E \) from it. These points are defined as “directly reachable” from \( x_1 \). A generic point \( x_k \) is “reachable” from \( x_1 \) if there exists a path \( x_1, x_2, \ldots, x_k \) so that \( x_{k+1} \) is directly reachable from \( x_k \). Reachable points from \( x_1 \) form a cluster, i.e., a dense region. Points that are not reachable from \( x_1 \) are called “outliers”, and may either form a separate cluster if they belong to another dense region, or fall in the “noise” region if it is not the case. \( \minPoints \) and \( E \) are two tunable parameters that can be set by a domain expert if the data to be processed is well understood. \( \minPoints \) defines the minimum size of a cluster and has little impact on the final results. \( E \) instead is a key parameter. If set too small, it leads to a high number of small clusters and lots of unclustered points. If set too large, it leads to few clusters with lots of (heterogeneous) points. Sensitivity analysis is thus essential to properly choose \( E \). We better detail parameter choice impacts in Sec. V-B.

IV. SCENARIO AND DATASET

In this section we provide an overview of the technologies used to record network traffic and of the tools used to extract useful information. We consider a scenario in which a sniffer passively monitors the traffic generated by a group of hosts, e.g., hosts in a LAN network, or households connected to a Point-of-Presence (PoP) of an Internet Service Provider (ISP). The sniffer is capable of identifying HTTP requests, and log them to a file for later postprocessing.

In our case, we capture traffic at the PoP of an European ISP where approximately 20,000 customers are connected. Most of them are residential customers accessing the Internet via ADSL modems. We instrument the PoP with a passive probe to monitor the traffic generated by residential users. The probe runs Tstat [22], a passive monitoring tool that rebuilds each TCP flow, tracks it, and, when the connection is closed, logs a record reporting statistics in a simple textual format. When the application protocol is HTTP, Tstat extracts the URL and logs it in file. In case multiple HTTP transactions are present due to the usage of HTTP-persistent option, multiple records are logged. We let Tstat collect URLs for an entire day, generating more than 100GB of data.

We have also access to a commercial Intrusion Detection System (IDS) that we use to label URLs as possibly malicious. The IDS has at its disposal an internal database of rules modeling network threats. If some URL matches one (or more) of these rules, the IDS raises an alert and flags the URL with a Threat-ID, i.e., a numeric code identifying a specific threat. For our purposes, we enabled signatures for a specific malware called TidServ (see Sec. V for a description of the malware) that is known to use polymorphic strings in the URLs to evade detection techniques. We identified 14 hosts to be infected by the malware in our dataset.

In the following, we consider one dataset. Table I provides some statistics about characteristics in terms of volume, number of URLs, etc. Our dataset considers traffic generated by the 14 hosts infected by the TidServ malware, i.e., for which the IDS flagged at least one flow as malicious, and 20 additional hosts randomly selected from the population of users; none of the URLs of this second group of hosts are flagged by the IDS. In total, more than 411,000 URLs are present, 78,421 of which are unique. For the sake of completeness, Table I details also the statistics considering only the 14 infected hosts. Out of the 255,000 total URLs, 43,479 are unique, of which only 228 are flagged as TidServ.

V. RESULTS

In this section, we present experiments conducted to tune CLUE parameters. We provide evidence of its effectiveness, and show case studies and examples of the analysis it enables.

A. TidServ Overview

We firstly provide some highlights on the traffic produced by TidServ, a popular Trojan Horse also known as Alureon, TDSS or TDL [4]. After infecting an host and transforming it in a bot, this malware communicates with a Command-and-Control (C&C) server to receive commands. Communications

\(^1\)The C&C servers run on central computers that attackers use to update, instrument and control infected hosts.

<table>
<thead>
<tr>
<th>All hosts</th>
<th>TidServ infected hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP Flows</td>
<td>287,393</td>
</tr>
<tr>
<td>HTTP Volume</td>
<td>89.99 GB</td>
</tr>
<tr>
<td>Total URL</td>
<td>411,727</td>
</tr>
<tr>
<td>Unique URL</td>
<td>78,421</td>
</tr>
<tr>
<td>Unique TidServ URL</td>
<td>228</td>
</tr>
</tbody>
</table>
with C&C servers are typically established using HTTP so to evade firewalls. Originally, static URLs were used and security software could easily block the communications using, e.g., static rules and blacklisting. However, malware started to successfully evade such rules by using polymorphic approaches, e.g., randomly generating and rotating hostnames for C&C servers, or adding randomness in the URL path. This makes the compilation of static blacklists based on string matching a more difficult task and in turn less effective.

TidServ adopts this expedient by changing periodically the URLs to contact C&C servers. To give the reader the intuition of how random a TidServ URL can appear, Table II reports four examples of URLs that the IDS flagged. Hostnames and paths change, but some common parts (in bold) are still visible. In some cases the common pattern may be very long, but in others as few as 4 characters are found in common, suggesting different communication patterns may be present. Observing these patterns is easy if one is provided the correct set of URLs, but spotting them when mixed in the hundred thousands of URLs generated by a host makes the detection very challenging.

### B. Parameter Settings

We now run CLUE on the overall dataset. We fix \( \text{minPoints} \) equal to 4 to let CLUE consider clusters of at least 4 points. This choice is not critical since normally one is interested in observing clusters with much higher number of points. The impact and choice of \( E \) needs instead a preliminary study since it can radically change clustering results. Intuitively, a too small value of \( E \) leads to a large number of very small clusters and to lots of points falling in a non-dense region, i.e., to a large number of outliers. Conversely, a too large value of \( E \) leads to few gigantic clusters, where all points are core points, sometimes artificially connected together by a single path.

Fig. 3 reports results when considering \( E \in [0.175, 0.5] \). Recall \( d_{URL} \leq 1 \) by construction. Specifically, Fig. 3a and 3b report the total number of clusters and the percentage of outliers, respectively. As expected, both curves decrease as \( E \) increases. Since we are interested in obtaining an amount of clusters in the order of hundreds, these results suggest to use large values of \( E \) (between 0.4 and 0.45). Notice that for \( E > 0.45 \), a sudden drop in the number of outliers is observed due to the appearance of gigantic clusters.

We now focus on how TidServ labeled URLs are clustered. Intuitively, we would like to have them all clustered together, possibly in different clusters (cfr. Fig. 2), with none of them left in the noise region, and with only TidServ URLs being included in each cluster. We run CLUE for increasing values of \( E \), and we observe results by leveraging the IDS labels as ground truth. We report results in Fig. 4. We consider two performance metrics: (i) TidServ in Outliers – \( N_{out} \), i.e., the number of URLs labeled as malicious by the IDS that are left in the noise region, and with only TidServ URLs being included in each cluster. We run CLUE for increasing values of \( E \), and we observe results by leveraging the IDS labels as ground truth. We report results in Fig. 4. We consider two performance metrics: (i) TidServ in Outliers – \( N_{out} \), i.e., the number of URLs labeled as malicious by the IDS that are left unclustered; (ii) Total Clustered TidServ URL – \( N_{clue} \), i.e., the total number of URLs belonging to any cluster in which there is at least a TidServ URL. This includes both TidServ labeled URLs and eventual other URLs not labeled as malicious, but labeled as TidServ.

Fig. 4a reports \( N_{out} \) versus \( E \). If set too small, a large number of TidServ URLs results unclustered, since the randomness added in the URLs makes \( d_{URL} \) too large compared to \( E \). Conversely, for values of \( E \geq 0.375 \), all TidServ labeled URLs are assigned to a cluster. Look now at Fig. 4b, depicting...
Fig. 4: Number of TidServ elements left unclustered, and total number of URLs in TidServ clusters.

*C. TidServ Findings*

Table III reports details for each of the 7 TidServ clusters. It shows the total number of URLs, the number TidServ labeled ones, the number of unique hostnames, and the longest common substring found. Clusters are sorted by number TidServ labeled URLs. The table shows the polymorphic behavior of the malware. Several hostnames are used for the C&C server, which only 228 are labeled by the IDS. By manually checking this second set, they all present very strong similarities with the TidServ labeled URLs. Considering also the other clusters, overall CLUE associates to TidServ 357 unique URLs, of which only 228 are labeled by the IDS. By means of manual validation, we verified that all the non-labeled elements are very likely malicious communications related to TidServ. We hypothesize the IDS signatures do not cover them, which can be thus considered false negatives. To give the reader more insights, Table IV reports some of the URLs of the last cluster TidServ identified by CLUE as belonging to the same cluster. Only the first one was identified by the IDS. Common patterns are highlighted in bold.

TABLE III: TidServ clusters identified by CLUE. $E = 0.4$.

<table>
<thead>
<tr>
<th>All URLs</th>
<th>TidServ URLs</th>
<th>Hostnames</th>
<th>Hostnames common string</th>
</tr>
</thead>
<tbody>
<tr>
<td>192</td>
<td>118</td>
<td>14</td>
<td>.com</td>
</tr>
<tr>
<td>79</td>
<td>75</td>
<td>1</td>
<td>wuptwywj4.cn</td>
</tr>
<tr>
<td>32</td>
<td>18</td>
<td>2</td>
<td>clickhxsanlb.com</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>bciwq1dipzkwqzmj.com</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>zl091kha444.com</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>3</td>
<td>zhakaurth.cn</td>
</tr>
</tbody>
</table>

TABLE IV: TidServ URLs identified by CLUE as belonging to the same cluster. Only the first one was identified by the IDS.

| gnsukou0t.com/IVY900p9PZY5a119h3yVv9TPQnmcxZiaWQP9NWJ3NWFt9MJE1 | zAwMDEmc1lkPTAcmcmQMQZIbhmc94d343Lmdvb2dzS5pCZaPWxWlzIhGNyDIZwZXM=16h |
| lkckckl1i1.com/1VY9mz5X5Va1XZu213x2nPTmNCZiaWQP9NWJ3NWFt9MJE1 | zAwMDEmc1lkPTAcmcmQMQZIbhmc94d343Lmdvb2dzS5pCZaPWxWlzIhGNyDIZwZXM=16h |
| lkckckl1i1.com/TVmz5X5Va1XZu213x2nPTmNCZiaWQP9NWJ3NWFt9MJE1 | zAwMDEmc1lkPTAcmcmQMQZIbhmc94d343Lmdvb2dzS5pCZaPWxWlzIhGNyDIZwZXM=16h |
| lkckckl1i1.com/TVmz5X5Va1XZu213x2nPTmNCZiaWQP9NWJ3NWFt9MJE1 | zAwMDEmc1lkPTAcmcmQMQZIbhmc94d343Lmdvb2dzS5pCZaPWxWlzIhGNyDIZwZXM=16h |
| lkckckl1i1.com/TAR3vUsX844mg1c5Y2zrPLta4zCZiaWQP9NWJ3NWFt9MJE1 | zAwMDEmc1lkPTAcmcmQMQZIbhmc94d343Lmdvb2dzS5pCZaPWxWlzIhGNyDIZwZXM=16h |
| lkckckl1i1.com/TAR3vUsX844mg1c5Y2zrPLta4zCZiaWQP9NWJ3NWFt9MJE1 | zAwMDEmc1lkPTAcmcmQMQZIbhmc94d343Lmdvb2dzS5pCZaPWxWlzIhGNyDIZwZXM=16h |
silhouette analysis, an unsupervised methodology to find how well each object lies within its cluster. In practice, a silhouette coefficient \( s(i) \) measures how close a point \( i \) is assigned to its cluster. It computes the average distance \( a(i) \) of \( i \) with all points in the cluster, and \( b(i) \) as the minimum average distance of \( i \) to points in other clusters. In formulas, \( s(i) \) is defined as:

\[
s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}
\]

where \( a(i) = \frac{1}{N} \sum_{j \in C(i)} d_{URL}(i, j) \), \( C \) being the cluster \( i \) belongs to, and \( b(i) = \min_{j \in \overline{C} \setminus \{i\}} \left( \frac{1}{N} \sum_{j \in \overline{C}} d_{URL}(i, j) \right) \). It results \( s(i) \in [-1, 1] \). Values close to 1 indicate that the sample is far away from the neighboring clusters, and that core point \( i \) is very close to all other core points in its cluster, i.e., cluster \( C \) is very compact. Instead, values close to 0 indicate that \( i \) is on or very close to the decision boundary between two neighboring clusters. Finally, negative values indicate that those samples might have been assigned to the wrong cluster. The average \( S(C) = E[s(i), i \in C] \) over all points in cluster \( C \) is a measure of how tightly grouped all the elements in the cluster are.

Since silhouette analysis allows one to understand the quality of clustering, we firstly study the distribution of \( S(C) \) among clusters. We consider only those clusters with more than 20 elements. We observe that 14% of clusters have a silhouette coefficient lower than 0.3, 37% between 0.3 and 0.5, 29% between 0.5 and 0.7 and, finally, 20% of clusters with a coefficient greater than 0.7. In a nutshell, most of the clusters have elements with a strong connectivity between them \( (S(C) \geq 0.5 \text{ for } 49\% \text{ of clusters}) \), and only a small subset of clusters appears to be sparse. These results suggest that the choice of \( E \) and \( \text{MinPoints} \) parameters are suitable for our use-case.

E. Digging into Clusters – Insights and Findings

1) Mining strongly connected clusters:
Silhouette analysis can ease the choice of the cluster to analyze. For instance, it suggests the analyst those clusters whose points are very similar inside the cluster, and very different from points in the other clusters. Table V shows the effectiveness of CLUE in giving the analyst hints about the traffic generated by the system. Focus first on the top part of the table, which ranks clusters by decreasing effectiveness of CLUE. Mined all clusters have elements with a strong connectivity between them \( (S(C) \geq 0.5 \text{ for } 49\% \text{ of clusters}) \), and only a small subset of clusters appears to be sparse. These results suggest that the choice of \( E \) and \( \text{MinPoints} \) parameters are suitable for our use-case.

TABLE V: Clusters information sorted by Silhouette coefficient on the top, and for number of elements in the bottom. In all cases, clusters clearly pinpoint specific services.

<table>
<thead>
<tr>
<th>( S(C) )</th>
<th>Main hostname (unique number)</th>
<th>Elements</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.92</td>
<td>skype_streaming-s.akamaehd.net (1)</td>
<td>551</td>
<td>Streaming</td>
</tr>
<tr>
<td>0.91</td>
<td>ad.doubleclick.net (1)</td>
<td>99</td>
<td>Advertising</td>
</tr>
<tr>
<td>0.87</td>
<td>cookex.amp.yahoo.com (1)</td>
<td>61</td>
<td>Malware</td>
</tr>
<tr>
<td>0.85</td>
<td>static.simply.com (1)</td>
<td>25</td>
<td>File Hosting</td>
</tr>
<tr>
<td>0.81</td>
<td>d24w6bsrhdhe9d.cloudfront.net (1)</td>
<td>63</td>
<td>File Hosting</td>
</tr>
<tr>
<td>0.81</td>
<td>mfdclk001.org (1)</td>
<td>27</td>
<td>Malware</td>
</tr>
<tr>
<td>0.78</td>
<td>adserver.webads.it (1)</td>
<td>35</td>
<td>Advertising</td>
</tr>
<tr>
<td>0.77</td>
<td>.com (3)</td>
<td>37</td>
<td>TidServ</td>
</tr>
<tr>
<td>0.75</td>
<td>pixel.quotserve.com (1)</td>
<td>57</td>
<td>Advertising</td>
</tr>
<tr>
<td>0.72</td>
<td>watson.microsoft.com (1)</td>
<td>29</td>
<td>Windows Debug</td>
</tr>
<tr>
<td>0.70</td>
<td>coadvertise.cohecdn.net (1)</td>
<td>36</td>
<td>Advertising</td>
</tr>
<tr>
<td>0.69</td>
<td>atdmt.com (2)</td>
<td>768</td>
<td>Tracking</td>
</tr>
<tr>
<td>0.65</td>
<td>su.ff.avast.com (1)</td>
<td>82</td>
<td>Avast Update</td>
</tr>
<tr>
<td>0.64</td>
<td>log.dmtry.com (1)</td>
<td>24</td>
<td>Advertising</td>
</tr>
<tr>
<td>0.61</td>
<td>clickpixelabn.com (1)</td>
<td>32</td>
<td>Malware</td>
</tr>
</tbody>
</table>

2) Mining bigger clusters:
The second part of Table V reports clusters sorted according to the number of unique URLs. The largest cluster contains many elements with a huge number of different hostnames. It aggregates all the “normal” traffic of the network, whose URLs are well-formed, with syntactically simple hostnames and paths. Given the very large number of services, it is possible to transform a URL into a similar one so that a large dense area is identified. Notice that this cluster is the only one with many services with less legitimate purposes. Some, e.g., cookex.amp.yahoo.com, mfdclk001.org, clickpixelabn.com, are associated to malware activities. Also in this case, a simple Google search unveils immediately clues to understand the picture. The first cluster, matching the hostname cookex.amp.yahoo.com, for instance, is associated to spyware actions. Beside the common hostname, we find the common part “http://ad.yieldmanager.com/imp” in the URL path. Probably this malware is connected to click fraud activities, and the C&C server hosted at cookex.amp.yahoo.com instruments the bots to perform fake-clicks on legitimate ads hosted on the Yieldmanager infrastructure. The second cluster, mfdclk001.org, is linked to a modified version of TidServ, while the last, clickpixelabn.com, is a C&C server of a different malware. Also in this case, the analysis of URLs gives interesting hints about the possible malicious activities, not discussed for the sake of brevity. In all cases, the availability of several URLs ease the signature extraction job, e.g., to augment IDS or firewall coverage.
a bad Silhouette index. It would be recommended to run again CLUE on such cluster with a smaller choice of $E$ to identify URLs subgroups. The second largest cluster aggregates URLs from the Facebook CDN, as easily highlighted by the common substring found in hostnames. It serves all images, videos, and static Facebook content. The third largest cluster collects all outliers. Here we find the 7,352 URLs that fall outside any dense area. Again, it would be appropriate to eventually re-run CLUE with a different choice of $E$.

Going down in the list, we observe URLs generated automatically by blog and advertising platforms, by newspapers websites, and by third-party tracking services [24]. These services follow users during their online activities, identifying them using different techniques. For instance, user identifiers and other information about advertising (e.g., timestamp or banner size) are often sent to the server embedded into URL queries. Since these elements are obviously created artificially and follow a regular syntax, they are easily identifiable by CLUE. This is why this category of services appears repeatedly in our results. By analyzing the URLs, it is possible to unveil some of the mechanisms they use to track online activities. For instance, doubleclick.net, an advertising service subsidiary of Google, communicates using artificial URLs with a key-value pattern, e.g., “u=”, “kv1id=”, “kp1d=” or “sz=”, that clearly carries information about user and her navigation. CLUE can thus also help researchers, network or security analysts to detect (and eventually filter) these services.

In a nutshell, CLUE proves very useful in grouping similar URLs, thus easing the analysis of HTTP traffic and naturally letting common patterns emerge. Applications for security and privacy are straightforward.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we presented CLUE, an unsupervised system to mine URLs from passive HTTP traffic traces. Based on unsupervised machine learning techniques and algorithms, it offers the analyst the ability to explore well defined clusters where similar URLs are grouped. Thanks to the cohesiveness of the formed clusters, clues about the processed traffic easily emerge and let the analyst identify possibly malicious traffic or advertisement, tracking and third-party services.

While we clearly showed the effectiveness of our approach by presenting produced results on a real but small dataset, an additional effort is still required to make the system scale at higher traffic volumes or, possibly, runtime usage. This is, by now, a limitation of our solution that we plan to tackle in future works. As further improvements, we are also working on the design of a hierarchical approach with which clusters can be further analyzed by changing CLUE parameters, allowing the identification of fine-grained subset of URLs.

REFERENCES


Testing for traffic differentiation with ChkDiff: the downstream case

Riccardo Ravaioli
Université Nice Sophia Antipolis
Laboratoire I3S/CNRS UMR 7271
Sophia Antipolis, France
ravaioli@i3s.unice.fr

Guillaume Urvoy-Keller
Université Nice Sophia Antipolis
Laboratoire I3S/CNRS UMR 7271
Sophia Antipolis, France
urvoy@i3s.unice.fr

Chadi Barakat
Inria
Sophia Antipolis, France
chadi.barakat@inria.fr

Abstract—
In the past decade it has been found that some Internet operators offer degraded service to selected user traffic by applying various differentiation techniques. If from a legal point of view many countries have discussed and approved laws in favor of Internet neutrality, confirmation with measuring tools for even an experienced user remains hard in practice. In this paper we extend and complete our tool ChkDiff, previously presented for the upstream case, by checking for shaping also on the user’s downstream traffic. After attempting to localize shapers at the access ISP on upstream traffic, we replay downstream traffic from a measurement server and analyze per-flow one-way delays and losses, while taking into account the possibility of multiple paths between the two endpoints. As opposed to other proposals in the literature, our methodology does not depend on any specific Internet application a user might want to test and it is robust to evolving differentiation techniques that alter delays or induce losses. We provide here a detailed description of the downstream tool and a validation in the wild for wired, WiFi and 3G connections.

I. INTRODUCTION
The increasing popularity of bandwidth-hungry applications, like peer-to-peer and video streaming, has induced some Internet Service Providers (ISPs) in the last decade to deploy some traffic management techniques that offer degraded performance instead of best-effort service to specific traffic flows. Reported cases abound: from blocking of BitTorrent traffic when a user is actively sharing files [1], to reduced performance of Netflix by a few US operators [2], [3] and throttling of YouTube during peak hours in the evening [4], [5]. Also competing services such as VoIP have been the target of ISPs, for instance when all Vonage calls were systematically blocked by a regional mobile operator [6] and when Apple’s FaceTime was disabled for mobile customers who did not opt for a more expensive data plan [7].

All these examples constitute clear violations of Internet neutrality, a principle according to which a network should treat all its incoming traffic equally, without deliberately offering worse or better performance to any traffic of its choice. There has been discussion [8] about whether the Internet has been conceived and implemented as a strictly level-playing field, but from a broader point of view it is generally agreed upon that all attempts that selectively deteriorate certain types of Internet traffic over others are, to say the least, controversial. Because of this, legislative efforts aiming at prohibiting cases like the above ones have appeared in a number of countries, the first of which to approve such laws were Chile [9] in 2010 and the Netherlands [10] in 2012.

In the literature, several tools [11]–[16] have been described in recent years to try to establish whether an ISP is applying traffic differentiation to specific applications (e.g., BitTorrent, YouTube, Skype, etc.) and with the use of specific differentiation techniques (e.g., port blocking, token-bucket shaper, etc.).

The solution that we propose in this paper, ChkDiff, directly addresses the problems of scalability to different user applications and of applicability to different shaping techniques affecting delays and losses. We achieve this by performing active measurements with the real user traffic, comparing the performance of a flow against the performance of the rest of the replayed traffic, and by analyzing for each flow its delays and losses, which reflect any alteration introduced by a shaper inside the network.

This complements our previous work [17], in which we focused on the user’s upstream traffic and replayed it with low TTL values in a traceroute-like manner against the routers at the first few hops away from the user in order to detect differentiation and localize shapers. We extend this with a new experiment in the downstream direction, where we replay the user’s incoming traffic from a server, measure one-way delays and losses, and check for differentiation on a per-flow basis. We describe in details the measures taken by ChkDiff in order to successfully deliver the replayed trace and validate the tool in two differentiation scenarios, with the server located in three different data centers, and over wired, WiFi and 3G connections.

The paper is organized as follows: in Section II we provide a detailed description of the methodology we used in ChkDiff; in Section III we validate the tool; we discuss our method in Section IV and assess it with respect to related work in Section V; we give closing remarks in Section VI.
II. METHODOLOGY

The design of a new tool for the detection of traffic differentiation has to necessarily consider two weak points of existing methods: the difficulty to scale to different applications and the limitation to specific differentiation techniques. We overcome this in three steps: a) we use the real traffic of a user and not a synthetic trace; b) we minimize the modifications to the trace needed for the experiment to work and c) we analyze the performance of a flow in terms of delays and losses with respect to the rest of the trace in order to infer neutrality violations: these two metrics alone are able to capture the effect of shapers at the IP layer.

A complete run of ChkDiff consists of two experiments, one that replays the user’s outgoing traffic (upstream direction) to the routers at the first few hops away from the user and one that replays the user’s incoming traffic (downstream direction) from a measurement server to the user. We report in Algorithm 1 an outline of a full execution of ChkDiff.

At first the tool dumps client traffic for a time window of typically 3-5 minutes, while the user is asked to run the applications and services of an Internet session she wishes to test. Next, packets are grouped into 5-tuple flows (source and destination IP addresses, transport protocol, source and destination port numbers), which are further arranged into an outgoing trace, \( trace_{out} \), and an incoming one, \( trace_{in} \). We now briefly describe the upstream experiment, as proposed in an earlier work [17], and then go on to illustrate in detail the methodology for the downstream case.

A. Upstream experiment, in a nutshell

Non-trivial outgoing traffic that an access ISP might want to differentiate includes media uploading, P2P file sharing and VoIP; a run of ChkDiff in the upstream direction should ideally test at least one type of such traffic. Before conducting the actual experiment, we shuffle \( trace_{out} \) in such a way that the position of the packets of each flow inside the trace follows a Poisson process, so that according to the PASTA property (Poisson Arrivals See Time Averages) [18], each flow will see the same network conditions when the trace is replayed. The order of packets within each flow is preserved.

As in the first half of Algorithm 1, we consider the first few hops away from the user, at or in proximity of her access ISP network (up to hop 3 or 4, as in Figure 1a), and for each hop we replay \( trace_{out} \) at a constant sending rate higher than the original one and with a modified IP TTL set to \( h \), so that the trace will expire on the router(s) at hop \( h \) and generate ICMP time-exceeded messages. We showed the validity of this ICMP feedback in a previous work [19], along with its robustness to ICMP rate-limitation. Each flow having its own set of Round-Trip Times (RTTs) and losses, we can now compare its performance up to a given hop to that of the rest of the flows along the same path. We use Kolmogorov-Smirnov test to analyze delays and a binomial-inspired test to analyze losses. By aggregating the results of each flow across consecutive hops we are able to infer the presence of a shaper and localize it in terms of number of hops from the client.

B. Downstream experiment

In this paper, we present the downstream version of ChkDiff and validate it experimentally. In brief, the experiment in the downstream direction consists in taking all necessary measures to replay the original incoming flows, having the server replay the trace to the client and finally analyzing the results in a way that takes into account the possibility of having multiple paths to the client (Figure 1b).

The second half of Algorithm 1 outlines the main steps of the downstream experiment, which we describe in details in the remainder of this section. First, we need to shuffle \( trace_{in} \) in the same way we did in the upstream experiment. This allows us to eliminate cross traffic noise from the effects of possible neutrality violations. Then, for a replayed flow to be able to successfully reach the client, we need to deal with possible Network Address Translation (NAT) and firewall devices a user might be behind and also other possible middleboxes that might be deployed along the path from the

Fig. 1: The two experiments in ChkDiff.
server. After all connections are initiated from the client side, the server replays the shuffled $\text{trace}_\text{in}$ to the client at a rate higher than the original one. We compute One-Way Delays (OWD’s) for each flow and note the number of losses, if any. In order to infer differentiation, we run a clustering analysis on delays so that we can distinguish when different delay distributions are due to shaping and when they are due to a variety of paths. Lastly, a test on flow losses completes the analysis. We elaborate now on each of the above actions.

1) Getting ready for replaying. As opposed to the upstream experiment [17], we do not pad packets in $\text{trace}_\text{in}$ to a fixed size (the maximum packet size in the trace). In the upstream experiment the low variability of the total delay along a short wired path is in the same order of magnitude as the variability of the transmission delay, which is proportional to the packet size. That makes it impractical to replay packets exactly as captured, since flows with large packets over a wired connection experience larger delays solely because of their packet size. For the downstream case, we examine a much longer path in terms of hops and delays, and such source of error is canceled out by the inherent delay variability along a larger path. Therefore we are able to replay the packets with their original payloads, as seen by the client upon receiving them.

Replaying incoming traffic from a single source (i.e. our server) means that we cannot keep the original source IP addresses of the user trace, for two main reasons. Firstly, most access networks today are configured to drop outgoing packets with a source address that does not belong to the address space of the access network itself. In other words, they do not allow IP address spoofing [20]. Secondly, as we will see shortly, if a NAT device is present, we can replay a flow only if we find the mapping applied by the NAT to that flow for external endpoints. Since we are not in control of other endpoints (that is to say, all network applications or services run by the user) than our measurement server, we need to overwrite the IP source address of each packet in $\text{trace}_\text{in}$ with the IP address of the server; original port numbers are retained. Conversely, in the upstream experiment the original source and destination IP addresses are preserved. However, even if in the downstream case we lose the ability to reveal shapers based on the source IP address, we can combine upstream and downstream experiments to overcome the limitations of both. Furthermore, the commercial shapers studied in a recent work [21] do not use this piece of information to classify flows.

After being shuffled, $\text{trace}_\text{in}$ is sent via FTP to the server. While we deploy a server with a public IP address, listening on a known port, a client is likely less easy to reach: a few network elements need to be considered before we can replay the trace to the client.

- **NAT’s.** In today’s networks, where IPv4 addresses are running out, deploying a NAT device has become a widespread practice. For our purposes, this means that the client’s view of a flow might not be the same as the server’s. Since the trace to replay is originally as seen by the client, we might need to modify the destination IP address and port number in order to reach the client from an external endpoint (Figure 1b). NAT devices are usually defined according to how they map the same source pair $X:x$ of IP address $X$ and port number $x$ of an internal network when different external destination IP addresses and port numbers are reached (for instance $Y_1:y_1$ and $Y_2:y_2$) [22], [23]. We distinguish four cases: 
  - $i$) in the simplest scenario the mapping is independent of the external destination endpoint and will not change for the same source pair $X:x$; 
  - $ii$) some NAT’s generate the same mapping only with same external destination IP address ($Y_1 \equiv Y_2$ or $Y_1 \equiv Y_3$) with the same external destination IP address and port number ($Y_1 \equiv Y_2$ and $y_1 \equiv y_2$); 
  - $iii$) the mapping might be connection-dependent and vary each time a new connection to the same external destination pair is initiated.

- **Firewall.** We assume that any user is protected by a firewall from the outer network. Consequently, all flows need to be initiated from the user side (the so-called hole punching) before the server can replay the trace. For TCP flows, we reproduce the whole 3-way handshake without the interaction of the kernel on both endpoints. We assign an initial sequence number for each side of a TCP flow and modify it accordingly in the data packets. For UDP flows, we only send one probe from the client.

- **Initial sequence numbers.** It has been observed [24], [25] that some middleboxes overwrite the initial sequence number of TCP flows. Since firewalls can easily keep track of the sequence numbers and reject inconsistent packets, it is important to intercept the overwritten

---

**Algorithm 1 ChkDiff complete execution**

1: Capture user traffic, store into $\text{trace}_\text{out}$ and $\text{trace}_\text{in}$
2: $\triangleright$ Upstream experiment
3: $\text{for each hop } h \in \{1,2...k\}$ $\text{do}$
4: $\text{for each run } r \in \{1,2,3\}$ $\text{do}$
5: $\triangleright$ shuffle $\text{trace}_\text{out}$
6: $\triangleright$ replay $\text{trace}_\text{out}$ with $TTL \leftarrow h$
7: $\text{collect ICMP time-exceeded replies}$
8: $\text{end for}$
9: $\text{detect shaped flows at hop } h$
10: $\text{end for}$
11: aggregate results and locate shaper(s), if any
12: $\triangleright$ Downstream experiment
13: $\text{for each run } r \in \{1,2,3\}$ $\text{do}$
14: $\triangleright$ shuffle $\text{trace}_\text{in}$
15: $\text{for each flow } f \text{ in } \text{trace}_\text{in}$ $\text{do}$
16: $\triangleright$ find NAT mapping for $f$
17: $\triangleright$ initiate connection from client
18: $\text{end for}$
19: $\triangleright$ replay $\text{trace}_\text{in}$ from server to client
20: $\text{compute one-way delays and losses}$
21: $\text{end for}$
22: $\text{detect shaped flows}$

---

297
sequence number from the SYN packet of the TCP handshake and modify all subsequent sequence and acknowledgement numbers individually for each TCP flow.

