2018 30th International Teletraffic Congress

A popularity-based approach for effective Cloud offload in Fog deployments

Marcel Enguehard
Cisco Systems & Telecom ParisTech

Giovanna Carofiglio
Cisco Systems

Dario Rossi
Telecom ParisTech

***Abstract*—Recent research has put forward the concept of Fog computing, a deported intelligence for IoT networks. Fog clusters are meant to complement current cloud deployments, providing compute and storage resources directly in the access network – which is particularly useful for low-latency applications. However, Fog deployments are expected to be less elastic than cloud platforms, since elasticity in Cloud platforms comes from the scale of the data-centers. Thus, a Fog node dimensioned for the average traffic load of a given application will be unable to handle sudden bursts of traffic. In this paper, we explore such a use-case, where a Fog-based latency-sensitive application must offload some of its processing to the Cloud. We build an analytical queueing model for deriving the statistical response time of a Fog deployment under different request Load Balancing (LB) strategies, contrasting a *naive*, an *ideal* (LFU-LB, assuming a priori knowledge of the request popularity) and a *practical* (LRU-LB, based on online learning of the popularity with an LRU filter) scheme. Using our model, and confirming the results through simulation, we show that the LRU-LB achieves close-to-ideal performance, with high savings on Cloud offload cost with respect to a request-oblivious strategy in the explored scenarios.**

I. Introduction

In [1], the authors highlight the need for deploying compute platforms geographically close to end-user devices. It is in particular necessary to enable low-latency or location-aware applications. Thus, the authors introduce the concept of Fog computing, a highly-virtualized platform that sits in IoT access networks and is complementary to the Cloud. Since then, interest has grown in the research community for Fog computing, encompassing areas such as workload placement [2], caching [3], [4], or application profiling [5].

As stated in [1], the Fog layer is not intended as a replacement for the Cloud, but rather as a complementary compute and storage platform. For instance, with frameworks such as AWS Greengrass [6], Fog operators can use their own devices (e.g., an IoT gateway or a local compute node) to perform some stateless pre-processing on data on its path to the Cloud. In addition, Fog nodes do not enjoy the same elasticity as Cloud datacenters. Indeed, while Cloud platforms inherently scale thanks to their size and their high number of tenants, Fog compute have strong physical limits that cannot be infringed. Thus, they are not fit to handle sudden bursts of requests, for instance in the case of flash crowds. In such a case, the natural solution is to offload part of the computation normally done in the Fog to the Cloud [7]. At the same time, if done incorrectly, Cloud offload could actually worsen the response latency. Furthermore, renting compute and storage power in the Cloud is expensive, as opposed to user-owned Fog devices. In this paper, we thus look at the problem of Cloud-offload and investigate the following question: *how to minimize cloud offloading costs while offering statistical latency guarantees in case of sudden bursts of requests in a Fog network?*

In particular, we explore how understanding request patterns can be used to optimize the usage of the Fog platform. Such understanding is typically obtained through Information-Centric Networking (ICN), whose *name-based forwarding* exposes request semantic at the network layer. Moreover, name-based forwarding also enables *connection-less communications*. For instance, ICN packets can convey request semantic to the network while keeping the communication secure through *object-based security*. Finally (and crucially for our architecture), it provides *application ubiquity*, as the forwarding is performed using an identifier instead of a locator. Our approach leverages all of these ICN features, using *naming* as a proxy to understand request popularity patterns to optimally exploit the *caching* resource of the Fog node.

Without loss of generality, we adopt a per-application view and consider a single Fog node, with compute and storage capabilities, that receives homogeneous requests from a single stateless application (e.g., lambda function). The Fog provider, who handles the Fog infrastructure, offers statistical latency guarantees to the application developer. However, the Fog node has fixed capacities and cannot handle a high arrival rate for a prolonged period. Thus, it offloads some of its processing to a Cloud platform, where the operator rents compute and storage resources, while still respecting the latency agreement.

To that end, we introduce *LFU-LB*, a new (ideal) load-balancing strategy between the Fog and Cloud that exploits *perfect knowledge* of request popularity, as well as *LRU-LB*, a new (practical) load-balancing strategies able to learn the request arrival pattern *with minimal knowledge*. Using an analytical model, we show the sizeable benefits of LFU-LB over a naive randomized load-balancing strategy, and additionally, show that LRU-LB performance is within 6% of the LFU-LB bound. Summarizing our contributions: (i) we propose a queueing model for the performance of a Fog-enabled application under different offloading strategies (Section III); (ii) we propose two new offloading strategies, called *LFU-LB* and *LRU-LB*, respectively based on knowledge and inference of request popularities (Section IV); (iii) we use an offline method for optimizing the performance of LFU-LB and LRU-LB, providing principled bounds on the achievable gain under

978-0-9883045-5-0/18/$31.00 ©2018 ITC
DOI 10.1109/ITC30.2018.00016

each strategy (Section V-B); (iv) we validate the model with a packet-level simulator, that we make available as open-source (Section V-C).

II. Problem description

A. Reference Fog architecture

We consider an IoT architecture composed of three main components: (i) IoT networks, where sensors, actuators, and users are connected; (ii) an access network which connects these IoT networks together and with the internet; and (iii) a Cloud platform, used for compute and storage. On top of this cloud platform, a Fog compute node is available in the access network. Both the Fog and the Cloud are equipped with caches that follow the Least-Recently-Used (LRU) policy. A load-balancer is in charge of redirecting incoming requests from the IoT networks to either Cloud or Fog. We summarize that architecture in Figure 1. We consider Fog applications that work in the following way: (i) the application retrieves *raw data* from one or several sensor nodes (e.g., an image or a temperature from several sensors); (ii) it performs some computation to transform the raw data into *processed data* (e.g., JSON file indicating detected shapes, or the average of the measured temperatures); (iii) the processed data is retrieved by users or actuators which use it to make decisions. As security is paramount for Fog applications [1], both processed and raw data are encrypted during network transmissions, for example with (D)TLS as common nowadays. In particular, we consider a pull-based model driven by client requests, of which we illustrate two of the possible paths in Figure 1. The user application starts by issuing requests (step 1). These requests reach the *load balance* function (for the sake of illustration, we only show cases where it decides to route the requests to the Fog node). In the Fog node, the request is matched against a cache (step 2) for the availability of processed data. In case of a *cache hit* (red dots · · ·), the processed data is sent back directly to the user. In case of a *cache miss* (green dashes - - -), the raw data must be retrieved from the sensor (step 3), before the computation can take place (step 4) and the processed data can then be served back to the user (step 5).

