2018 30th International Teletraffic Congress

TimeWeaver: Opportunistic One Way Delay Measurement via NTP

Ramakrishnan Durairajan*, Sathiya Kumaran Mani†, Paul Barford†‡, Robert Nowak†, Joel Sommers§
*University of Oregon, †University of Wisconsin-Madison, ‡comScore, Inc, §Colgate University

***Abstract*—One-way delay (OWD) between end hosts has important implications for Internet applications, protocols, and measurement-based analyses. We describe a new approach for identifying OWDs via passive measurement of Network Time Protocol (NTP) traffic. NTP traffic offers the opportunity to measure OWDs accurately and continuously from hosts throughout the Internet. Based on detailed examination of NTP implementations and in-situ behavior, we develop an analysis tool that we call TimeWeaver, which enables assessment of precision and accuracy of OWD measurements from NTP. We apply TimeWeaver to a ∼1TB corpus of NTP traffic collected from 19 servers located in the US and report on the characteristics of hosts and their associated OWDs, which we classify in a precision/accuracy hierarchy. To demonstrate the utility of these measurements, we apply iterative hard-threshold singular value decomposition to estimate the missing OWDs between arbitrary hosts from the highest tier in the hierarchy. We show that this approach results in highly accurate estimates of missing OWDs, with average error rates on the order of less than 2%.**

I. Introduction

End-to-end network latency plays a fundamental role in behavior and performance at different layers of the protocol stack. As such, measurements of network latency are included in many protocols and systems in order to adjust their behavior to network conditions. A canonical example is TCP's measurement of round-trip time (RTT) to adjust its sending behavior. Additionally, numerous network-focused applications have been developed over the years that are based specifically on latency measurements including network positioning and distance estimation (*e.g.,* [1]–[4]), available bandwidth estimation (*e.g.,* [5], [6])) and IP geolocation (*e.g.,* [7], [8])) to name a few.

Latency depends intrinsically on the routes that packets traverse between end points. If routes between hosts are symmetric, then RTT measurements would be appropriate for many protocols and applications. RTT measurements would also be a reasonable proxy for understanding network proximity, *e.g.,* in the context of selecting a "closest server". Indeed, this notion of latency-based proximity estimation is the basis for standard client redirection in content delivery networks (CDNs) [9] and certain DHTs (*e.g.,* [10]). However, routes are frequently asymmetric [11], [12], which reduces the utility of RTT measurement for proximity estimation and for real-time applications (*e.g.,* streaming and gaming), and makes one way delay (OWD) an important measurement objective.

Measuring end-to-end latency (RTT or OWD) has several basic challenges. First, there are a number of factors that contribute to latency in addition to the physical path including queuing and processing delays by nodes in the network. While one can consider the physical path to be relatively stable over moderate timescales, the latter two factors can introduce variability into measurements on shorter timescales [13]. This implies the need to specify a target measurement metric, *e.g.,* average RTT or minimum OWD (minOWD), and to devise an appropriate methodology for collecting and analyzing measurement data. Central to measurement methodology design are the precision and accuracy requirements. These may imply a relatively simple measurement system or a complex infrastructure based on dedicated hardware. Finally, measuring OWD has the additional requirement that host clocks must be synchronized.

In this paper, we consider the problem of measuring OWDs in the Internet. Our specific interest is in measuring OWDs at scale (*i.e.,* from many hosts in the Internet), with high precision and accuracy and without the need for complex, dedicated systems. We posit that such measurements could be applied to a wide variety of timely and important problems including those mentioned above.

The basis for our work is the vast quantity of OWD measurements that can be extracted from Network Time Protocol (NTP) packet exchanges—both from client to server, and server to client. NTP is pervasively used by hosts in the Internet to synchronize their clocks with high fidelity time sources [14, §2]. An intrinsic component of the protocol is estimation of OWD, which is used in the client clock adjustment algorithm.

We begin by developing a method for extracting accurate OWDs from NTP data. As pointed out in [15], OWD measurements extracted from NTP packets are not always accurate. Our filtering method is based on a detailed analysis of the NTP codebase [16]. Our analysis reveals regimes in the NTP packet exchange process that are observable in the traces, and which can be used to infer strong synchronization between clocks and thus accurate OWD measurement. We realize our filtering method in a framework that we call TimeWeaver, which organizes OWDs into measurement precision tiers that are based on observed client behavior. The tiers provide context for understanding the accuracy and utility of the measurements.