- **Timeouts.** In NetFilter [26], the standard firewall provided in Linux, the TCP timeout for established connections is 5 days, while for UDP it is only 30 seconds whether packets have been seen in one or both directions. This means that we need to make sure that the time elapsing between two consecutive packets of each UDP flow, initial probes included, is less than 30 seconds. NAT mappings expire too in order to remove stale entries from the NAT table. Given the large amount of commercial NAT devices, we refer in this case to the requirements and guidelines found in the RFC’s. For UDP flows [22] the timeout should not be less than 2 minutes, while 5 minutes is recommended; for TCP [23] if the connection is not yet in the established state, the timeout should not be less than 4 minutes, and if it is already in the established state it should be no less than 2 hours and 4 minutes. These values are all large enough for the purpose of our experiment and do not interfere with ChkDiff. Finally, to avoid unnecessary synchronization between client and server, we do not actually send any acknowledgments from the client side for the TCP flows we replay. In the Linux kernel, the socket parameter TCP_USER_TIMEOUT sets the maximum amount of time that data can be transmitted without being acknowledged. Its default value is 20 minutes [27], which is roughly one order of magnitude larger than a single run of ChkDiff.

Given that our goal is not to classify all possible devices along a path but to rapidly deliver the trace to the client, we take a conservative approach and assume that the most stringent restrictions among the above ones are in place. We expect the NAT to be connection-dependent and perform a per-flow mapping discovery already during the hole punching initiated by the client against her firewall. For each flow, we encrypt the client’s view of its source IP address and port number and add it to the payload of its SYN packet or UDP probe, as NAT devices are expected to overwrite every occurrence of the client’s own IP address in a packet. We set the client-side initial sequence number of TCP flows in accordance to the acknowledgement numbers used in the trace (since no packets are sent from the client during the replaying phase, the acknowledgement number of a TCP flow sent by the server is constant across packets of the same flow). From the server side, for TCP flows, we keep track of incoming SYN packets along with the observed sequence numbers and client’s source pair, and mimic the TCP handshake; for UDP flows we just keep track of client’s and server’s views of the client-side source pair. When these two views differ, we modify the client-side IP address and port number of the corresponding flows in trace_20 to with the pair as seen by the server. The server also overwrites the acknowledgement number of a TCP flow, when the number in the trace does not match the sequence number seen in the received SYN packet. Additionally, in order to overcome the relatively short timeout on UDP flows, we enforce a maximum interval of 30 seconds between any two UDP packets of the same flow when shuffling trace_20 and start a timeout on the client side to make sure that we do not exceed 30 seconds between the initial UDP probe of a flow and its first occurrence in the trace. We discard the current run of the experiment and start a new one if ever this timeout expires. In any case, during an ordinary execution of ChkDiff only a few seconds elapse between hole punching and the start of the replaying phase.

We avoid the overhead of opening a socket for each flow and replaying the trace from the application layer by injecting packets with tcpreplay [28] directly between the IP layer and the Network Interface Card. Since we are emulating TCP and UDP flows below the IP layer, we need to prevent our packets from reaching their respective client applications, which could cause unsolicited traffic or unexpected application behaviour. Also, our hole-punching probes target ports on the server on which no process should be listening and will trigger TCP reset packets for TCP SYN probes, ICMP port-unreachable messages for UDP probes and real application packets if ever a process is indeed listening. As error messages cross a firewall on their way to the user, the corresponding newly-created connection entries are removed. Therefore, we need a way to distinguish between experiment packets and regular traffic, so that we can drop the former right before they reach the IP stack and we can allow the latter to pass through. We achieve this by assigning a unique number to each user session and overwrite with this value the 2-Byte IP ID field of each experiment packet between a given user and the server. Through a combined use of tcpdump and iptables, as shown in Figure 2, we dump the experiment packets right at the Network Interface Card and drop them before they reach the kernel.

2) **Replay.** We are now ready to replay the trace from the server at a constant packet rate higher than the original one (by default, we replay it at twice the original rate).

We dump the replayed trace on both the server and the client and then for each flow we measure One-Way Delays and note the number of lost packets. For simplicity, we avoid any clock synchronization between user and server: any effect
due to clock skewing is not expected to disrupt the measured delays for the short time window of one experiment (a few tens of seconds) and in any case will affect all flows equally, as they are evenly spread across the trace.

Once the server has completed the replaying phase, the client closes the emulated TCP connections by sending an RST packet for each TCP flow in trace_out. This has the added benefit of clearing space in the open connection table of the firewall, if ever a per-user restriction is active.

3) Results analysis. The study of delays between two endpoints across a path of several hops has to necessarily take into account the possibility of multiple paths. Discovering the paths taken by each flow would be cumbersome: first of all, we do not know the exact hash function applied in a load balancing decision and we observed that some data centers, where the measurement server could be deployed, make a massive use of load balancers; second of all, it would take some extra time, as we would need to probe at low rates (i.e. 1 packet per second) to bypass ICMP rate limitation [19] and have a complete view of each path. An example is provided in the timeseries of Figure 3, where we can visually identify at least five different paths; a direct comparison between any two flows becomes harder in this case. In the absence of the ground truth, we can rely on the fact that non-differentiated flows following the same path will have similar delay distributions, which a clustering algorithm can group together. A differentiated flow going on any of the available paths will show a distribution significantly different from that of all other flows and should not belong to any of the discovered groups. We combine this with a loss analysis in order to capture the behaviour of shapers.

• Delays. Our choice of clustering algorithm for delays is dbscan [29], which groups together points that are in the same high-density area and labels as outliers those that do not belong to any found cluster. As a representative point for each flow we take its 25th percentile: it is close enough to the real path delay, it discards possible queueing delays and it is robust to delay variations due for example to WiFi. The algorithm then takes two parameters: the minimum number $n$ of core samples to form a cluster and the maximum distance $\epsilon$ between any two samples for them to be included as core points in a cluster. Since we expect shaped flows to stand out from non-shaped flows, we set $n$ to 2. As for $\epsilon$, we need a value that reflects the delay variations of a path: we take the core values (2nd quartile range, i.e. the 25th-50th percentile range) of the delay distribution of each flow, we aggregate them and then pick for $\epsilon$ a large value in this set, the 75th percentile. The output of dbscan will be a set of clusters of flows and a set of outliers, which we label as having failed the delay analysis.

• Losses. We compare the losses of a flow to the loss rate of the rest of the trace as a whole. The reasoning is the following: if a flow $i$ with $s_i$ packets has not been differentiated, its number of lost packets can be modeled as a binomial random variable of parameters $B(s_i, p)$, where $p$ is the loss rate of the rest of the trace. If we approximate this binomial to a normal random variable of parameters $N(s_i p, s_i p(1-p))$, we can verify whether the number of lost packets $l_i$ of flow $i$ lies within $\alpha$ standard deviations of the normal mean, where $\alpha$ is approximated to 2.58 for our chosen significance level of 99%. Since we are interested to know whether a flow experienced more losses than it should have with the global loss rate $p$, we check that $l_i < s_i p + \alpha \sqrt{s_i p(1-p)}$. If the condition does not hold, the flow is rejected by our loss analysis. Since a shaper affects the delays or the losses of a flow, or both, we reject a flow if it fails either analysis.

We repeat the whole experiment three times in order to remove transient errors and claim that a flow was differentiated if in all three runs it failed the combined analysis of delays and losses.

III. Validation

We validate the downstream experiment of ChkDiff in wired, WiFi and 3G setups (Figure 4) with a client located in France and the server located in three different Amazon data centers: Germany, Ireland and Oregon (USA). The client is directly connected to a middlebox, where the shaper is deployed and which serves also as the client’s gateway. In the WiFi setup, the client is connected to the gateway through a dedicated WiFi network operating on the same channel as
the local University WiFi network to cause more link-level collisions. In the 3G setup, the client is connected to the middlebox via a wired connection and the middlebox is connected via WiFi to a mobile phone functioning as hotspot. We test ChkDiff in two differentiation scenarios: given a set of flows we want to differentiate, in Scenario 1 we throttle their bandwidth and in Scenario 2 we apply a uniform packet drop rate to them. We configure dummynet [30] on the middlebox to shape incoming traffic, as shown in the upper part of Figure 4. Flows to shape are forwarded to the upper pipe, which applies the desired differentiation technique according to the scenario we test; flows that we do not intend to differentiate go through the lower pipe, which only adds a constant delay equal to the transmission delay of the upper pipe in Scenario 1 and has no effect in Scenario 2. This way, in Scenario 1 the difference in delays between shaped and non-shaped flows is due only to the queueing delay at the upper pipe. A final pipe, where all flows eventually go, emulates a 100 Mbit/s link. In Scenario 2 this last pipe causes uniform drops on the whole trace.

In all experiments we replay a trace of approximately 9000 packets captured during an Internet session of 3 minutes that included watching a short streaming video, browsing a news website and making a call on Skype. In dummynet, our pipes have a buffer length of 100 packets and use droptail buffer management policy.

A. Shaping pipe (Scenario 1).

In this scenario, we compute in trace\(_{\text{ids}}\) the overall sending rate of the flows to shape and set the shaping pipe to a fraction \(k_{\text{bw}} < 1\) of this value. The second parameter we vary is the fraction \(fr\) of packets we shape. When picking which flows to differentiate, we choose iteratively the flow closest to the target \(fr\), until the desired size is reached. All flows we differentiate go through the same shaping pipe. By combining delay and loss analysis, we show that we can effectively identify all shaped flows as long as the fraction \(fr\) does not constitute most of trace\(_{\text{ids}}\), which is what we expect when we take the whole trace as baseline for comparison.

We present in Figure 5 the results in terms of recall\(^1\) for each of the three server locations over wired, WiFi and 3G connections. For constraints of space, we do not include the results in terms of precision, commonly defined as the number of true positives over the number of positives, since we found it for all experiments to be the optimal one, at 100%; in short, we never flagged a non-differentiated flow erroneously. For each pair of \(fr\) and \(k_{\text{bw}}\) we show a pie where the color of the upper-left quarter represents the result of the delay analysis, the color of the upper-right quarter represents the result of the loss analysis, and the color of the lower half is the outcome of the combined analysis. If for \(fr \leq 40\%\) in almost all cases the delay analysis suffices to detect all shaped flows, for \(fr = 60\%\) we see that combining the delay and loss analysis is essential for a correct output. When \(k_{\text{bw}} = 1.0\) the shaping pipe is configured with the average bit rate of incoming packets, so the flows that are supposedly being differentiated might not be shaped at all, hence the uncertain outcome on the top row of each graph.

In this scenario, ChkDiff appears to cope just as well with a wired connection as it does with WiFi and 3G. Over the two wireless connections, there seems to be more noise that is in any case neutralized for the most part when combining the two analysis.

B. Uniform drops (Scenario 2).

We consider now a shaper that uniformly drops packets of selected flows at a loss rate \(lr\) (upper-right pipe in Figure 4) and of the whole trace at a loss rate \(lr_{\text{all}}\) (left pipe in the same figure). As opposed to the previous scenario, we deploy here a dedicated shaping pipe for each flow we want to differentiate. We vary \(lr\) and \(lr_{\text{all}}\), as well as the fraction \(fr\) of traffic impacted by \(lr\). Shaped flows will thus have an overall loss rate \(lr_{\text{shaped}}\) equal to \(1 - (1 - lr)(1 - lr_{\text{all}})\). For reasons of space we only show the results for the server located in the data center in Germany, over the three types of connection considered previously (Figure 6). Nevertheless, results across the three server locations are qualitatively similar. Since in this second scenario the differentiation we apply does not affect delays, we focus only on the outcome of the loss analysis. For completeness, results are shown also for high values of \(lr_{\text{all}}\), even if in practice a global loss rate of \(20\%\) is already able to disrupt TCP connections. We observe for both wired and wireless setups that when \(fr\) is less than half of the trace, ChkDiff is able to detect all differentiated flows except for the cases on the bottom diagonal, where the difference in loss rate between differentiated and non-differentiated flows is the lowest (5%, 10% and sometimes 20%). Experiments over WiFi and 3G seem to be only slightly worse than over wired for the lowest values of \(lr_{\text{all}}\).

IV. DISCUSSION

A full run of ChkDiff in upstream and downstream directions is able to detect differentiation when, regardless of its implementation, it directly worsens the throughput, packet delay and losses of user applications. This is the typical effect introduced by a shaper. Even though in this paper we used the terms shaping and differentiation interchangeably, the former is a subset of the latter and shaping is what we aim at detecting with our current tool. As we saw in the validation section, we cannot precisely reveal shaping when most of the user traffic is affected, as our baseline in the analysis would mainly be made of differentiated flows. To counteract this, a user should be running different applications during the capturing phase, so as to have a variety of flows to check against. In the extreme case, if really an ISP throttles the bandwidth of all traffic for a given user, it would not be possible to discern it from severe network congestion from the view point of this particular user.

In this work, we assume that ISPs select flows to differentiate based on packet fields (IP addresses, ports and...
payload). We do not directly address classification based on flow bandwidth, but we make sure that we replay the trace at a higher global rate than the original one.

Also, we do not detect differentiation when it aims at providing better treatment to selected traffic. Such behaviour could be the result of an agreement between a content provider and an ISP and it does not necessarily imply any worse conditions for the rest of the traffic than in normal network conditions. It would be interesting to redefine the delay analysis in both upstream and downstream experiments to account for this, as it could reveal what services and applications are favored in a given network. We plan to address this in future work.

The tool is available for Linux machines on the web page of the project\(^2\). We currently provide a server located in our lab, where no traffic differentiation is taking place.

V. RELATED WORK

Many tools for the detection of traffic differentiation have appeared in the literature in recent years. Among the first ones, BT-test [12] checks for injected RST packets while emulating a BitTorrent packet exchange between a user and a server. Other tools [13]–[16] compare the performance of a synthetic

\(^2\)http://chkdiff.gforge.inria.fr/
(a) Recall when $fr = 20\%$, wired connection.
(b) Recall when $fr = 40\%$, wired connection.
(c) Recall when $fr = 60\%$, wired connection.
(d) Recall when $fr = 80\%$, wired connection.

(e) Recall when $fr = 20\%$, WiFi connection.
(f) Recall when $fr = 40\%$, WiFi connection.
(g) Recall when $fr = 60\%$, WiFi connection.
(h) Recall when $fr = 80\%$, WiFi connection.

(i) Recall when $fr = 20\%$, 3G connection.
(j) Recall when $fr = 40\%$, 3G connection.
(k) Recall when $fr = 60\%$, 3G connection.
(l) Recall when $fr = 80\%$, 3G connection.

Fig. 6: Recall of loss analysis as we vary $fr$ in Scenario 2. Results are from the server located in Germany over wired, WiFi and 3G connections.

application flow to the performance of a similar flow with some modified or randomized packet fields (port numbers or payloads), so that a shaper targeting such application would affect the former and not the latter. Glasnost [13] looks for differences in throughput between these two flows, while DiffProbe [14] attempts to create congestion in the ISP network by scaling up the replaying rate of application and control flows, and then analyzes their delay and loss distributions. ShaperProbe [15] expands on DiffProbe by considering the case of a shaper implemented as a token bucket and tries to infer its parameters as the received rate at the destination shows a level shift. Packsen [16] also tries to identify the shaper type and its parameters, but claims to use a more efficient statistical analysis. As opposed to ChkDiff, these tools are limited to the set of application traces made available by their authors (e.g., Skype, BitTorrent, YouTube, etc), which would make it hard to maintain them in the long run.

A recent work, Differentiator Detector [21], aims at solving this by replaying between user and server a captured user trace and reproducing the same original application behaviour (ports, payloads, inter-packet times) at the application layer, first through a direct path between the two endpoints (application flow) and then through VPN tunnel to a middlebox (control flow). It measures throughput, RTT distribution and losses in order to detect shaping. Even though our tool replays separately upstream and downstream traffic, in the upstream direction it tests the real hops where the original packets went instead of testing the path between client and server, and in the downstream direction it deals in a more robust and scalable way with NAT devices. Moreover, differentiation of VPN’s is not unheard of [31] and the released version of Differentiation Detector only provides a set of pre-recorded traces to replay.

NetPolice [11] makes use of TTL-limited probes from numerous vantage points to ingress and egress routers of backbone ISPs to find differentiation in backbone networks. It replays a few synthetic application flows along with an HTTP flow, which serves as control flow, and looks for differences in loss rates on the same path segments.

Zhang et al. [32] propose a theoretical framework where they conduct measurements on target links from a large number of vantage points and try to derive differentiation from link properties typical of non-differentiated networks, in a way inspired by network performance tomography. This method, if deployed, would need a substantial and diverse user base in order to have enough vantage points and would require a central server to process measurement results.
Nano [33] differs from existing solutions in that it carries out passive measurements on user traffic and compares it against a data set of other users in the same geographical area, with comparable machine setups, at the same time of the day, but connected to a different ISP. While this method is undeniably independent of user applications and differentiation techniques, its main disadvantage is that it needs a fairly large number of users for it to be operational.

VI. Conclusion

We extended with a downstream experiment ChkDiff, a tool which enables users to detect differentiation on their own traffic. After first checking for degraded traffic performance on upstream traffic, the tool replays user incoming flows from a measurement server to the user and analyzes delays and losses to verify whether each flow experienced the same network conditions as the rest of the trace. While in the upstream direction our tool proved to be robust to rate limitation in the ICMP feedback generated by routers, in the downstream case we successfully cope with NAT’s and middleboxes in front of the client and with end-to-end measurements possibly representing a diversity of paths between server and client. We validated ChkDiff in the wild, with two differentiation scenarios over three types of connections: wired, WiFi and 3G. We showed that it correctly identifies shaped flows when up to half of a user trace is affected.

In future work, we envisage to include in the tool tests that check for differentiation techniques that do not necessarily alter delays and losses, as for instance TCP RST injection. We also intend to run a study with volunteers in a variety of wired and mobile setups in order to have a mapping of the current practices of ISPs.

ACKNOWLEDGMENTS

This work was funded by the French Government (National Research Agency, ANR) through the “Investments for the Future” Program reference #ANR-11-LABX-0031-01.

REFERENCES

ModelGraft: Accurate, Scalable, and Flexible Performance Evaluation of General Cache Networks

Michele Tortelli*, Dario Rossi*, Emilio Leonardi†
* Télécom ParisTech, Paris, France  † Politecnico di Torino, Torino, Italy

Abstract—Large scale deployments of general cache networks, such as Content Delivery Networks or Information Centric Networking architectures, arise new challenges regarding their performance evaluation for network planning. On the one hand, analytical models can hardly represent in details all the interactions of complex replacement, replication, and routing policies on arbitrary topologies. On the other hand, the sheer size of networks and content catalogs makes event-driven simulation techniques inherently non-scalable.

We propose a new technique for the performance evaluation of large-scale caching systems that intelligently integrates elements of stochastic analysis within a MonteCarlo simulative approach, that we colloquially refer to as ModelGraft. Our approach (i) leverages the intuition that complex scenarios can be mapped to a simpler equivalent scenario that builds upon Time-To-Live (TTL) caches; it (ii) significantly downscales the scenario to lower computation and memory complexity, while, at the same time, preserving its properties to limit accuracy loss; finally, it (iii) is simple to use and robust, as it autonomously converges to a consistent state through a feedback-loop control system, regardless of the initial state.

Performance evaluation shows that, with respect to classic event-driven simulation, ModelGraft gains over two orders of magnitude in both CPU time and memory complexity, while limiting accuracy loss below 2%. In addition, we show that ModelGraft extends performance evaluation well beyond the boundaries of classic approaches, by enabling study of Internet-scale scenarios with content catalogs comprising hundreds of billions objects.

I. INTRODUCTION

Caching systems, from their simplest single-cache incarnation to more complex general networks, have attracted remarkable attention over the years. Different approaches have been proposed to study their performance, from exact analytical models [17] with a high computational cost, to refined approximations able to reduce the computational cost while preserving their accuracy within acceptable bounds [8, 15, 9, 18, 32, 7, 10, 31, 25, 22]. However, with Content Distribution Networks (CDNs) first, and then with the advent of a new networking paradigm, namely Information Centric Networking (ICN) [28], caches become the atomic part of globally deployed networks. Under CDN/ICN architectures, several factors like content replacement algorithms, cache decision policies, and forwarding strategies, interact with each other at a global scale: such intricate dependencies heavily influence the performance of the whole system, thus making it hard to rely only on pure analytical models to accurately predict network key performance indicators (KPIs).

It follows that, especially as a first step to assess the performance of new complex protocols, event-driven simulation techniques represent an appealing alternative: indeed, simulative techniques make it simpler to describe algorithmic interactions between all the different entities with the desired level of fidelity. At the same time, large-scale simulations require massive computational resources (both CPU and memory), to the point that the very same fidelity of simulation results may be compromised by the intrinsic scalability limit of the technique itself.

To show why this may be the case, we provide in Fig. 1 an illustrative example. Whereas it is well known that Internet-scale catalogs approaches trillion objects [26], these scales are hardly achievable in simulation: as such, simulation-based studies typically resort to naive downsampling of the scenario under investigation, to comply with memory constraints and CPU time budgets. Using the mean hit ratio as KPI, Fig. 1 contrasts the performance of a 4-level binary tree where the Zipf exponent is varied as $\alpha \in \{0.8, 1.2\}$, and the ratio between catalog cardinality $M$ and cache size $C$ is kept constant to $C/M = 0.01\%$, while both $M$ and $C$ are jointly downscaled. Results in Fig. 1 clearly show that barely downsampling the simulated scenario by linearly reducing the size and cardinality of all the components does not preserve its original properties at all: the relative error between the smallest vs largest scenario is between 50% and 100%. This fact has profound implications: indeed, rather typically, crucial parameters of the scenario under investigation, such as the Zipf exponent $\alpha$ (or Mandelbrot Zipf plateau $q$), are measured over real Internet catalogs like YouTube [6] or the BitTorrent ecosystem [12]) which are not only growing, but already had a catalog in the order of hundred millions objects about 10 years ago [6]. Clearly, Fig. 1 also implies that merely applying catalog parameters inferred from large-scale measurements to
small-scale simulations does not make any sense, as it induces excessive distortion of the KPIs to make results of practical relevance.

Inspired by **hybrid models** proposed over the years in different domains (see Sec. IX), we argue that **grafting** components of stochastic modeling into simulative techniques can increase the overall scalability by orders of magnitude, without compromising the accuracy of the resulting technique, at the same time. Specifically, our methodology exploits a synergy between stochastic analysis of Least Recently Used (LRU) caches [7, 22] and Monte Carlo approaches based on Time-To-Live (TTL) caches [15, 9, 23, 25]. In particular, our intuition consists in using the characteristic time $T_C$ of Che’s approximation [7] for LRU caches as the TTL parameter in Monte Carlo simulations: as a consequence, the complexity is significantly reduced by simulating TTL-based caches in place of their more complex LRU counterpart, and by subsampling the original catalog in a way that preserves its key statistical properties. Given that $T_C$ is a complex scenario is not known a priori, our system uses a feedback loop to iteratively converge to the correct $T_C$ value, even when the initial guess is wrong by two orders of magnitude.

Developing this intuition further to build a fully integrated system, making the following contributions:

- we propose a novel hybrid methodology for the performance evaluation of cache networks, that reduces CPU time and memory usage by over two orders of magnitude, while limiting accuracy loss to less than 2%;
- we implement the methodology as an alternative simulation engine, so that users can seamlessly switch between event-driven vs ModelGraft simulation, on the very same scenario;
- we make the technique (and scenarios) available as open source in the latest release of ccnSim [1].

In the remainder of this paper, we first provide background information about the building blocks leveraged by our methodology (Sec. II). We next provide a succinct overview of our proposal (Sec. III) followed by an in-depth description of each component (Sec. IV–VI). We next validate the technique (Sec. VII) and showcase its performance in very large scenarios (Sec. VIII), before covering related work (Sec. IX), and concluding the paper (Sec. X).

II. MODELING INTUITION

In this section we first introduce background material on Che’s approximation [7] (Sec. II-A), and then formalize our intuition about the equivalence between LRU caches and (opportunistically configured) TTL-based caches [15, 9, 23, 25] (Sec. II-B), which constitutes a fundamental building block of our methodology.

A. Background

Che’s approximation [7], which emerged in the last few years as one of the most flexible models for cache networks, is essentially a mean-field approximation which greatly simplifies interactions between different contents inside a cache. In particular, for an LRU cache, Che’s approximation consists in confusing the cache eviction time $T_C(m)$ for content $m$ (i.e., the time since the last request after which object $m$ will be evicted from the cache, unless the object is not requested again in the meantime), with a constant characteristic time $T_C$, which is a property of the whole cache but does not depend on the content itself. As a consequence, content $m$ is considered to be in the cache at time $t$, if and only if, at least one request for it has arrived in the interval $(t - T_C, t)$. Supposing an Independent Request Model (IRM), with content catalog of size $M$, and request process for content $m$ to be Poisson of rate $\lambda_m$, the probability $p_{hit}(m)$ for content $m$ to be in a LRU cache at time $t$ can be expressed as:

$$p_{hit}(\lambda_m, T_C) = 1 - e^{-\lambda_m T_C}.$$

Since, by construction, cache size $C$ must satisfy:

$$C = E[C] = E \left[ \sum_m \mathbb{1}_{\{m \text{ in cache at } t\}} \right] = \sum_m p_{hit}(\lambda_m, T_C),$$

where $\mathbb{1}_{\{A\}}$ is the indicator function for event $A$, then the characteristic time $T_C$ can be computed by numerically inverting (2), which admits a single solution [7]. Finally, KPIs of the system, such as the cache hit probability $p_{hit}$, can be computed using the PASTA property as:

$$p_{hit} = E_\lambda[p_{hit}(\lambda_m, T_C)] = \sum_m \lambda_m p_{hit}(\lambda_m, T_C)/\sum_m \lambda_m.$$

Although apparently simple, a theoretical explanation for the accuracy of Che’s approximation under the IRM model has been provided only recently in [10], which shows that $T_C(m)$ converges to a constant independent of $m$ as the cache grows large, and extended in [19] to renewal request models. As far as a single cache is concerned, Che’s approximation was originally proposed for LRU caches [7], but it has been extended in more recent times to FIFO or Random replacement [10], or LRU caches with probabilistic insertion [22], possibly depending on complex cost functions [3].

As far as network of caches are concerned, however, further approximations are required as an alternative approach to the computationally and algorithmically challenging characterization of the miss stream at any node in the network [22, 25]. In arbitrary networks with shortest path [32] or more complex routing policies [33], it has been shown that inaccuracies can potentially cascade, with significant degradation of the accuracy with respect to simulation [34]. Finally, analytical approaches often assume stationary conditions, thus lacking in characterizing transient periods, although a model has been recently proposed only for a single cache [11]. All these reasons thus justify the quest for an hybrid approach, such as the one we propose in this work.

B. Intuition

Observe that, given the characteristic time $T_C$ under Che’s approximation, the dynamics of the different contents become completely decoupled. As a consequence, we can resort to simpler caching than LRU to simplify the analysis of complex and large cache networks. One such alternative is constituted by Time-to-Live (TTL) based caches [15, 9, 23, 25]: contents
Observation 1. We argue that, the dynamics of a LRU cache with characteristic time \( T_C \), fed by an IRM process associated to a catalog \( M \) with cardinality \( M \), and request rates \( \lambda_m \) drawn from a distribution \( \Lambda \), become indistinguishable from those of a TTL based cache with deterministic TTL parameter set equal to \( T_C \) operating on the same catalog:

\[
p_{\text{hit}}(\Lambda, T_C) = \mathbb{E}_\Lambda[1 - e^{-\sum \lambda_m T_C}] = \mathbb{E}_\Lambda[p_{\text{in}}(\lambda_m, T_C)] = p_{\text{hit}}^{\text{TTL}}(T_C)
\]

Specifically, (4) equals the average hit probability of the original LRU system to that of its TTL-based equivalent [9, 22]. Leveraging further on this intuition, we argue:

Observation 2. Large-scale LRU networks can be analyzed through a downscaled system with \( M' \ll M \), where each cache is replaced by its TTL equivalent, with TTL set to the characteristic time \( T_C \) of the original LRU cache. However, it is necessary to maintain the stochastic properties of the original catalog \( M \), while downsizing down. This, in turn, requires to average system performance over multiple MonteCarlo realizations, each lasting for a duration \( \delta T \), where rates \( \lambda_m' \) for individual objects in each realization \( M' \) of the downsized catalogs are drawn from \( \Lambda \). Thus, expanding (4):

\[
\mathbb{E}_\Lambda[p_{\text{in}}(\lambda_m', T_C)] = \sum_{\lambda_m} p(\lambda_m = \lambda_m')p_{\text{in}}(\lambda_m, T_C) = \frac{\sum_{\lambda_m} \lambda_m p_{\text{in}}(\lambda_m, T_C)}{\sum_{\lambda_m} \lambda_m}
\]

We expect, therefore, that decoupling the dynamics of different contents would allow to downscale the system, thus significantly reducing both memory and CPU complexity on the one hand, while still accurately representing complex interactions and correlations among different caches at the same time.

Observation 3. In particular, an alternative approach is to let \( \delta T \to 0 \), and a convenient approximation is to vary \( \lambda_m' \) at each new request, still satisfying (4).

The remainder of this paper illustrates, describes, and validates in greater details the methodology that is built upon these observations.

### III. ModelGraft Overview

The hybrid methodology presented herein, colloquially referred to as ModelGraft, performs MonteCarlo simulations of an opportune downscaled system, where LRU caches are replaced by their Che’s approximated version, implemented in practice as TTL based caches. Before delving into the details of the approach, it is worth to both placing it into a broader context, as well as illustrating at high level each of its building blocks.

We implement ModelGraft as a simulation engine available in the latest version of ccnSim [1]. As illustrated in Fig.2(a), starting from a unique scenario description, users can analyze the performance of cache networks via either an analytical model [22] (left), a classic event-driven simulation engine (right), or via the ModelGraft engine (middle). ModelGraft depends on a single additional parameter, namely the downscaling factor \( \Delta \), which can be automatically set, or it is anyway very easy to tune (according to guidelines in Sec.VI-B).

As introduced in Sec. II-B, ModelGraft requires in input, \( T_C \) values for each cache in the network. One option could be to bootstrap ModelGraft with informed guesses of \( T_C \) gathered via, e.g., analytical models (notice the \( T_C(\text{model}) \) switch in Fig.2(a)), which would, however, limit the appeal of the methodology. A more interesting approach, used by default in ModelGraft, is instead to start from uninformed guesses of \( T_C \) (notice the default wiring to the \( T_C(\text{guess}) \) switch in Fig.2(a)), and let the system iteratively correct the \( T_C \) value. In other words, ModelGraft is conceived to auto-regulate, so that by design it achieves accurate results even when the input \( T_C \) values, that the user does not even need to be aware of, largely differ from the correct ones.

Details of this iterative design are exposed in Fig.2(b), showing each of the blocks that are thoroughly described in the following sections. In a nutshell, ModelGraft starts with the configuration of the downsampling and sampling process (Sec. IV), before entering the MonteCarlo TTL-based (MC-TTL) simulation (Sec. V). During the MC-TTL phase, statistics
are computed after a transient period (Sec. V-A), where an adaptive steady-state monitor tracks and follows the dynamics of the simulated network in order to ensure that a steady-state regime is reached without imposing a fixed threshold (e.g., number of requests, simulation time, etc.) a priori (Sec. V-B). Once at steady-state, a downscaled number of requests are simulated within a MC-TTL cycle (Sec. V-C), at the end of which, the monitored metrics are provided as input to the self-stabilization block (Sec. VI); a consistency check decides whether to end the simulation (Sec. VI-A), or to go through a $T_C$ correction phase (Sec. VI-B), before starting a new simulation cycle.