B. Fog vs Cloud load-balancing

IoT applications can roughly be categorized as: (i) *latency-critical*, when processed data must be received within 1-10ms, (ii) *latency-sensitive*, where the timescale of user interaction is in the order of 100ms [8], and (iii) *latency-tolerant*, that have no specific delay constraint. Whereas latency-critical applications cannot run in the Cloud (and would rather run in the device or at Fog level), latency-tolerant applications can be scheduled off-peak in the Fog. We thus argue that offloading strategies are most relevant for applications of the latency-sensitive class, where the computing bottleneck in the Fog may force to offload part of request processing to the Cloud. At the same time, the use of faraway Cloud resources not only increases the *cost* for the Fog operator but may additionally increase the *service latency*. As such, the Fog operator needs to carefully decide which requests to offload to the Cloud.

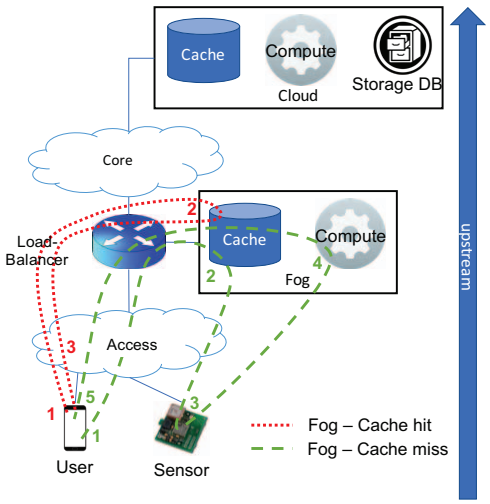


Fig. 1: Reference IoT, Fog and Cloud architecture

To devise such strategies, we assume that the application runs both in the Cloud and in the Fog, and consider costs for the Fog operator. In the Cloud, resources are elastic and one can increase the capacity as the incoming load increases. Furthermore, we consider that the Cloud always stores the raw data for archiving and monitoring purposes (we neglect the cost of raw data archival as it must be paid in any case). This comes on top of the cache for processed data, whose size is defined by the amount of storage rented in the cloud. Moving data to/from the Cloud through the core network also has a cost for the Fog operator. On the other hand, application deployment in the Fog comes at no cost for the Fog operator (since it owns the infrastructure). As Fog nodes have limited storage, the Fog cache is only used for processed data. The Fog node also has a finite amount of computing resources, which must be equally shared between all incoming requests. Thus, in case of a high load, the Fog node might have a high response time or even start dropping requests, which may violate the agreement set up with the application Fog developer.

The need for a proper offloading function ϕ between the Fog and Cloud resources thus becomes clear: such a function should *minimize the cost* of renting Cloud resources while *respecting the agreed upon latency constraint*, which we outline below (and formalize in Section III):

$$\begin{cases} \text{min.} & \text{Cost}(\phi) \\ \text{s.t.} & \mathbf{E}[T(\phi)] + \kappa\sigma(T(\phi)) \leq \Delta \end{cases} \quad (1)$$

where T is the stochastic variable describing the system response time. We use a *statistical* latency constraint, which guarantees that the bulk of requests are served under Δ (where κ allows to more precisely tune the fraction of in-profile requests). The advantage of such formulation is clear considering that it enables us to express the constraint in closed form in our queueing model, which simplifies tractability. At the same time, we acknowledge that Fog operators may be interested in tuning their offer to different operational points

TABLE I: Variables used in the model

Application specific	
Space of possible requests	R
Cumulated arrival rate	λ
Necessary work per request	X_{comp}
Raw data size	s_{raw}
Processed data size	s_{proc}
Application latency constraint	Δ
Optimization variables	
Load-balancing function	$\phi \in (0,1)^R$
Cloud cache size	$s_{cache,c}$
Total request serving time	$T(\phi, s_{cache,c})$
Cost function	$\Pi(\phi, s_{cache,c})$
Fog characteristics	
Fog compute capacity	$C_{comp,f}$
Fog cache size	$s_{cache,f}$
Access network capacity	$C_{acc,d}^{u}$
Access network propagation time	τ_{access}
TLS establishment delay	$\tau_{TLS,f}$
Cloud characteristics	
Cloud compute capacity	$C_{comp,c}$
DB query delay	τ_{DB}
Core network capacity	C_{core}
Core network propagation time	τ_{core}
TLS establishment delay	$\tau_{TLS,c}$
Cloud pricing	
Compute price	p_c
Network price	p_n
Storage price	p_s
Miscellaneous	
Cache hit probability	$h_{f_c}(r, s_{cache,f_c})$

in the cost-vs-delay trade-off. While this formulation leads to optimal results in static settings, it is also a useful reference to evaluate the performance of dynamic approaches, as discussed in Section VI.

III. An Analytical Model for Fog Networks

To understand the performance of a given load-balancing function, we define a queueing model that describes the systemic behaviour of the IoT architecture. We introduce the necessary variables for our model in Table I.

A. Application model and request distribution

Let us consider a single application running on a sliced Fog deployment. This application is described by its latency constraint Δ, its raw data size s_{raw}, its processed data size s_{proc}, and its job size X_{comp}. In particular, we assume that s_{raw} and s_{proc} are constant, while X_{comp} is a stochastic variable following an exponential distribution.

Let now R be the total number of possible requests, and $\{r_1, \ldots, r_R\}$ these requests. Following previous work, we consider that the request popularity distribution q follows a Zipf distribution [3], [9], [10], [11], i.e., for a request r arriving in the system, $q(k) = \mathbf{P}[r = r_k] = \gamma k^{-\alpha}$ where $\alpha > 0$ is the skew parameter and γ a normalization factor. In particular, requests arrivals are user-driven, and are thus well modeled by a Poisson process of parameter λ; it follows that the arrival process for the request r_k is a Poisson process of parameter $\lambda_k = q(k)\lambda$ and we assimilate $\{r_1, \ldots, r_R\}$ to $\{1, \ldots, R\}$. Additionally, the use of an independent requests model (IRM) is known to under-estimate of the cache benefits [12], [13], so we expect our results to be conservative.