We assembled a ∼1TB corpus of NTP packet data, which was collected from 19 US-based NTP servers over a period of 30 days.There are 162,798,893 IPv4 and 6,056,609 IPv6 unique client addresses evident in the data. Examination of the client IP addresses indicates hosts from around the world are configured to synchronize to these servers.

We apply TimeWeaver to our NTP data and assess the re-

978-0-9883045-5-0/18/$31.00 ©2018 ITC
DOI 10.1109/ITC30.2018.00036

sulting OWDs vs. a prior filtering method and vs. active probe-based RTT measurements. We find that the sum of forward and reverse paths OWDs from the high precision/accuracy tier correlate well with probe-based measurements, and that TimeWeaver filters with much higher accuracy than [15]. Our analysis of OWD measurements reveals diverse characteristics including a range of OWDs in all tiers but the range is more narrow (typically < 100ms) in the high precision/accuracy tier.

Next, we further demonstrate the utility of OWD estimates based on opportunistic NTP measurements by applying them to the problem of inter-host distance estimation. While prior studies have considered this problem (*e.g.,* [17], [18]), our formulation differs in that we consider minOWDs (as opposed to round-trip times) in the context of a Euclidean space. Our matrix completion method for estimating inter-host distance is based on iterative hard-threshold singular value decomposition [19]. This algorithm iterates between truncating the SVD of the current estimate to a user-specified rank r, and then replaces the values in the observed entries with their original (observed) values. For Euclidean space, we consider $r = 4$ and apply the algorithm to minOWD measurements from NTP clients that contact more than one server. We find that the resulting distance estimates are highly accurate, with errors on the order of about 2%.

II. Background

The NTP ecosystem is composed of a hierarchy of servers. Starting at the top-level are servers with high-precision time sources such as GPS-based and atomic clocks. These servers, referred to as stratum 0, offer high-quality timing information to servers in the next level, stratum 1, which are also known to as *primary* servers. Stratum 2 or *secondary* servers connect to stratum 1 servers, etc., all the way down to stratum 15. In addition to connecting to a source up the hierarchy, NTP servers may also peer with others at the same level for redundancy.

Hosts in the Internet typically synchronize time with more than one server in order to compute a precise time estimate. Even though a host running a commodity operating system is configured with default NTP server(s) to synchronize time (*e.g.*, time.windows.com, time.apple.com, 0.pool.ntp.org), it can be manually configured to use a specific NTP server or a set of servers. Recent efforts like `ntp.org` also maintain lists of stratum 1 and stratum 2 servers that can be used after acquiring permission from the server administrators. NTP hosts or clients typically connect to reference clocks that are stratum 2 or higher.

Every NTP client host runs the `ntpd` daemon, which in turn runs several filtering algorithms and heuristics to synchronize its clock with reference clocks in the Internet. At a high level, `ntpd` operates by exchanging timestamps with its reference servers (in a process called *polling*). When and how often reference servers are polled is governed by the *clock discipline algorithm* [20]. In most operating systems, the polling interval starts with `minpoll` (64s) intervals and may eventually increase in steps to `maxpoll` (1024s) intervals. As part of its operation, the algorithm measures round-trip delay, jitter and oscillator frequency wander to determine the best polling interval [21].

Four timestamps are included in NTP packets as a result of each polling round: the time at which a polling request is sent (t_0), the time at which the request is received at the server (t_1), the time at which the response is sent by the server (t_2), and the time at which the response is received by the client (t_3). These timestamps are not set until after the completion of a handshake between client and server, which is indicated by the inclusion of an IPv4 address or hash of an IPv6 address in the `ref id` field of a request packet. Unfortunately, the logs do not contain explicit information regarding whether a client's clock is in "good" synchronization with the server, which is when differences between timestamps would offer the most accurate indication of OWD. In addition, a sizable number of hosts in the Internet use Simple NTP (SNTP) [22], which sets all packet fields to zero except the first octet which mainly contains metadata (*e.g.*, version number, stratum, poll interval, etc.). Due to the lack of explicit synchronization information in the NTP packets and the prevalence of SNTP clients (*e.g.*, mobile and wireless hosts) [23], we develop a framework to classify the precision of timestamps in an NTP packet as we discuss in §IV.