IV. DOWNSCALING AND SAMPLING

A. Design

The ModelGraft workflow starts with a proper downsizing of the original scenario, controlled by the downsizing factor $\Delta \gg 1$: specifically, the equivalent TTL-based system has a target cache size $C' = C/\Delta$, a catalog comprising $M' = M/\Delta$ objects, and a target number of simulated requests at steady-state $R' = R/\Delta$. In order to avoid the pitfalls caused by a naive downscaling process (recall Fig. 1) we need to ensure that the downscaled catalog preserves the main features of the original one, like its popularity distribution. While our methodology is not restricted to a specific popularity law, in what follows we develop the case where object popularity follows a Zipfian probability distribution with exponent $\alpha$ – which is also the most interesting case from a practical viewpoint. Hence, we denote with $\Lambda$ the aggregate arrival rate of all objects in the catalog and with $\lambda_m = \Lambda n^{-\alpha} / \sum_{k=1}^{M'} k^{-\alpha}$ the rate for the $n$-th object in the original catalog.

The proposed approach, sketched in Fig. 3, consists in splitting the original catalog into a number of $M'$ bins having the same cardinality $\Delta$, i.e., $|\mathcal{M}_n| = \Delta$, where $\mathcal{M}_n$ refers to the $n$-th bin with $n \in [1, M']$. In ModelGraft, each bin of the original catalog is represented by a single “meta-content” in the downscaled system, i.e., the active catalog comprises $M'$ meta-contents. The key idea is to let each meta-content $n$ to be requested with an average request rate, $\bar{\lambda}_n$, which closely approximates the average request rate of the contents within the respective bin in the original catalog. More formally, for the $n$-th meta-content, with $n \in [1, M']$, it is required that:

$$\bar{\lambda}_n = \frac{1}{\Delta} \sum_{m=(n-1)\Delta+1}^{n\Delta} \lambda_m. \quad (6)$$

where the interval $[(n-1)\Delta+1, n\Delta]$ comprises contents of the original catalog that fall within the $n$-th bin. This design can be achieved by (i) considering $M'$ parallel request generators, i.e., one per each meta-content, each of which is identified by a fixed $n \in [1, M']$; (ii) considering any given meta-content $n$, varying its instantaneous request rate at each new request, so that its average complies with (6).

B. Implementation

It is easy to see that the simplest implementation of the above requirements boils down to bind the probability of the rate $\lambda'_n$ at which the $n$-th meta-content is requested $P(\lambda'_n = \lambda_m)$ with the popularity distribution of the $\Delta$ contents inside the respective bin $m \in [(n-1)\Delta + 1, n\Delta]$:

$$P(\lambda'_n = \lambda_m) = \frac{\lambda_m}{\sum_{j=(n-1)\Delta+1}^{n\Delta} \lambda_j} \approx \frac{1}{\Delta} \frac{\lambda_m}{\bar{\lambda}_n}. \quad (7)$$

While the above requirement (6) is met, a significant downside of this naive implementation is its space complexity. Indeed, since it is based on the classic inverse transform sampling, this approach would require to store $M'$ Cumulative Distribution Functions (CDFs) having each a size $\Delta$, with an overall memory allocation equal to $M'/\Delta = M$ elements, as in the original scenario. Given that $M$ is the dominant factor driving the overall memory occupancy, it is clear that such a simple implementation is not compatible with our goals.

We therefore resort to a better sampling technique called Rejection Inversion Sampling [14], which is an acceptance-rejection method that efficiently generates random variables from a monotone discrete distribution (in this case Zipf distribution) without allocating memory-expensive CDFs, and which is characterized by a $O(1)$ runtime complexity. Originally proposed in [14] for $\alpha > 1$, this technique has only very recently [2] extended to all non-negative exponents $\alpha > 0$. Recall now that power-laws (and hence Zipf distribution) exhibit a scale-independent, or self-similar, property according to which the scale exponent $\alpha$ is preserved independently of the level of observation. Hence, by means of rejection inversion sampling, we can consider a single interval $[1, \Delta]$ (i.e., with the same cardinality of one bin) from which extracting request rates, at each new request, from a Zipf distribution with exponent $\alpha$. Indeed, if the request generator associated to the $n$-th bin, with $n \in [1, M']$, needs to schedule the next request rate for the $n$-th meta-content, an integer $t \in [1, \Delta]$ is extracted with the aforementioned technique: the relative request rate is then, computed as $\lambda'_n = \lambda_{(n-1)\Delta+t}$, thus satisfying condition (6). Due to lack of space, we point the interested reader to our technical report [35] for an extensive justification.

V. MC-TTL SIMULATION

A. Transient

Once the scenario is properly downscaled, and initial uninformed guesses for $T_C$ are provided to ModelGraft, the warm-
up phase of the first MC-TTL simulation cycle is started. Given that the duration of the warm-up can be affected by many parameters (e.g., the presence of a conservative cache decision policy, like Leave Copy Probabilistically (LCP) [4], where the reduced content acceptance with respect to Leave Copy Everywhere (LCE) yields much longer transient durations), ModelGraft automatically adapts the duration of the transient period, in order to guarantee the statistical relevance of the monitored KPIs.

B. Steady-state monitor

The convergence of a single node \( i \) is monitored via batch means on the Coefficient of Variation (CV) of the measured hit ratio, \( \bar{p}_{hit}(i) \). In particular, denoting with \( p_{hit}(j, i) \) the \( j \)-th sample of the measured hit ratio of node \( i \), node \( i \) is considered to enter a steady-state regime when:

\[
CV_i = \sqrt{\frac{1}{W} \sum_{j=1}^{W} (p_{hit}(j, i) - \bar{p}_{hit}(i))^2} \leq \varepsilon_{CV}, \quad (8)
\]

where \( W \) is the size of the sample window, and \( \varepsilon_{CV} \) is a user-defined convergence threshold. To avoid biases, new samples are collected only if (i) the cache has received a non-null number of requests since the last sample, and (ii) its state has changed, i.e., at least a new content has been admitted in the cache since the last sample. To exemplify why this is important, consider that with a LCP\( (p) \) cache decision policy, where new contents are probabilistically admitted in the cache, the reception of a request is correlated with the subsequent caching of the fetched content only in 1 out of 1/\( p \) cases.

At network level, denoting with \( N \) the total number of nodes in the network, and given a tunable parameter \( Y \in (0, 1] \), we consider the whole system to enter steady-state when:

\[
CV \leq \varepsilon_{CV}, \quad \forall i \in Y, \quad (9)
\]

where \( Y = \{Y, N\} \) is the set of the first \( YN \) nodes satisfying condition (8). The rationale behind this choice is to avoid unnecessarily slow down the convergence of the whole network by requiring condition (8) to be satisfied by all nodes: indeed, due to particular routing protocols and/or topologies, there are nodes that have low traffic loads (hence, long convergence time), and, at the same time, a marginal weight in network KPIs. Although further information about a sensitivity analysis on \( Y \) can be found in [35], we anticipate that its typical settings lay in the \( Y \in [0.75, 1) \) interval.

C. Simulation cycle

For the original system, the duration of a simulation cycle \( T \) at steady-state is computed as \( T = R/(\Delta N_C) \), where \( R \) is the target number of requests, \( \Lambda = \sum_{r \in M} \lambda_i \) is the aggregate request rate per client, and \( N_C \) the number of clients. In ModelGraft simulations, instead, the total request rate per each client is \( \tilde{\Lambda} = \sum_{r \in M'} \lambda'_i \approx \Lambda/\Delta \). Keeping the simulated time \( T' = T \) constant, it follows that the number of simulated events per each cycle of a MC-TTL simulation is \( R' = R/\Delta \) – with an expected significant reduction of the CPU time required to simulate a cycle.

VI. Self-Stabilization

As described in Sec. III, one of the desirable properties of our hybrid methodology is a self-contained design that allows to simulate large scale networks even in the absence of reliable estimates of characteristic times \( T_C \). This is achieved through a feedback loop, which ensures that our methodology self-stabilizes, as a result of the combined action of two elements: a measurement step, referred as consistency check, and a controller action, where inaccurate \( T_C \) values are corrected at each iteration.

A. Consistency check

The consistency check is based on the observation that with a downscaling factor \( \Delta \), and when \( T_C = T_C' \), a TTL cache stores, on average, \( C' = C/\Delta \) contents at steady-state. Indeed, adapting (2) to the downscaled scenario (i.e., \( M' = M/\Delta \)), we have:

\[
E[C'] = \sum_{n=1}^{M'} p_{in}(\lambda'_n, T_C') = C/\Delta. \quad (10)
\]

However, unlike LRU caches that have a fixed size (and the oldest content is selected for eviction in LRU fashion), TTL caches have unbounded size (as the old contents remains soft-state in the cache for a fixed TTL, but are not otherwise evicted due to cache size limit). Considering that there exists a strong correlation between the eviction time \( T_C \) and the number of cached contents, it follows that we can consider the measured cache size \( \tilde{C} \) as the controlled variable.

In particular, for each TTL cache we maintain an online average of the number of stored contents as:

\[
\tilde{C}_i'^{(z)}(k+1) = \tilde{C}_i'^{(z)}(k) t(k) + B_i'^{(z)}(k+1) \frac{[t(k+1) - t(k)]}{t(k+1)}, \quad (11)
\]

where \( \tilde{C}_i'^{(z)}(k) \) is the online average of the cache size of the \( i \)-th node at \( k \)-th measurement time during \( z \)-th simulation cycle, and \( B_i'^{(z)}(k+1) \) is the actual number of contents stored inside the TTL cache of the \( i \)-th node at the \( (k+1) \)-th measurement time during the \( z \)-th simulation cycle. Samples for the online average are clocked with \( miss \) events and collected with a probability 1/10, so that they are geometrically spaced.

At the end of each MC-TTL simulation cycle (i.e., after the simulation of \( R' \) requests), a consistency check evaluates the accuracy of the measured cache size \( \tilde{C}_i'^{(z)} \), with respect to the target cache \( C' \), by using the following expression:

\[
\frac{1}{YN} \sum_{i \in Y} \left| \frac{\tilde{C}_i'^{(z)}(k_{\text{end}}) - C'}{C'} \right| \leq \varepsilon_{C}, \quad (12)
\]

where \( \tilde{C}_i'^{(z)}(k_{\text{end}}) \) is the online average of the measured cache size of \( i \)-th node at the end of the \( z \)-th simulation cycle, \( C' \) is the target cache, supposed to be equal for all the nodes without loss of generality, and \( \varepsilon_{C} \) is a user-defined
consistency threshold. For coherence, measures are taken on those $|\mathcal{V}| = K N$ nodes that have been marked as stable in Sec. V-B. If condition (12) is satisfied, the MC-TTL simulation ends, otherwise a new MC-TTL cycle needs to be started: $T_C$ values are corrected (as in the next subsection), all the caches are flushed, and the online average measures are reset.

B. $T_C$ correction

The direct correlation that exists between the target cache size $C^\prime$ and the actual one $C$ is expressed through equations (11)-(2)-(4)-(12), represents the basis for the controller action. Intuitively, there exists a linear proportionality between $C^\prime$ and $T_C$, i.e., the average number of elements $C^\prime$ stored in a TTL cache with TTL=$T_C$ grows as $T_C$ grows.

Therefore, if the consistency checking block reveals that the measured cache size $C$ of a particular node is smaller than its target cache $C < C^\prime$, it means that the respective $T_C$ value provided as input is actually smaller than the actual one, and that it needs to be increased in the next step. Vice versa, for a $C > C^\prime$, the $T_C$ of the correspondent node should be decreased.

As a consequence of the linear relationship, we employ a proportional (P) controller to compensate for $T_C$ inaccuracies. That is, if condition (12) is not satisfied at the end of $\zeta$ simulation cycle, the $T_C$ values are corrected, before starting the next one, as:

$$T_{C,i}^{(z+1)} = T_{C,i}^{(z)} \left( \frac{C^\prime}{C^{\prime}(z)} \right)_i,$$

where $T_{C,i}^{(z)}$ is the TTL value assigned to the $i$-th node during the $z$-th simulation cycle. In practice (13) guarantees a fast convergence towards the right $T_C$ values (see Sec. VII-C), avoiding at the same time any divergence of the control action (provided that measures on $C^\prime$ are taken at steady-state). This allows ModelGraft to guarantee considerable gains, even when multiple simulation cycles are necessary, due to significantly inaccurate input $T_C$ values.

There is an important condition worth highlighting: i.e., the controller needs to react on measurable quantities (which happens whenever $C^\prime / C^{\prime}(z)$ is reliably measured), as opposite to noisy measures (which happens whenever the numerator $C^\prime$ is too small). In particular, this translates into a very simple practical guideline, as it introduces a lower bound to the target cache size of the downscaled system, i.e., $C^\prime = C/\Delta \geq 10$, practically upper bounding the maximum downsampling factor to $\Delta \leq C/10$. For a detailed sensitivity analysis regarding all the parameters involved in the ModelGraft design, we refer the reader to [35].

VII. VALIDATION

Now we validate the ModelGraft engine against classic event-driven simulation, both (i) in the case where we provide accurate $T_C$ values as input (to assess the gain of a single MC-TTL cycle), as well as (ii) in the case where we provide completely wrong $T_C$ guesses (to assess the self-stabilization capabilities). All the results presented in this section have been obtained by executing both event-driven and ModelGraft simulations on the same commodity hardware, i.e., an Intel Xeon E5-1620, 3.60GHz, with 32GB of RAM.

A. Very-large Scale Scenario

To assess ModelGraft accuracy, we consider the largest scenario we can investigate via event-driven simulation gathered via ccnSim, already shown to be among the most scalable ICN software tools [34]. To stretch the boundaries reachable by event-driven simulation, we integrate the rejection inversion sampling – to the best of our knowledge, this represents the first performance evaluation of ICN networks with content catalogs in the order of billions.

The validation scenario represents an ICN access tree network [24], where the topology is a $N=15$-nodes 4-level binary tree depicted in Fig. 4(a). A single repository, connected to the root node, stores a $M=10^9$ object catalog, where objects have Zipf popularity distribution with exponent $\alpha = 1$. An overall $R = 10^9$ requests, following an Poisson IRM process, are injected at each leaf nodes, at a rate of $\Lambda = 20$req/s per leaf.

The cache size of each node is fixed at $C = 10^8$, resulting in a cache to catalog ratio of $C/M = 0.01\%$. Three different cache decision policies are considered for the comparison: (i) LCE, where fetched contents are always cached in every traversed node; (ii) LCP(1/10), that probabilistically admits content in the cache (configured so that one out of ten fetched contents is cached on average); (iii) 2-LRU [16, 22], where cache pollution is reduced by using an additional cache in front of the main one, with the purpose of caching only the names of requested contents: the fetched contents will be stored in the main cache only in case of a hit event in the first cache.

According to our rule of thumb $C^\prime = C/\Delta \geq 10$, the maximum downsampling factor is $\Delta = 10^5$. Additionally, equations (9) and (12) are computed considering $Y = 0.75$, $\varepsilon_{CV} = 5 \cdot 10^{-3}$, and $\varepsilon_C = 0.1$: in other words, we test convergence of 75% of the caches in the network, by requiring the coefficient of variation of the hit rate to be below $5 \cdot 10^{-3}$, and iterate MC-TTL simulations until the measured average cache size of those nodes is within 10% of the expected size $C^\prime = C/\Delta$. It is worth stressing that while we cannot report a thorough sensitivity of the above parameters in this paper due to size limits, an extensive account of their (limited) impact is available to the interested reader at [35].
To illustrate values taken by the characteristic time $T_C$ in this large scale scenario, and its dependency on the cache decision policy, Tab. I reports $T_C$ measured by event-driven simulation at different depths of the tree. It clearly appears that the cache decision policy has the largest impact on the $T_C$ values (even larger than topological position in the network). Specifically, the more conservative the policy (e.g., LCP or 2-LRU), the larger the $T_C$ values, which can vary up to one order of magnitude among different policies. Intuitively, with caching policies that are more conservative in admitting new contents, (popular) stored contents are cached for longer time (which increases the overall hit probability).

To validate the ModelGraft workflow, we start with a single MC-TTL cycle: to do so, we feed ModelGraft with accurate $T_C$ estimates, so that no iteration is necessary. Results are reported in Tab. II, where mean values of 10 different runs for three KPIs are reported: mean hit ratio $p_{hit}$, CPU time, and memory occupancy (for which relative gains are also highlighted). Results are noteworthy in that (i) the discrepancy between $p_{hit}$ measured by event-driven vs ModelGraft remains always under 2%, (ii) ModelGraft achieves significant gains, of about two orders of magnitude, for both CPU time and memory occupancy. Additionally, it is interesting to notice that there is neither trace of the (iii) accuracy/speed trade-off, as one could typically expect [27], nor trace of (iv) memory/CPU tradeoff [13], i.e., cases where an algorithm either trades increased space (e.g., cached results) for decreased execution time (i.e., avoid computation), or vice versa.

### TABLE I

<table>
<thead>
<tr>
<th>Level</th>
<th>$T_C$ values [s]</th>
<th>LCE</th>
<th>LCP(1/10)</th>
<th>2-LRU (Name/Main)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Root)</td>
<td>$16.7 \cdot 10^3$</td>
<td>$16.7 \cdot 10^4$</td>
<td>$20.0 \cdot 10^3/76.4 \cdot 10^4$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$32.5 \cdot 10^5$</td>
<td>$31.4 \cdot 10^3$</td>
<td>$38.1 \cdot 10^3/12.5 \cdot 10^6$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$63.0 \cdot 10^4$</td>
<td>$56.9 \cdot 10^3$</td>
<td>$71.9 \cdot 10^3/20.4 \cdot 10^6$</td>
<td></td>
</tr>
<tr>
<td>3 (Leaves)</td>
<td>$1.1 \cdot 10^4$</td>
<td>$88.3 \cdot 10^3$</td>
<td>$11.1 \cdot 10^4/22.6 \cdot 10^6$</td>
<td></td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>Cache Decision Policy</th>
<th>Technique</th>
<th>$p_{hit}$</th>
<th>CPU Gain</th>
<th>Mem [MB]</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCE</td>
<td>Simulation</td>
<td>33.2%</td>
<td>11.4 h</td>
<td>6371</td>
<td>168s</td>
</tr>
<tr>
<td>ModelGraft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCP(1/10)</td>
<td>Simulation</td>
<td>35.4%</td>
<td>7.3 h</td>
<td>6404</td>
<td>168s</td>
</tr>
<tr>
<td>ModelGraft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-LRU</td>
<td>Simulation</td>
<td>37.0%</td>
<td>10.8 h</td>
<td>8894</td>
<td>234s</td>
</tr>
<tr>
<td>ModelGraft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### VIII. RESULTS

We finally employ the validated ModelGraft engine to venture scenarios that are prohibitively complex for classic event-driven simulation, due to both CPU and memory limitations.

#### A. Internet-scale Scenarios

We now aim at investigating Internet-scale scenarios, whose content catalogs are estimated [26] to be in the order of $O(10^{10}) - O(10^{15})$, i.e., two orders of magnitude larger than those considered in the previous section.

We consider two scenarios: the ICN-like one, depicted in Fig. 4(a), which models a 4-level binary tree with a single repository connected to the root, serving a catalog with cardinality $M = 10^{10}$. Cache size is $C = 10^6$, which limits the maximum downscaling to $\Delta = 10^2$. The second scenario, depicted in Fig. 4(b), models a more complex CDN network, where three repositories, serving a catalog with cardinality $M = 10^{11}$, are connected to backbone nodes interconnected as the classic Abilene network, and where an access tree is
further attached to each backbone node. In this scenario, we let the cache size be \( C = 10^7 \), which allows to increase the downscaling to \( \Delta = 10^6 \).

As before, we set \( Y = 0.75 \), \( \varepsilon_{CV} = 5 \cdot 10^{-3} \), and \( \varepsilon_C = 0.1 \) and run experiments on the same Intel Xeon E5-1620, 3.60GHz, with 32GB of RAM. Clearly, we cannot instrument event-driven simulations at such large scale due to both physical memory limits (a hard constraint), as well as time budget (a soft constraint). At the same time, we can estimate the expected memory occupancy and CPU times by fitting and cross-validating several scenarios with simple models. Despite related to the very specific implementation of the ccnSim simulator, estimates are, however, useful to project ModelGraft gains. While a thorough analysis is provided in [35] due to lack of space, general trends are discussed herein: on the one hand, the event-driven approach of ccnSim has a \( O(M) + O(NC) \) memory cost (being \( N \) the number of caches in the network), meaning that the allocation of memory space is mostly influenced by the cardinality of the catalog, rather than by the cache size (being \( M \gg C \)). On the other hand, the CPU time varies linearly with the number of total requests, i.e., \( O(R) \).

**B. Projected gains**

Results for the Internet-scale scenarios are shown in Tab. III, which reports the mean values of \( p_{init} \), CPU time, and memory usage, along with the number of MC-TTL cycles gathered via ModelGraft, and estimates of CPU time and memory occupancy for the classic event-driven approach.

**TABLE III**

<table>
<thead>
<tr>
<th>Topology</th>
<th>Parameters</th>
<th>Technique</th>
<th>( p_{init} )</th>
<th>CPU / Memory Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-level binary tree ((N = 15))</td>
<td>( M = 10^6 ) ( R = 10^6 ) ( C = 10^6 ) ( \Delta = 10^6 ) ( Y = 0.75 )</td>
<td>Simulation (estimate)</td>
<td>n.a.</td>
<td>4.5 days</td>
</tr>
<tr>
<td>CDN-like ((N = 67))</td>
<td>( M = 10^6 ) ( R = 10^6 ) ( C = 10^6 ) ( \Delta = 10^6 ) ( Y = 0.75 )</td>
<td>ModelGraft 34.0%</td>
<td>1 cycle</td>
<td>1 cycle</td>
</tr>
</tbody>
</table>

Consider the ICN-like scenario first. Notice that, whereas memory requirements for event-driven simulation increased with respect to scenarios in the previous section (due to the need of mapping seed copies with respective repositories), memory usage in ModelGraft increases only slightly (since the mapping scales as \( M' \)); consequently, memory gain increases significantly. Second, notice that CPU gains maintain to \( 270 \times \), which happens since in this case our initial guess of \( T_C \) was accurate enough to let ModelGraft converge after one cycle.

For the CDN-like scenario, instead, memory gain increases by one further order of magnitude, in reason of the larger downscaling factor \( \Delta = 10^6 \). Conversely, CPU time gain reduces due to the larger number of cycles needed to end the simulation: specifically, given that our initial \( T_C \) guess was not accurate enough, ModelGraft took three cycles to converge, reducing gains of the techniques to a still very significant \( 96 \times \).

**IX. RELATED WORK**

Hybrid approaches have been considered to make it practical to study large scale networks even with commodity hardware, and at a reasonable time scale. The concept of inferring key aspects of large systems from the study of equivalent and scaled-down versions has been adopted in several domains, from cosmology and biology, to the more closer communication networks, in the forms of large scale IP networks [30, 21], wireless sensor networks [20], and control theory [5].

What presented in this paper finds a strict correspondence with the work in [30], where large IP networks are scaled to reduce the computational requirements of simulations and simplify performance prediction. The idea consists in feeding a suitable scaled version of the system with a sample of the input traffic, while changing the scaling rule according to the type of TCP/UDP flows traversing the network.

TCP networks are also considered in [21], where the authors propose a scalable model which is easily comparable with discrete event simulators due to its time-stepped nature. In particular, by refining a known analytical model [29] based on ordinary differential equations, they show that their approach yields accurate results with respect to those of the original networks, and that, at the same time, it is able to speedup the completion time of orders of magnitude with respect to packet level and discrete events simulators like ns.
This work is the first to apply these concepts to the study of cache networks. Clearly, caching dynamics are intrinsically different from system-level aspects of wireless sensor networks [20], or the steady state throughput of TCP/IP networks [30, 21]. Our methodology is not only novel, but also practical, as it is fully integrated in open-source tools [1].

X. CONCLUSION

This work proposes ModelGraft, an innovative hybrid methodology addressing the issue of performance evaluation of large-scale cache networks. The methodology grants elements of stochastic analysis to MonteCarlo simulation approaches, retaining benefits of both methodology classes. Indeed, ModelGraft inherits simulation flexibility, in that it can address complex scenarios (e.g., topology, cache replacement, decision policy, etc.). Additionally, ModelGraft is implemented as a simulation engine to retain simulation simplicity: given its self-stabilization capability, ModelGraft execution is decoupled from the availability of accurate input $T_C$ values, which is completely transparent to the users. Results presented in this paper finally confirm both the accuracy and the high scalability of the ModelGraft approach: CPU time and memory usage are reduced by (at least) two orders of magnitude with respect to the classical event-driven approach, while accuracy remain within a 2% band.

ModelGraft features, like decoupled dynamics of content requests and the use of TTL caches, pave the way for a further improvement, which we keep as future work: by resorting to a map-reduce approach, different cores could simulate the request process for different parts of the downscaled catalog (i.e., bins can be assigned in order to equalize to load of each core); then, once a consistency check is needed at the end of a MC-TTL cycle, measurements about the actual cache occupancy, coming from all the cores, can be linearly aggregated (i.e., by simply summing them) in order to check for (12). In case a further cycle would be needed, the $T_C$ would be corrected, and the different cores will restart to simulate their part of the catalog. Since there would be no need of message passing between the cores during the MC-TTL simulation, this approach could remarkably reduce the CPU time, especially if projected in a multi-core cluster environment.

ACKNOWLEDGMENTS

This work has been carried out at LINCS (http://www.lincs.fr). This work benefited from support of NewNet@Paris, Cisco’s Chair “NETWORKS FOR THE FUTURE” at Telecom ParisTech (http://newnet.telecom-paristech.fr). Any opinion, findings or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of partners of the Chair.

REFERENCES

Abstract—The growing popularity of mobile multimedia content and the increase of wireless access bitrates are straining backhaul capacity in mobile networks. A cost-effective solution to reduce the strain, enabled by emerging all-IP 4G and 5G mobile backhaul architectures, could be in-network caching of popular content during times of peak demand. In this paper we formulate the problem of content caching in a mobile backhaul as a binary integer programming problem, and we propose a 2-approximation algorithm for the problem. The 2-approximation requires full information about the network topology and the link costs, as well as about the content demands at the different caches, we thus propose two distributed algorithms that are based on limited information on the content demands. We show that the distributed algorithms terminate in a finite number of steps, and we provide analytical results on their approximation ratios. We use simulations to evaluate the proposed algorithms in terms of the achieved approximation ratio and computational complexity on realistic mobile backhaul topologies.

I. INTRODUCTION

The penetration of high speed mobile access technologies, such as HSDPA and LTE, together with the proliferation of powerful handheld devices has stimulated a rapid increase of user demand for mobile multimedia content in recent years. The traffic growth is predicted to continue in coming years, with an estimated 10-fold increase in mobile data traffic in 5 years and an increasing peak-to-average traffic ratio, and puts significant strain on mobile backhaul capacity.

Recent measurement studies of mobile data traffic indicate that caching could be an effective means of decreasing the mobile backhaul bandwidth requirements: caching could reduce the bandwidth demand by up to 95% during peak hours and could at the same time reduce content delivery time by a factor of three [1]. At the same time, mobile traffic is dominated by downloads; up to 75% of daily traffic load comes from download traffic, and the demand shows significant diurnal fluctuations with low loads during early morning hours [2].

While tunelling imposed by previous 3GPP standards made backhaul in-network caching technically challenging, allowing only caches at the network edge, in emerging all-IP mobile backhaul architectures the caches could be co-located with every switch and could implement cooperative caching policies throughout the backhaul. Since fairly accurate content popularity predictions can be obtained for Web and video content [3], [4], the most popular contents could be downloaded into the caches of the mobile backhaul in the early morning hours when the load is relatively low, thereby alleviating the traffic demand during peak hours.

Given predicted content popularities, a fundamental problem in-network caching in a mobile backhaul is to find efficient content placement algorithms that take into consideration the characteristics of mobile backhaul topologies and of mobile data traffic. The algorithms should achieve close to optimal bandwidth cost savings and should have low computational complexity. Furthermore, they should require as little information as possible, e.g., about content popularities and network topology, in order to allow fully distributed operation and scaling to large topologies with small communication overhead. While previous works proposed centralized and distributed content placement algorithms for two-level hierarchical topologies [5], general topologies with an ultrametric [6], and topologies in a metric space [7], efficient distributed algorithms based on limited topological information have received little attention.

In this paper we formulate the problem of content placement in a mobile backhaul based on predicted demands as a 0-1 integer programming problem. We show that a 2-approximation to the problem can be obtained using a distributed greedy algorithm when global information is available, and propose two computationally simple distributed algorithms that do not require global information. We evaluate the algorithms through extensive simulations on various network topologies. Our results show that information about object demands at descendants is not sufficient for achieving good performance, but the proposed h-Push Down algorithm achieves consistently good performance based on a limited amount of information about object placements.

The rest of the paper is organized as follows. Section II describes the system model and provides the problem formulation. Section III describes the 2-approximation algorithm based on global information, and Section IV describes the distributed algorithms based on limited information. Section V shows performance results based on simulations. Section VI discusses related work and Section VII concludes the paper.

II. SYSTEM MODEL AND PROBLEM FORMULATION

We consider a typical mobile backhaul, and model its active topology by a symmetric acyclic directed graph $G(N, E)$, where the vertices $N$ are routers that connect cell sites and may
aggregate traffic from other routers (and thus cell cites), and for every connected pair of nodes \(i, j \in \mathcal{N}\) there exist edges \((i, j) \in E\) and \((j, i) \in E\). Observe that since \(\mathcal{G}\) is connected and acyclic, \(\mathcal{G}\) is a tree. We denote by \(\mathcal{L}\) the set of leaf nodes in \(\mathcal{G}\), by \(\mathcal{I}\) the set of internal nodes and by \(\mathcal{N}_0\) the root node, i.e., \(\mathcal{N} = \mathcal{L} \cup \mathcal{I} \cup \mathcal{N}_0\). We denote the unique simple path from node \(i\) to node \(j\) by \(P_{i,j} = \{(i, v_1), (v_1, v_2), \ldots, (v_{|P_{i,j}|}, j)\}\), and we denote by \(|P_{i,j}|\) the number of edges in path \(P_{i,j}\).

Observe that \(|P_{i,j}| = |P_{j,i}|\). We define the level \(l(i)\) of node \(i \in \mathcal{N}\) as the number of edges from node \(i\) to the tree’s root node \(i_0\) in the unique path from \(i\) to \(i_0\), i.e., \(l(i) = |P_{i,i_0}|\). We denote the children of node \(i \in \mathcal{N}\) by \(C(i) \triangleq \{j | (i,j) \in E \land l(j) > l(i)\}\) and the parent of node \(i\) by \(P(i)\), where \(P(i) \in \mathcal{N}\) such that \(i \in \mathcal{C}(P(i))\). We denote by \(P^0(i)\) the \(i^0\)-ancestor of node \(i\), e.g., \(P^0(i) = P(P(P(\cdots P(i)))))\). By definition \(P^0(i)\) is \(i\). We refer to an edge \((i,j)\) as the downlink direction if \(j \in C(i)\) and as the uplink direction if \(i \in C(j)\).

We say that two nodes are siblings if they have the same parent, and define the sibling set \(S(i) \triangleq \{j | P(j) = P(i) \land i \neq j\}\). We denote the descendants of node \(i\) by \(D(i) \triangleq \{j | l(j) > l(i) \land LCA(i,j) = i\}\), where \(LCA(i,j)\) denotes the lowest common ancestor of nodes \(i\) and \(j\), furthermore we use the notation \(G_i(N_i,E_i)\) for the subgraph induced by \(N_i = \{i\} \cup D(i)\). rooted in \(i\).