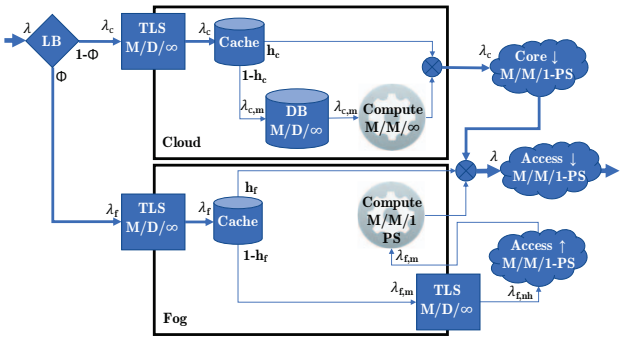


Fig. 2: Queueing network

B. Queueing model

Whenever relevant, we follow seminal work [7], [14], [15], [16] to select the most fitting queue to describe each resource. Particularly, we select an M/M/1-PS queue for the *Fog compute* as in [7], [14]: processor-sharing policy makes sense here since the Fog has a fixed amount of resources that must be shared between all the incoming requests. On the other hand, as we consider that the *Cloud compute* is elastic and scales on demand, we represent it by an M/M/∞ queue, and we represent *Cloud-database* access as a constant-time M/D/∞ queue. For *network resources*, we chose the M/M/1-PS model, as common in the literature [7], [15], [16]. As *load-balance* decisions and cache lookup should be done at line-rate, we consider that their impact is minimal w.r.t. other queues; and since they do not impact the comparison between Cloud and Fog service time anyway, we neglect them in what follows. Finally, we model the TLS endpoints as M/D/∞ queues, neglecting the computation time of the TLS handshake. Assuming that TLS is running in version 1.3, only 1 round-trip is necessary to establish the TLS connection, i.e.:

$$\tau_{TLS,f} = 2\tau_{acc}$$
$$\tau_{TLS,c} = 2(\tau_{acc} + \tau_{core}).$$

The resulting queueing system is depicted in Figure 2. One can note that the request transmission time is uniquely taken into account in the TLS queue: since IoT requests have a negligible size, their transmission time is indeed dominated by their propagation time.

We represent the load-balancing strategy by a function $\phi : \{1, \ldots, R\} \mapsto [0, 1]$, which associates to each request a probability of being forwarded to the Fog. In particular, given a popularity distribution q and a load-balancing strategy ϕ, the popularity distribution of requests arriving in the Fog is $q_f(r) = \gamma_f \phi(r) q(r)$, where γ_f is a normalization factor. For computing the hit probability in the Fog cache, we use the formula proposed by Che et al. [17]:

$$h_f(r) \approx 1 - e^{-q_f(r) t_s} \quad (2)$$

TABLE II: Arrival rate per queue in network

LB	Load-balancer	λ
	Access down.	
Fog	TLS - Fog	$\lambda_f = \sum_{r\in R} \phi(r)\lambda q(r)$
	Cache - Fog	
	Access up.	$\lambda_{f,m} = \sum_{r\in R} \phi(r)(1-h_f(r))\lambda q(r)$
	Compute - Fog	
Cloud	TLS - Cloud	$\lambda_c = \sum_{r\in R}(1-\phi(r))\lambda q(r)$
	Cache - Cloud	
	Core down.	
	DB	$\lambda_{c,m} = \sum_{r\in R}(1-\phi(r))(1-h_c(r))\lambda q(r)$
	Compute - Cloud	

where t_s is the unique root of $\sum_{r=1}^{R}(1-e^{-q_f(r)t}) = s_{cache,f}$. We use a similar model for the Cloud cache, where it suffices to replace the probability ϕ by its complement $1-\phi$.

C. Computing the statistical latency

First, let us point out that since processor-sharing queues are quasi-reversible processes, the exit distribution of an M/G/1-PS queue is a Poisson process (Theorem 3.6 of [18]). This is also true for the M/G/∞ queue [19], justifying that all the queues have a Markovian input. We can then easily derive the expected sojourn time in each of the queues. In particular, the expected service time for requests with job size X and Poisson arrival rate λ in an M/G/1-PS of capacity C is given by:

$$\mathbf{E}[T] = \frac{1}{(\mu-\lambda)} \quad \text{where } \mu = \frac{C}{\mathbf{E}[X]} \tag{3}$$

We can compute the arrival rate at each queue depending on the offloading strategy ϕ and the cache hit probabilities h_f and h_c at the Fog and Cloud caches respectively, obtained using Equation (2). We report the arrival rate per queue in Table II.

Computing the standard deviation of the queue is a more complicated task, as it depends on the service time distribution. However, since all M/G/1-PS queues have an exponential distribution for the job size in our model (M/M/1-PS queues), we can use the result of Ott [20]:

$$\sigma^2(T) = \frac{(2+\lambda\mu)\mu^2}{(1-\lambda\mu)^2(2-\lambda\mu)} \tag{4}$$

We thus get the equation for the expected queueing delay and the variance of the queueing delay in Equation (5). We explicit the expected latency and variance for the service time in the Fog $T_f(r)$ and $T_c(r)$ in the cloud in Table III, where the expected response time and the variance of M/M/1-PS queues (such as $\mathbf{E}[T_{acc,d}]$ and $\sigma^2[T_{acc,d}]$ for the access downstream) can be derived from Equation (3) and Equation (4) respectively.

$$\begin{aligned}\mathbf{E}[T] &= \sum_r q(r)\Big[\phi(r)\mathbf{E}[T_f(r)] + (1-\phi(r))\mathbf{E}[T_c(r)] + \mathbf{E}[T_{acc,d}]\Big]\\ \sigma^2(T) &= \sum_r q(r)\Big[\phi(r)\sigma^2(T_f(r)) + (1-\phi(r))\sigma^2(T_c(r)) + \sigma^2(T_{acc,d})\Big]\end{aligned} \tag{5}$$

TABLE III: Expected value and variance for the sojourn time in the Fog and Cloud

Fog

$$\begin{aligned}&\mathbf{E}[T_f] = \tau_{TLS,f} + (1-h_f(r))\left(\tau_{TLS,f} + \mathbf{E}[T_{acc,u}] + \mathbf{E}[T_{comp,f}]\right)\\ &\sigma^2(T_f) = (1-h_f(r))\left(\sigma^2(T_{acc,u}) + \sigma^2(T_{comp,f})\right)\end{aligned}$$

Cloud

$$\begin{aligned}&\mathbf{E}[T_c] = \tau_{TLS,c} + (1-h_c(r))\left(\tau_{DB} + \frac{\mathbf{E}[X_{comp}]}{C_{comp,c}}\right) + \mathbf{E}[T_{core,d}]\\ &\sigma^2(T_f) = (1-h_c(r))\left(\frac{\mathbf{E}[X_{comp}]}{C_{comp,f}}\right)^2 + \sigma^2(T_{core,d})\end{aligned}$$

D. Computing the cost function

The *hourly* operation cost consists of a network, a compute, and a storage term. We approximate that the compute power rented in the Cloud is synchronized with the incoming load (i.e., the Cloud spawns a container process at each new request). In particular, the cost of running the Cloud increases proportionally to the requested load: $p(c,s,n) = p_c c + p_s s + p_n n$, where c (resp. s) is the amount of compute (resp. storage) resources rented on the Cloud, and n is the egress Cloud traffic.