III. NTP Data Collection

To collect the NTP log data used in our study, we contacted several NTP administrators and explained our research goals. Eight administrators responded by providing datasets in the form of full pcap ("tcpdump") traces from a total of 19 NTP servers. Table I summarizes the basic statistics from each of the NTP server logs which includes 5 stratum-1 servers and 14 stratum-2 servers with a combination of both IPv4 and IPv6 support. These logs include a total of 6,369,784,837 latency measurements to 162,798,893 IPv4 and 6,056,609 IPv6 worldwide clients, as indicated by unique IP addresses, collected over a period of one month from Nov., 2015 to Dec., 2015. To facilitate network latency analysis, we developed a lightweight tool (about 800 lines of C code) to process/analyze the NTP logs.[1] Our efforts to amass server logs from NTP administrators can be replicated by anyone in the community due to the ubiquity of NTP servers in the Internet (*e.g.,* there are over 3.6K servers in `pool.ntp.org` alone [25]).

IV. Extracting One Way Delays from NTP data

In this section, we describe a framework called TimeWeaver for extracting and classifying OWDs from NTP packets. Our examination of NTP traces along with observations of others (*e.g.*, [15], [26], [27]) imply that latency measurements available through NTP packet exchanges may be skewed. A key aspect of the TimeWeaver framework is that in addition to adjusting NTP-derived latency measurements for skew, we

[1]The tool extends `netdissect.h` and `print-ntp.c` modules of tcpdump [24]. Each round of processing took ∼3 hours on a 16 core Intel workstation with 64GB RAM. The tool and the datasets collected are available upon request.

TABLE I
Summary of NTP server logs used in this study.

Server ID	Server Stratum	IP Version	Server Organization	Total Measurements	Total IPv4 Clients	Total IPv6 Clients
AG1	2	v4	Independent	349,917,829	12,889,722	0
CI1	2	v4/v6	ISP	23,201,076	1,337	88
CI2	2	v4/v6	ISP	22,163,583	1,009	71
CI3	2	v4/v6	ISP	24,836,284	701	62
CI4	2	v4/v6	ISP	23,846,973	573	44
EN1	2	v4/v6	ISP	13,166,749	581	46
EN2	2	v4/v6	ISP	13,381,258	536	41
JW1	1	v4	Commercial	11,498,989	337,015	0
JW2	1	v4	Commercial	40,330,009	864,845	0
MW1	1	v4	University	5,451,294	20,589	0
MW2	2	v4	University	1,850,765,317	60,682,989	0
MW3	2	v4	University	386,487,947	26,997,177	0
MW4	2	v4	University	355,913,460	16,758,046	0
MI1	1	v4	Commercial	1,899,642,404	27,133,385	0
PP1	2	v4/v6	Independent	10,090,072	690,486	0
SU1	1	v4/v6	ISP	590,431,652	16,206,848	6,052,784
UI1	2	v4/v6	University	302,622,909	58,967	1,363
UI2	2	v4/v6	University	270,990,678	98,159	1,147
UI3	2	v4/v6	University	175,046,354	55,928	963

assign measurements into different *precision tiers*[2] as we discuss below.

A. OWD precision framework

Overview. The TimeWeaver framework adopts the notion of precision discussed by Paxson [28], specifically that it is "the maximum exactness that a tool's design permits." Thus, the basic assumption we start with is that our precision assignment framework must be *NTP-specific*. That is, given the information available within the NTP packet traces (*e.g*, timestamps relative to client and server clocks, polling values), we do not expect to have success with a naïve approach like excluding extreme OWD values, or by only including values close to the minimum observed OWD. The reason, again, is that there is no meta-information available in protocol messages to indicate whether a client has reached good/close synchronization with the server. Instead, our approach is explicitly designed to exploit the ways in which the protocol *behaves* in response to good synchronization or events that degrade synchronization to create *tiers of precision*, each of which is suitable for various applications of interest. Extracted OWDs are assigned to a specific tier based on the inferred level of synchronization and the number and quality of measurements. Specifically, we define the following four precision tiers:

- *Tier 0*: These samples are from SNTP/NTP clients issuing a one-shot synchronization request. Unfortunately, no OWD information is available in these samples.
- *Tier 1*: This tier includes OWD measurements derived from clients using NTP which often exchange multiple packets with servers. The clients are either moving towards or away from close synchronization with the servers and the OWDs extracted are typically greater than one second with respect to the reference.
- *Tier 2*: Similar to Tier 1, OWD measurements in their tier are from clients that exchange multiple packets with servers and cannot be confirmed to be in close synchronization. The main difference with Tier 1 is that the OWDs are less than one second.
- *Tier 3*: This tier includes highly accurate OWD measurements from clients which are observed to be tightly synchronized with their NTP references.