A. Objects, Demand and Storage

We denote the set of objects requested by mobile nodes by \(\mathcal{O}\). We follow common practice and consider that every object has unit size \([8],[9]\), which is a reasonable simplification if content is divisible into unit-sized chunks. We denote the average request rate (demand) predicted for the peak hours for object \(o \in \mathcal{O}\) at the cell site connected to node \(i\) by \(w_i^o\).

Every node \(i \in \mathcal{N}\) is equipped with a cache, and we denote the size of the cache at node \(i\) by \(A_i \in \mathcal{O}\). We use the shorthand notation \(A_V \triangleq (A_i)_{i \in V}\), where \(V \subseteq \mathcal{N}\), and \(A_{V'} \triangleq (A_i)_{i \in \mathcal{N} \setminus \{V\}}\). We denote by \(\mathcal{A}_i\) the set of object placements that satisfy the storage constraint at node \(i\), i.e. \(\mathcal{A}_i = \{A_i \in 2^\mathcal{O} | |A_i| \leq K_i\}\), where \(2^\mathcal{O}\) is the powerset of \(\mathcal{O}\). Finally, we denote the set of objects stored at node \(i\) and at its descendants by \(R_i(A) = A_i \cup \bigcup_{j \in D(i)} R_j(A)\). Figure 1 shows an example topology with a maximum level of 2, illustrating some of the commonly used notation.

B. Cost model

We denote the unit cost of using edge \((i,j)\) by \(c_{i,j}\). Since during peak hours most of the traffic in a mobile backhaul is flowing downlink (serving users’ requests for content) \([1],[2]\), we consider that uplink edges have zero unit cost, i.e., \(c_{i,j}P(i,j) = 0\). Without loss of generality, the cost of downlink edges is \(c_{D}(i,j) > 0\). We consider that edge costs are additive, i.e., if a request for object \(o\) arrives at node \(i\) and is served from node \(j\) then the unit cost is \(d_{i,j} = \sum_{(i,w) \in P_{i,j}} c_{v,w}\). We call \(d_{i,j}\) the **distance** from node \(j\) to node \(i\). Note that the terms \(c_{v,w}\) are zero if they correspond to an uplink, i.e., if \(w = P(v)\). Furthermore, observe that in general \(d_{j,i} \neq d_{i,j}\), thus distance is not symmetric (hence it is a hemimetric).

A request for object \(o\) generated by a mobile user connected to the cell site at node \(i \in \mathcal{N}\) is served locally if \(o \in A_i\). Otherwise, if node \(i\) has a descendant \(j \in D(i)\) for which \(o \in A_j\), the node forwards the request to the closest such descendant. Otherwise, node \(i\) forwards the request to its parent \(P(i)\), which follows the same algorithm for serving the request. If an object \(o\) is not stored in any node (i.e., \(o \notin \mathcal{A}_N\)) then it needs to be retrieved through the Backbone via the root node \(n_0\) at a unit cost of \(c_{0}\).

Given a placement \(A = (A_i)_{i \in \mathcal{N}}\) we can define the unit cost to serve a request for object \(o\) at node \(i\) as

\[
d_i(o,A) = \begin{cases} 
  d_{i,j} & \text{if } o \in A_j \\
  d_{i,n_0} + c_0 & \text{if } o \notin A_N 
\end{cases}
\]

which together with the demand \(w_i^o\) determines the cost incurred by node \(i\) as

\[
C_i(A) = \sum_{o \in \mathcal{O}} C_i^o(A) = \sum_{o \in \mathcal{O}} w_i^od_i(o,A).
\]

Finally, we define the total cost \(C(A) = \sum_{i \in \mathcal{N}} C_i(A)\).

C. Problem formulation

Motivated by minimizing the congestion in the mobile backhaul during peak hours, our objective is to find a placement that minimizes the total cost \(C(A)\). We refer to this as the mobile backhaul content placement problem (MBCP), which can be formulated as finding \(A = \arg \min_{A \in \mathcal{A}} C(A)\).

It is easy to see that the MBCP problem can be formulated as the following 0 – 1 integer linear program

\[
\min \sum_{i \in \mathcal{N}_0 \in \mathcal{O}} \sum_{j \in \mathcal{N}_i \neq i} w_i^o(d_{i,j}x_{i,j,o} + (d_{i,n_0} + c_0)x_{i,-1,o}) \quad \text{s.t.} \quad \sum_{o \in \mathcal{O}} x_{i,o} \leq K_i, \quad \forall i \in \mathcal{N}, \quad x_{i,j,o} \leq x_{j,o}, \quad \forall i,j \in \mathcal{N}, \quad o \in \mathcal{O} \quad \sum_{j \in \mathcal{N}_i \neq i} x_{i,j,o} + x_{i,-1,o} \geq 1, \quad \forall i \in \mathcal{N}, \quad o \in \mathcal{O} \quad x_{i,o}, x_{i,j,o}, x_{i,-1,o} \in \{0,1\}
\]

where \(x_{i,o}\) indicates whether object \(o\) is in the storage of node \(i\) (i.e. \(x_{i,o} = 1 \Leftrightarrow o \in A_i\)), \(x_{i,j,o}\) indicates whether a request for object \(o\) at node \(i\) is served from node \(j\), and \(x_{i,-1,o}\) indicates whether object \(o\) is retrieved from the Backbone, i.e., the level of the Backbone is indicated with \(-1\).

It can be shown that for 4 or more nodes the constraint matrix is not totally unimodular and solving the MBCP
would be computationally infeasible already for moderate sized instances of the problem. We are thus interested in finding computationally feasible, scalable distributed algorithms to approximate the solution.

III. DISTRIBUTED 2-APPROXIMATION ALGORITHM BASED ON GLOBAL INFORMATION

In what follows we show that if global information is available about the object demands and placements at every node of the network, then it is possible to obtain a 2-approximation to the optimal solution using the Depth First Greedy (DFG) algorithm. The DFG algorithm is based on a depth-first traversal of the graph $\mathcal{G}$, i.e., an ordering $i_1, \ldots, i_{|N|}$ of the vertices in $N$, and can be executed by the nodes in an iterative (distributed) manner. The algorithm starts with an empty allocation ($A_i = \emptyset$); at iteration $1 \leq k \leq |N|$ node $i_k$ populates its cache with $K_{i_k}$ objects, one at a time, that provide the highest global cost saving. The DFG algorithm is shown in Figure 2.

![DFG Algorithm](image)

**Theorem 1.** The DFG algorithm is a 2-approximation algorithm for the MBCP problem in terms of cost saving, i.e., $\frac{C(\emptyset) - C(A)}{C(\emptyset)} \leq 2$.

Before we prove the theorem we introduce some definitions and previous results.

**Definition 1.** Let $E$ be a finite set and let $\mathcal{F}$ be a collection of subsets of $E$. The pair $(E, \mathcal{F})$ is a partition matroid if $E = \bigcup_{i=1}^{l} E_i$ is the disjoint union of $l$ sets, $l_1, \ldots, l_k$ are positive integers and $\mathcal{F} = \{F | F = \bigcup_{i=1}^{l} F_i \text{ s.t. } F_i \subseteq E_i, |F_i| \leq l_i, i = 1, \ldots, k \}$.

**Definition 2.** Let $E$ be a finite set, and $f : 2^E \rightarrow \mathbb{R}$ a real valued function on subsets of $E$. Then $f$ is submodular if for every $A, B \in E$

$$f(A \cup B) + f(A \cap B) \leq f(A) + f(B).$$

Let us now recall a fundamental result about the maximization of submodular functions over partition matroids.

**Lemma 1.** [10] Let $\mathcal{F}$ be a partition matroid over a set $E$, and $f : \mathcal{F} \rightarrow \mathbb{R}$ be a non-decreasing submodular function with $f(\emptyset) = 0$. Then the DFG algorithm achieves a 2-approximation of $\max_{F \in \mathcal{F}} f(F)$.

In what follows we show that MBCP can be formulated as the maximization of a non-decreasing submodular function over a partition matroid. Let us define for every object $o \in O$ one fictitious object $(o, i)$ per node $i \in N'$, i.e., $(o, i) \in O \times N$. The set of fictitious objects that can be assigned to node $i$ is then $\mathcal{E}_i = \{(o, i) | o \in O\}$ and we define the set $\bar{\mathcal{E}} = \bigcup_{i \in N'} \mathcal{E}_i$. We denote by $\mathcal{A}$ the family of subsets of $\bar{\mathcal{E}}$, defined as $\mathcal{A} = \times_{i \in N'} \mathcal{N}_i$, where $\mathcal{N}_i \subseteq E_i$, $|\mathcal{N}_i| \leq K_i$, is the set of object placements that satisfy the storage capacity constraint at node $i$, as defined in Section II-A.

**Proposition 2.** The pair $(\bar{\mathcal{E}}, \mathcal{A})$ is a partition matroid.

**Proof.** Consider an allocation $A \in \mathcal{A}$ and a fictitious object $(o, i) \in A_i$. If we remove $(o, i)$ from $A_i$, i.e., $A_i' = A_i \setminus \{(o, i)\}$, then $A_i' \subseteq \mathcal{E}_i$ will still hold as well as $A_j \subseteq \mathcal{E}_j$, for $j \in N' \setminus \{i\}$, which implies that $(\bar{\mathcal{E}}, \mathcal{A})$ is an independence system.

Consider now two allocations $A_1, A_2 \in \mathcal{A}$. If $|A_1| < |A_2|$ then $\exists E_i$ such that $|A' \cap \mathcal{E}_i| > |A \cap \mathcal{E}_i|$, which implies that there is a node $i \in N'$ with at least one free space in its cache, i.e., $|A_i| < K_i$. Therefore, there is an $(o, i) \in (A' \setminus A) \cap \mathcal{E}_i$ such that $A \cup \{(o, i)\} \in \mathcal{A}$.

**Proof of Theorem 1.** We prove the theorem by showing that the function $C(A) = -C(A')$ is a nondecreasing submodular function on $A$. Let us define the change of the global cost after inserting an object $o$ in the cache of node $i$ as $\Delta C(A) = C(A \cup \{(o, i)\}) - C(A)$, where $A \in \mathcal{A}$ and $\exists E_i$ for which $|A_i| \geq K_i$. We show that $C(A \cup \{(o, i)\}) - C(A) \geq C(A' \cup \{(o, i)\}) - C(A')$ for all $A \subseteq A' \subseteq \mathcal{A}$ and $(o, i) \in \mathcal{E}_i \setminus A_i'$. We now distinguish between two cases. If $\exists j$ such that $(o, j) \in A_i' \setminus A_j$ then the difference $\Delta C(A)$ is

$$\Delta C(A) = c_0 \sum_{k \in \mathcal{N}_i \setminus \text{LCA}(k, i) = n_0} \sum_{k \in \mathcal{N}_i} w^o_k + \sum_{t=1}^{l_{i-1}} \left( c_0 + d_{P_i(j, n_0)} \right) \sum_{k \in \mathcal{N}_i \setminus \text{LCA}(j, i) = n_0} w^o_k,$$

and the difference $\Delta C(A')$ is

$$\Delta C(A') = c_0 \sum_{k \in \mathcal{N}_i \setminus \text{LCA}(j, i) = n_0} \sum_{k \in \mathcal{N}_i} w^o_k + \sum_{t=1}^{l_{i-1}} \left( c_0 + d_{P_i(j, n_0)} \right) \sum_{k \in \mathcal{N}_i \setminus \text{LCA}(j, i) = n_0} w^o_k.$$

Since $\text{LCA}(j, i) \geq 0$, it holds that $\Delta C(A) > \Delta C(A')$. Otherwise, if $\exists j$ such that $(o, j) \in A_j$ or if $\exists \bar{j}$ such that $(o, j) \in A_{\bar{j}}$ then $\Delta C(A) = \Delta C(A')$. The result then follows by applying Lemma 1 to $C(\emptyset) - C(A)$.

Observe that the approximation ratio is bounded for arbitrary traversals of the graph. Nonetheless, a pre-order depth-first traversal allows for a distributed implementation of DFG with a communication overhead of $\sum_{t=1}^{l-1} (|N| - k_i)K_i$.

It is important to note that DFG differs from the distributed global greedy (DGG) algorithm used in [5], [11]. DGG chooses in every iteration the fictitious item $(i, o)$ that maximizes the cost saving, and thus has computational complexity $O(|N|^2 \max_i K_i |O| \log(|N| |O|))$. In contrast, DFG populates the caches of the nodes one-by-one, and thus
has computational complexity $O(|\mathcal{N}| \max_k K_k |\mathcal{O}| \log(|\mathcal{O}|))$. Unfortunately, DFG requires global information at every node of the network, which may cause significant communication overhead. We therefore turn to distributed approximation algorithms based on limited information.

IV. DISTRIBUTED ALGORITHMS UNDER LIMITED INFORMATION

In what follows we propose two distributed algorithms that do not need global information about the demands and the network topology.

A. Local Greedy Swapping (LGS) Algorithm

The first algorithm, called Local Greedy Swapping (LGS), allows nodes to swap objects with their parents based on the aggregate demands and the object placements in their descendants only. Denoting the placement at node $i$ at iteration $k$ by $A_i(k)$, the LGS algorithm starts with an initial empty object placement $A_{0}(0) = (A_{0}(0))_{i \in \mathcal{N}}$ in which each node $i \in \mathcal{N}$ stores $K_i$ objects. At iteration $k$ the algorithm computes the set of beneficial swaps $T(A(k)) \subset \mathcal{N} \times \mathcal{O}^2$. A triplet $(i,o,p) \in T(A(k))$ corresponds to that node $i$ can swap object $p \in A_i(k)$ with object $o \in A_{P(i)}(k)$ at its parent node $P(i)$. For $i = n_0$, i.e., $(n_0,o,p) \in T(A(k))$ the root node $n_0$ can evict object $p$ and can fetch object $o$ through the Backbone. The set of implemented swaps $S(A(k)) \subseteq T(A(k))$ is then chosen to increase the local cost saving greedily.

To define the set of beneficial swaps $T(A(k))$, let us introduce the function $I(i,o,p)$ to indicate whether the aggregate demand at node $i$ and its descendants $D(i)$ is higher for object $o$ than for object $p$,

$$I(i,o,p) = \begin{cases} 1, & \text{if } \sum_{j \in \mathcal{N}_i} (w^o_j - w^p_j) > 0 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Given a placement $A_i$, node $i$ might be interested in swapping object $p \in A_i$ with object $o \in A_{P(i)}$ at its parent if $I(i,o,p) = 1$ or if $p$ is available in the cache of node $i$’s descendants $D(i)$, i.e., $p \in \mathcal{R}_i \setminus A_i$, as in this case node $i$ can retrieve object $p$ at no cost even if $p \notin A_i$. We use this observation to define the set of node-object triplets that would be beneficial for swapping at placement $A_i$.

$T(A) = \{(i,o,p) | i \in \mathcal{N}, o \in A_{P(i)} \setminus \mathcal{R}_i, p \in A_i, (p \in \mathcal{R}_i \setminus A_i) \lor (p \notin \mathcal{R}_i \setminus A_i \land I(i,o,p) = 1)\}$.

The algorithm terminates at iteration $k$ if the set $T(A(k))$ is empty. The pseudo-code of LGS is shown in Fig. 3.

To complete the definition of the algorithm, we now describe a greedy algorithm to choose the set $S(A(k)) \subseteq T(A(k))$ at iteration $k$. Given $T(A(k))$, we choose a node $i_k$ with a child that would like to swap (i.e., $\exists j \in C(i_k)$ and $(j,o,p) \in T(A(k)))$. Given $i_k$ we select the best swap $(j_k,o_k,p_k)$ of its children, i.e., the one that maximizes the local cost saving in the subtree $N_{i_k}$ (swap with parent), and we then allow every child node $j \in C(i_k)$ to insert into its cache objects $o \in A_{i_k}(k) \cup \{p_k\}$, if doing so would increase the local cost saving (copy from parent). The algorithm is shown in Algorithm 1.

**Algorithm 1: LGS Algorithm**

1. $k \leftarrow 0$
2. while $|T(A(k))| > 0$ do
3. $A(k+1) \leftarrow A(k)$
4. for each $(i,o,p) \in S(A(k))$ do
5. $A_i(k+1) \leftarrow (A_i(k) \cup \{o\} \setminus \{p\})$
6. if $p \notin A_{P(i)}(k)$ then
7. $A_{P(i)}(k+1) \leftarrow (A_{P(i)}(k) \cup \{p\} \setminus \{o\})$
8. end if
9. end for
10. $k \leftarrow k + 1$
11. endwhile

**Lemma 2.** The global cost $C$ decreases strictly at every swap.

**Proof.** Consider $(i,o,p) \in S(A(k))$ at iteration $k$. For every node $j \in \mathcal{N} \setminus n_i$, it holds $d_j = d_{P(i)} + c_{P(i),i}$, hence $d_j(o,A(k+1)) = d_j(o,A(k))$ and $d_j(p,A(k+1)) = d_j(p,A(k))$. Consequently, $C_j(A(k+1)) = C_j(A(k))$ for all $j \in \mathcal{N} \setminus n_i$.

Consider now node $j \in n_i$. Since $S(A(k)) \subseteq T(A(k))$, it follows that $o \notin \mathcal{R}_j$ and $o \in A_{P(i)}$. Hence $d_j(o,A(k)) = d_{ij} + c_{P(i),i}$, $d_j(o,A(k+1)) = d_{ij}$, and the difference in the cost $\Delta C(k+1)$ before and after the swap is

$$\Delta C(k+1) = \sum_{j \in \mathcal{N}_i} [C_j(A(k+1)) - C_j(A(k))]$$

$$= \sum_{j \in \mathcal{N}_i} [w^o_j d_{ij} - w^p_j (d_{ij} + c_{P(i),i}) + w^p_j d_j(p,A(k+1))]$$

$$- w^p_j d_j(p,A(k))]$$

$$= \sum_{j \in \mathcal{N}_i} [-w^o_j c_{P(i),i} + w^p_j (d_j(p,A(k+1)) - d_j(p,A(k)))]$$

Similarly, $S(A(k)) \subseteq T(A(k))$ implies that $p \in A_i(k)$, hence $d_j(p,A(k)) \leq d_{ij}$. We now distinguish between two cases. If $d_j(p,A(k)) < d_{ij}$, then $d_j(p,A(k+1)) = d_j(p,A(k))$, which implies that $\Delta C(k+1) < 0$. Otherwise, if $d_j(p,A(k)) = d_{ij}$, then $d_j(p,A(k+1)) = d_{ij} + c_{P(i),i}$. Since $I(i,o,p) = 1$, then $\Delta C(k+1) = c_{P(i),i} \sum_{j \in \mathcal{N}_i} (w^o_j - w^p_j) < 0$. This proves the lemma.

We can use this result to show that the algorithm terminates after a finite number of iterations.

**Theorem 3.** The LGS algorithm terminates after a finite number of iterations.

**Proof.** Consider iteration $k$ of the LGS algorithm. Call $S(A)$ the object placement that results from applying swap $s = (j,o,p)$ to object $p \in \mathcal{R}_j \setminus A_j(k)$ or if $I(j,o,p) = 1$. Therefore, from the proof of Lemma 2, it follows that $C_j(S(A(k))) \leq C_j(A(k))$ for any swap $s = (j,o,p) \in S(A(k))$. Hence, for any swap $s = (j,o,p) \in S(A(k))$ and any node $l \in \mathcal{N} \setminus n_j$, it holds $R_l(A(k)) \cap A_l(k) = R_l(s(A(k))) \cup s(A_l(k))$ and hence $C_l(s(A(k))) = C_l(A(k))$. Since for every $l \in C(i_k)$, $j \neq l$ it holds $l \notin \mathcal{N}_j$, we can consider each node $j \in C(i_k)$ separately.

Consider swap $s = (j,o,p) \in S(A(k))$. It follows from (3) that either $p \in \mathcal{R}_j \setminus A_j(k)$ or $I(j,o,p) = 1$. Therefore, from the proof of Lemma 2, it follows that $C_j(S(A(k))) \leq C_j(A(k))$ for every node $j \in C(i_k)$.
Algorithm 1 \( S(A(k)) = \text{populateS}(A(k), i_k) \)

1. Select the best swapping opportunity at the children of \( i_k \),
\[
(j_k, o_k, p_k) \leftarrow \arg \max_{(j, o, p) \in T(A(k))} \sum_{n \in \mathcal{N}_j} c_{i,j}(w_n^o - w_n^p)
\]
\[
S(A(k)) \leftarrow (j_k, o_k, p_k)
\]

2. Further decrease the cost function through allowing nodes in \( C(i_k) \) to insert objects available at \( \{A_{c_k}(k) \cup \{p_k\}\} \).
\[
\text{PE}_j \leftarrow (A_{c_k}(k) \cup \{p_k\}) \cap \mathcal{A}_j(k)
\]
\[
\text{PO}_j \leftarrow (A_{c_k}(k) \cup \{p_k\}) \setminus \mathcal{R}_j(k)
\]
\[
\text{while } \exists (j, o, p) \text{ s.t. } o \in \text{PO}_j \text{ and } p \in \text{PE}_j \text{ and }
\] \[
(p \in \mathcal{R}_j \setminus \mathcal{A}_j(k)) \lor (p \notin \{\mathcal{R}_j \setminus \mathcal{A}_j(k) \}\cup I(j, o, p) = 1)
\] \[
\text{do}
\]
\[
S(A(k)) \leftarrow S(A(k)) \cup \{(j, o, p)\}
\]
\[
\text{PE}_j \leftarrow \text{PE}_j \setminus \{p\}
\]
\[
\text{PO}_j \leftarrow \text{PO}_j \setminus \{o\}
\]
\[
\text{end while}
\]

Algorithm 2 \( A' = \text{PushDown}(i, A) \)

1. \( t \leftarrow 0 \)
2. \( A^0 \leftarrow A \)
3. \( \text{do} \)
4. \( n \leftarrow \mathcal{P}(i) \)
5. \( s' \leftarrow \arg \min_{s \in \mathcal{A}(i, s)} \mathcal{C}(i, \{s\}) \)
6. \( A_{n+1} \leftarrow A_n \cup \{s'\} \)
7. \( A_{n+1} = A_{n+1} \setminus \{s'\} \)
8. \( t \leftarrow t + 1 \)
9. \( \text{while } n \neq n_0 \)
10. \( \text{return } A' \)

An iteration of the algorithm consists of two steps. The first step is an eviction operation at some node \( i \). The second step is a \( \text{PushDown} \) move, a sequence of placement updates such that at each update one object \( o \in \mathcal{A}(i) \) is moved from \( \mathcal{P}(i) \) to \( \mathcal{P}(i) \) for \( l = 1, 2, \ldots, k \), where \( \mathcal{P}(i) \) is the last update of the \( \text{PushDown} \) move, i.e., \( l = k \), one object is retrieved through the Backbone and stored at the root node \( \mathcal{P}(i) \). The pseudo-code of the \( \text{PushDown} \) move of the \( \text{h-PushDown} \) algorithm is shown in Algorithm 2.

Central to the algorithm is the LCA of node \( i \) and the node from which node \( i \) would retrieve object \( o \) in the placement \( \{\emptyset, A_{n_0}\} \), i.e., if it had no objects cached,
\[
\mathcal{P}(i, A_{n_0}) = \text{LCA}(i, \arg \min_{j \in \mathcal{N} \setminus \{i\} \cup \{A_j\}} d_{i,j})
\]
Similarly, we define \( \mathcal{P}(i, A) \) for placement \( A \), i.e., \( \mathcal{P}(i, A) = i \) if \( o \in \mathcal{A}_i \), otherwise \( \mathcal{P}(i, A) = \mathcal{P}(i, A_{n_0}) \).

The following lemma shows an important property of the \( \text{PushDown} \) move.

Lemma 3. A move \( A' = \text{PushDown}(i, A) \) always decreases the global cost by
\[
|t| \left(i\right) \bar{\mathcal{C}}_D(i, A) \leq \mathcal{C}(A) - \mathcal{C}(A') = \sum_{t=0}^{t(i)} \mathcal{C}(A') = \sum_{t=0}^{t(i)} \mathcal{C}(A') = \sum_{j \in T(t)} w_j^o,
\]
where \( T(t) = \{ j \in \mathcal{N} \setminus \{i\} \cup \{A_j\} \} \).

Proof. Consider iteration \( t \) of move \( A' = \text{PushDown}(i, A) \). Since \( c_{n, p}(n) = 0 \), for all \( j \in \mathcal{N} \setminus \{i\} \), it holds that \( d_j(o', A') = d_j(o', A^{t+1}) \).

For simplicity, we restricted ourselves to a single \( i_k \) per iteration when defining \( S(A(k)) \), but the above results hold for any set of nodes that are not each others’ descendants, hence the algorithm can be executed in parallel.

B. \( \text{h-PushDown} \) Algorithm

In the \( \text{LGS} \) algorithm, every node \( i \) swaps objects based on the information about the object placement and the aggregate demand for objects at its descendants \( D(i) \). In the following we provide a distribution algorithm that allows node \( i \) to leverage additional information on placements and on aggregate demands for objects. In the \( \text{h-PushDown} \) algorithm, every node \( i \) has information about the placement \( \mathcal{A}_j \) and about the object demands \( w_k^j \), for every ancestor \( j \) that lies within its information horizon \( h_i \), i.e., for \( j = \mathcal{P}(i) \) for \( 0 < l < h_i \).

The algorithm starts with an object placement \( (\mathcal{A}_j(0))_{i \in \mathcal{N}} \) in which each node \( i \in \mathcal{N} \) stores \( K_i \) objects that have the highest aggregated demands in the subnetwork \( \mathcal{N}_i \) and that are not available in the cache of node \( i \)’s descendants \( D(i) \).

317
1: \( k \leftarrow 0 \)
2: \( Z^{0} \leftarrow \{ i \in N \text{ such that } |Z_{i}(A(0))| > 0 \} \)
3: \( A \leftarrow A(0) \)
4: while \( |Z^{k}| > 0 \) do
5: \( \text{Pick } i_{k} \in Z^{k} \)
6: \( \text{Compute the least cost eviction} \)
7: \( \phi^{k} \leftarrow \arg \min_{o \in Z_{i_{k}}} |C_{EV}(i_{k}, o, A)| \)
8: \( \Delta C^{h}_{PD}(i_{k}, A) = \sum_{t=0}^{\infty} c_{P^{t+1}(i_{k}), P^{t}(i_{k})} \sum_{j \in T(0)} w_{j}^{t} \)
9: if \( \Delta C^{h}_{PD}(i_{k}, A) + \Delta C_{EV}(i_{k}, \phi^{k}, A) > 0 \) then
10: \( \text{If } \phi^{k} \leftarrow \text{PushDown}(i, (A_{i_{k}} | \phi^{k}), \text{A}_{-i_{k}}) \)
11: \( k \leftarrow k + 1 \)
12: \( A \leftarrow A(k) \)
13: \( Z^{k} \leftarrow \{ i \in N \text{ such that } |Z_{i}(A(k))| > 0 \} \)
14: end if
15: end while

The change in the global cost caused by the eviction of object \( o \) at node \( i \) observe that \( \Delta C_{EV}(i, o, A) \leq 0 \).

The pseudo-code of the \( h \)-Push Down algorithm is shown in Figure 4. We start with showing that the algorithm terminates in a finite number of iterations.

**Theorem 4.** The \( h \)-Push Down algorithm terminates after a finite number of iterations.

**Proof.** We prove the theorem by showing that the global cost \( C(A) \) decreases at every iteration of the \( h \)-Push Down algorithm. From Lemma 3 it follows that

\[
\Delta C_{PD}(i_{k}, (A(k))_{i_{k}}(\phi^{k}), (A(k))_{-i_{k}}) \geq \Delta C_{PD}(i_{k}, A(k)).
\]

(5)

By definition, the variation of the global cost at iteration \( k \) can be written as the sum of the variation due to the eviction and the variation due to PushDown move, i.e., \( \Delta C_{EV}(i_{k}, o^{k}, A(k)) + \Delta C_{PD}(i_{k}, (A(k))_{i_{k}}(\phi^{k}), (A(k))_{-i_{k}}) = C(A(k)) - C_{PD}(A(k) + 1)). \) The proof of the theorem follows from (5).

Furthermore, similar to LGS, the algorithm does not make any changes to an optimal placement, as shown next.

**Corollary 2.** The optimal content placement \( \bar{A} \) is stable with respect to the \( h \)-Push Down algorithm.

**Proof.** The proof is analogous to the proof of Corollary 1.

Observe that the computation of \( \Delta C_{PD}^{l}(i_{k}, A(k)) \) depends only on the object demands and the placements at the nodes in the set \( N^{P^{l+1}(i_{k})} \). Furthermore, in order to compute \( \Delta C_{EV}(i_{k}, o^{k}, A(k)) \), node \( i_{k} \) only requires information about placements and demands in the subnetwork \( N^{P^{l+1}(A_{-i_{k}})} \), which lies within node \( i_{k} \)’s information horizon \( h \).

V. Numerical Results

We use simulations to evaluate the approximation ratio and the convergence rate of the proposed algorithms. To generate backhaul topologies, we use the Manhattan model, in which \( |N| \) nodes are randomly placed on a \( |N| \times |N| \) grid. Given the node placement, we build a weighted complete graph by setting the weight on edge \((i, j)\) equal to the Euclidean distance between nodes \( i \) and \( j \), computed based on their coordinates. We then run Kruskal’s algorithm [12] on the resulting weighted complete graph to compute a minimum spanning tree to obtain the topology \( G \). We consider two different cost models. In the distance cost model the edge costs \( c_{P(i),i} \) are equal to the weights used for generating the tree. In the descendants cost model the edge costs \( c_{P(i),i} \) are proportional to the size of the subtree \( N_{i} \), as larger subnetworks likely lead to higher peak loads and less available bandwidth on the links serving them.

The object demands \( w_{o}^{i} \) follow Zipf’s law. For the ranking of the object demands at the nodes we consider two models. In the case of homogeneous demands, the object demands have the same rank at all nodes. In the case of heterogeneous demands, every demand \( w_{o}^{i} \) for object \( o \) at node \( i \) is ranked as in the case of homogeneous demands with 0.5 probability. With 0.5 probability, the rank of \( w_{o}^{i} \) is picked uniformly at random. The results shown are the averages of 500 simulations, and the error bars show 95\% confidence intervals.

As a baseline for comparison, we use a selfish distributed algorithm called Distributed Local-Greedy (DLG), which is based on global information about the object demands and placements at every node of the network. Following the DLG algorithm, starting from a randomly chosen allocation, at iteration \( k \) node \( i_{k} \) optimizes its placement of objects \( A_{i_{k}}(k) \) so as to minimize the cost for serving the requests from the local cell site, given the placement of objects \( A_{-i_{k}}(k) \) at the other nodes in the network [13], [14], [15]. As there is no guarantee that the DLG algorithm terminates [15], we run it for \(|N| \) iterations and we set \( i_{k} = k \). Note that although DLG is seemingly similar to DFG, DFG minimizes the global cost based on global information, while DLG minimizes the local cost based on global information, hence it is algorithmically simpler.

A. Performance of distributed algorithms

In order to compare the performance of the proposed algorithms, as well as to evaluate the tightness of the analytical results, we computed the optimal placement \( \bar{A} \) and the cost-approximation ratio \( C(A)/C(\bar{A}) \) for each algorithm. To make the computation of the optimal placement feasible, we considered a relatively small scenario with \( |N| = 20 \), \( |O| = 100 \) and \( K_{i} = 2 \) for all \( i \in N \). Figure 5 shows the cost-approximation ratio as a function of the Zipf exponent of the object demand distribution for LGS, DLG, DFG and for the \( h \)-Push Down algorithm with global information, i.e., for \( h = \max_{i \in N} l(i) \), for the descendants cost model.