1) Compute cost: We consider an elastic consumption for the compute resources in the cloud. If $Q_{comp,c}(t)$ is the number of customers in the Cloud compute M/M/∞ queue, the instantaneous number of instantiated cloud compute platforms is: $c(\phi, s_{cache,c})_t = Q_{comp,c}(t)$ According to [19], the expected value for $c(\phi, s_{cache,c})$ is thus:

$$\mathbf{E}[c(\phi, s_{cache,c})] = \frac{\lambda_{c,m}(\phi, s_{cache,c})}{C_{comp,c}/\mathbf{E}[X_{comp}]}$$

2) Storage cost: The storage cost depends on the cache size in the cloud: $s(\phi, s_{cache,c}) = s_{cache,c} s_{proc}$

3) Network cost: For each incoming request, s_{proc} is transferred downstream as a reply. Given that λ_c (in Hz) requests are forwarded in average to the cloud every second, $3600\lambda_c$ objects of size s_{proc} are forwarded every hour. Thus:

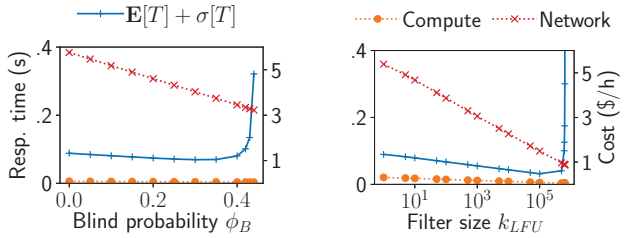
$$\mathbf{E}[n(\phi, s_{cache,c})] = (3600\lambda_c(\phi))s_{proc}$$

Thus, the total cost function reads:

$$\begin{aligned}\Pi(\phi, s_{cache,c}) =& \frac{p_c\lambda_{c,m}(\phi, s_{cache,c})}{C_{comp,c}/\mathbf{E}[X_{comp}]} + p_s s_{cache,c} s_{proc}\\ &+ p_n(3600\lambda_c(\phi))s_{proc}\end{aligned} \tag{6}$$

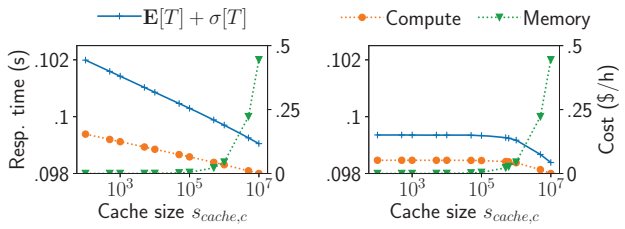
IV. Load-balancing strategies

The objective cost function defined by Equation (6) and the stochastic latency expressed in Equation (5) constitute the building blocks of the optimization problem defined in Equation (1). At the same time, the problem is still not well specified since the optimization variables $(\phi, s_{cache,c}) \in [0,1]^R \times [0,R]$ reside in high-dimensional space. Clearly, this is not only impractical but also irrelevant, as we argue that to reduce both delays and costs, a load balance function should divert popular content toward the local Fog resource as long as this does not introduce a CPU bottleneck. We can thus reduce the large space of LB functions ϕ to a



(a) Blind-LB, $s_{cache,c}$=$3.1{\cdot}10^5$ (b) LFU-LB, $s_{cache,c} = 0$

Fig. 3: Response time (left) and cost (right) of the system vs ϕ for the Blind- and LFU-LB at fixed cache size



(a) Blind-LB, $\phi = 0.42$ (b) LFU-LB, $k_{LFU} = 6.1 \cdot 10^5$

Fig. 4: Response time (left) and cost (right) of the system vs $s_{cache,c}$ for the Blind- and LFU-LB at fixed ϕ

few relevant families. In this section, we present three load-balancing strategies to which we particularize the optimization problem. Specifically, we introduce a baseline *Blind* load-balancer strategy (Section IV-A), as well as two strategies based on request popularity: *LFU-LB* (Section IV-B) and *LRU-LB* (Section IV-C).

A. Blind load-balancer

In this case, we consider that the load-balancer has no information about the request (e.g., because it is encrypted), thus it blindly balances all traffic with i.i.d. probability $\phi(r) = \phi_B$. In particular, Equation (1) can be rewritten as such:

$$\begin{cases} \text{min.} & \Pi(\vec{\phi}_B, s_{cache,c}) \\ \text{s.t.} & \mathbf{E}[T(\vec{\phi}_B, s_{cache,c})] + \kappa\sigma\left(T(\vec{\phi}_B, s_{cache,c})\right) \leq \Delta \end{cases}$$

with $\vec{\phi}_B = (\phi_B, \ldots, \phi_B)$. In this particular case, since $(\phi_B, s_{cache,c}) \in [0,1]\times[0,R]$, the problem is easy to optimally solve numerically.

As such, system performance depends on two variables: the probability ϕ_B and the cloud cache size $s_{cache,c}$. For the sake of illustration, we investigate their respective importance in the setup described further along in Section V-A, using the parameters reported in Table IV for the numerical evaluation. In particular, we examine the variation of the costs and constraint functions depending on either ϕ_B of $s_{cache,c}$.

In Figure 3a, we represent the variation of the constraint function (+, left side) and of the compute (•, right side) and network (×, right side) costs for a fixed cache size $s_{cache,c} = 3.1 \cdot 10^5$. We do not represent the storage cost as it is constant (since $s_{cache,c}$ is fixed). The first takeaway is that, as expected from the costs in Table IV, the network cost is dominant w.r.t. the compute and memory cost. We also notice that the constraint function diverges towards $+\infty$ when ϕ_B grows close to 0.45, as the Fog compute queue becomes unstable and cannot handle the request rate.

In Figure 4a, we next vary the cache size $s_{cache,c}$ for a fixed load-balancing probability $\phi_B = 0.42$ (the sweet spot in Figure 3a). We show the statistical response time (+, left side), and the compute (•, right side) and memory (▼, right side) costs. We do not represent the network cost since it is constant when ϕ is constant. First, we note that varying the cache size has a limited impact on both the cost and the constraint function. Furthermore, in this case, given that the compute is almost one order of magnitude more expensive than the memory and the cloud popularity distribution is sufficiently skewed, it is interesting to cache highly popular requests.