We first exploit NTP behavior by considering the polling operation of clients, dividing them into two basic classes: constant or non-constant polling. Our motivation is similar to that of prior work [15] in that we attempt to take advantage of polling behavior in order to detect whether the client is in good or poor synchronization with the server. For example, an intended protocol behavior is for a client to increase its polling rate (reduce the polling interval) in response to poor synchronization. Likewise, in response to detection of good synchronization, a client may reduce its polling rate (increase the polling interval). Unfortunately, this is not sufficient, as there are clients that do not vary their polling rate at all. For these clients, we use similar heuristics to those within the NTP protocol [21] and code [16] to identify high-quality latency samples as we discuss below.

Algorithm. As part of developing this algorithm, we conducted a detailed study of the NTP codebase [16], request and response transactions, protocol behaviors, packet fields and packet selection heuristics. We also experimented with different NTP client/server configurations (*e.g.*, Mac OS, Linux, and Windows) and path properties (*e.g.*, latency, loss, asymmetry, etc.) in a controlled laboratory setting. Our goal was to understand the *operational aspects* of NTP in detail. Specifically, we conducted measurements in different settings: *(i)* distant client synchronizing with a local NTP server, *(ii)* local client synchronizing with a distant NTP server and *(iii)* local client synchronizing with a local server. From our source code analysis and controlled experiments we identified two specific features of the protocol to leverage in our filtering algorithm: the client-estimated ground truth RTT (gtRTT) value, which is used when a client polls at a *constant* rate, and the jiggle counter heuristic [21] used in NTP's client selection algorithm, which is used when a client exhibits *non-constant* polling behavior. In addition to these NTP-specific filtering methods, we found that we needed to eliminate spikes in OWD samples, as we discuss below.

Non-constant polling behavior. For clients with non-constant polling values, we use insights from the client selection heuristic [21] to select what are likely to be high-quality latency samples. For a given polling value ($2^{P_e^c}$, where P_e^c is the polling exponent), the algorithm requires at least N samples (where N = $30/P_e^c$)[3] before deciding to increase or decrease the polling interval. This algorithmic detail implies that when we observe the same polling value for fewer than N samples, we infer that the clock is going to a bad state

[2]Our original goal was a quantitative precision framework based on standard deviations of repeated measurements. However, the highly dynamic nature of NTP renders this approach unreliable. We will show that our tiered framework provides a reliable and useful context for interpreting the OWDs extracted by TimeWeaver.

[3]This expression was derived by NTP's designers through years of experimentation and experience [21].

(*i.e.*, losing synchronization) and assign the corresponding measurements to tier 2. When we observe exactly N samples in our logs with the same polling value followed by samples with an increased polling interval (*i.e.*, polling rate decreases), we infer that the N samples must have been accepted by NTP's algorithm as good clock values and we therefore accept the N samples too. Similarly, N samples followed by a decreased polling interval corresponds to clock values that we infer to be of poorer quality, thus we assign these N samples to tier 2. If the number of samples is greater than N, we infer the client's clock to be in an unstable state, either shifting from a bad state to a good state (if polling interval increases), or from a good state to a bad state (if polling interval decreases). In either case, we cannot determine which samples are good and hence we assign all these samples to tier 2.

Constant polling behavior. When a client sends a request to an NTP server, it sets the origin timestamp to be equal to the transmit timestamp from the previous server response. We refer to this behavior as *timestamp rotation*. Since our logs are captured at the server, we can obtain the server-to-client (s2c) latencies because of timestamp rotation as t_3 - t_2. Similar to clients, servers also rotate timestamps when they send out an NTP response. Hence we can also estimate client-to-server (c2s) latencies as t_1 - t_0. Timestamp rotation is an expected NTP protocol behavior and is used to prevent replay attacks (see [29], p.28).

We can also recover the client-computed gtRTT between a client and a server which is reported by the client's `ntpd` after correcting the system clock. After the initial handshake between a client and a server, the client sets both the `ref id` and `root delay` fields in outgoing NTP request packets to the server's IP address and gtRTT estimate respectively. Thus, when we see a value in `ref id` set to the IP addresses of one of our NTP log collection servers, we can get the client's estimate of gtRTT to our servers from the `root delay` field. This offers an opportunity to enhance the filtering process. Furthermore, we found that for some client implementations the `ref id` field is set to a wide variety of different NTP server addresses, thus providing client-computed RTT values between the client and multiple other servers.