The most salient feature of the figure is that the approximation ratio of the LGS algorithm increases exponentially with the Zipf exponent at a fairly high rate. The reason for the poor performance in the case of homogeneous demands is
that the LGS algorithm populates the set \( S(A(k)) \) exclusively based on the rankings of the object demands and not based on their values. As the Zipf exponent increases, the demand of the most popular content increases and the optimal solution might differ significantly from the allocation reached by the LGS algorithm. In order to validate this hypothesis, we computed the redundancy of a placement \( A \) using the index

\[
\text{r}(A) = \frac{\sum_{i \in N} \sum_{j \in A \setminus \{i\}} \left( 1 - \frac{\min(K_i, K_j) - |A_i \cap A_j|}{\min(K_i, K_j)} \right)}{|N|(|N| - 1)}.
\]

Intuitively, \( r(A) \) is the average ratio of objects common between all pairs of placements \( A_i \) and \( A_j \).

In Figure 5 we plot the average \( r(A) \) index of the final placements reached by the algorithms, for the same scenario as Figure 5. The figure confirms that as the Zipf exponent increases, the LGS algorithm fails to introduce redundancy, which explains its poor performance.

Comparing the performance of h-Push Down to that of DFG we observe that h-Push Down (with global information) performs better than DFG, which is also reflected by the redundancy index, which is very close to the optimal (cf. Fig. 5). Finally, it is noteworthy that the DLG algorithm, which corresponds to selfish local optimization, fails to achieve performance close to the optimal, despite the availability of global information.

In order to evaluate the performance of the algorithms for larger scenarios, in the following we use the DLG algorithm as a baseline for comparison, as it is prohibitive to compute the optimal placement. Recall that the DLG algorithm optimizes the placement of objects in order to minimize the local cost, which would make it a reasonable simple choice in absence of more elaborate distributed algorithms.

To capture the performance of the algorithms relative to DLG we define the performance gain of an algorithm as the ratio between the cost of the placement reached by the DLG algorithm and the cost of the placement reached by the algorithm. It follows from (1) that the performance gain is also a measure of the increased hit rate achieved by the algorithm relative to DLG. Figure 7 shows the performance gain for the LGS, DFG and h-Push Down (for two values of the information horizon) algorithms, as a function of the number of nodes for \( K_i = 20 \). The results are shown for heterogeneous demands using a Zipf exponent of 1, for the two cost models. We observe that the performance gain for the DFG and the h-Push Down algorithms increases with the number of nodes. Furthermore, the figure shows that h-Push Down outperforms DFG (i.e., it is close to optimal) for both values of the horizon \( h \). The figure also shows that LGS performs just slightly better than DLG, with a decreasing gain as the network size increases.

Figure 8 shows the number of iterations needed to compute the final object placement corresponding to the results shown in Figure 7. Recall that the DFG algorithm starts with an empty allocation and terminates in \( \sum_{i \in N} K_i \) iterations, and can thus be used as a baseline in terms of convergence. The results show that LGS performs worst, while h-Push Down for \( h = 4 \) requires almost an order of magnitude less iterations to terminate than DFG.

Figure 9 shows the performance gain as a function of the cache sizes for \( |N| = 50 \). The figure shows that for higher cache sizes the performance gain of the DFG and h-Push Down algorithms over the DLG algorithm increases faster than exponentially. In the case of global information, the h-Push Down algorithm outperforms the DFG algorithm, while in the case of non-global information, i.e., for \( h = 4 \), it achieves performance close to the DFG algorithm. Furthermore, the performance gap between the h-Push Down algorithm with global and non-global information increases for higher cache sizes. The figure also confirms that the LGS and DLG algorithms achieve a comparable total cost.

B. Impact of the information horizon (h)

Finally, we evaluate the impact of the information horizon \( h \) on the performance of h-Push Down. We define the performance gain \( \text{PG}^h(A) \) for horizon \( h \) as the ratio between the cost of the placement \( A^h \) reached by the h-Push Down algorithm with \( h = 1 \) and the cost of the placement \( A^h \) reached with horizon \( h_i \), i.e. \( \text{PG}^h(A) = \frac{\text{Cost}(A^h)}{\text{Cost}(A^1)} \).

Figures 10 and 11 show the performance gain \( \text{PG}^h(A) \) and the number of iterations, respectively, for the h-Push
Fig. 7: Performance gain vs number of nodes $|\mathcal{N}|$ for the $h$-Push Down, LGS and DFG algorithms on the Manhattan graph with descendants and distance cost model, $|\mathcal{O}| = 5000$, $K_1 = 20$.

Fig. 8: Number of iterations vs number of nodes $|\mathcal{N}|$ for the $h$-Push Down, LGS and DFG algorithms on the Manhattan graph with descendants and distance cost model, $|\mathcal{O}| = 5000$, $K_1 = 20$.

Fig. 9: Performance gain vs. cache size $K_i$ for $h$-Push Down, LGS and DFG on the Manhattan graph with descendants and distance cost model. Results for $|\mathcal{O}| = 5000$, $|\mathcal{N}| = 50$.

Fig. 10: Performance gain vs. horizon $h$ for cache sizes $K_i \in \{10, 20\}$ on the Manhattan graph with descendants and distance cost models. Results for $|\mathcal{O}| = 5000$ and $|\mathcal{N}| = 100$.

the problem, they design a distributed amortizing algorithm that achieves a constant factor approximation. The model considered in [16] is based on the ultrametric cost model introduced in [6], which differs from our model on the assumption of symmetric costs between nodes. The authors in [5] give insights in the structure of the optimal placement in a regular two level hierarchical network, and they develop a greedy distributed 2-approximation algorithm. The authors in [11] consider a hybrid network with in-network caching and they propose a $(1 - 1/e)$-approximation greedy algorithm. A more generic cost model was considered in [7], where the authors develop a 10-approximation algorithm by rounding the optimal solution of the LP-relaxation of the problem. [17] proposed a set of centralized, polynomial time algorithms with approximation guarantees, for the joint problem of request routing and content replication under strict bandwidth constraints at the storage sites. In contrast to [16], [5], [11], [7], [17], in our work we developed two distributed algorithms for computing a content placement based on limited information on the content demands and on the network topology, that can be used to solve large problem instances with prohibitive space complexity.

Related to ours are recent works on game theoretical analyses of distributed selfish replication on graphs [13], [18], [19], [20], [21], [14], [15], as they can serve as a basis for distributed
content placement algorithms. Equilibrium existence when the access costs are homogeneous and nodes form a complete graph were provided in [13], and results on the approximation ratio (referred to as the price of anarchy) were provided in [18], [19] for homogeneous costs and a complete graph. Non-complete graphs were considered in [20], [21], [14], and results on the approximation ratio of a distributed greedy algorithm were given for the case of unit storage capacity and an infinite number of objects in [20], [21] considered a variant of the problem where nodes can replicate a fraction of objects, and showed the existence of equilibria, while convergence results were provided for the integer problem in [14] in the case of homogeneous neighbor costs. The case of heterogeneous neighbor costs, for which the non-convergence of distributed greedy replication was shown in [15] is a generalization of our model, and thus the negative result provided in [15] may not apply to our case. Different from these works, in this paper we consider caches managed by a single entity, and thus we consider the minimization of the total cost as opposed to the selfish minimization of the cost of the individual nodes. Our objective of minimizing the total cost also sets this work apart from recent work on cache networks in the context of content centric networks, e.g., [22].

VII. Conclusion

We considered the problem of minimizing the bandwidth demand in a mobile backhaul through cooperative caching, and formulated it as a 0-1 integer linear program. We proposed a 2-approximation distributed algorithm that is based on global information. Furthermore, we proposed a low complexity distributed algorithm based on information about object demands at descendants, and an algorithm with an adjustable level of available information. We proved convergence and stability of the algorithms. We used extensive simulations to evaluate the performance of the proposed algorithms. Our results show that information about object demands at descendants is insufficient for good cooperative caching performance, but the proposed h-Push Down algorithm achieves consistently good performance despite limited information availability, consistently better than greedy optimization based on global information.

References


Performance Evaluation for New Web Caching Strategies
Combining LRU with Score Based Object Selection

Gerhard Hasslinger
Deutsche Telekom
Darmstadt, Germany
gerhard.hasslinger@telekom.de

Konstantinos Ntougias
Athens Information Technology
Athens, Greece
kontou@ait.gr

Frank Hasslinger
Darmstadt Univ. of Tech.
Darmstadt, Germany
frank.hasslinger@stud.tu-darmstadt.de

Oliver Hohlfeld
RWTH Aachen
Aachen, Germany
oliver@comsys.rwth-aachen.de

Abstract — The topic of Internet content caching regained relevance over the last years due to the extended use of data center infrastructures in CDNs, clouds and ISP networks to meet the capacity and delay demands of multimedia services. In this study, we evaluate the performance of web caching strategies in terms of the achievable hit rate for realistic scenarios of large user populations. We focus on a class of score gated least recently used (SG-LRU) strategies which combine the simple update effort of the LRU policy with the flexibility to keep the most important content in the cache according to a predefined score function.

Caching efficiency is evaluated via simulations assuming Zipf request pattern, which have been confirmed manifold in the access to popular web platforms for video streaming and other types of content. We analyze the possible hit rate gain of alternative web caching strategies over pure LRU for the standard independent request model (IRM) within the complete relevant range of the three basic system parameters. The results confirm that the absolute hit rate gains of 10%-20% over LRU being observed in some case studies for special caching strategies are a realistic estimation in general.

Moreover, we compare IRM evaluations with results for dynamic request pattern over time using Wikipedia statistics, which recently have been made available as daily top-1000 page requests. Simulations are extended to show the impact of varying object popularity on the caching efficiency and to adapt a score-based caching strategy to increasing popularity dynamics.

Keywords — Web caching strategies, Zipf distributed requests, least recently used (LRU), score gated LRU, least frequently used (LFU), hit rate, simulation, Che approximation

I. INTRODUCTION

A. Web caching strategies for content delivery on the Internet

Web caching systems are widely applied at global scale on the Internet to improve delivery of streaming, IPTV and many other IP services. Today, a major portion of IP traffic is transferred from content delivery networks (CDNs) and distributed data centers in cloud architectures [12][17][22][26], yielding essential traffic savings on expensive intercontinental and inter-domain links. The main benefits are reduced load and delays as well as higher throughput, when caches serve requests to popular data on shorter transport paths to the users. Caches are also present in local networks, as nano data centers [29], or in home gateways and browsers on the end devices [5][11].

In this regard, the efficiency of a cache is driven by the replacement strategy for identifying and storing the most relevant objects. A basic and widely used caching strategy follows the least recently used (LRU) principle, employing a simple cache update scheme. However, LRU can't offer flexibility to prioritize objects according to cost and benefit aspects, regarding the size, the transport path from the origin or other preferences from the content providers' or users' perspective [3].

A number of studies have evaluated LRU web caching efficiency in terms of the hit rate [10][21][28][30]. They partly recommend LRU, whereas on the other hand, alternatives are shown to achieve higher hit rates [3][13][16][18][20][24]. Thus, no clear picture can be concluded from literature whether LRU is appropriate for web caching in a tradeoff with more efficient but often more complex alternatives.

This tradeoff is discussed in detail for the ARC caching strategy proposed by Megiddo and Modha [20]. ARC is shown to outperform LRU by more than 1.5-fold hit rate in most of a large set of measured and synthetic traces. However, the update effort for ARC is high for maintaining two lists and checking half a dozen cases for each update as depicted in Fig. 1 in [20]. Consequently, the ARC method isn't widely adopted.

B. Score gated LRU (SG-LRU) web caching strategy

For cache optimization, we recently proposed a score-gated LRU (SG-LRU) approach [14], which combines a LRU stack implementation with arbitrary scores being attributed to each object. SG-LRU can even undercut the update effort of pure LRU web caching, when using an appropriate score function.

LRU always puts the requested object on top of the cache, while the bottom object of a full cache is replaced if cache storage is exhausted. SG-LRU admits a new external object to enter a full cache only if it has a higher score than the bottom object and otherwise puts the bottom object on top, instead of the requested object. Figure 1 illustrates (SG-)LRU updates for a usual implementation as a cyclic double linked list. If the score updates are simple, e.g., when only the score of the requested object is changing, SG-LRU requires slightly higher update effort in the caching list than pure LRU. However, LRU is loading an object into the web cache for each request to an external object, i.e. for each cache miss, whereas SG-LRU can avoid most uploads for seldom requested, low-scored objects. Thus, SG-LRU usually undercut pure LRU update effort.

The detailed evaluation of the performance gain in SG-LRU hit rates as compared to pure LRU is a main goal of this study. We extend and complement experience from many case studies [3][13][16][18][20][24] based on measurement of specific web applications by an exhaustive simulative evaluation of the tradeoff between LRU and the least frequently used (LFU) principle for independent Zipf distributed requests. Zipf distri-
contributions of this study, we perform an upper bound of the caching efficiency. As the main new for the independent request model (IRM) and, thus, provides an optimum hit rate for the large user community has access to a large set of objects. The LFU principle keeps those objects in the cache that have been most frequently requested in the past. Pure LFU is not a valid web caching strategy because it cannot adapt to a changing popularity of objects, but it achieves an optimum hit rate for the independent request model (IRM) and, thus, provides an upper bound of the caching efficiency. As the main new contributions of this study, we perform:

- an extensive simulation study of the efficiency of SG-LRU caching strategies including advanced accuracy control estimation developed in [15], which is also used to check the Che approximation [6] for LRU hit rates,
- an exhaustive evaluation of the hit rate optimization potential left open by LRU compared to SG-LRU and LFU for the standard IRM model with Zipf distributed requests, and
- an extension of the study for dynamically changing popularity of objects applied to request pattern for Wikipedia pages.

II. SCENARIO CLASSIFICATION

Next, we address several preconditions in order to clarify our main focus within the broad spectrum of caching applications.

A. Web caching vs. caches in computing & database systems

Caching for support of IP services has basic commonalities with caching in computer and database systems, i.e. to get faster access to a part of the data that fits into a cache of limited size, but the workloads and the effects on costs and benefits are entirely different for both cases [2][3]. In computing and database systems, periodic sequences of requests are relevant whereas random and Zipf distributed request sequences are usual for web access. Frequent LRU data uploads from an original or higher layer web server to a cache are much more expensive than for data in a local system. Therefore, results from this study on web caching strategies cannot be transferred to caches in computing systems, although some technical discussions seem to mix up the different scenarios [21].

B. Cachable web content and HTTP caching guidelines

Caching is known to be inapplicable for a part of the web content, including highly dynamic content or one-timers [2], i.e. web pages that are only requested once over a long period. Moreover, a content owner or provider can mark objects as non-cacheable in network or in end system caches. Nonetheless, the major portion of IP traffic for video streaming and IPTV services is distributed from caches in CDN and cloud architectures. A recent update of caching options for HTTP [8] by the IETF standardization provides guidelines on how to avoid inappropriate data in caches. We always refer to the fraction of cacheable data in evaluations of caching efficiency.

C. Cache updates per request versus daily updates

Web caching strategies basically assume updates to be done per request, but they can partly be deferred to low traffic periods to reduce peak hour traffic. Cache providers often combine updates per requests with daily updates [17][23][26]. In this work, we focus on updates per request. The need for fast updates depends on the dynamics in the popularity of objects, as discussed in section VI.

D. Caches for large versus small populations

Caching systems are often organized hierarchically with central caches serving a large population and lower level caches for smaller regions. Even a cache in a browser for a single user can save 20% of download traffic for repetitive requests [5]. The request pattern is different for each user and varies for small communities. Our focus is on a large population with access to a large web platform, where the request pattern is known to be universal and characterized by a Zipf law [1].

E. Fixed versus variable size objects

Files and other objects representing cacheable web data have different size up to the GB range for videos. For small caches,
bin-packing problems can arise and therefore size has been considered as an important factor in studies which assume complete files as transport unit. However, coding schemes for video streams or files in client-server and peer-to-peer systems are nowadays segmenting data into small chunks in the kB range, while storage size is steadily increasing. Therefore, we simply assume objects of fixed size corresponding to data chunks. Files of different size can still be represented as sets of fixed size objects with equal score according to the file popularity. Moreover, Helbeda and Saleh [16] suggest to assign linear decreasing scores to fixed size data chunks of the same video because users often stop video streams after some time, such that the first data chunks of a video are more relevant.

We continue in Section III with an overview of the concept and preconditions for simulative evaluation of (SG-)LRU web caching, a brief overview of a random generator for Zipf distributions (III.A-B), score functions surveying statistics of past requests (III.C) and the control of the precision of hit rate distributions (III.A-B), score functions surveying statistics of past requests (III.D). Section IV presents hit rate results for the entire relevant parameter range for independent (IRM) Zipf distributed requests. The Che approximation is validated against simulation results in Section V. Section VI extends the performance evaluation for SG-LRU strategies to dynamically changing object popularity with a case study of requests to Wikipedia pages. Finally, main results are summarized in the conclusions.

III. SIMULATION OF CACHING FOR ZIPF REQUEST PATTERN

A. Relevance of Zipf request pattern

Many studies have confirmed Zipf’s law as an appropriate model for access pattern to content on the Internet including web shops and user-generated content such as videos hosted on YouTube [1][30], for channels in IP-TV systems [4][25] and for P2P file sharing systems such as Gnutella and BitTorrent [16][33]. According to Zipf’s law, a small set of popular web objects attracts most user requests, which is favourable for the efficiency of small caches.

When a finite set of N objects is considered for web caching, Zipf’s law assigns decreasing request probabilities z(r) corresponding to the objects’ popularity ranks r ∈ (1, 2, …, N):

\[ z(r) = \alpha r^{-\beta} \quad \text{for } \alpha, \beta > 0; \quad \alpha = z(1) = 1/\sum_{r=1}^{N} r^{-\beta} \quad (1) \]

where \( \beta \) is an adaptive shape parameter and \( \alpha \) is a normalization constant. Access probabilities becoming more unbalanced for \( \beta \to 1 \). \( \beta \) has been determined for Zipf models that were adapted to different sets of request measurement traces in [1][6][16] resulting in the range 0.56 ≤ \( \beta \) ≤ 0.88 covering all studied cases. Therefore we focus our caching simulations on Zipf distributed requests in the range 0.5 ≤ \( \beta \) ≤ 1.

B. Inversion method for a random Zipf rank generator

Despite of Zipf’s law relevance in Internet access, efficient random generators for Zipf ranks are missing in literature. The Mathematica tool set [32] refers to an acceptance-rejection method for Zipf random variates proposed by Devroye [7] which only covers the range \( \beta > 1 \) for infinite support. Instead, we derived the following inversion formula for selecting a Zipf rank \( r \) from a uniform random variate \( R \in [0, 1] \) for finite sets of \( N \) objects [15]

\[ r = N \left\lfloor \frac{1 - R(1 - (1/2)^{-\beta})}{1 - Z_{CDF}(N/2)} \right\rfloor \quad (2) \]

In particular, the Zipf rank generator (2) is confirmed in [15] to return the correct rank or a neighbor rank, i.e. to deviate by no more than ±1 from the correct rank \( r \) for \( N \leq 10^6 \) and \( \beta = 0.1, 0.2, \ldots, 3.0 \). The correctness of the rank \( r \) is verified by checking \( Z_{CDF}(r - 1) < R \leq Z_{CDF}(r) \). The cumulative Zipf distribution \( Z_{CDF}(r) \) is computed and stored for \( r \in \{1, \ldots, N\} \) in the starting phase of a simulation. A fast random Zipf rank generator is a prerequisite for simulating billions of requests.

C. Score functions for SG-LRU web caching

In our web caching simulations, we perform cache updates for LRU and SG-LRU strategies, whereas uploading and delivery of objects is not included. Basic data structures are set up for a cache of size \( M < N \) and a fixed set of \( N \) objects. The cache is implemented as a double chained cyclic list shown in Figure 1 with a top pointer to enable updates at low constant effort per request. Moreover, for each object we store \( Z_{CDF}(r) \) according to eq. (1) in order to control the Zipf rank generator as well as a score value in case of SG-LRU.

In general, SG-LRU opens flexibility to prefer cache content according to any arbitrary score function. Usual demands for fast updates impose the restriction of only a few score modifications per request. Scores which represent the number of requests to each object following the LFU principle are simply obtained by incrementing the score of the requested object.

Since pure LFU can’t adapt to changing popularity, other approaches with limited count statistics have been proposed and evaluated in literature [6][14][20]. A class of such strategies is attributed as LRFU spectrum [18], which includes LRU and LFU caching schemes as extreme cases. Two basic schemes covering the LRFU spectrum are

- sliding window, restricting LFU to request counts within a window of the \( W \) most recent requests, and
- geometrical fading, introducing a decreasing factor \( \rho^k \) for the \( k \)th recent request and ranking an item according to the sum of weights, as specified in equation (3).

Let \( \delta_{kj} = 1 \), if the \( k \)th recent request was addressing an object \( O_j \) of the set \( O_1, \ldots, O_N \), and otherwise \( \delta_{kj} = 0 \), where \( k \leq 1 \) refers to the most recent request. Then we can define the score functions for sliding window \( S^{SW} \) and geometrical fading \( S^G \)

\[ S^{SW}_{O_j} = \sum_{k=1}^{W} \delta_{kj} \quad S^G_{O_j} = \sum_{k=1}^{\delta_{kj}} \delta_{kj} \rho^k \quad 0 < \rho < 1 \quad (3) \]

Both schemes behave similar for \( \rho = 1 - 1/W \). An unlimited LFU scheme is approached as one extreme for \( W \to \infty \). On the other hand, geometric fading is equivalent to LRU for \( \rho \leq 0.5 \), when the most recent request dominates the weights.
Two score updates are needed for sliding window per new request, where the score of the requested object is incremented and the score of another object, whose former request is falling out of the window, is decremented. Therefore, the sequence of the last \( W \) requests has to be stored.

For geometrical fading, a direct implementation of the score function \( S^{gr} \) requires an update of all objects by a factor \( \rho \) per request and an increment of the score of the requested object. Instead, we add \( (1/\rho)^r \) only to the score of the \( \rho \)th requested object. In this way, we achieve the same ratio of scores of different objects as for direct computation of \( S^{gr} \). The only drawback is that \((1/\rho)^r\) is steadily growing. Therefore, we have to scale down all scores if the updated score exceed a threshold. Nonetheless, we prefer geometrical fading with fading factor \( \rho = 1 - 1/\rho \) in the evaluations instead of sliding window, because geometrical fading has smaller mean update effort per request and sliding window often needs to resolve equal scores of objects by a tie breaking mechanism.

The usual approach for score based caching maintains a sorted list of objects in the cache according to their scores, which leads to prohibitively high update effort for reinserting objects with modified scores into a sorted list, e.g. via a heap sort method used in [18]. Instead, we confirm SG-LRU to be sufficient for collecting high scored objects in the cache at low LRU update effort without the need for a strictly sorted list.

### D. Evaluation of SG-LRU web caching

Next, we compare the achievable hit rates for LRU, SG-LRU and LFU in simulation studies. We start with a first evaluation example assuming independent and Zipf distributed requests. For caching evaluations of the independent request model (IRM) we consider three basic parameters:

- the size \( N \) of a fixed set of objects,
- the cache size \( M \) \((M < N)\), and
- the shaping parameter \( \beta \) of the Zipf distribution, which determines the request probabilities \( z(r) = z(1)^r/\rho^r \) to the objects.

We simulate how the cache content is developing for a sequence of \( K \) requests, starting from an empty cache. During a filling phase of the cache until \( M \) different objects have been addressed, the caching strategies behave identical. As soon as the cache is full, pure LRU already has entered steady state regarding the object set in the cache. Thus, it is sufficient to exclude the cache filling phase as a non-representative start phase for pure LRU simulations. For LFU and SG-LRU, the convergence to a steady state depends on stabilizing score ranks of the objects, which takes much longer than the cache filling phase. LRU scores count the requests to each object. In a sequence of \( k \) successive requests, an object in rank \( r \) gets a binomially distributed number of requests in \([0, ..., k]\) with mean \( k z(r)\). We exclude the first quarter of each simulation run from evaluations of the hit rate, in order to converge to stable score ranks when the run time persists sufficiently long.

Figure 2 shows SG-LRU evaluation results for an example with \( N = 10^6 \) objects, a cache for \( M = 1000 \) objects and independent Zipf distributed requests with \( z(r) = 0.01337 \cdot r^{-0.8} \). In this case, the LRU hit rate is \( h_{LRU} \approx 10\% \), which is also confirmed by the Che approximation [6], whereas LFU achieves the maximum hit rate under IRM assumptions:

\[
h_{LFU} = z(1) + ... + z(M) \approx 20.68\%
\]

The score gated SG-LRU results for different \( \rho \) fall between both extremes. For sufficiently long simulation runs with \( K > 10^7 \) requests, the hit rate is observed to stabilize, where SG-LRU stays close to \( h_{LRU} \) for \( \rho \leq 10^{-3} \) and comes close to \( h_{LFU} \) for \( \rho \geq 10^{-3} \). For shorter simulations, the SG-LRU hit rate is still increasing with \( K \) towards a saturation level depending on \( \rho \). Since a fading factor \( \rho \) has similar effect as sliding window with window size \( W = 1/(1-\rho) \), it is obvious that the number of simulated requests is partly smaller than \( W \). Then the scores are still evolving in a transient phase with increasing hit rates. On the other hand, the SG-LRU results in Figure 2 are close to their saturated maximum level as soon as \( K > 10^6 \cdot W = 10^6(1-\rho) \) requests are evaluated.

The results confirm that SG-LRU with scores based on previous requests limited by sliding window or geometrical fading can fully adapt to LRU as well as LFU hit rates as extreme cases based on a single parameter, \( \rho \) or \( W \) respectively. The parameter can be automatically tuned to approach the LFU hit rate up to \( h_{LFU} - \epsilon \) in the IRM case, because the SG-LRU hit rate is monotonously increasing with \( \rho \) and \( W \).

### E. Hit rate estimator and variance of the simulation results

The evaluation of the hit rate from simulations is subject to variability that can be indicated by confidence intervals or by the standard deviation based on a number of simulation runs. Let \( h(k) \) be 1 if and only if the \( k^{th} \) request is a cache hit. In order to characterize the variability in hit rate simulations, we evaluate the second order statistics \( \sigma(h_{k}) \) that indicates the standard deviation of a stochastic process over request sequences of different length \( K \). \( \sigma(h_{k}) \) is defined and computed from the mean values over \( K \) successive requests of the process \( h(k) \):

\[
h_{k}(j) = \sum_{h(k) = 1}^{x} h(k) / K \quad \Rightarrow \quad \sigma(h_{k}) = \sqrt{E(h_{k}^2(j)) - \mu^2(h)} \quad \mu(h) = E(h_{k}(j)) = Pr(h(k) = 1).
\]

Figure 2: SG-LRU hit rate simulations for different run times
Note, that the mean $\mu(h)$ equals the hit rate in all time scales $K$ for a process in steady state, whereas $\sigma(h_{(K)})$ is expected to decrease with $K$, e.g. $\sigma(h_{(K)}) = \sigma(h_{(1)})/\sqrt{K}$ for a process of i.i.d. random values. Note also, that the cache filling phase and a start phase of a quarter of the requests is not included in the evaluations. In order to evaluate $\sigma(h_{(K)})$ during caching simulations, we consider successive request sequences of length $K = 10, 10^2, ..., 10^6$ of a simulation run over $10^{K+1}$ requests and take the usual estimate of the standard deviation:

$$\sigma(h_{(K)}) = \sqrt{\frac{\sum_{j=1}^{10^{K+1}} h_{(K)}^2(j) - \mu^2(h) \cdot \left(10^{K+1}/K - 1\right)}{\sum_{j=1}^{10^{K+1}} h(k)/10^{K+1}}}.$$  

For simulations based on the independent request model we can estimate the hit rate in two ways [15]: (i) by counting the hits or (ii) by the sum of the request probabilities of all objects in the cache over the considered request sequence. We denote the standard deviations of both estimators by (i) $\sigma(h_{(K)})$ and (ii) $\sigma(\pi_{(K)})$, respectively. Figure 3 shows a $2^{nd}$ order analysis of both hit rate estimators during a simulation for SG-LRU caching with geometrical fading scores for independent Zipf requests ($\beta = 0.8$; $N = 1000$ objects; $\rho = 0.9999$). Four different cache sizes ($M = 2, 13, 87$ and 342) are considered, which achieve about 10%, 25%, 50% and 75% SG-LRU hit rate. The four curves for the standard deviation of the hit count $\sigma(h_{(K)})$ are almost linear $\sigma(h_{(K)}) = \sigma(h_{(1)})/\sqrt{K}$. The four standard deviation curves for the sum of cache probabilities $\sigma(\pi_{(K)})$ as hit rate estimator start at a low level $\sigma(\pi_{(1)}) < 0.005$ already for snapshots of a single request, are remaining almost constant over several time scales and finally are also decreasing. On all time scales we observe that $\sigma(\pi_{(K)}) < \sigma(h_{(K)})$ and therefore prefer the sum of probabilities of cached objects as hit rate estimator.

Figure 3: $2^{nd}$ order statistics for SG-LRU caching simulations

IV. Exhaustive LFU / LRU Hit Rate Evaluations for Zipf Distributed Independent Requests (IRM)

In discussions whether LRU could be sufficient for web caching, the gain of alternative strategies is evaluated in measurement driven case studies [13][16][18][20] but it remains open how the performance depends on system parameters. We complement the experience based on measurement by a generalized simulative evaluation of the gain of LFU over LRU for the IRM model with Zipf distributed requests.

Therefore we extend our web caching simulations over the entire relevant range of the three characteristic parameters: (i) the number of objects $N$, (ii) the cache size $M (M < N)$, and (iii) the shaping parameter $\beta$ of the Zipf distribution.

Figure 4 shows results from extended evaluations for different size $N = 10^2$ and $N = 10^4$ of the set of objects. The achievable hit rate is compared for LRU and LFU for varying cache sizes $M$ and for $\beta = 0.5, 0.6, ..., 1$, actually with $0.9999$ instead of 1. Beyond simulations, the LFU optimum hit rate under IRM equals the sum of request probabilities $h_{(LFU)} = \pi(1) + ... + \pi(M)$. In case of LRU, the Che approximation can be applied and is confirmed to provide accurate results as discussed in Section V. We also checked that SG-LRU closely approaches the LRU and LFU results as extreme cases.

Regarding the potential advantage of alternatives over LRU for independent Zipf distributed requests, we observe a 10%-15% absolute hit rate gain of LFU over LRU for the entire relevant range, i.e. whenever LRU achieves a hit rate in the range 10% - 50%. The relative gain is largest especially for small caches. When LRU hit rates are below 10%, then the LFU optimum is at least twice as high. Moreover, the caching efficiency can be expressed in terms of the LFU versus LRU cache size required to obtain a demanded hit rate. In order to achieve 20% hit rate, LRU requires 3-fold to 8-fold larger cache size than LFU.

Figure 4: Comparing LFU/LRU hit rates for IRM Zipf requests
Finally, we evaluate the LFU improvement potential beyond LRU for each grid point combining the main parameters:

- $\beta = 0.4, 0.5, ..., 1.1$ for Zipf distributed requests due to IRM,
- $N = 10^2, 10^3, ..., 10^7$ for the number of objects, and
- $M = M_{10^1}, M_{10^2}, ..., M_{10^6}$ for the cache size,

where $M_{\text{min}}$ is the minimum cache size required to obtain an LRU hit rate of $x\%$ depending on $\beta$ and $N$. On the whole, $8 \cdot 6 \cdot 99 = 4752$ simulations have been performed to cover this range. Each simulation runs at least $10^7$ requests in the evaluation phase in order to reduce the standard deviation of the hit rate results below $5 \cdot 10^{-3}$ as checked according to Section III.E.

The results of the detailed simulation study are summarized in Figure 5. It is already visible from Figure 4 that the difference between LRU and LFU hit rates is primarily depending on the achieved LRU hit rate. For each combination of $\beta$ and $N$ we first evaluate the minimum cache sizes $M_{10^1}, ..., M_{99\%}$ to obtain $1\%, ..., 99\%$ LRU hit rate and then compute the LFU hit rate for the same cache size. Figure 5 shows the minimum, mean and maximum absolute LFU gain over all cases with the same LRU hit rate of $x\%$.