B. LFU-LB strategy

Let us now consider the case where the load-balancer is aware of network-names. This can, for instance, be done using ICN, where the content name is directly encoded in the network layer as the forwarding instruction. A network-layer load-balancer would then be able to distinguish requests. Note that nothing prevents this name from being entirely or partially encrypted for security reasons, as the LB does not require any understanding of the name to derive its popularity.

We then consider the following strategy: forward the most popular requests to the Fog until you reach the latency constraint, and offload the least popular ones to the cloud. We call this strategy LFU-LB (for Least-Frequently Used, since it is reminiscent of a cache with the LFU eviction policy). In particular, if k_{LFU} request classes are forwarded to the Fog:

$$\phi(r) = \delta_{r \leq k_{LFU}} = \begin{cases} 1 & \text{if } r \leq k_{LFU} \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the LFU-LB selects the k_{LFU} most popular classes for processing in the Fog. Thus, Equation (1) becomes:

$$\begin{cases} \text{min. } \Pi(\vec{\delta}_{k_{LFU}}, s_{cache,c}) \\ \text{s.t. } \mathbf{E}[T(\vec{\delta}_{k_{LFU}}, s_{cache,c})] + \kappa\sigma\left(T(\vec{\delta}_{k_{LFU}}, s_{cache,c})\right) \leq \Delta \end{cases}$$

with $\vec{\delta}_{k_{LFU}} = (\delta_{1\leq k_{LFU}}, \ldots, \delta_{R\leq k_{LFU}})$. Like the Blind-LB, we reduced the problem to a two-dimensional optimization.

The *rationale* behind this strategy is simple: to optimize the hit rate in the Fog cache, we artificially reduce the space of incoming requests at the Fog. Thanks to this much-higher hit rate, we can then forward more requests in the Fog without overloading the access upstream or the Fog compute. In Figure 3b (resp. Figure 4b), we represent the evolution of

the cost and constraint functions while setting $s_{cache,c} = 0$ (resp. $k_{LFU} = 6.1 \cdot 10^5$) for the setup in Table IV. At a first glance, Figure 3b indicates that a proper choice of k_{LFU} allows to decrease the network cost at levels unreachable with the Blind-LB while respecting the constraint. Furthermore, as in Section IV-A, the dominant factor in terms of cost is the number of offloaded requests. Both of these insights point towards LFU as a good strategy for Fog/Cloud load-balancing.

Additionally, Figure 4b shows that for small values of $s_{cache,c}$, the compute cost stays constant. This is due to the popularity distribution at the Cloud cache, which only contains the long tail of the Zipf distribution. Thus, for small cache sizes, the hit probability is low and the cache almost useless.

C. The LRU-LB strategy

While the LFU-LB is efficient, we are aware that it is an ideal policy, difficult to realize in practice if the popularity distribution is not known in advance (as estimating the popularity is difficult and slow[1]). To derive a practical LB policy, we argue that the LB does not need to learn the popularity of each specific request. It only needs to flag whether a request is popular, acting as a low-pass filter. Let us now consider a virtual filter that, for each incoming request, updates a small fixed-size database of requests, which are evicted according to the LRU policy. As this database only stores *names* of previously seen requests, but never stores any content, its size is small. It is immediate to recognize that such an LRU filter would probabilistically store only the most popular requests names. Compared to the aforementioned counter solution for the LFU-LB, this has four main advantages: (i) it does not require prior knowledge of the application; (ii) it keeps memory constrained to the size of the filter, instead of the size of the catalogue; (iii) it is flexible w.r.t. changes in the popularity distribution; (iv) it requires minimal effort for integration in ICN forwarders as the LRU structure is already used for caches.

We thus propose the LRU-LB strategy where the LB is equipped with an LRU filter that performs the forwarding function; i.e., for each incoming request, a name hit means forwarding the request to the Fog and a miss means offloading the request to the Cloud. This approach to optimize cache hit is similar to a 2-LRU cache, whose effectiveness is already known [21], but, to the best of our knowledge, the use of an LRU filter for load-balancing and offload is a novel contribution. Indeed, in our case, we do not only consider the cache hit, but the impact of the filter size on the service time of the Fog compute. This leads to a different optimization problem, that we now explicit.

To incorporate LRU-LB in the model, we must compute the load-balancing function ϕ depending on the filter size k_{LRU}. Since our filter is a virtual LRU cache, we have $\phi(r) = h_{k_{LRU}}(r)$ where $h_{k_{LRU}}(r)$ is the hit probability for the request r in an LRU cache of size k_{LRU} with input distribution q, which can be derived straightforwardly from Equation (2). Integrating this in Equation (1), we end up with a constraint and a cost functions that depend only on k_{LRU} and $s_{cache,c}$:

$$\begin{cases} \text{min. } \Pi(\vec{h}_{k_{LRU}}, s_{cache,c}) \\ \text{s.t. } \mathbf{E}[T(\vec{h}_{k_{LRU}}, s_{cache,c})] + \kappa\sigma\left(T(\vec{h}_{k_{LRU}}, s_{cache,c})\right) \leq \Delta \end{cases}$$

with $\vec{h}_{k_{LRU}} = (h_{k_{LRU}}(1), \ldots, h_{k_{LRU}}(R))$. Compared to the LFU-LB, realizing and optimizing the LRU-LB only requires knowing the popularity skewing factor α and the arrival rate λ instead of the actual per-content popularity distribution.

[1]This requires either offline analysis of the popularity distribution, or to keep counters of incoming requests. Both solutions are not flexible to popularity changes and are difficult to implement efficiently.

TABLE IV: An example application

Deployment		Application	
$C_{comp,f}$	3GHz	R	10^7
$s_{cache,f}$	1GB	λ	2kHz
C_{acc}	10Gbps	$\mathbf{E}[X_{comp}]$	10^7 CPU cycles
τ_{acc}	2 ms	s_{raw}	1MB
$C_{comp,c}$	2GHz	s_{proc}	10KB
τ_{DB}	1ms	α	1
C_{core}	1Gbps	Δ	100ms
τ_{core}	20ms	κ	1

Cloud pricing	
p_n	\$0.08 per GB
p_s	\$0.004446 per GB and hour
p_c	\$0.033174 per vCPU and hour

V. Evaluation

To understand the performance of our strategies, we conduct in this section a thorough evaluation of their behaviour. In Section V-A, we describe the setup used to perform the evaluation. In Section V-B, we use the analytical model to infer the general behaviour of each LB strategies depending on the most important parameters of the system. Finally, in Section V-C, we use simulation to augment our evaluation with the packet-level performance of the strategies.