We apply our observations on timestamp rotation behavior and the inclusion of gtRTT to filtering latency samples for clients that exhibit a constant polling interval. First, we check if the packets between clients and servers contain gtRTT values. Next, if the gtRTT values are present, we extract them from the root delay field and select only those packets in which the sum of OWDs is less than or equal to gtRTT. If the gtRTT value is absent, we default to the mean plus one-sigma deviation filter similar to prior work [15].

Sample smoothing. Finally, even with these NTP-specific techniques for filtering OWD samples, there may yet be spikes that cause inaccuracies. We apply a simple EWMA filter to smooth spikes in latency measurements in all tier 3 accepted samples. To choose the weighting factor (α), we iterate from 0.1 to 0.9 such that the mean-squared error of filtered latencies is minimized.

Apart from the measurements with constant or non-constant polling value, a number of samples are either from SNTP clients with one-shot requests or from NTP clients sending (single or multiple) one-shot requests in our logs. Since some or all of the timestamps are empty in the one-shot NTP and SNTP measurements, we assign them to tier 0 if OWD cannot be inferred. In addition, we found that a sizable number of measurements also exhibited similar behavior despite multiple one-way requests sent by NTP clients but with OWD information that we cannot verify. We assign such measurements to tier 1.

Putting it all together. We implemented our precision framework in about 1,250 lines of C++ code and applied it to our NTP traces. The result is a set of measurements assigned to one of the following four tiers. The raw number of measurements assigned to each tier are depicted in Table II.

- First, we assign all the one-shot measurements (*e.g.* SNTP) to *tier 0*. These include measurements with empty values in the timestamp field, partially filled timestamp fields, etc. We note again that we cannot infer any latency information from these measurements.
- Next, we assign the measurements that we filter using our heuristics-based technique to *tier 3*. OWD measurements in this tier are from well-synchronized clients. As a by-product of the tight synchronization between clients and references, we have a set of accurate client-to-server/server-to-client OWD measurements in this tier.
- Next, those measurements that are rejected by our filter are assigned to *tier 2*. The measurements in this tier are from clients trying to achieve synchronization with references, but are not well-synchronized. We note that we also apply a constant bound for measurements: if the OWDs are less than 1000ms, they are assigned into tier 2.
- Finally, in the above tier 2 classification, OWDs greater than 1000ms are classified as *tier 1*. Furthermore, since we cannot infer the level of synchronization for a number of SNTP clients despite having multiple OWD samples, we include those measurements in *tier 1*.

To illustrate concretely, Figure 1 shows the *raw* polling values (top plot; dashed green curve) and *unfiltered* latencies (2nd to top plot; solid yellow curve) extracted from NTP logs for a client in our lab. In addition to the polling values and unfiltered latencies, the figure shows gtRTT/2 (from the root delay field) extracted from NTP logs (bottom plot; grey curve). Observe that between times 25000–37500 the polling interval increases and, indeed, there is a stabilization of OWDs to accurate values. Likewise, between times 60000–70000 the polling interval decreases and there is some corresponding loss of stability in OWD samples. The bottom two plots of Figure 1 show that during time 40000–60000 and the stabilization of OWD (and likewise, the longest increasing trend in polling value), latency samples are classified and filtered as tier 3 measurements (bottom plot; black curve), while before and after that stable period samples are classified as tier 2 (3rd

plot from top; magenta curve). Also shown in the bottom plot are the min., max. and avg. of the tier 3 filtered latencies. For this experiment, only tier 2 and 3 latencies resulted from processing the raw samples.

The tier 3 filtered samples shown in the bottom plot of Figure 1 represent the most accurate estimates from the TimeWeaver framework. We observe that the `ntpd` client's computed latency value (gtRTT/2) aligns well with our filtered OWD estimate. Notice also the figure shows several spikes in unfiltered OWD samples (*e.g.*, two spikes between times 40000 and 50000) among other deficiencies, which are effectively addressed through the TimeWeaver framework. Through extensive examination of many individual client traces, results for which are not shown here due to space constraints, we found that our filtering technique correctly and consistently eliminates poor samples and spikes that would otherwise pollute OWD estimates.