In fact, we can confirm at least 9.8% potential gain of LFU whenever the LRU hit rate is in the range of 10% - 50%. The mean LFU gain in this range is 13.7% and the maximum goes up to 15% - 20%. The maximum is most often reached for the case $\beta = 1.1, N = 100$. Then, the LRU and LFU cache sizes required for $x\%$ hit rate are often unchanged for several values of $x\%$. The minimum gain is due to different cases, most of which are in the largest set of items $N = 10^7$. Again, the relative differences in LFU versus LRU caching efficiency are largest in small caches. When the LRU hit rate is, e.g., 5% then an LFU cache of equal size achieves at least 12.8% hit rate and 15% in the mean over all 8-6 cases of $N \times \beta$ combinations.

The additional storage required to compensate for 10%-20% LRU hit rate deficit ranges from about twice the capacity for $N = 1000$ objects up to 10-fold and more storage required in examples with $N = 10^7$ for large content platforms. Measurement based studies [28] show similar curves for moderate LRU hit rates up to 25%, which indicate even 10-100-fold higher storage requirement to obtain 10%-20% more LRU hit rate.

![Figure 5: LFU gain over LRU for IRM Zipf requests: Result summary of 4752 cases as a grid on the relevant $\beta \times N \times M$ range](image)

V. ACCURACY OF THE CHE APPROX. CHECKED BY SIMULATION

Finally, we compare the Che approximation [6] with simulated LRU hit rates, confirming a surprisingly good match as also experienced in [10]. Note, that high accuracy is supported by mathematical arguments in [10] but without quantitative analysis in terms of concrete boundaries. Thus we checked the deviations between simulation results and the Che approximation for all combinations of parameters in a grid range $\{(\beta, N, M) \in \{0.4, 0.5, ..., 1.1\} \times \{10^2, 10^3, ..., 10^7\} \times \{M_{10^1}, M_{10^2}, ..., M_{100\%}\}\}$ for Zipf distributed requests. As the main result, we observe a maximum absolute difference of 0.4% while the majority of evaluated differences is below 0.02%. Each simulation result is again based on $10^6$ or more requests and is expected to be subject to a standard deviation of less than $5 \cdot 10^{-5}$ [15]. Partly, longer simulations runs are needed because the precision of the Che approximation is often in the same range. The largest deviations are encountered for small cache sizes $M$ and for $\beta \rightarrow 1$.

![Figure 6: Che approximation: Deviations from simulated results](image)

VI. EVALUATION FOR DYNAMIC POPULARITY BASED ON DAILY WIKIPEDIA PAGE REQUEST STATISTICS

A. CONTENT POPULARITY DYNAMICS OVER TIME

In contrast to the IRM assumption, many measurement studies indicate how the popularity of Internet content varies over time

- for user-generated content, e.g., videos [9][19][27][28],
- especially for file sharing systems [33], and
- for IP-TV applications [4][25].

327
Some interesting conclusions can be drawn from these papers:

- The temporal dynamics in object popularity are noticed on the timescales of days, weeks, or months [4]. A study on the effect of dynamics for caching based on YouTube video traces [28] concludes that request correlations on time scales of a few hours can be ignored. Instead the time scale of a few days/weeks is experienced as most important.

- The popularity evolution of each object is characterized by a rapid growth towards maximum popularity followed by a phase of slow decrease [33]. When the uploaded content is young, significant rank changes are encountered mainly from low initial popularity, stabilizing at the maximum.

- Studies on P2P and IP-TV systems indicate content popularity dynamics to be relatively low. Only 1-3% daily drift in the popularity of the top 1000, 1000 and 10 000 Gnutella files has been observed in [33]. The measurement study [25] reports that the cosine similarity value between the popularity of individual TV channels over 3 days is about 0.97, corresponding to a fairly stable ranking of the content.

In principle, unpredicted changes in content popularity make caching less efficient, if such dynamics is high enough to render content useless only a few requests after being loaded into the cache. The effect of popularity dynamics on cache hits mainly depends on the user population served by the cache and its request frequency, which can count into millions per day.

In fact, we experience higher dynamics in the day by day Wikipedia request pattern than the previously referred studies on video, P2P and IP-TV platforms, see Section V.I.D for more details. Nevertheless, Zipf distributed requests and the conclusions on caching efficiency of the previous sections are confirmed to be still relevant.

B. Daily Wikipedia top-1000 statistics & Zipf request pattern

For realistic access pattern of a popular web platform, we refer to statistics being published by Wikipedia [31]. Even if the data volumes are smaller than for popular video streaming platforms, we expect similar dynamic Wikipedia page request pattern. We evaluate daily statistics of the number of requests to the top-1000 pages, which are provided since August 2015.

In a first step, we adapt Zipf distributions to daily top-1000 request distributions. We measure the deviation of the Wikipedia top-1000 requests for the cumulative distribution function \( W_{\text{CDF}}(k) = R_k/R(1000) \) and a Zipf adaptation \( Z_{\text{CDF}}(k) \) due to eq. (1-2), where \( R_k(k) \) is the sum of requests to the top-\( k \) pages \( k = 1, \ldots, 1000 \) at day \( d = 1, \ldots, 184 \) for the 6 month period from Aug. 2015 - Jan. 2016. Our measure of deviation \( \Delta_{\text{w-z}}(k) \) is defined by the difference in the ranks, for which both CDF distributions achieve the same level, i.e. we define

\[
\Delta_{\text{w-z}}(k) = j_z - k \quad \text{where} \quad Z_{\text{CDF}}(j_z) \leq W_{\text{CDF}}(k) < Z_{\text{CDF}}(j_z + 1).
\]

Therefore, the Zipf parameter \( \beta \) is determined in order to minimize the mean absolute rank deviation \( \langle |\Delta_{\text{w-z}}(k)|/1000 \rangle \). We found the minimum according to the considered deviation criterion always in the range \( 0.5 < \beta < 0.6 \) except for two days in the time frame with larger \( \beta \) up to 0.75.

Figure 7 shows the minimum, maximum and mean absolute rank deviations of Zipf distributions adapted to each daily top-1000 request statistics for optimum \( \beta \). All rank deviations are in the range \([-27, 9]\) and mean absolute rank deviations are always less than 10. On 4 out of 184 days, the Wikipedia top-1000 requests can be perfectly matched by a Zipf distribution with rank deviations of only \(-1, 0\) or \(1\) on all 1000 ranks.

The largest deviations from Zipf distributions are experienced in the top-10, whereas the tail of the top-1000 distributions can be closely fitted. This is due to a large statistical variation of requests especially for the top-1 page, ranging from \(3 \cdot 10^7\) to \(6 \cdot 10^8\) requests, while the total number of daily top-1000 request is less variable between \(2.4 \cdot 10^7\) and \(3.4 \cdot 10^7\) when we consider the distribution given by the mean number of top-\( k \) requests over increasing time periods, then we experience a convergence very close to a Zipf distribution.

Figure 7: Zipf adaptations to daily Wikipedia page requests

C. Daily Wikipedia top-1000 statistics & IRM cache hit rates

In a first evaluation of caching efficiency based on the Wikipedia data, we assume independent requests per day with request probabilities according to the top-1000 request frequencies and separated simulations for each day.

Figure 8 shows SG-LRU results for several fading factors \( \rho \) and for cache sizes 25 and 200, respectively. Although the variability in the daily hit rates is high, from 4.3%–24.9% for LRU with \( M = 25 \) and from 29.3%–54.9% with \( M = 200 \), the potential gain for SG-LRU is almost constant each day close to the maximum LFU hit rate under IRM conditions. The mean LRU hit rates over 184 days are 8.2% for \( M = 25 \) and 36.5% for \( M = 200 \), as compared to mean SG-LRU hit rates of 20% (\( M = 25 \)), \( \rho = 0.9999 \)) and 49.4% (\( M = 200 \), \( \rho = 0.99999 \)). This confirms the 10%-20% gain similar to the results in Figure 5 also for a real request pattern due to Wikipedia statistics. For those evaluations, all requests are assumed to address top-1000 web pages of a day. With regard to a 10-fold higher number of requests to all Wikipedia web pages, larger cache sizes are required in order to achieve the same hit rates.

D. Cache simulation under daily changing request pattern

From the daily top-1000 page request statistics we can also gain insights into the dynamics of page popularity. The rate of change in the top-\( k \) pages is expressed in Table 1 by the fraction of requests addressing top-\( k \) pages of the previous day.
The evaluation is again based on 184 days from Aug. 2015 - Jan. 2016. At least more than half of the requests for \( k = 25 \) and even 75% for \( k = 1000 \) are referring to yesterday’s top-\( k \) pages. On the other hand, 20% - 24% of the requests are for new pages which didn’t appear among yesterday’s top-1000.

We still apply the SG-LRU caching strategy with geometrical fading scores. In a range \( 500 \leq N_c \leq 1000 \), the SG-LRU hit rate doesn’t approach the LRU hit rate for \( M = 25 \) even up to \( 50\% \) of the daily Wikipedia requests. In fact, the considered population and the generated number of requests \( R_d/N_c \) per cache per day is a main characteristics for the efficiency. When \( R_d/N_c \) is smaller, then the dynamics due to daily changes is becoming more relevant.

We still apply the SG-LRU caching strategy with geometrical fading score function. The results shown in Figure 10 fall into three different categories depending on \( N_c \):

- In the range \( N_c \leq 200 \) (for \( M = 25 \) up to \( N_c \leq 1000 \)), the LFU hit rate limit for independent requests (IRM) is closely approached by SG-LRU with appropriate fading factor \( \rho \).
- In a range \( 500 \leq N_c \leq 10000 \), the SG-LRU hit rate doesn’t exploit the LFU limit under IRM conditions, but still significantly improves over pure LRU.
- In a range \( N_c \geq 20000 \), SG-LRU does not essentially improve pure LRU hit rates for geometrical fading scores.

Note that ranges for \( N_c \) can also be expressed by the number of requests per cache per day via \( R_d/N_c = 2 \times 10^7 \). As a difference compared to the static IRM case, SG-LRU hit rates are now increasing up to an optimum \( \rho_{opt} \) and decreasing beyond \( \rho_{opt} \). In the case \( N_c = 10000 \) and \( M = 25 \), we observe SG-LRU hit rates going up to 16.6% for \( \rho = 0.999 \) and then falling down to 11.4% already for \( \rho = 0.9999 \). With dynamically changing request pattern, we experience SG-LRU hit rates \( h(\rho) \) generally as a function of \( \rho \) starting to increase from LRU for \( \rho < 0.5 \) towards an optimum \( \rho_{opt} \) with maximum hit rate \( h_{max} \) and then falling for \( \rho > \rho_{opt} \) down to hit rates even below the LRU hit rate. This behaviour allows to automatically determine \( \rho_{opt} \) and the maximum hit rate. For further study, we are testing more sophisticated and further optimized score functions, trying to predict rising popularity of new pages.

Table 1: Fraction of requests to yesterday’s top-\( k \) pages

<table>
<thead>
<tr>
<th>Top-( k ) pages: ( k )</th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y’s Top-( k ) → Top-( k )</td>
<td>54.7%</td>
<td>58.8%</td>
<td>62.4%</td>
<td>66.7%</td>
<td>71.9%</td>
<td>76.1%</td>
</tr>
<tr>
<td>1 − (Y’s Top-1000 → Top-( k ) )</td>
<td>22.6%</td>
<td>21.8%</td>
<td>21.2%</td>
<td>20.8%</td>
<td>20.9%</td>
<td>23.9%</td>
</tr>
</tbody>
</table>

Figure 8: Cache hit evaluation for top-1000 daily requests

Figure 9: Changes in the daily top-200 request statistics

Figure 10: Cache hit rate for daily changing request pattern
For dynamics on the time scale of hours, the Wikipedia statistics has no data available per page. Therefore we leave an analysis in smaller time scales open for future work based on more detailed alternative traces. If we interpolate a daily change in the requests to a page from $R_i$ to $R_{i+1}$ over the hours as a monotonous trend, then the dynamics is smoother, leading to improved caching efficiency as compared to a hard shift from $R_i$ to $R_{i+1}$. From our current experience, we agree to [28] that dynamics in the time scale of days/weeks is more relevant.

CONCLUSIONS AND OUTLOOK

We devoted detailed simulation studies to the efficiency of caching for usually observed Zipf distributed request pattern. The least recently used (LRU) caching strategy is compared to score gated SG-LRU methods, combining low LRU update effort with flexible score-based selection of the cache content.

In a first part on the standard model of independent (IRM) Zipf distributed requests, we confirm that LRU hit rates in the range 10-50% generally leave further 10-20% absolute hit rate potential unused compared to SG-LRU, as also shown in several measurement studies for more complex caching strategies [20].

In a second part, we include dynamic popularity changes based on Wikipedia page request statistics, which again exhibit Zipf-like request pattern. Although >20% new pages appear among the top-1000 pages every day, the cache hit rates for Wikipedia pages are still close to IRM conditions for the Zipf-like daily request distribution when a cache serves a large population, i.e. when a cache handles at least 50 000 requests per day. Dynamic request pattern have more impact on the performance of smaller caches, leading to less homogeneous results also depending on local environments and user preferences.

On the whole, the results indicate that caching efficiency is not restricted to the prevalently considered LRU case, but can go up to the essentially higher LFU hit rate limit for Zipf distributed requests. Including statistics about past requests by the proposed SG-LRU method provides a simple extension to exploit the hit rate potential beyond pure LRU for IRM and for more realistic dynamic content popularity scenarios. For future work, we plan to optimize SG-LRU score functions for request pattern based on an extended set of web access measurements.

ACKNOWLEDGEMENTS

This work has received funding from the European Union’s Horizon 2020 research and innovation programme 2014-2018 under grant agreement No. 644866 and within the EU COST ACROSS activity 1304. This work reflects only the authors’ views and the European Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[12] G. Hasslinger and F. Hartleb, Content delivery and caching from a network provider’s perspective, Special Issue on Internet based Content Delivery, Computer Networks 55 (Dec. 2011) 3991-4006
[18] D. Lee et al., LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies, IEEE Transactions on Computers 50/12 (2001) 1352-1361
[23] PeerApp Inc. <www.peerapp.com>
[27] G. Szabo and B.A. Huberman, Predicting the popularity of online content, ACM Communications 53/8 (2010) 80-88
[29] C. Valancius et al., Greening the Internet with nano data centers, Proc. ACM CoNEXT Workshop, Rome, Italy (2009)
A Power Efficient and Robust Virtual Network Functions Placement Problem

Antonio Marotta and Andreas Kassler
University of Karlstad
Universitetsgatan 2, 65188 Karlstad, Sweden
Email: antonio.marotta, andreas.kassler {@kau.se}

Abstract—Reducing the CAPEX and OPEX is a major concern for Telecom Operators (TOs): to this extent, Network Function Virtualization (NFV) has been considered a key aspect to virtualize network functions and push them to the NFVI Infrastructure. Virtual Network Functions (VNFs) can be deployed as a set of components running on several cooperating Virtual Machines (VMs) inside modern data centers. As a consequence, it becomes crucial for network operators to minimize the power consumption of their NFVI infrastructure, by using the minimum set of physical servers and networking equipment subject to the constraints that VNFs impose on the infrastructure in terms of compute, memory, disk and network resources requirements. In this work, we present a joint resources and flow routing assignment problem for VNFs placement, with the objective of minimizing both the power consumption of the servers and switches needed to deploy the overall virtualized infrastructure and the routing graph. In contrast to many existing works assuming perfect knowledge on input parameters, such as VNFs CPU demands, which is difficult to predict, we propose a novel mathematical model based on the Robust Optimization (RO) theory to deal with data uncertainty. Our numerical evaluation focuses on a specific use-case, that is the deployment of a virtualized Evolved Packet Core (vEPC), namely the core for next generation mobile networks. We demonstrate that with our model, a vEPC operator can trade-off between two important aspects: the power consumption minimization on one side, and the protection from severe deviations of the input parameters on the other (e.g. the resources requirements).

Index Terms—Network Function Virtualization, VNF Placement Problem, Evolved Packet Core, Mixed Integer Optimization, Robust Optimization.

I. INTRODUCTION

Telecom Operators (TOs) currently use a combination of vendor specific hardware and software to implement their network functions such as Load-balancers, Firewalls, Mobility Management Entity and so on. With the advent of Cloud Computing, those functions are increasingly virtualized, leading to the concept of Network Function Virtualization (NFV) [1]. This is a promising technology which leverages the benefits of virtualization in the telecommunication realm. The purpose is to offer telecommunication features as a service, so that mobile network operators can take advantages of the cloud paradigm. Indeed by deploying Virtualized Network Functions (VNFs) on Virtual Machines (VMs), operators are allowed to run telecommunication services inside virtualized data centers on commodity hardware with the advantage of reducing capital expenses. More importantly, by changing VMs resources dynamically (e.g. by adding more compute or memory resources), the VNFs may be scaled according to the load or new VMs can be dynamically spawned on demand, and this significantly simplifies the VFN operation and management.

In general a Virtual Network Infrastructure (VNI) consists of different VNFs interconnected by well-defined interfaces and thus creating a VNF forwarding graph (e.g. service chain). A VNF itself consists of different components (VNFCs), each one executing a clear task. VNF instances need to be deployed on VMs in the NFVI infrastructure which is composed of compute, storage and network resources. The virtualization layer maps the virtual resources to the physical servers, disks and network nodes. One of the main aspects of virtualization is the resources consolidation, since more VMs may reside on the same physical server. However, the more VMs are hosted on the same physical machine, the higher the potential for contention and, thus, the possibility of SLA violations. On the other hand, deploying each VNFC on a different server may result in high energy consumption, which may be unnecessary during low load. From a NFVI infrastructure point of view, it is obvious to pursue the objective of saving as much energy as possible, in order to reduce the electricity costs and the CO\textsubscript{2} footprint, leading towards the concept of Green NFVI Infrastructure operation. In order to maximize the energy efficiency, TOs need to place as many VNFCs as possible on the smallest set of physical servers, by trying to guarantee the expected quality of service to the final users. In the literature, several optimization models have been proposed to solve different versions of the VNFs placement problem. A common assumption of those models is that input data is precisely known beforehand, which is very difficult to achieve in practice. For instance, it is hard to predict how much CPU a VNF will require or how much data VNF A will send towards VNF B during its execution time. Unfortunately, the presence of uncertain data may produce solutions to optimization problems that are useless in practice [2], [3]. This is due to the fact that small deviations in input data values may usually lead to situations where an optimal solution, previously found, is not even feasible any more. By taking into account variable CPU demands, if during runtime some VNFC components allocated on the same server require more CPU than the expected one, contention may occur leading to severe service degradation. Consequently, we need to develop models that cope with data uncertainty by applying e.g. Stochastic Programming or Robust Optimization (RO).

In this paper, we present an optimization model based on
RO to solve the joint resources and flow routing assignment problem when designing a VNI, with the objective of minimizing both the power consumption of the servers and switches needed to deploy each VNF under uncertainty. In particular we are considering uncertainty on the VNFC resource demands, since it is likely that the CPU utilization may vary according to the VNF processing load. We model the service chain as a set of communicating VNFCs, with traffic demands and latency constraints. Our model allows a TO to protect against parameter deviations, by avoiding overloading situations that are a result of aggressive VNF consolidation but, at the same time, incurring in the so-called price of robustness [4]. Our numerical results demonstrate that it is possible to achieve a trade-off between the power consumption and the protection from deviations in the CPU utilization that may lead to low and unexpected QoS levels. By taking into account more severe and unlikely deviations of VNFCs requirements, the model will provide a higher protection but also a higher power consumption. Alternatively, a more risky placement will offer less protection at a lower power consumption. Our model can be used by a VNI operator to balance between the two conflicting goals, according to the sensitiveness of the VNF to resources contention. In our numerical evaluation we consider a virtual Evolved Packet Core as use-case, namely the cornerstone of next generation mobile networks.

The rest of the paper is structured as follows: in section II the related work on the VNFs placement problem is discussed. In section III, the RO theory is introduced; while in section IV the problem formulation is described, alongside with the robust model. In section V the use case for the experimental evaluation is illustrated and some numerical results for the vEPC case are presented. The last section concludes the paper and lists our future work.

II. RELATED WORK

The VNFs placement is a well-studied problem in the literature due to its importance for Telecom Operators. In [5], the authors present an optimization model for the embedding of Virtual Mobile Core Networks. They face the problem of resource allocation for a core network service chain which is intended as a combination of VNFs that the user or control plane related traffic needs to traverse. In their formulation latency is nicely modelled as a combination of processing, packet queuing and propagation delay, where the first two variables depend on the traffic utilization of the node the VNF is placed on, while the last one is a function of the path length. Authors show numerical results of the model on a real network topology. However, in their assumption input parameters are precisely known in advance and they do not study the impact of such uncertainty. [6] presents an Integer Linear Programming (ILP) model for VNF orchestration. The problem consists in finding the number of necessary VNFs and allocating them in order to minimize the total network related cost and the resources fragmentation. The model is optimally solved, even if a dynamic programming based heuristic is also used for large instances. In [7] an interesting approach is applied to the aim of reducing the energy consumption in telecommunication networks, by consolidating the available resources. A game theoretic based procedure is applied to drive the resource consolidation and achieve a good trade-off between energy efficiency and network resiliency. The work in [8] investigates the benefits of the application of two approaches to the telecommunication networks: the network functions virtualization on one hand, and the Software Defined Networking paradigm for functions decomposition on the other. A model to solve the VNFs placement is proposed, whose objective is to minimize the total network load overhead, by taking into consideration several parameters, such as the data plane delay and the SDN control overhead. In [9], two constraint-based heuristics are applied for the deployment of a virtualized infrastructure providing Evolved Packet Core services. The authors discuss all the involved parameters to determine the components needed to run the VNF and they show the results in terms of average number of used CPU cores and aggregate throughput for two placement strategies. In all those models and algorithms, perfect knowledge on all the parameters is assumed and optimal values are computed based on such precise input assumptions. However, if input parameters later vary the optimal solution previously found may be totally infeasible, by making those approaches impractical. We address those deficiencies in our approach by modelling the VNFs placement problem using Robust Optimization. We also consider the network topology in the model, and we model the collaborating VNFCs as a service chain. By varying the protection level, the model is able to output different solutions that trade off between the total power consumption of physical resources and protection against SLA violations (for instance due to resource contention or link overloading). The theory of RO has been already applied in a different context successfully. For example, in Chaisiri et al. [10], authors present a robust cloud resource provisioning approach (RCPRA) that considers fluctuation in resource demands and cloud providers resource price. In [11] RO theory is applied to a well-known problem, namely the Virtual Machines Consolidation. [12] present an approach to mitigate network jammers in the wireless context based on a very interesting extension of RO that allows to model multi-band robustness. Also in [13] the RO theory is applied for solving the Virtual Network Embedding (VNE) problem on a physical network substrate. The objective is to maximize the revenue that comes from the embedding of virtual nodes and links with a constraint on the capacity budget. In order to solve large instances, they propose a very interesting two phase heuristic which is based on Γ-robustness to deal with capacity requests variability.

III. INTRODUCTION TO ROBUST OPTIMIZATION THEORY

Robust Optimization [2], [3], [4] tries to mitigate problems that arise in optimization under input data uncertainty or deterministic variability. One difference to stochastic optimization is that the probability distribution of the parameters uncertainty is not known beforehand: instead, it is specified by a so-called uncertainty set, representing the parameters space which the problem is optimized over. According to [3] we can write an
uncertain linear optimization problem as:

\[
\min \mathbf{c}^T \mathbf{x} \\
\text{s.t. } A \mathbf{x} \leq \mathbf{b}, \quad \forall \mathbf{x} \in X
\]

(1)

where \( \mathbf{x} \in \mathbb{R}^n \) is the vector of decision variables and \( X \) is a deterministic polyhedron. The uncertain parameters may assume arbitrary values from a given uncertainty set \( \mathcal{U} \). The aim is to find the minimum cost solutions \( \mathbf{x}^* \) among all feasible ones for any possible realization of the unknown coefficients. In other words, the constraints need to be satisfied for all the possible values out of the given uncertainty set \( \mathcal{U} \).

The problem can be translated into the robust counterpart as in [14]:

\[
\min \mathbf{c}^T \mathbf{x} \\
\text{s.t. } A \mathbf{x} \leq \mathbf{b}, \quad \forall \mathbf{a}_1 \in \mathcal{U}_1, \ldots, \mathbf{a}_m \in \mathcal{U}_m, \quad \mathbf{x} \in X
\]

(2)

where \( \mathbf{a}_i \) is the \( i \)-th row of the uncertain matrix \( A \), taking values from the uncertainty set \( \mathcal{U}_i \subseteq \mathbb{R}^n \). We call a solution robust feasible if it satisfies all the uncertain constraints \( \mathbf{a}_i^T \mathbf{x} \leq \mathbf{b}, \forall \mathbf{a}_i \in \mathcal{U}_i \) and any optimal solution of (2) is called a robust optimal solution. The robust counterpart (2) has typically infinitely many constraints and provides solutions that are worse than the ones provided by the original (non-robust) problem, since RO tries to mitigate the effects of uncertainty.

An important aspect of the robust optimization is the definition of the uncertainty set. Bertsimas and Sim [4] considers an uncertainty set that allows to specify a sort of uncertainty budget \( \Gamma \geq 0 \). For a given uncertain matrix \( \mathbf{A} = (a_{ij}) \) we can assume that each coefficient \( a_{ij} \) has a nominal value \( \bar{a}_{ij} \) and a possible symmetric maximum deviation \( \tilde{a}_{ij} \geq 0 \), thus lying in the interval \( [\bar{a}_{ij} - \tilde{a}_{ij}, \bar{a}_{ij} + \tilde{a}_{ij}] \). We allow that at most \( \Gamma_i \) coefficients of row \( i \) may deviate from their nominal value and \( \Gamma_i \) denotes the budget of uncertainty of constraint \( i \). Then, we can define the robust uncertainty set as all coefficients for which the sum of the relative deviations from the nominal values is at most \( \Gamma_i \). More formally, given the parameter \( a_{ij} \), we define \( z_{ij} = (a_{ij} - \bar{a}_{ij})/\bar{a}_{ij} \) and we require that

\[
\sum_{j=1}^{m} |z_{ij}| \leq \Gamma_i \quad \forall i,
\]

(3)

\( \Gamma_i \) defines the maximum number of parameters, whose values deviate towards their maximum value. When \( \Gamma_i = 0 \), all the parameters are at their nominal values and the solution is not protected against any uncertainty. On the other hand, if \( \Gamma_i = n \), the \( i \)-th constraint is fully protected against uncertainty, leading to the most conservative solution. Any trade-off in between is possible: by tuning \( \Gamma_i \), we can now obtain more robust solutions characterized by higher \( \Gamma_i \) and leading to worse objective function values, but, at the same time, protecting from more parameter deviations. Or otherwise we can accept more opportunistic solutions with a lower \( \Gamma_i \), leading to better objective values, but also to a higher risk. This uncertainty model allows to compute an upper bound to the probability of constraints violation as shown in [4]. Consequently, a robust solution remains feasibly with a high probability and \( \Gamma_i \) controls the trade-off between the constraint violation probability and the impact on the objective function.

IV. Problem Formulation

Our main goal is to derive a mathematical optimization model for the robust VNFs placement problem, where several parameters are not known precisely. The objective is to minimize the overall energy consumption of the VNI in terms of power needed to drive the physical servers, switches and other networking equipment. This will be achieved by placing the VNFs on the smallest set of physical servers and powering down unused switches, switch-ports and physical servers. An important aspect is that too many VNFs should not be allocated on the same server as we need to take into account the uncertainty in demand fluctuations of the components, in terms of compute resources. In order to derive such a model, we need to look into the power consumption models for the physical infrastructure in a data center.

A. Server and Switch Power Models

Most part of the whole VNI power consumption is due to physical servers that run the VNFCs. The servers power consumption depends on several factors such as CPU load, memory, cache states, and so on. As stressed out by several papers in the literature, the most influential subsystem on the server’s power consumption is the CPU. When a server is powered on but not experiencing any load, it consumes \( P_{idle} \).

Hence, the power consumption can be simplified as:

\[
P_j(t) = P_{idle} + \left( P_{max} - P_{idle} \right) \cdot \text{used}_{CPU}^j(t)
\]

(4)

where \( \text{used}_{CPU}^j \) is the usage (in percentage) of the processor (value between 0 and 1). At each time instant the power consumption is linearly increasing with the CPU utilization, due to the running tasks on the physical machine. If the physical server does not run any VNF, we power down that server.

In our work we also model the energy consumption of the network that connects the physical servers running the set of VNFs. As a consequence, we need to model the switches power consumption, which can be expressed as [15]:

\[
P = P_{ch} + n_c \cdot P_c + \max_{r=1}^{\max_s} n_{pr}^r \cdot P_{pr} \cdot u_p
\]

(5)

The power consumption of a switch is given by:

- a static power consumption related to the chassis, namely the frame for mounting the circuit components, \( P_{ch} \), and a number \( (n_c) \) of line cards providing the network interfaces.
- the power consumption by a number \( (n_{pr}^r) \) of powered on ports operating at a specific rate and characterized by a total utilization \( u_p \).

We simplify (5) to obtain the switch power consumption as:

\[
P = P_{static} + \sum_{p \in Q_{active}} P_p
\]

(6)
where $P_{\text{static}}$ is the static power component and the sum involves the power of all the switch active ports. We assume that if a port is not carrying any traffic, we can power down it. Finally, if all the ports are powered down, we can power down the whole switch.

**B. Problem Definition**

As introduced in I, a VNI can consist of different cooperating VNFs, whose functionality is accomplished through different VNFCs, each one with a well defined interface. We are considering a group of Service Chains (SCs) to model a single VNF: each service chain is a group of VMs that communicate among each other, with a certain amount of traffic. In order to provide service guarantees, we associate a certain maximum tolerable latency for each service chain. In order to illustrate this concept, assume a generic VNF which is composed of two service chains as it follows:

$$VNF_1 = \{sc_1, sc_2\}$$
$$SC_1 = \{vm_{m1}, vm_{m2}, vm_{m3}\}$$
$$SC_2 = \{vm_{n4}, vm_{n5}\}$$

Demands $= \{(vm_{m1}, vm_{m2}), (vm_{m2}, vm_{m3}), (vm_{n4}, vm_{n5})\}$

Latencies $= \{(vm_{m1}, vm_{m2}), (vm_{n4}, vm_{n5})\}$

The first SC contains three VMs and it is characterized by two traffic demands; the second one is composed by two VMs and has a single traffic demand. For each service chain we define the maximum latency which is the latency for packets sent by the first VM and forwarded to the last one in the same chain. In our example, the first SC has one traffic demand between VM1 and VM2 and another between VM2 and VM3, but the latency constraint is applied to any combination of paths the traffic originating in VM1 can be sent over in order to reach VM3. Consequently, we need to find the set of links in the communication graph that fulfill the latency constraint given by the service chain.

We suppose that the cloud environment consists in a number of $s$ physical machines, $v$ VNFs to deploy, $s$ service chains and $m$ VMs where to run the different VNF components. Each physical server can be connected to a particular node in the network topology, consisting of $n$ nodes (e.g. switches). The static allocation of each server to a network node in the topology is given by the binary matrix $al$. A node in the topology is considered as a switch with a specific power model and a set of links, each of which has a power consumption, an expected latency and a maximum bandwidth. Given these assumptions, we tackle the following **Power Efficient Robust VNFs Placement Problem**:

Given the amount of resources available at each server, the amount of CPU and RAM requirements for the VNF components, the CPU uncertainty model and the power profile of the servers and switches, the problem consists in finding the placement for the VNFs and jointly the traffic flows routing that minimize the total power consumption due to the active servers, switches and links.

The objective function of the problem is:

$$f = \sum_{j \in J} P_j + \sum_{n \in N} P_n$$

The placement must guarantee that the used resources on each physical server do not exceed the available amount, the traffic on the links should not be greater than the available bandwidth and the latency constraint for each service chain is respected.