A. An example application

Let us consider a deployment with the characteristics described in Table IV. We select an application with a medium compute difficulty (10ms on a 1GHz processor) and medium processed data size. For our Fog deployment, we consider that our application has a slice of a compute platform amounting to a 3GHz CPU and 1GB of cache. Finally, to make the evaluation more realistic, we particularize it using public pricing of the Google Compute infrastructure as of October 2017, set the delay target to Δ =100ms and the multiplicative standard-deviation factor to $\kappa = 1$.

To find the optimal behaviour of each strategy, we numerically solve the optimization problems defined in Section IV. In particular, we use the Method-of-moving-asymptotes [22] in its NLopt implementation [23]. We complement our queueing model with a packet-level simulator, consisting of about 2.3k lines of C code, that we make available as open-source software for the community[2]. This simulator enables easy construction and evaluation of queueing models without the

[2]https://github.com/marceleng/fog-cloud-offload-sim/

TABLE V: Optimal costs and parameters per LB

Method	$\mathbf{E}[\phi]$	$s_{cache,c}$	Π
Blind	0.42	$3.1 \cdot 10^5$	3.4$/h
LFU	0.844 ($k_{LFU} = 6.1 \cdot 10^5$)	0	0.95$/h
LRU	0.840 ($k_{LRU} = 1.3 \cdot 10^6$)	0	0.97$/h

complexity of full-blown network simulators. In particular, it is useful to quickly prototype systems such as the Fog/Cloud offload problem and gain insight on their performance depending on the various parameters.

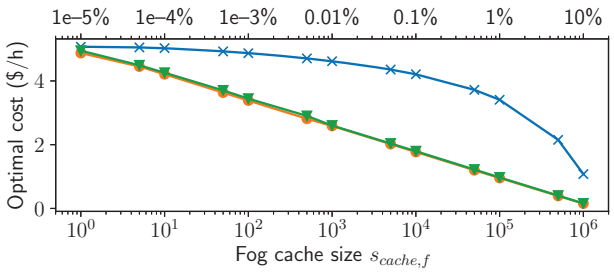
B. Numerical solution

In a first step, we show in Table V the optimized values for our example application. We first note that using the LRU and LFU-LB allows the Fog to handle more than twice as many requests as with the Blind-LB. This results in a decrease in offload cost of more than 70%. Furthermore, it shows that the LRU-LB has similar performances to the LFU-LB, with a 2% relative difference in offload cost.

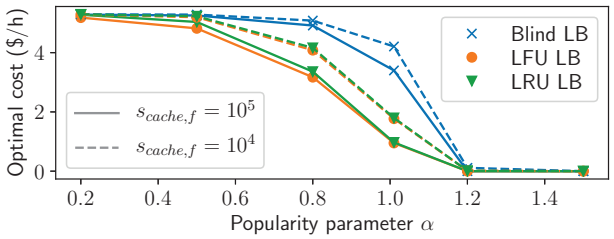
In Figure 5a, we show the influence of the Fog cache size w.r.t. to the optimal cost for the Blind-, LFU-, and LRU-LB strategies. This figure points out that the LFU and LRU strategies are especially interesting for mid-range scenarios. Indeed, if $s_{cache,f}$ is small, they do not benefit from the much-improved hit-rate in the cache seen in other cases. On the other hand, if $s_{cache,f}$ is big, the Blind-LB strategy also provides a high hit-rate, thus closing in on LFU and LRU. However, for a reasonable cache size (between 0.1% and 1% of the catalogue size, see top x-axis), both the LFU- and the LRU-LB offer large gains in terms of offloading cost. Furthermore, this graph confirms that the LRU-LB is an extremely good approximation of the ideal LFU-LB, and regardless of the Fog cache size.

Similar remarks can be made if we study various skews in the popularity distribution. In Figure 5b, we vary the parameter α in the Zipf popularity distribution. For small α values, the popularity distribution converges towards a uniform distribution, thus diminishing the impact of popularity-based LBs. For large α values, the relative importance of the first items is such that the cache hit is always high, regardless of the LB strategy. However, for $\alpha \in [0.5, 1.1]$, the LFU- and LRU-LB strategies allow for a largely reduced optimal cost for both Fog cache sizes that we tested. Furthermore, we see that the LFU- and LRU-LB strategies with $s_{cache,f} = 10^4$ are more efficient than the Blind-LB strategy with $s_{cache,f} = 10^5$. This indicates that our strategies also allow for more efficient provisioning of Fog resources. Once again, we note that the performance of the LRU-LB is close to the LFU-LB, varying by at most 6% (for $\alpha = 0.8$).

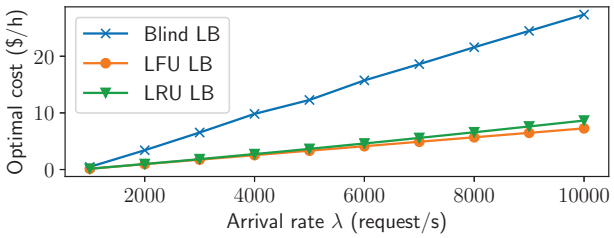
Finally, we consider the impact of the arrival rate on the efficiency of our scheme in Figure 5c. Interestingly enough, the optimal cost increases linearly w.r.t. the arrival rate for all three cases, with slopes at $3.0 \cdot 10^{-3}$ \$/Hz for the Blind-LB, $7.9 \cdot 10^{-4}$ \$/Hz for the LFU-LB, and $9.5 \cdot 10^{-4}$ \$/Hz for the LRU-LB. This confirms that the LFU- and LRU-LB strategies cope better with increased loads (e.g., flashcrowds) than the Blind-LB. This is typically due to the improved



(a) Fog cache size $s_{cache,f}$



(b) Zipf parameter α



(c) Arrival rate λ

Fig. 5: Cost of the Blind-, LFU-, and LRU-LB letting the application-independent parameters vary

hit rate at the Fog cache, which absorbs a large part of the increased arrival rate. Particularly, if the cost of the Blind-LB diverges with respect to the LFU-LB and the LRU-LB for an increasing arrival rate, their ratio stays however constant. The ratio between the absolute costs for the LFU-LB (LRU-LB) over the Blind-LB is of $3.8\times$ $(3.2\times)$[3]. Thus, when the arrival request rate increases, the relative gain of using the LFU-LB (LRU-LB) over the Blind-LB also increases, which shows the LRU- and LRU-LB to be quite robust to high arrival rates.