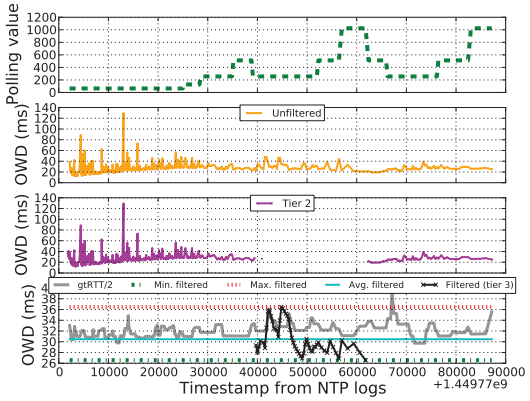

Fig. 1. **Latencies extracted from NTP logs from a lab-based client before and after applying filtering. Tier 3 and tier 2 measurements are shown in black and magenta, respectively.**

B. Comparison with ping measurements

To assess the effectiveness of our approach, the administrators of several of the NTP servers (MW1-4) used `ping` to send 10 probes each to a random sample of more than 20,000 client hosts identified in their logs. Ping measurements were issued simultaneously on a day that the NTP data was collected. We do not argue that ping measurements provide ground truth, rather that they provide a useful perspective on the NTP measurements. We include clients for which our algorithm assigned tiers 1, 2 or 3 for OWD samples. 6,370 out of 21,443 target clients responded to the pings.

Figure 2 shows a scatterplot comparison of the minimum of $s2c{+}c2s$ filtered latencies derived from NTP logs for the 6,370 clients compared with the corresponding minimum RTT values from ping measurements. Latencies from 3,708 clients were assigned to tier 3 by our framework and are shown as green circles, and 2,662 latency samples were in tiers 2 and 1 and are shown in red triangles and blue dots respectively. From these data points, we observe that there are no extreme outliers that are colored green. This indicates that our precision assignment approach is effective in assigning poor latency samples to lower tiers. Furthermore, a number of clients were assigned to lower tiers by our algorithm even though the $c2s{+}s2c$ latencies were comparable with the RTT measurements. On detailed examination of such clients, we found that the polling values were oscillating at the time when the packets were captured at these four servers. In such cases, without further information we must treat the latency samples as indeterminate and thus assign them to a lower tier. Overall, our results show that latency samples from NTP packet traces can indeed be used to derive OWD estimates of different precisions that are suitable for various applications.

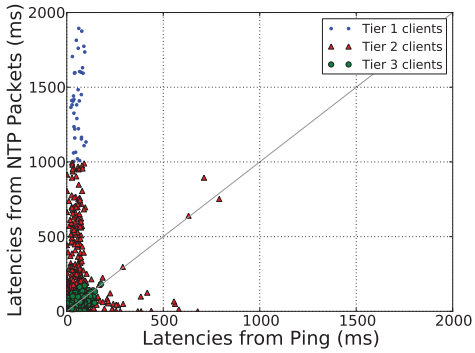

Fig. 2. **Comparison of measured minimum $c2s + s2c$ latencies from NTP packets and RTT from ping measurements. Clients assigned to tier 3, 2 and 1 are denoted by green circles, red triangles, and blue dots, respectively.**

C. TimeWeaver vs. prior NTP filtering

A natural question is whether the filtering approach described in prior work (see §4 in [15]), can be used to remove bad OWD measurements? To answer this question, we used the code from [15] and compared its filtering output vs. TimeWeaver. Specifically, we randomly selected logs from our data corpus from multiple NTP servers across multiple days and compared the client and latency characteristics of TimeWeaver versus those produced from the prior method. Based on comparisons using one day's-worth of data from the JW1 server[4], we found that the filtering approach used in [15] is not widely applicable for the following reasons:

Client characteristics. *(1)* Out of the 18,620 unique clients seen in the log of JW1 server on a randomly-selected day, [15] only considers 8,804 clients due to its US-only filtering constraint. On the contrary, TimeWeaver considers all 18,620 clients spread across many countries. *(2)* Of the 8,804 clients considered by the prior method, a large fraction of clients (*i.e.*, about 3,631) were rejected due to missing timestamps, negative latency values, and other reasons. TimeWeaver, on the other hand, assigns such discarded measurements to lower tiers, making it possible to use less accurate OWD values in applications that have less stringent accuracy requirements.

Latency characteristics. Apart from considering the US-only clients, the method from [15] also uses a 100ms OWD

[4]Results from other days and other NTP servers exhibited similar characteristics.

TABLE II
Number of measurements assigned to each tier by TimeWeaver.