**C. Robust VNFs Placement Model**

The problem has been modelled as an optimization model with robustness constraints on resources requirements and traffic demands. All the input parameters and the decision variables are summarized in Table I. The overall model is shown in Table II.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{ij}$</td>
<td>the amount of resource $i$ available at server $j$</td>
</tr>
<tr>
<td>$sc_{nm}$</td>
<td>is 1 if the VNFC $m$ belongs to the service chain $s$</td>
</tr>
<tr>
<td>$r_{im}$</td>
<td>is the amount of resource $i$ requested by VNFC $m$</td>
</tr>
<tr>
<td>$\Delta r_i,m$</td>
<td>is the max variation in the usage of resource $i$ by VNFC $m$</td>
</tr>
<tr>
<td>$P_{\text{idle},n}$</td>
<td>is the static power consumption of node $n$</td>
</tr>
<tr>
<td>$P_{\text{idle},j}$</td>
<td>is the idle power consumption of server $j$</td>
</tr>
<tr>
<td>$P_{\text{max},j}$</td>
<td>is the maximum power consumption of server $j$</td>
</tr>
<tr>
<td>$e(s, d, b, pw, \text{lat})$</td>
<td>is the source, destination, max bandwidth, power consumption and latency of link $e$</td>
</tr>
<tr>
<td>$d_{m1,m2}$</td>
<td>is the traffic demand between $m1$ and $m2$</td>
</tr>
<tr>
<td>$\text{lat}_s$</td>
<td>is the maximum latency which can be tolerated by service chain $s$</td>
</tr>
<tr>
<td>$al_{jn}$</td>
<td>represents which network node $n$ the server $j$ is connected to</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>is the protection level from parameters deviation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{jm}$</td>
<td>is 1 if VNFC $m$ is allocated to server $j$</td>
</tr>
<tr>
<td>$\text{act}S_{rfj}$</td>
<td>is 1 if server $j$ is active, 0 otherwise</td>
</tr>
<tr>
<td>$Uncr_{\Delta r_i,m}$</td>
<td>is the uncertain usage of resource $i$ by VNFC $m$</td>
</tr>
<tr>
<td>$P_{f,rj}$</td>
<td>is the power consumption of server $j$</td>
</tr>
<tr>
<td>$P_{f,n}$</td>
<td>is the power consumption of node $n$</td>
</tr>
<tr>
<td>$f_{m1,m2}^n$</td>
<td>is the flow demand $d_{m1,m2}$ on the link $e$</td>
</tr>
<tr>
<td>$h_{m1,m2}^e$</td>
<td>is 1 if the link $e$ is carrying the demand $d_{m1,m2}$</td>
</tr>
<tr>
<td>$H_e$</td>
<td>is 1 if the link $e$ is used for any traffic</td>
</tr>
<tr>
<td>$F_e$</td>
<td>is the total traffic on the link $e$</td>
</tr>
<tr>
<td>$\text{act}Sw_{mn}$</td>
<td>is 1 if node $n$ is active, 0 otherwise</td>
</tr>
</tbody>
</table>

Table I: Model Parameters

Each server is connected to a network node: this is specified through the binary variable $al_{jn}$. A server is active if and only if it is hosting at least one component belonging to a VNF after the placement. The binary decision variable indicating if the server hosts any VNFCs is $\text{act}S_{rfj}$. (1 means the server is up, 0 the server is shut-off). Similarly, we use a binary decision variable, $\text{act}Sw_{mn}$, to describe that a network switch is active, meaning at least one of its links is actually carrying traffic.
The Optimization Model

\[
\begin{align*}
\min f &= \sum_{j=1}^{J} P_{j,j} + \sum_{n=1}^{N} P_{j,n} \\
\text{s.t.} & \\
util_{j} &= \sum_{m} x_{jm} \cdot (r_{im} + \text{Unc Req}_i \cdot \Delta r_{im,j}) \quad \forall j, \forall i \\
\sum_{m} \frac{\text{Unc Req}_i \cdot \Delta r_{im,j}}{\Delta r_{im}} &\leq \Gamma \\
\forall j, i = CP \\
\sum_{j} x_{jm} &= 1 \quad \forall m \\
act Sr_{j} &\leq \sum_{m} x_{jm} \quad \forall j \\
act Sr_{j} &\geq x_{jm} \quad \forall j, \forall m \\
util_{ij} &\leq a_{ij} \cdot act Sr_{ij} \quad \forall j, \forall i \\
\forall n \\
P_{j,n} &= \text{P}_{\text{idle}_n} + \text{act Sr}_{n} + \sum_{e : s = n} \text{H}_{e} \cdot e.pw \quad \forall n \\
\sum_{e : d = n} \text{P}_{e}^{m_1} - \sum_{e : s = n} \text{P}_{e}^{m_2} &= d_{m_1,m_2} \cdot \sum_{e} ((x_{jm} \cdot a_{jn}) - (x_{jm} \cdot a_{jn})) \quad \forall n, m_1, m_2 \\
h_{e}^{m_1,m_2} &\leq (2 - \sum_{j} (x_{jm} \cdot a_{jn}) - \sum_{j} (x_{jm} \cdot a_{jn})) \quad \forall n, e, m_1, m_2 \\
h_{e}^{m_1,m_2} &\leq \sum_{j} (x_{jm} \cdot a_{jn}) \quad \forall n, e, s = n, m_1, m_2 \\
h_{e}^{m_1,m_2} &\leq \sum_{j} (x_{jm} \cdot a_{jn}) \quad \forall n, e, d = n, m_1, m_2 \\
F_{e} &= \sum_{m_1,m_2} \text{P}_{e}^{m_1,m_2} \quad \forall e \\
F_{e} &\leq e.b \cdot \text{H}_{e} \quad \forall e \\
h_{e}^{m_1,m_2} \cdot d_{m_1,m_2} &= \sum_{e} \text{P}_{e}^{m_1,m_2} \quad \forall e, m_1, m_2 \\
act Sr_{n} &\leq \sum_{e : s = n} \text{H}_{e} \quad \forall n \\
act Sr_{n} &\geq \text{H}_{e} \quad \forall n, e.s = n, e.d = n \\
\sum_{m_1,m_2} d_{m_1,m_2} \cdot h_{e}^{m_1,m_2} &\leq \text{lat}_{s} \quad \forall s, \forall m \in \text{service chain } s \\
x_{jm} &= \{0,1\}, act = \{0,1\}, act_n = \{0,1\} \\
h_{e}^{m_1,m_2} &= \{0,1\}, \text{H}_{e} = \{0,1\} \\
\end{align*}
\]

Table II: Problem Model

We consider a set of \( m \) possible VNF components, each of which can be deployed inside a single VM. Each server and switch has a power consumption model whose description is given in section IV-A. The network topology consists of a set of links connecting the switches, with a given capacity, a maximum latency and an estimated power consumption. All the components belonging to a service chain are exchanging traffic according to a given pattern and have specific resource demands. A specific traffic demand is traversing all the VNFs and each service chain has a maximum latency bound. To cope with the uncertainty of the resource demands of a given service chain, we assume that the resource demands needed by a VM running a given VNF component are not precisely known. Rather, we assume that we know a mean resource demand and a maximum allowed deviation from the mean, specified as symmetrically distributed upper and lower bounds, as specified for the coefficients \( a_{ij} \) in section III. In more detail, we consider that a VM which is running a specific VNF \( m \) requires an expected demand \( \Delta r_{i,m} \) of resource \( i \) (e.g. memory or CPU). We model the uncertainty of the demand by introducing a random variable \( \text{Unc Req}_i \cdot \Delta r_{i,m} \), which is symmetrically distributed between \( [\Delta r_{i,m} - \Delta r_{i,m}] \cdot \Delta r_{i,m} \) and with 0 mean.

The objective to minimize the total power consumption due to the active server, switches and links after the placement is computed in (13). Each server shows a resource utilization which is computed by considering all the requests by all VNFCs placed on it and the uncertainty on the demands (14). The constraint (15) assures that the sum of the deviations of the uncertain resource (over all the possible VNFCs) should not exceed the protection level \( \Gamma \). The server power profile is computed through (16): if the physical machine is not hosting any VNFCs, it can be shut-off and its power consumption is set to 0. Otherwise the power consumption is linearly increasing with the CPU usage (14). The binary decision variable \( x_{jm} \) represents the placement of the VNF \( m \) on the physical server \( j \). The model assures that each VNF \( m \) is placed on a single server (17). Constraint (18) guarantees that the server is shut-off, if it is not hosting any VNFCs; on the other side, if at least one VNF is placed on it, it should be active (19). The constraint (20) avoids that the used resources on a server exceed the maximum available amounts.

The power consumption of the switch is computed as in (21). If the switch is powered on, its power consumption is the static component plus the dynamic one which is dependent on the active outgoing links. If the switch is not used, then the power is set to 0. (22) is the so-called flow conservation constraint: the sum of the flows entering one node should be equal to the sum of the ones exiting from the node (except for the source and the sink). In other words, for each traffic demand and network node, if the flow is generated by the considered node and directed to a different one, then the demand is multiplied by -1 (the traffic source is \( m_1 \)). Otherwise, when the node is attracting the flow, the demand is multiplied by 1. For intermediate nodes, the difference between the sums should be zero. Constraint (24) assures that, given a demand between \( m_1 \) and \( m_2 \), if both the source and destination are placed on a server which is connected to the same node, this traffic is not flowing into the network. Considering a link, its source node \( n \) and a demand between \( m_1 \) and \( m_2 \), if \( m_1 \) is not allocated to any server connected to \( n \), the flow exiting from the node should be zero (25). The same happens for incoming links (26). The total flow on a link \( e \) is the sum of all the demands forwarded through it (27), and it should be not greater than the available capacity.
of the active link (28). If a link \( e \) is used to carry traffic between \( m_{12} \) and \( m_{22} \), then the flow \( f_{m_{12},m_{22}} \) should be equal to the demand itself; otherwise it must be zero (29). In this model we compute a single path for each traffic demand, in other words the flow is unsplitable. The constraints (30) and (31) assure that a switch is active if and only if at least one of its connecting links is used to forward traffic; if no link is active, then the switch can be shut-off. The last constraint (32) regards the latency bound for each service chain: the sum of all the path latencies used for the traffic flows of a specific service chain should not exceed the maximum tolerable one.

V. A NUMERICAL EVALUATION: THE VEPC USE-CASE

The experimental evaluation of the model was conducted on a particular example of VNF, namely the Evolved Packet Core (EPC). With the advent of cloud and virtualization, the Telecom Operators are moving parts of the next generation networks infrastructure into the cloud, leading to solutions named EPCaaS (EPC as a Service). The EPC is thus composed of different entities, each of which can be considered as a stand-alone VNFC running on a dedicated VM as shown in Figure 1. Among the components there are:

- Evolved NodeB (eNB). The user gets connected to the mobile access network after being associated to an eNB, which represents the base station for the radio technology.
- The Mobility Management Entity (MME) is the component which processes control plane related information (e.g. the signalling traffic for handling the user mobility and security for mobile network access). Another task accomplished by the MME is to keep tracking of User Equipments (UEs) in idle-mode. The MME usually consists of the Signalling Load Balancer (SLB) and the Mobility Management Processor (MMP). Their combination is useful to scale the control traffic originating in eNBs and to balance it among different MMPs. These components execute the real processing tasks of the MME.
- The gateways, Serving GW and Packet Data Network (PDN) GW, process data related to the user plane, as they forward packets from the UE towards external destinations. The S-GW represents the interconnection between the radio access and the core of the EPC and it determines the routing for the packets coming from the UEs. The Serving GW is the interconnection point between the radio-side and the EPC. As its name indicates, this gateway serves the UE by routing the incoming and outgoing IP packets. The PDN GW connects the EPC to the external world and it is responsible for routing the packets to and from the PDNs.
- The Home Subscriber Server (HSS) is the component that handles the database where all the subscribed users related data is residing. Besides the HSS has other functionalities, such as user mobility management, call session establishment and security tasks (e.g. user authentication and access authorization).
- The Policy and Charging Rules Function (PCRF) is in charge of applying policy rules in next-generation networks. It allows the creation of rules and the enforcement of decisions for the subscriber in the network. Among the tasks of the PCRF there are billing, rating, charging, session and call establishment with guaranteed quality of service.

A. Numerical Results

For the experimental evaluation we considered the placement of two VNFs, as discussed in the previous section. The experimental values used for the servers, switches, VNFCs and links are shown in Table III. For the CPU requirements uncertainty we took into account a maximum deviation, \( \Delta r_{1,m} \).
Table III: Experimental Values

<table>
<thead>
<tr>
<th>Physical CPU cores</th>
<th>18, 14, 15, 12, 5, 4, 1, 2, 2, 2, 3, 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical RAM (GB)</td>
<td>9, 5, 7.5, 5, 2.5, 2, 0.5, 1, 1, 1, 1.5, 1.5</td>
</tr>
<tr>
<td>Server Idle Power Consumption (%)</td>
<td>20, 30, 20, 30, 40, 40, 30, 30, 30, 20, 30, 30</td>
</tr>
<tr>
<td>VNFC CPU demands</td>
<td>5, 4, 4, 4, 4, 6, 4, 3, 3, 2.5, 1.5, 1.5, 2.5, 2, 3, 1.5, 1.5, 2</td>
</tr>
<tr>
<td>VNFC RAM requirements</td>
<td>2, 2, 1.5, 1, 1, 3.5, 1, 1, 1, 1.5, 1.5, 0.5, 1, 1.5, 1, 1.5, 1</td>
</tr>
<tr>
<td>Switches Idle Power (W)</td>
<td>100, 70, 140, 120</td>
</tr>
<tr>
<td>Links Power (W)</td>
<td>35.4, 7.4, 35.4, 35.4, 35.4, 35.4, 35.4, 7.4, 35.4, 35.4</td>
</tr>
<tr>
<td>Links Bandwidth (Mbps)</td>
<td>80, 80, 30, 50, 80, 30, 100, 80, 50, 100</td>
</tr>
</tbody>
</table>

range from 10% to 30% of the demand, with step 10. We increase the protection level from 0 (no protection, or in other words deterministic problem) up to the total number of VMs (we assume all VMs demands may deviate to the max). The dimensioning of the VNFs is realized through the parameters found in [9]. By considering the control plane load, measured in terms of events per hour, we compute the number of instances for each type of VNFC required to sustain the estimated hourly traffic bundle. We assume 500,000 signalling events per hour generated by UEs connected to the eNBs. In particular we used 8 eNB, 1 MPP, 1 SLB, 1 S-GWC, 6 S-GWU, 1 HSS and 1 PCRF. Since the deterministic model is already very hard to solve and the additional uncertainty even increases its complexity, we reduced the number of service chains, by considering some random set of VNFCs exchanging traffic out of all the possible combinations. To be more clear, a control plane related service chain can consist of any possible instance of the eNBs sending a certain amount of traffic to the SLB, which in turn forwards packets to the MMP and so on. For our use case we considered these values:

- 12 physical servers and 19 VNFCs;
- 4 network nodes and 10 links;
- 16 service chains and 27 traffic demands.

The model was implemented in Matlab [16] using the Robust Optimization Made Easy (ROME) [17] toolbox. ROME transforms the robust model into its deterministic form, which is then solved using CPLEX [18]. The experimental evaluation was conducted on a cloud based system (Abisko [19]), which is part of the High Performance Computing (HPC) cluster in Uppsala. It is comprised of 328 nodes (a total of 15888 CPU cores), each of which is equipped with 4 AMD Opteron 6238 (Interlagos), 12 cores, 2.6 GHz processors.

We use a very simplified network topology that is shown in Figure 2: for convenience, the bottom layer switches coincide with the top of racks where the physical servers are allocated. Hence, this figure points out the association of each server to a network node: in particular servers 1, 2, 3, 4, 8, 9, 10 are allocated to node 1, while servers 5, 6, 7, 11, 12 are associated with node 2. As we can see from Figure 3, for each demand deviation we present two graphs: the upper one shows the total power consumption of the VNI (in W, left axis) and the constraint violation probability \( Pr(\omega, \Gamma) \) for a given \( \Gamma \) (varied on x-axis). The expected power consumption is computed over all the active servers, switches and links after the placement, assuming that the CPU demands are at the mean value, while the risk adjusted power assumes that a given number of CPU requests are deviating to the maximum \( \Delta \) (e.g., \( \Gamma = 1 \), one CPU requirement deviating at the maximum \( \Delta \)). On the lower graph we plot the number of active servers, switches and links which are needed to deploy the VNI for different values of the protection level \( \Gamma \) (outcome of the placement).

Figure 3(a) shows the results for maximum CPU demand deviation equal to 10%. When \( \Gamma = 0 \), the solution of the model is the deterministic one (all input parameters are known precisely). This solution allocates all the VNFCs in the servers 1, 2, 3, 4 that are connected to the node 1 in the network topology and no traffic is going out on any link. The first three servers have a 100% CPU utilization, while server 4 has a 75% CPU usage with a total power consumption of 791 W. Since \( \Gamma = 0 \) means we do not protect from CPU demand deviations, the expected power consumption coincides with the risk adjusted one. The probability of constraint violation is 56.8%. Consequently, when considering perfect knowledge on all input parameters, there is 56.8% possibility of constraint violation if at least one VNFC component deviates from its nominal CPU demand, leading to resource overload and potential SLA violations. When \( \Gamma = 1 \), the model reallocates the VNFCs in order to block enough CPU resources on each server to accommodate one single VNFC CPU demand variation. This allocation has a total nominal power consumption of 737 W and a higher risk-adjusted power consumption of 742 W (a single CPU demand may be allowed to deviate to the maximum \( \Delta \)). When the demand of VNFC CPU requirements are varying to the max allowed, four servers are no longer...
sufficient to protect the solution. Hence, a new server is activated (11) with a CPU utilization equal to 33.33%. For all the $\Gamma$ values, the VNFCs are allocated so that no traffic flow is being forwarded in the network. Besides when $\Gamma$ increases, both the expected and the risk adjusted power consumption increase: in particular the expected power is stable after $\Gamma \geq 7$, while the risk adjusted power increases. The expected power is also changing when $1 \leq \Gamma \leq 7$, even if the VNFCs allocation always involves 5 servers. This is due to the fact that the used servers have different levels of energy efficiency.

The same trends can be also observed when we allow a larger deviation percentage. In particular when the maximum variability is 20% (Figure 3(b)), the model protects from deviations by using one more server for each $\Gamma \leq 4$. The maximum number of activated servers is achieved when the protection level is equal to 5, with a total expected power consumption of 1254 W. The number of used servers is stable when considering higher values of $\Gamma$. When the allocation is fully protected, the maximum gap between the power values is 12%. In Figure 3(c), the results for a maximum deviation of 30% are shown. The number of used servers is increased of 2 when the protection level goes from 1 to 2, with a total relative unused CPU utilization equal to 20% and an expected power consumption of 1482 W. The highest number of physical machines needed to deploy all the VNFCs and to cope with the CPU variability is reached when $\Gamma = 6$: 9 servers are needed and the total expected power consumption is 1776 W, while the risk adjusted one is 7.2% higher. Interestingly, when the maximum deviation is 30%, the energy consumption is significantly increasing. This is due to the fact that for $\Gamma = 2$, the VNFCs allocation is not suitable to avoid traffic in the network. Instead, the new allocation requires 4 demands to be forwarded in the network, leading to the activation of 3 routers (1, 2, 4) and 2 links (1-4, 4-2), with a total flow equal to 28 on them. For higher protection levels, also more servers are needed leading to further increase in energy consumption. Instead, the same network configuration is operated for higher $\Gamma$ values, even if with a different total amount of flow according to the allocation scheme.

In Figure 4(a), the total relative unused CPU is plotted: this is the sum of the available CPU of all the active servers minus the total CPU requested by the VNFCs, normalized to the total available CPU. In general, Figure 4(a) shows that when $\Gamma$ is increasing, the more the allocation sets aside spare CPU to cope with potential deviations. In particular, when the maximum deviation is 10% and $\Gamma = 2$ the relative unused CPU is growing from 5% to 9%, due to the fact that a new server was activated. Then the same servers are utilized until $\Gamma = 5$, when the server 11 is shut-off and server 5 is used instead, with a total relative unused CPU equal to 12.5%. When the allocation scheme is fully protected, the gap between the expected power and the risk-adjusted one is 6.2% (a maximum risk adjusted power of 989 W). Finally Figure 4(b) summarizes the risk adjusted power for different maximum allowed CPU demand deviations for different protection levels.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we modelled the Robust Virtual Network Functions Placement Problem by taking into consideration the physical servers and network resources along with their energy efficiency. The objective of the problem is to find the VNFs placement that jointly minimize the power consumption due to the servers and the switches needed to deploy all the required virtualized functions. The theory of Robust
Optimization was applied to the optimization problem to cope with uncertain input parameters, assuming that we have a budget of uncertainty in terms of cardinality constraints, i.e. the number of uncertain parameters that are allowed to deviate from their nominal values. Our numerical evaluation shows how a Telecom Operator can balance the energy consumption and the protection from input parameters that are deviating from their nominal values. By modifying the protection level of the solutions, the operator can calculate more conservative solutions that consume higher total energy or more opportunistic ones at lower energy consumption, having a higher risk of SLA violation. In the future we plan to apply stochastic algorithms to the problem in order to be able to solve also very large instances and develop fast solution heuristic for online optimization, which may be integrated into an ETSI MANO framework for NFV Orchestration such as e.g. Tacker which is based on OpenStack and OpenDaylight.

ACKNOWLEDGMENTS

Parts of this work has been funded by the High Quality Networked Services in a Mobile World (HITS) and the READY projects, partly funded by the Knowledge Foundation of Sweden.

REFERENCES

Elastic Network Service Provisioning with VNF Auctioning

Mathis Obadia*, Mathieu Bouet†, Vania Conan*, Luigi Iannone†, Jean-Louis Rougier†

*Thales Communications & Security
{firstname.name}@thalesgroup.com
†Telecom ParisTech, LTCI
{firstname.name}@telecom-paristech.fr

Abstract—Network Function Virtualization (NFV) is an emerging approach that has received attention from both academia and industry as a way to improve flexibility, efficiency, and manageability of networks. NFV enables new ways to operate networks and to provide composite network services, opening the path toward new business models. As in cloud computing with the Infrastructure as a Service model, clients will be offered the ability to provision and instantiate Virtual Network Functions (VNF) on the NFV infrastructure of the network operators.

In this paper, we consider the case where leftover VNF capacities are offered for bid. This approach is particularly interesting for clients to promptly provision resources to absorb peak or unpredictable demands and for operators to increase their revenues. We propose a game theoretic approach, using Multi-Unit Combinatorial Auctions to select the winning clients and the price they pay. Such a formulation allows clients to express their VNF requests according to their specific objectives. We solve this problem with a greedy heuristic and prove that this approximation of economic efficiency is the closest attainable in polynomial time and provides a payment system that motivates bidders to submit their true valuations. Simulation results show that the proposed heuristic achieves a market valuation close to the optimal (less than 10% deviation) and guarantees that an important part of this valuation is paid as revenue to the operator.

I. INTRODUCTION

The rise of Network Functions Virtualization (NFV) represents a major shift in the way telecommunication networks and services are operated and used. So far, network functions such as Firewall, Deep Packet Inspection (DPI), Intrusion Detection Systems (IDS), Video Encoding, Load Balancing, and Routing were supported by hardware middleboxes from multiple vendors. With the NFV approach [1], these functions can now be virtualized — becoming Virtual Network Functions (VNFs). These individual functions run on commodity hardware and can be chained to form complex offers, thus promising manageability and cost-efficiency. VNFs can be instantiated on Points-of-Presence (PoPs), which already host embedded cloud infrastructure next to legacy middleboxes.

Yet, NFV goes beyond a mere technical change. Network operators can leverage this approach to enrich their offer and develop new business models. We can envision network operators directly selling NFV resources as infrastructure, with clients choosing where to place their VNFs inside such network to better suit their needs. This can be described as Virtual Network Function as a Service [2]. When provisioning their own network services, the clients can estimate with some accuracy their average and peak needs, and how much gain they can expect from satisfying these demands. If they want to always satisfy all demands, over-provisioning is the norm, but the additional cost of reserving resources left mostly unused is not cost efficient. At the same time, the operator has often unsold resources that could be offered to anyone interested, using an auction mechanism with dynamic prices to sell those available resources. In such a system, clients bid on leftover resources, for instance to satisfy their peak and unpredictable demands. In this way, they only satisfy those demands when the competition for those resources is not too aggressive and the price paid remains low. Hence, from client’s perspective, the idea is to buy resources to absorb peak demand at low price, instead of over-provisioning. For the network operator, instead, this system can allow to sell resources that would otherwise be unused (generating no income). A similar auctioning system has been successfully used in the cloud environment (Amazon EC2 [3]), providing benefits to both clients and cloud operators [4].

In the NFV setting, instead Virtual Machines, clients want to acquire a set of ordered VNFs linked together called Service Function Chains (SFCs). The problem of mapping SFCs requests with respect to available network resources has been already explored in the literature, e.g., see [5]. In such formulation, to maximize network utility, two main issues need to be addressed: (i) choose which clients to serve when resources are limited (winner determination); (ii) accepted services should be placed/allocated in the network infrastructure, meeting potentially complex demands in term of service chaining (mapping).

In this paper, we tackle the problem of maximizing network operator utility in economic terms, introducing and addressing a third issue: (iii) decide how much the clients should pay (payment plan). We propose a VNF auction which, acting as an interface between the operator and the clients, is tasked to accept or reject clients’ demands and fix their price to maximize market value.

With the proposed approach, clients run their own VNF composition and optimization process, taking into account their own constraints and needs and what the operator advertises as available resources, and request VNF function chains. In order to solve the problem of winner determination and
find the payment plan, we use Multi-Units Combinatorial Auctions [6] scheme, which allow us to model the demands of a bundle of VNFs (i.e., a VNF chain) and provides: (i) the most economically efficient clients to serve, and (ii) a dynamic pricing for each service function chain as a whole.

We solve this problem with a greedy heuristic which is proved to be the best approximation attainable in a polynomial time for the winner determination problem. We also demonstrate that it provides a payment plan that is incentive compatible and strategy proof. In other words, the goal of the payment plan is to give incentives to clients to submit their true valuations as a bid. We want to avoid clients strategically placing their bids to game the system, as it could result in inefficiencies and instabilities. This is also desirable as it is fair to clients who gain no value from the additional information that they could have about their competitors and their bids [7].

The rest of the paper is structured as follows. Sec. II presents some related work. Sec. III introduces the proposed VNF auction scheme. Sec. IV provides the market model Sec. V presents the different algorithms that can be used to allocate resources and decide the price of each SFC. Those algorithms are evaluated and compared in Sec. VI. Finally, Sec. VII concludes the paper.

II. RELATED WORK

Some work have proposed approaches to find optimal solutions and heuristics for the VNF placement and chaining problem ([8], [9], [10]). They all look at the problem where demands are expressed as chains with a certain number of constraints and the operator is in charge of both the optimization and the orchestration processes. We propose a system that is similar to the cloud markets, where demands are expressed directly into infrastructural needs to the cloud operator [11].

The proposed cloud market schemes using combinatorial auctions is similar to [12], which uses double sided auctions to sell cloud resources in a marketplace similar to the electricity market, using a combination of spot and future market. Different types of cloud markets and pricing models are reviewed in [13]. Auctions have also been used for bandwidth reservation in networks [14] and can be solved using second price auctions in case where demand is elastic [15]. The main difference between our approach and these markets is that we apply this paradigm to network function embedding in a NFV setting. We want to express the fact that a service function chain is only useful if all the components are successfully bought by the client and implemented by the operator. One way to model such constraints is to use Multi-Unit Combinatorial Auctions.

Combinatorial auctions have been used in a variety of settings from spectrum auctions [16] to airport time slot allocation [17]. With this settings, a variety of single goods are sold in an auction where bidders try to acquire a bundle of goods. Combinatorial auctions are important tools of the so-called "mechanism design" theory, because they are applicable to a wide range of different settings. They still pose some computational challenges, that were addressed in [18]. This work introduces an algorithm for the case of single unit combinatorial auctions. Our approach extends their approach for multi-unit combinatorial auctions, where each goods have a certain limited quantity that can be greater than one, which is more adapted to selling network resources, such as link capacity where we don’t sell a full link but bandwidth on this link.

Works in multi-unit combinatorial auction have focused on finding the optimal allocation (winning bids) which is a computationally intensive task even with adapted heuristics ([19], [20]). Incentive compatible payment systems, with optimal winner determination, have also been found on easier instances of the problem as in [21], where each bidder is limited to buying a quantity of one unit per good.

Our work adapts auctions to the combinatorial setting of NFV and service chaining, which is the main difference from the classical cloud setting.

III. VNF AUCTIONING

A. Motivations

Because of bursty demand, over-provisioning is a necessity, but buying resources at full price in a regular fixed price market for mostly unused resources comes at a high price for clients. Our system aims at offering an alternative method to over-provisioning in order to accommodate unexpected demands, by quickly buying additional resources for a usually smaller price. We can also envision that some of the resources that were bought in the fixed price market because of over-provisioning can then be auctioned if they are left unused. In Fig. 1 we consider a simple use case with two clients who want to deploy parts of a Content Delivery Network. They both compete for the operator resources: the two clients who want to deploy parts of a Content Delivery Network. They both compete for the operator resources: the two clients are both interested by the same caching resources located in PoP B and the same link capacity between PoP A and PoP B. In the case where the operator does not have sufficient leftover resources (the lighter (blue) colored part of VNFs in the operator domain box in Fig. 1) to satisfy both clients, it
will need to make a choice based on the bid price submitted by each client. The operator cannot make decisions based on each VNF individually, because a single un-deployed VNF in the service chain makes the whole chain useless: for instance, it is useless to obtain a firewall for client 2 if he does not get the caching resources.

**Clients’ motivations**: Provisioning is a complicated process, the clients need to understand the variability of their demands, and the business objectives related to their needs. Each of the client’s users is providing a given utility (e.g. the price paid by the users) and his demands can be estimated to buy most of the fixed reserved resources in the network (the darker (blue) colored part of VNFs in the operator domain box in Fig. 1). But part of the demand is variable and unpredictable. Thus, instead of over-provisioning, the client can choose to serve the peak users only if the cost of buying additional capacity does not exceeds the utility of serving them. The client can then use this valuation of serving additional clients as the bid for the desired resources (resources which are displayed in the client bid boxes in orange and red in Fig. 1).

**Operator’s motivations**: The operator has spare resources that can be sold to a number of different possible clients. Since clients are only interested in full service chains, the auctioning system needs to let clients express their demands as bundles of resources associated with the bid price associated with the whole chain. The operator benefits from such a common format as it frees him from the need to know what are the most relevant SFCs that should be proposed to clients. The system is based on an auction mechanism: this introduces dynamic pricing with prices paid according to supply and demand, as in cloud auction systems for virtual machines. In this way the operator is able to distribute his limited resources to the clients who value them most (and are thus willing to pay more). The market decides which requests are satisfied and then the whole VNF chain is deployed in the operator’s infrastructure. For a client a request is either not satisfied (because the price to pay would be higher than the valuation associated to it), or the entire service function chain is accepted (and the price is below its valuation). This is achieved using Multi-Unit Combinatorial Auctions that take into account that each resource has a maximal capacity and that they are bought in bundles. For example, a client who submitted a bid for a video streaming service chain with 10 mb/s links and a video encoder for $100 might only pay $80 if his bid is accepted and will always pay $0 if it is not accepted.