C. Packet-level simulation

In the previous section, we derived some statistical insights that showed potential benefits for using the LFU- and LRU-LB. However, these insights are based on the smoothed behaviour of the system and do not reflect its packet-level behaviour. We thus build a simulator and run a simulation campaign to complement the numerical insights gathered so far. In Figure 6, we show for instance the empirical distribution

[3]The relative cost gain of the LFU-LB (LRU-LB) over the Blind-LB $(\Pi_{Blind-LB} - \Pi_{LRU-LB})/\Pi_{Blind-LB}$ is deceptive here as it asymptotically grows to 100%, already reaching 80% at 10kHz

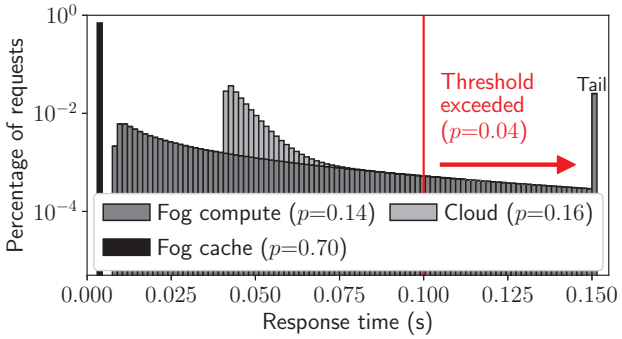


Fig. 6: Distribution of the response time for the LRU-LB ($k = 1.3 \cdot 10^6$) for 10 simulations of 10^8 arrivals

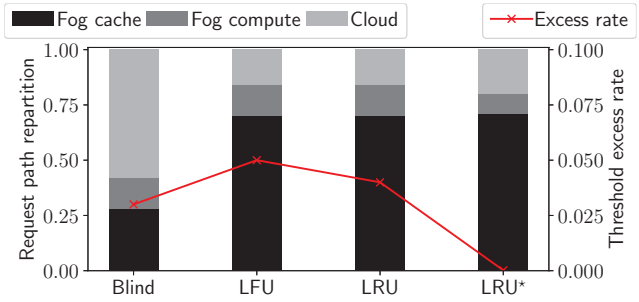


Fig. 7: Request repartition (grey bars, left y-axis) and threshold excess rate (red line, right y-axis) for the Blind-, LFU-, LRU-LB (with optimal k and extra-budgeted $k^\star$ tunings)

(EDF) of service times for 10^9 arrivals over 10 runs of the simulator for the application described in Section V-A using the LRU-LB with the optimal value $k_{LRU} = 1.3 \cdot 10^6$. For the sake of readability, different components (i.e., fog cache, compute, and cloud) are highlighted with different shades of grey. Note that, as expected, a significant fraction of the requests hit the Fog cache. At the same time, it can be seen that the Fog compute exhibit a long tail due to the processor-sharing scheduling, which causes service times to grow up to *possibly several seconds* in the worst case. We also gather that while the statistical constraint in Equation (1) is respected, the probability of a packet being served in more time than the target deadline $\Delta = 0.1$s is 4%.

Clearly, whereas our objective function and statistical latency constraint are specific, the LRU-LB mechanism and the optimization framework are general enough to accommodate other operational points in the cost-vs-latency space, which can simply be done by *altering the LRU-LB filter size*. An easy way to tune this trade-off is, for instance, to let the multiplicative factor of the latency standard deviation grow beyond $\kappa > 1$ in Equation (1), to ensure that a larger fraction of the requests fit the target latency profile. Another option is to allow a specific budget increase w.r.t. the optimal cost, thus computing first the optimal cost and then the value of k_{LRU} corresponding to that budget increase. As an illustration, we depict the breakdown of the requests (grey bars, left y-axis) and the fraction of requests exceeding the delay target (red line, right y-axis), in Figure 7 for the Blind- and LFU-LB, and for two settings of the LRU-LB (the cost-optimal one, and the LRU* which corresponds to a 25% budget increase). It shows that the extra +25% budget (1.21$ overall hourly cost) would allow reducing the fraction of requests served in more than Δ by 2 orders of magnitude (to 0.02%), additionally cutting the tail to sub-second service time (0.5sec) *in the worst case*. As stated earlier, this could be simply achieved by shrinking the LRU-LB filter size to $k^\star_{LRU-LB} = 7.3 \cdot 10^5 < k_{LRU-LB} = 1.3 \cdot 10^6$ – which is intuitive, since reducing the filter size reduces the probability that less popular content is forwarded to the Fog. Notice that, despite the budget increase with respect to the minimum cost, the LRU* operating point still corresponds to a 61% reduction with respect to the offload budget necessary with the Blind-LB strategy. Whereas where to exactly place the cursor in the cost versus latency trade-off is a Fog operator decision, the LRU-LB filter size constitutes a unique and simple knob to tune the system to reach the desired operational point.

VI. Discussion and Future Work

In this paper, we use the LRU-LB as an approximation for the LFU-LB. However, setting the size of the filter currently requires offline computation depending on the arrival rate λ and the Zipf popularity parameter α. While λ is easy to measure live and Figure 5c seems to indicate that the optimal filter size increases linearly with λ, the linear coefficient depends on α, which is notably difficult to compute online. As a next step, we thus set out to find an *online method* that approximates the optimal filter size *without any knowledge of the popularity distribution* using simple telemetry such as the Fog cache hit rate or the request service time. As stated in Section V-C, it should also be tunable to the Fog operator preferred point on the cost-vs-delay trade-off. As a start, we might consider the following strategy: (i) find a maximum filter size $k^{(m)}_{LRU}$ (considering a low α and a high λ) and set the filter size as such; (ii) keep the filter size to $k^{(m)}_{LRU}$, but only forward to the Fog requests in position up to $k' < k^{(m)}_{LRU}$, where k' is tuned through a control-loop feedback mechanism depending on the operational point wished by the operator. Such a method would also be able to adapt to changes in the popularity distribution or the arrival rate.

Furthermore, our evaluation is limited to a single application. In future work, we plan to look at the performance of the LFU- and LRU-LB depending on the relative values of s_{proc} and X_{comp}: the size of the data vs the amount of compute necessary to produce it. Having a high s_{proc} and low X_{comp} would, for instance, reduce the incentive of hitting the cache, and thus limit the benefits of our strategies. Finally, to encompass all possible applications, we must look at other queueing models. In particular, we might want to look at GPU-powered applications (e.g., image processing), whose compute model differs from processor-sharing of exponentially-distributed jobs.