	AG1	CI1	CI2	CI3	CI4	EN1	EN2	JW1	JW2	MW1	MW2	MW3	MW4	MI1	PP1	SU1	UI1	UI2	UI3
Tier 0	1.5e8	9.4e6	9.7e6	1.1e7	1.1e7	5.4e6	5.5e6	3.0e6	1.7e7	2.4e6	5.2e8	1.8e8	2.6e8	5.5e8	4.7e6	2.4e8	1.9e8	6.7e7	2.7e7
Tier 1	1.3e8	5.2e6	5.2e6	4.9e6	5.4e6	4.9e5	8.5e5	2.8e6	1.6e7	1.1e6	2.6e8	1.1e8	1.1e8	4.6e8	4.0e6	2.2e8	3.6e7	5.2e7	2.0e7
Tier 2	4.2e7	7.0e6	6.5e6	7.7e6	6.9e6	6.5e6	6.2e6	3.3e6	4.5e6	9.2e5	3.7e7	5.4e7	1.7e7	3.9e8	6.4e5	6.8e7	5.5e7	1.2e8	9.6e7
Tier 3	3.0e7	1.5e6	8.4e5	1.4e6	7.0e5	7.8e5	8.4e5	2.4e6	2.9e6	1.0e6	1.3e7	1.7e7	6.0e6	5.1e8	7.8e5	5.6e7	1.9e7	3.6e7	3.2e7

threshold to limit wired vs. wireless hosts. As a consequence, the observed latency characteristics after applying TimeWeaver (bottom) are completely different from [15] (top), as depicted in Figure 3. First, the maximum of the extracted minimum OWD is 100ms in prior work, whereas the maximum value for a client in TimeWeaver's tier 3 category is 992ms. Second, about 80% of the clients filtered using [15] exhibited a latency less than 50ms, while only 14% of the tier 3 clients had OWDs less than 50ms using TimeWeaver due to a more flexible and NTP-specific filtering approach.

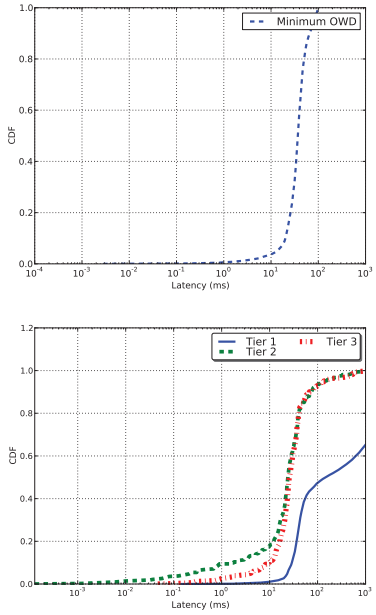

Fig. 3. **CDF of minimum OWD latencies extracted using Durairajan *et al.* [15] (top) versus TimeWeaver (bottom). Note that the x axis for TimeWeaver (right) is cut at 1000ms to make the plots more comparable.**

V. Internet distance estimation from OWDs

In this section, we demonstrate the utility of OWDs extracted for predicting unobserved latencies between clients as well as for predicting latencies between NTP servers and clients that were not measured. The predictions are based on the tier 3 subset of OWDs extracted through TimeWeaver's precision assignment algorithm. A key observation we make is that a matrix of Euclidean distances between points in the 2-dimensional plane has rank 4. The matrix of geodetic distances [30] on the sphere is not exactly low-rank, but is well-approximated by a low-rank matrix. This implies a significant level of correlation must exist among the pairwise latencies, which our algorithm exploits.

A. Problem setup

We organize the latency matrix X in the following block-form. Given m NTP servers and n clients, the latency matrix is $(m+n) \times (m+n)$ and can be arranged as follows

$$X = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$$

where A is the m x m sub-matrix corresponding to inter-NTP server latency measurements, B is the n x m block that corresponds to the c2s from m NTP servers to n clients, and B^T is the m x n sub-matrix corresponding to the s2c distances from n clients to m NTP servers. Distance estimates from both B and B^T are extracted from the NTP logs and are partially incomplete sub-matrices. The matrix C corresponds to the c2c distances, and is completely unobserverd.

Although C is completely unobserved, it may still be possible to estimate these latencies from the measured data. To understand why, suppose that we fully observe A and B. If rank$(X) = 4$ (as discussed above), $m >= 4$, and the first m rows/columns have rank 4, then we can complete C with

$$C = BA^{\dagger}B^T \quad (1)$$

where $A^{\dagger}$ is the *pseudo-inverse* of A.

Unlike prior efforts on distance estimation that assume a full matrix of RTT measurements [1]–[4], the situation we face is more challenging. The matrix B is only partially observed in the logs and the matrix A is not observed at all. To deal with this, we propose the following: *(1)* estimate the latencies in A from the *known* physical distances between NTP servers, and *(2)* employ a low-rank matrix completion method to deal with the missing entries in B and C.