**B. Market phases**

The problem of allocating resources is solved by using a multi-unit combinatorial auction market on a time slot basis. This means that all resources are freed after each time slot and that each time slot is considered as independent one. Service function chains required to span more than one time slot have no guarantees to win all the auctions. This does not introduce any issue in cases like time slot longer than the demanded time (which is the peak duration, in our use case), or when used to treat time shift-able demands. This system is adapted for selling unused residual resources, which were not sold with the traditional (non-auction based) long term plans offered by the operator. The two systems (high price guaranteed duration and lower price auction) need to coexist to offer clients the reliability that their service chain requires. This is similar to a cloud system such as Amazon Spot Instances that can be disconnected any time when the real time price exceeds the maximum target price chosen by the client [22]. In the same way the client is assured that he will always pay less than the bid price and will only pay for the instant dynamic price which is typically quite low but in turn is not offered a guarantee on the duration of the services supplied. We do not make any assumption on the duration of the time slot at this stage, rather focusing in this paper on what happens during each time slot.

To attribute resources for each time slot, the VNF auctioning system can be devised in five consecutive phases:

1) **Resource advertisement phase**: The operator has to advertise the VNFs that it can provide, the virtual topology of his network, e.g., the available PoPs, the different VNF availability within these PoPs and their capacity, and the capacity of the connecting virtual links. This will inform clients about what are the available resources in order for them to build the service function chains they need and can buy.

2) **Bidding phase**: The clients choose bundles of available resources that represent a service function chain. They also submit the bidding price which represents the maximum price they are willing to pay for that bundle.

3) **Winner determination phase**: After the bidding phase, the auction selects which bids are accepted and which bids are refused, because of a lack of resources.

4) **Price computation phase**: The auction will also determine how much each bidder has to pay. We assume that losing bidders pay zero and winning bidders pay a price that is less than or equal to their bidding price.

5) **VNF instantiation phase**: All previous phases have to end before the beginning of a time slot to allow service function chains to be properly instantiated.

**IV. VNF auctioning model**

We model the VNF auctioning so as to represent the information exchanged in the system: the offer from the operator and the clients’ demands are expressed as bids. Fig. 2 shows an example of a possible mapping between two service function chains from clients’ demands and the virtual NFV infrastructure of the operator.

**A. Network operator model - seller**

Network operators control the Infrastructure as a Service (IaaS) and sell directly the virtual resources of bandwidth in their links and the VNFs that they want to offer. We use the general term of service for both kind of resources:

- **Bandwidth**: the available bandwidth the operator wants to sell, expressed in Mb/s, between ingress, egress nodes and PoPs.
VNF: the available VNF functions offered by the operator. Each VNF demand is defined by a quantity of the service expressed in a unit, specified in the service description (most relevant to this particular function). For instance, DPI might be expressed in Mb/s of inspected flows. Firewalls might be expressed in number of flows/s that can be inspected.

Each of the M services \{S_1, S_2, ..., S_M\} sold in the market is specific to a particular PoP or link. For instance, a client can choose to buy a Firewall (FW) VNF in a PoP in New York or in Los Angeles. In Fig. 2 for instance, there are a total of M = 18 services offered on 5 PoPs and 6 links (S_1, S_2, S_3 in PoPA, S_4, S_5 in PoPB, etc.). The maximal available capacity for service j is C_{S_j}.

B. Client’s demand model - bidder

Each client’s demand is expressed as an abstract service function chain with some requirements on how this chain is embedded in the network. For example the service function chain of client 2 in Fig. 2 links three VNFs: Firewall (FW), Deep Packet Inspection (DPI) and Cache (CA). It can be expressed as (S8:10Mb/s)→(FW:10Mb/s)→(DPI:10Mb/s)→(CA:10Mb/s)→(egress:20Mb/s), could be instantiated in a number of different ways. It is up to the client to submit bids that are in accordance with his constraints. The proposed bid in our example is \{S8:10Mb/s, S10:10Mb/s, S12:10Go, S17:10Mb/s\}. If a client has multiple bids that satisfy his requirements, he can submit alternative bids: \{S8:10Mb/s, S10:10Mb/s, S12:10Go, S17:10Mb/s\} OR \{S8:10Mb/s, S11:10Mb/s, S12:10Mb/s, S17:10Mb/s\}. This assures the client to win only a single service function chain, while increasing his chances.

More formally we denote Q_i = \{q_1^i, q_2^i, ..., q_M^i\} the set of services demanded by bidder i, where q_M^i denotes the quantity of service S_j required by bidder i. We assume that each bidder i has a valuation \nu_i for the entire set of services \cal{Q}_i. This means that client i values the service as zero if he does not get at least the demanded quantity for each service. A bid is defined as a tuple (Q_i, b_i) where Q_i represents the services demanded and b_i represents its bid price, this price can be chosen strategically by the client to maximize his utility. The number of bidders is expressed by N.

C. Market’s properties

As seen in previous sections, the market needs to perform two tasks: winner determination and price computation. We denote W the set of accepted bids. We denote as p_i the price bidder i has to pay. We assume that losing bids do not pay anything (i.e., p_i = 0 if the bid is not accepted). Note that the price p_i is not necessarily the same as his bid price b_i. The way p_i is computed has a big influence on the strategy that bidder i will use on choosing his bid price b_i. The following are some desirable game theory properties that a market can have, but that cannot be respected all at the same time.

Incentive compatibility: the market is made in such a way that the best possible outcome for each participant is to reveal truthfully its private parameters required by the system, i.e. b_i = \nu_i, \forall i in our case. In our mechanism, the price paid by the winning bidder is always lower or equal to the submitted bid price, thereby restraining the strategic behaviors of clients trying to submit the lowest possible bid that would still allow them to win the desired service function chain. This also means that the price paid by the client does not depend on his own bid price — this price is only used to determine whether he wins or not.

Maximize operator revenue: maximizing the payments made by the winning bidders \sum_{i \in W} p_i.

Economic efficiency: In auction theory, economic efficiency refers to allocating the services to bidders who value it the most. This means maximizing the total sum of all accepted valuations \sum_{i \in W} \nu_i. Another possible objective is to maximize the social welfare, i.e. the global utility of the system, composed of the seller and all the bidders. This represents the utility of the bidders \sum_{i \in W} (\nu_i - p_i) and the utility (revenue) of the seller \sum_{i \in W} p_i. In other words, it represents the total valuation of the accepted bids \sum_{i \in W} \nu_i.

\begin{table}[h]
\centering
\caption{Notations.}
\begin{tabular}{|c|c|}
\hline
b_i & Bid price submitted by bidder i \\
\hline
C_{S_j} & Capacity of Service S_j \\
\hline
M & Number of services \\
\hline
N & Number of bids \\
\hline
p_i & Price paid by bidder i \\
\hline
q_M^i & Quantities of each services asked by bidder i \\
\hline
S & \{S_1, S_2, ..., S_M\} Services offered \\
\hline
\nu_i & Utility of bidder i \\
\hline
v_i & Valuation of bidder i \\
\hline
W & Set of accepted bids \\
\hline
x & Boolean: 1 if i is a winning bid 0 otherwise \\
\hline
\end{tabular}
\end{table}
In this particular case, economic efficiency and social welfare maximization are aligned. There is a trade-off between revenue and efficiency maximization; it is a well-known problem that has been extensively studied [18]. Nevertheless, keeping efficiency high is very important when considering long-term business benefits. Further justifications of this choice are provided in Sec. V-E. The heuristic we designed follows the first and last properties: incentive compatibility and economic efficiency. It is not possible to maximize operator revenues together with those two properties; however, we will discuss how our system affects operator revenue in the following section. Notations are summarized in Table I.

D. Dummy Services

In a system where there can be more than one way to implement a service for a client, it is useful for clients to submit alternative bids, as explained in sec. IV-B. The problem of representing alternative bids is that it is easier to solve classic markets bids where only one set has a value for the bidder and all the rest has zero valuation (so-called "single minded bidders" in auction theory). However, in order to allow a more realistic representation of the bidder valuation, it is possible to represent alternative (OR) bids with dummy services [23]. Each alternative bid is split into normal bids with an added dummy service with a capacity of one asked by each bid. This ensures that only a single bid (among these alternatives) will be a winning bid. This is because when any of the alternative bid is chosen this consumes all the available units of the dummy service and it is not possible to accept any other. For instance, the bid: ((<S1:10>,<S2:20>), $100) OR ((<S3:10>,<S4:20>), $90), by creating the dummy service S5 (with capacity C_{S5} = 1) becomes: bid 1: ((<S1:10>,<S2:20>,<S5:1>), $100), bid 2: ((<S3:10>,<S4:20>,<S5:1>), $90). Note however that this can make the number of goods (i.e. services) grow quickly and that might pose some computational challenges. We prove in Sec. V-D that our heuristic gives a result in polynomial time according to the number of goods so it does not pose a problem for this particular heuristic.

V. MULTI-UNIT COMBINATORIAL AUCTION HEURISTIC

The problem of finding the optimal economic efficiency (i.e., maximizing the total valuation of winning bids) is proven to be NP-complete [18]. We thus propose a heuristic and detail it in this section.

A. Economic efficiency

Here we assume that we have an incentive compatible mechanism (as discussed in the previous section). This means that b_i = v_i, \forall i and maximizing the valuations of accepted bidders \( \sum_{i \in W} v_i \) is equivalent to maximizing the total of accepted bid prices: \( \sum_{i \in W} b_i \). The economic efficiency objective can be easily expressed with an Integer Linear Programming (ILP) representation: let \( x_i \) the Boolean variable representing whether bid \( i \) is accepted or not.

\[ \sum_{i=1}^{N} x_i b_i \]  \hspace{1cm} (1)

Subject to:

\[ \sum_{i=1}^{N} x_i q_j b_i \leq C_j \] \hspace{1cm} \forall j \in [1..M] \hspace{1cm} (2)

This objective corresponds choosing the accepted clients so as to maximize the sum of the accepted bid prices \( x_i b_i \) while not accepting clients if there is not enough leftover resources \( C_j \) on any link or VNF \( j \) to satisfy their demands \( q_j b_i \). This objective is solved in our optimal evaluation using the CPLEX linear solver. CPLEX can find the optimal of this optimization problem, however, since it is NP-complete, the time to solve it becomes very long for big instances of the problem.

B. Greedy heuristic

To alleviate the computation time problem of finding an optimal solution, we propose to use a greedy heuristic. The heuristic is presented in Algorithm 1 and has two main objectives. Firstly, finding an approximation in a polynomial time for the winner determination problem that is solved in the preceding ILP. Secondly, computing the prices paid by each winning bid. This heuristic is a \( \sqrt{V} \) approximation of the maximal sum of accepted bid prices [19], where \( V \) is the maximal sum (the one found using the ILP). The heuristic aims at solving the two parts in a consecutive manner:

**Algorithm 1** Greedy incentive compatible heuristic

initialization: Reorder the bids such that:

\[ \sqrt{\sum_{i=1}^{M} v_i^2} \geq \sqrt{\sum_{i=1}^{M} v_i^2} \geq \cdots \geq \sqrt{\sum_{i=1}^{M} v_i^2} \]

\( \forall j \in [1..M] \) \hspace{1cm} (3)

Output: \( W \): set of winning bids, \( p_1,\ldots,p_N \): price paid for each bid

1: // Part 1. Winner Determination
2: \hspace{1cm} for i=1\ldots N do
3: \hspace{2cm} if \( \forall j \in [1..M] \) \hspace{1cm} q_j b_i + U(j) \leq C_j \hspace{1cm} then
4: \hspace{3cm} W \leftarrow i
5: \hspace{2cm} \forall j \in [1..M] \) \hspace{1cm} U(j) = U(j) + q_j b_i
6: \hspace{1cm} end if
7: \hspace{1cm} end for
8: // Part 2. Price Computation
9: \hspace{1cm} for i \in W do
10: \hspace{2cm} if \( \forall j \in [1..M] \) \hspace{1cm} U(j) = 0
11: \hspace{3cm} for k = [1..N], k \neq i do
12: \hspace{4cm} if \( \forall j \in [1..M] \) \hspace{1cm} q_j b_k + U(j) \leq C_j \hspace{1cm} then
13: \hspace{5cm} \forall j \in [1..M] \) \hspace{1cm} U(j) = U(j) + q_j b_k
14: \hspace{4cm} end if
15: \hspace{2cm} end if
16: \hspace{2cm} p_i = \frac{\sum_{j=1}^{M} q_j b_j}{\sqrt{\sum_{j=1}^{M} q_j^2}}
17: \hspace{1cm} break
18: \hspace{1cm} end if
19: \hspace{1cm} end for
20: \hspace{1cm} end for
Winner determination: In this part (lines 1-7) we try to maximize the sum of the bid prices of all winning bids with a greedy algorithm. We reorder bids proportionally to their bid price and weight inversely with the square root of the sum of the units asked. We then add bids in that order if they are enough service units left until the last bid. This order is chosen because R. Gonen et al. [19] proves that it is the best approximation attainable in a polynomial time for this kind of problems. If V is the maximal sum of accepted bid prices, this will give a \(\sqrt{V}\) bid price sum in the worst case scenario.

**Price computation:** In this part (lines 8-20) we determine how much each winning bid should pay (remember that losing bids pay zero). For each winning bidder \(i \in W\), ranked \(i^{th}\) in the greedy order, we compute the optimal allocation without bid \((Q_i, b_i)\) and, at each step \(k\), checking whether satisfying its demand \(q_k\) for all \(j\) would have been possible. Since \(i\) is a winning bidder, the rank \(k\) for which his demand cannot be satisfied anymore is greater than \(i\). This rank \(k\) represents the maximum rank that \(i\) could have obtained while still winning the bid. The bid price that \(i\) should have had to be at rank \(k\) is \(p_i = b_k \sqrt{\sum_{j=1}^{k-1} q_{S_j} / \sum_{j=1}^{k-1} q_{S_j}'}\). This is the minimal bid price that bidder \(i\) could have used and still win.

**C. Incentive compatibility**

**Claim 1.** The greedy heuristic is incentive compatible.

**Proof.** Let’s denote \(u_i\), the utility function of bidder \(i\); \(u_i = v_i - p_i\) if bidder \(i\) wins the desired services \(Q_i\), and is zero otherwise. Note that from the way the algorithm is devised \(u_i \geq 0\). We want to prove that for any bidder \(i\), submitting bid price \(b_i \neq v_i\) or services \(Q_i' \neq Q_i\) and paying \(p_i'\) as a bid will not improve his utility, defined as \(u_i'\). Let’s define the relation \(\subseteq\) for service demands such as: \(Q_i \subseteq Q_i'\) if \(\forall j \in [1..M]\) \(q_{S_j} \leq q_{S_j}'\). The only way of obtaining all services \(Q_i\) for bidder \(i\) is if his bid \(Q_i'\) is such that \(Q_i \subseteq Q_i'\).

\(u_i'\) is the utility bidder \(i\) gets by submitting \((Q_i', b_i)\). If \((Q_i', b_i)\) is not a winning bid, \(u_i' = 0\) this cannot increase his utility. We then assume \((Q_i', b_i)\) is a winning bid. If \(Q_i \not\subseteq Q_i'\) bidder does not obtain the desired services but pays \(p_i'\), his utility is negative. We then assume \(Q_i \subseteq Q_i'\) and \((Q_i', b_i)\) is a winning bid.

We then show that the bidder will always be better off by reporting \((Q_i, b_i)\) rather than \((Q_i', b_i)\). Since \(Q_i \subseteq Q_i'\) \(\sum_{j=1}^{M} q_{S_j} \leq \sum_{j=1}^{M} q_{S_j}'\) and the bid is placed lower in the greedy order. Since the critical rank \(k'\) is the lowest rank that \((Q_i', b_i)\) could have been and still won the auction, and since \((Q_i', b_i)\) is asking for more resources than \((Q_i, b_i)\) then \(k' \leq k\).

This means that \(\sqrt{\sum_{j=1}^{k} q_{S_j}} / \sqrt{\sum_{j=1}^{k'} q_{S_j}'} \geq \sqrt{\sum_{j=1}^{k} q_{S_j}} / \sqrt{\sum_{j=1}^{k'} q_{S_j}'}\). The critical payment \(p_i'\) is then equal to \(\frac{b_k \sqrt{\sum_{j=1}^{k-1} q_{S_j} / \sum_{j=1}^{k-1} q_{S_j}'}}{\sum_{j=1}^{k} q_{S_j}} = p_i\).

It is left to show that bidding \((Q_i', v_i)\) is always better or equal than the winning bid \((Q_i, b_i)\). If \((Q_i, v_i)\) is a winning bid then reporting \(b_i \geq p_i\) will give the same result and same payment, reporting \(b_i \leq p_i\) will result in a losing bid and a worse or equal utility. If \((Q_i, v_i)\) is a losing bid and since \((Q_i, b_i)\) is a winning bid it means that the critical payment necessary to win this bid \(p_i'\) is greater than the true valuation \(v_i\). The utility is then \(v_i - p_i' \leq 0\).

Since the algorithm provides an incentive compatible payment system, the approximation of the winner determination also provides an approximation of economic efficiency (as discussed in Sec. V-A).

As a note we can remark that we proved that the algorithm is strategy proof for a single player. Meaning that no single player has an incentive to not be truthful about his valuation. But we do not make the claim that it is group-strategy proof, meaning that a coalition of bidders can gain from submitting untruthful bids. To see how that is possible consider the scenario where player \(i\) and \(j\) are in a coalition. If \(i\) is a winning bid, the price paid by \(i\) depends on the bid submitted by another player. If that other player is \(j\) then there can be an incentive for the player \(j\) to submit a bid price that is lower than his real valuation because it will lower the cost paid by \(i\). But by lowering his bid, player \(j\) also has a chance to lose his bid and utility gained. In a competitive environment this scenario should prove to be rare and not overly affect the system as the same issue is present in Amazon EC2 auctions [22].

**D. Computational complexity**

The heuristic we use starts by ordering all the bids, if we use the quicksort algorithm we get a complexity of at worst \(O(N^2)\) with \(N\) being the number of bids [24]. The algorithm then performs \(N^2 M\) operations to check if the bid can be accepted and for each accepted bid performs \(N^2 M\) operations to find the worst possible position. The complexity then stays at \(O(N^2 + M)\) in the worst case scenario. Having a high number of dummy bids that can make the number of services \(M\) grow quickly does not pose a particular challenge for this heuristic.

**E. Effect of incentive compatible on operator revenue**

It is easy to see that the incentive compatible greedy mechanism guaranties economically efficient winning bids, but has no guaranty about operator revenue. An operator could try to maximize his revenue by choosing \(p_i = b_i\) for all winning bids, after having maximized the total valuation of accepted bids. However, such mechanism gives an incentive for bidders to report bid prices that are lower than their valuations: they will try to guess what the lowest price they could submit and still win is, using the guess as a bid price. Depending on the conditions and how much risk bidders are willing to take, this auction could give economically inefficient results. It would also give an unfair advantage to clients that have more information about the competition and their respective bid submissions. Having an incentive compatible mechanism makes the task of submitting bid much easier for clients since there is no need to try to guess the bid prices of other clients: their task is limited to set the maximum price they are willing to pay for a service function chain, while being assured that
they will pay the minimum price possible (and maximize their own utility).

VI. Evaluation

A. Implementation and experimental setup

We used a simple model for our experimental setup to show with a few parameters how the whole system reacts when we introduce our NFV auctioning.

Operator model: The topology used is the GEANT network with 27 nodes, and is representative of a medium sized backbone network that is already hosting a number of services such as VPN or cloud virtualization. On this network, traffic is highly elastic with total load for 15 minutes varying during a day between 600 GBytes and 1.4 TBytes and some link resources are highly loaded (attaining 100% load) even if the network is mostly over-provisioned and has a mean max load of around 10%. [25]. In our evaluation nodes are considered as PoPs, and the 37 edges as high-capacity links. The services offered are bandwidth on each link and $N_{VN}F$ services on each node. Thus, the number of services is: $M = 37 + N_{VN}F \times 27$. The service capacity is $C$ and is the same for all services.

Bid model: The number of bids is equal to $N$. To define each bid, we need to model the demanded services (bandwidth and VNFs), their quantity, their valuation and their bid prices. Each bid represents a service function chain going from a random ingress node to a random egress node. Demanded services are: (i) bandwidth on all the links of the shortest path between those two nodes; (ii) additional services chosen at random on all the services offered by the on-path nodes. The number of additional services is chosen uniformly at random between 1 and $N_{add} = 7$ which is a reasonable number for the sake of our simple evaluation. The units requested for each service $j$ $q_{S_j}$ are chosen uniformly at random between 1 and $Q_{max} = 30$, which is a number that can only be interpreted in comparison to the unit and maximal capacity of the services. The influence of the service capacity is shown later. The valuation of the bid $v_i$ (in $\$) is chosen uniformly at random between 1 and $\sum_{j \epsilon [1..M]} q_{S_j}$, to account for the fact that, the more services are requested by a bid, the more his valuation has a chance to be higher. Since the system is incentive compatible, clients submit bid prices $b_i$ equal to their $v_i$.

B. Heuristic vs. optimal allocation

Fig. 3 shows the difference between the total valuation computed by our heuristic and the optimal maximal sum of valuations found using CPLEX. The curves stop at 600 bids because the solver ran out of memory. The figure shows how the optimal and the approximation stay close together (less than 10% deviation) but seem to get farther as the number of bids grows. Note that we are assured that this approximate total valuation is in the worst case at least the square root of the optimal total valuation [19], but actually it is much closer to the optimal ($\$880$ difference vs. a theoretical worst case of $\$9382$ difference). We can conclude that the greedy heuristic performs well in our simulation environment, providing a good and fast approximation.

C. Impact of the number of bids

We first show the importance of the number of bids on the operator revenue. Fig. 4a represents the market valuation and
D. Impact of service capacity

Fig. 5a shows what happens when the operator chooses to scale up the size of the offered services while demand does not change. The market value keeps growing because installing more capacity means that more bids can be accepted, even if the growth is not constant, since we start adding bids with a lower bid price. The revenue initially grows because there is a lot of competition and having more capacity means that more clients are accepted and pay the operator. However, adding capacity means that there is less and less competition for resources; hence the price paid by each winning bid gets smaller, as we can see in Fig. 5b. This is again because, with less competition, the minimal bid price for a winning bid \( i \) to be accepted \( (p_i) \) gets smaller. The maximum revenue for the operator is in our case at a capacity of \( C = 240 \), with revenue of about $10,000. However, this does not consider the operator’s cost of adding capacity, and depending on such cost the optimal might be anywhere between \( C = 0 \) and \( C = 240 \).

As we can see in Fig. 6 the optimal capacity is highly dependent on the demand level. During normal operations, leftover capacity will change from time slot to time slot on each PoP, so even assuming that demand is constant, revenue will change for the operator and it is not always possible to predict accurately the revenue that will be obtained on each time slot.

This simple model represents easily what might be the operator offers and clients bid, the general properties of the system are the same for a more complex model. The question of what is the optimal capacity to offer, and where should the capacity be placed in priority (as opposed to uniformly in our model) are important questions that the operator should answer when managing its NFV infrastructure and the auction.
chains in an NFV infrastructure that can be efficiently used to handle elastic network provisioning. The proposed approach offers a new business model and revenue opportunities to NFV operators. We proposed a Multi-Unit Combinatorial Auction system which accommodates clients by allowing them to directly express their service function chain demands as bids with a guaranty of obtaining the whole service function chain (and not isolated VNFs). The problem results being NP-Complete, but we proposed a heuristic that drastically simplifies the combinatorial nature of allocating bundles of resources, while guaranteeing that the approximation given is the best attainable in polynomial time. Moreover, our heuristic leads to an optimal economic efficiency and incentives clients to submit their true valuation. This is especially important in such a novel system since there is not yet good information on what exactly the utility of clients are, while operators cannot easily make assumptions on how they should price their services. Our evaluation results show that our heuristic gives a very good approximation of the economic efficiency (less than 10% deviation from the optimum) and that the operator can keep a good share of the submitted bids even if the payment system is incentive compatible and lets clients pay a lower price than their submission. Our results also show the interesting trade off facing operators in this elastic VNF provisioning system, where the revenue generated is highly dependent on the quantity of available resources compared to the unpredictable demand.

Future work will include a dynamic study of the auctioning system to show how clients and operators can adapt to demands that span more than one time slot, with a more detailed model of VNF requirements and implementation considerations. We can then show how clients need to balance between a classical model of paying a higher price for long demands and the auction system for less critical NFV needs. We will also study the problem of revenue sharing for multiple NFV operators: In this paper, only a single operator offers VNFs but a client might need multiple operators to satisfy his needs. Independent auctions are not possible since a client might win one auction while losing in another operator domain. We then need coordination between operators and a fair way to allocate the price paid by the client for the whole network service chain to the multiple operators that are deploying the VNFs.

Acknowledgments: This work was partially supported by the French ANR Reflexion project (ANR-14-CE28-0019).

REFERENCES

Author Index

Alameddine, Hyame Assem............................... 1
Anand, Narendra......................................... 60
Araldo, Andrea........................................... 139
Assi, Chadi................................................ 1
Ayoubi, Sara................................................ 1
Baccelli, François......................................... 94
Bannazadeh, Hadi......................................... 216
Barakat, Chadi............................................. 295
Barford, Paul............................................... 278
Bauer, Robert............................................. 225
Bavier, Andy................................................. 130
Beifuß, Alexander........................................ 35
Berger, Daniel S.......................................... 44
Bocchi, Enrico.............................................. 286
Bouet, Mathieu............................................ 340
Bouhtou, Mustapha....................................... 78
Boutaba, Raouf............................................ 216
Bronzino, Francesco.................................... 201
Cabrerizo, Sergio......................................... 112
Carle, Georg............................................... 35
Carra, Damiano........................................... 148
Cesari, Pablo................................................ 112
Cho, Jeong-Woo........................................... 94
Chuah, Teong Chee...................................... 87
Ciucu, Florin............................................... 184
Conan, Vania............................................... 340
Dai, Le Quang............................................... 18
Dán, György.................................................. 139, 313
Danielis, Peter............................................ 69
Develder, Chris........................................... 10
Durner, Raphael.......................................... 197
Eiermann, Andreas....................................... 212
Emmerich, Paul.......................................... 35
En-Najary, Taoufik........................................ 78
Feng, Mingdong.......................................... 148
Finze, Robert............................................. 193
Galambozi, Nandor...................................... 209
Gast, Nicolas.............................................. 157
Gebert, Steffen.......................................... 197, 234
Gray, Nicholas............................................ 197
Großmann, Marcel...................................... 212
Guerra, Ryan E.......................................... 60
Hasslinger, Frank....................................... 322
Hasslinger, Gerhard.................................... 322
Hemnings, Matthew.................................... 270
Hohlfeld, Oliver.......................................... 322
Huynh, Nguyen Van................................... 18
Hyytiä, Esa............................................... 166
Iannone, Luigi............................................ 340
Ingalls, Esa............................................... 166
Jarschel, Michael........................................ 209
Janardhan, Vaishnav.................................. 148
Karlsson, Gunnar....................................... 69
Karsten, Martin.......................................... 44
Kasera, Sneha............................................ 121
Kassler, Andreas......................................... 331
Khaledi, Mojgan......................................... 121
Kleinrouweler, Jan Willem........................... 112
Kline, Jeff.................................................. 278
Knightly, Edward W.................................... 60
Kouyoumdjieva, Sylvia T.............................. 69
Krahn, Robert............................................. 270
Lange, Stanislav......................................... 197, 234
Lee, Ying Loong.......................................... 87
Leonardi, Emilio.......................................... 304
Leon-Garcia, Alberto................................. 216, 252
Loo, Jonathan............................................. 87
Lorenz, Claas.............................................. 197
Lundrigan, Philip........................................ 121
Madeira, Edmundo....................................... 10
Marotta, Antonio........................................ 331
Maswood, Mirza Mohd Shahriar.................... 10
Mazumdar, Ravi R........................................ 261
McGeer, Rick............................................. 130, 270
Medhi, Deep............................................... 10
Mellia, Marco............................................. 286
Mentha, Michael......................................... 175, 193, 205
Metwalley, Hassan...................................... 286
Metzger, Florian......................................... 103
Michiardi, Pietro......................................... 148
Mitschele-Thiel, Andreas............................ 53
Moety, Farah.............................................. 78
Moldovan, Christian................................... 103
Morichetta, Andrea.................................... 286
Morper, Hans-Jochen............................. 209
Mukhopadhyay, Arpan.......................... 261
Nam, Tran Manh.................................. 18
Nasri, Ridha.................................. 78
Neglia, Giovanni............................. 148
Nguyen, Dinh.................................. 243
Ntougias, Konstantinos..................... 322
Obadia, Mathis................................ 340
Pacifici, Valentino........................... 313
Park, Byungchul.............................. 216
Poloczek, Felix............................... 184
Pries, Rastin.................................. 209
Puschmann, André............................ 53
Raumer, Daniel................................ 35
Ravaioli, Riccardo......................... 295
Raychaudhuri, Dipankar.................... 201
Reutter, Daniel............................... 193
Ricart, Glenn.................................. 130, 270
Righter, Rhonda.............................. 166
Röder, Marko.................................. 270
Rossi, Dario.................................. 139, 304
Rougier, Jean-Louis......................... 340
Roy, Rahul..................................... 261
Runge, Torsten M............................. 35
Saito, Hiroshi.................................. 26
Sankararaman, Abishek.................... 94
Schmidt, Mark.................................. 193, 205
Schmitt, Jens.................................. 44, 184
Schwartz, Christian......................... 234
Schwarzkopf, Fabian......................... 175
Seskar, Ivan.................................. 201
Shaik, Zubair................................. 53
Shariat, Ali.................................. 252
Shepard, Clayton............................ 60
Stege, Ulrike.................................. 270
Stockmayer, Andreas....................... 205
Subramanyam, Naveen Dasa............... 121
Sugiyama, Kohei............................. 243
Tagami, Atsushi.............................. 243
Thanh, Nguyen Huu......................... 18
Thomas, Charles............................. 278
Tizghadam, Ali................................ 216, 252
Tortelli, Michele............................. 304
Tran, Phuong Nga........................... 26
Tran-Gia, Phuoc............................. 234
Tsikgari, Dimitra............................ 148
Usroy-Keller, Guillaume............... 295
Van Der Mei, Rob......................... 112
Van Houdt, Benny........................... 157
Veith, Sebastian........................... 175
Virtamo, Jorma.................. 166
Wang, Li-Chun.............................. 87
Wolfinger, Bernd E.......................... 35
Zhang, Qi.................................. 216
Zhang, Sai Qian............................. 216
Zinner, Thomas............................ 197, 234
Zitterbart, Martina......................... 225
IEEE Computer Society Technical & Conference Activities Board

T&C Board Vice President
Cecilia Metra
Università di Bologna, Italy

IEEE Computer Society Staff
Evan Butterfield, Director of Products and Services
Lynne Harris, CMP, Senior Manager, Conference Support Services
Patrick Kellenberger, Supervisor, Conference Publishing Services

IEEE Computer Society Publications
The world-renowned IEEE Computer Society publishes, promotes, and distributes a wide variety of authoritative computer science and engineering texts. These books are available from most retail outlets. Visit the CS Store at http://www.computer.org/portal/site/store/index.jsp for a list of products.

IEEE Computer Society Conference Publishing Services (CPS)
The IEEE Computer Society produces conference publications for more than 300 acclaimed international conferences each year in a variety of formats, including books, CD-ROMs, USB Drives, and online publications. For information about the IEEE Computer Society’s Conference Publishing Services (CPS), please e-mail: cps@computer.org or telephone +1-714-821-8380. Fax +1-714-761-1784. Additional information about Conference Publishing Services (CPS) can be accessed from our web site at: http://www.computer.org/cps

Revised: 18 January 2012

CPS Online is our innovative online collaborative conference publishing system designed to speed the delivery of price quotations and provide conferences with real-time access to all of a project’s publication materials during production, including the final papers. The CPS Online workspace gives a conference the opportunity to upload files through any Web browser, check status and scheduling on their project, make changes to the Table of Contents and Front Matter, approve editorial changes and proofs, and communicate with their CPS editor through discussion forums, chat tools, commenting tools and e-mail.

The following is the URL link to the CPS Online Publishing Inquiry Form:
http://www.computer.org/portal/web/cscps/quote

352