Finally, our model should encompass other characteristics of ICN. For instance, the *object-based security model* allows for secure transfer of content without the need for TLS. In particular, it would mean discarding the round-trip time taken into account by the TLS queues in our model. Furthermore, the use of *in-network request aggregation* and *in-network caching* would diminish the arrival rate in our system and might change the content popularity distribution.

VII. Related Work

The importance of locating compute resources topologically close to users had been put forward under diverse forms in the community: Fog computing [1], Mobile-Edge-Computing [24], hybrid Cloud [7]. In particular, Niu et al. [7] explore a similar problem to ours: the use of a local cloud infrastructure to handle sudden bursts of traffic. They also use a Markov-chain based model for computing the expected response time and devise a scheduling algorithm between hybrid and public cloud using an optimization problem under budget constraints. However, they do not exploit any knowledge of the request popularity, thus falling under the hard limit that we exposed for the Blind-LB. Malawski et al. [25] look at costs optimization between a hybrid cloud and multiple public clouds with different pricing models under a deadline constraint. However, they focus on task optimization, looking at a model closer to scheduling for scientific computing rather than live optimization of user requests.

Using popularity to load-balance content in ICN networks has already been explored. In [9], the authors propose to count incoming packets and use exponential smoothing. As argued in Section IV-C, this approach is not flexible to popularity changes and requires knowledge of the application. Furthermore, the authors aim at load-balancing packets over homogeneous paths, whereas the Fog/Cloud offload problem is essentially heterogeneous. Similarly, Carofiglio et al. [10] propose the use a k-LRU filter to learn popularity for load-balancing ICN interests over multiple paths. They then measure per-name latency to optimize the distance to the next object. However, the authors do not specify the settings of the k-LRU filter, and only consider the effect of their load-balancing on the data creation process. Finally, in [4], the authors use the ICN-Fog node as a classifier between static and dynamic data, thus preventing upstream caches to store dynamic data. They do not, however, consider the data processing that happens in many Fog applications.

VIII. Conclusion

In this paper, we looked at the problem of computation offload for Fog computing deployment. Using queueing theory, we built an analytical framework to evaluate the performance of a given offloading strategy. We then presented three offloading strategies, that we evaluated through our analytical model. We showed that strategies using request popularity to load-balance requests performed measurably better than an optimized Blind load-balancer. We also proved that our approximated strategy, the LRU-LB, degrades performances by less than 6% compared to the optimal LFU-LB. Finally, using a packet-level simulator, we demonstrated that our LRU-LB reduces the Cloud offload cost by 70% while keeping the threshold excess rate stable.

Acknowledgments

This work benefited from the support of NewNet@Paris, Cisco's Chair "Networks for the Future" at Telecom ParisTech (https://newnet.telecom-paristech.fr).

References

[1] F. Bonomi, et al. "Fog computing and its role in the internet of things." In *Proc. 1st Edition Workshop Mobile Cloud Computing*, ACM, 2012.
[2] K. Hong, et al. "Mobile fog: A programming model for large-scale applications on the internet of things." In *Proc. 2nd SIGCOMM Workshop Mobile Cloud Computing*, ACM, 2013.
[3] J. Khan, et al. "A content-based centrality metric for collaborative caching in information-centric fogs." In *IFIP-Networking - ICFC*, 2017.
[4] M. Wang, et al. "Fog computing based content-aware taxonomy for caching optimization in information-centric networks." In *IEEE Conf. Comput. Commun. Workshops*, May 2017.
[5] Z. Chen, et al. "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance." In *Proc. 2nd ACM/IEEE Symp. Edge Computing*, ACM, 2017.
[6] "Aws greengrass." https://aws.amazon.com/greengrass.
[7] Y. Niu, et al. "Handling flash deals with soft guarantee in hybrid cloud." In *Proc. INFOCOM*, IEEE, 2017.
[8] R. B. Miller. "Response time in man-computer conversational transactions." In *Proc. AFIPS Fall Joint Comput. Conf.*, 1968.
[9] T. Janaszka, et al. "On popularity-based load balancing in content networks." In *Proc. 24th Int. Teletraffic Congr.*, p. 12, 2012.
[10] G. Carofiglio, et al. "Focal: Forwarding and caching with latency awareness in information-centric networking." In *Globecom Workshops*, IEEE, pp. 1–7, 2015.
[11] L. Breslau, et al. "Web caching and zipf-like distributions: Evidence and implications." In *Proc. INFOCOM*, vol. 1, IEEE, 1999.
[12] C. Imbrenda, et al. "Analyzing cacheable traffic in isp access networks for micro cdn applications via content-centric networking." In *Proc. 1st ACM SIGCOMM Conf. Inform.-Centric Networking*, Sep 2014.
[13] S. Traverso, et al. "Temporal locality in today's content caching: why it matters and how to model it." *ACM SIGCOMM Comput. Communication Review*, vol. 43, no. 5, 2013.
[14] B. Urgaonkar, et al. "An analytical model for multi-tier internet services and its applications." In *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, 2005.
[15] M. Nabe, et al. "Analysis and modeling of world wide web traffic for capacity dimensioning of internet access lines." *Performance evaluation*, vol. 34, no. 4, 1998.
[16] J. Boyer, et al. "Heavy tailed m/g/1-ps queues with impatience and admission control in packet networks." In *Proc. INFOCOM*, vol. 1, IEEE, 2003.
[17] H. Che, et al. "Hierarchical web caching systems: Modeling, design and experimental results." *J. Select. Areas in Commun.*, vol. 20, no. 7, 2002.
[18] F. P. Kelly. *Reversibility and stochastic networks*. Cambridge University Press, 2011.
[19] G. F. Newell. "The m/g/∞ queue." *J. Appl. Math.*, vol. 14, no. 1, 1966.
[20] T. J. Ott. "The sojourn-time distribution in the m/g/1 queue by processor sharing." *J. of Appl. Probability*, vol. 21, no. 2, 1984.
[21] D. Shasha and T. Johnson. "2q: A low overhead high performance buffer management replacement algoritm." In *Proc. 20th Int. Conf. Very Large Databases*, 1994.
[22] K. Svanberg. "The method of moving asymptotes—a new method for structural optimization." *Int. J. Numerical Methods in Eng.*, vol. 24, no. 2, 1987.
[23] S. G. Johnson. "The NLopt nonlinear-optimization package." http://ab-initio.mit.edu/nlopt.
[24] Y. C. Hu, et al. "Mobile edge computing—a key technology towards 5g." *ETSI white paper*, vol. 11, no. 11, 2015.
[25] M. Malawski, et al. "Cost minimization for computational applications on hybrid cloud infrastructures." *Future Generation Comput. Syst.*, vol. 29, no. 7, 2013.