B. Estimating inter-NTP server distances

To estimate the distances between the m NTP servers, we use the following approach. First, from the NTP pool server list [25], we obtain the physical coordinates of the m NTP servers used in our study. Using the Vincenty formula [31], we compute the line-of-sight physical distances between all NTP servers. We calculate the speed-of-light estimate of latencies as roughly $2/3^{rd}$ the speed of light in air [32] to obtain, A_{geo}, the geo-based estimate of A.

Next, we reached out to the NTP server administrators to measure the inter-NTP server latencies; 5 administrators (managing 11 servers) responded positively. We performed 10 ping measurements from each of the 11 NTP servers and used $rtt/2$ of the minimum of ping measurements to create A_{rtt}, the rtt-based estimate of A.

Lastly, since A_{rtt} is partially incomplete and A_{geo} only gives the *ideal* lower bound of distances, we use a scaling

factor, γ, to obtain A. To derive γ, we use a simple linear model to capture the sharing of information in the data (*e.g.*, all MW servers are located in Madison, WI) and γ is based on linear model coefficients β_0 and β_1. That is, the coefficients are obtained by solving a simple linear regression ($y = \beta_0 + \beta_1 x$) for non-zero entries in A_{rtt} and using the obtained β_0 and β_1 on A_{geo} for those measurements for which we do not observe distances in A_{rtt}. The final inter-NTP distance matrix A is a combination of A_{rtt} and γ applied on A_{geo}.

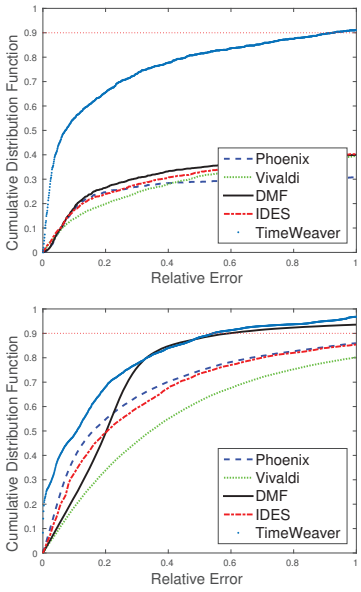

Fig. 4. **CDF of relative errors using our approach (labeled TimeWeaver), Vivaldi, Phoenix, DMF and IDES after matrix completion for incomplete (top) and completed (bottom) data.**

C. Distance estimation algorithm

The distance estimation algorithm we employ is based on iterative hard-threshold singular value decomposition (IHTSVD) [19]. This is an iterative algorithm that alternates between truncating the SVD of the current estimate to a user-specified rank k, and then replacing the values in the observed entries with their original (observed) values. The algorithm can be initialized by setting the missing entries in X arbitrarily. In our experiments we initialize the missing values with the mean latency in the NTP logs. Because Euclidean distance matrices in 2-dimensions have rank 4, we set $k = 4$ in the algorithm. We apply the algorithm to tier 3 minOWD values from NTP clients that contact four or more servers. Since mobile and wireless clients can confound our estimation, we identify and remove those clients using Cymru lookup [33].

D. Assessing predicted distances

Comparison with other techniques. We compare the relative error of the our distance estimates against prior distance estimation techniques including Vivaldi [1], IDES [2], Phoenix [3], and DMF [4]. Figure 4-(top) shows the CDF of relative error made by TimeWeaver-based predictions versus the other methods for predicting the missing values in incomplete matrix X, where the sub-matrix B is partially observed and sub-matrix A is completely unobserved. The same OWD data is used in all cases. For 50% of the estimates, TimeWeaver-based predictions were off by at most 6% from the original values, while for *only* 12% of the estimates, similar relative errors were achievable using the other distance estimation techniques.

Figure 4-(bottom) shows the CDF of relative error for predictions from the same set of entities (mentioned above) but for the completed matrix X. In this analysis, the estimates were randomly held-out and then predicted again. The plots show that TimeWeaver-derived distance estimates are perfectly accurate for 20% of the estimates, and have a relative error of 10% for 50% of the estimates. For 80% of the estimates, the predictions are off by 34% and beyond that the results are comparable with DMF.

We approach latency/distance estimation as a low-rank matrix completion problem. The basic ingredients in the algorithm (matrix factorization) are also used in the prior methods. However, our approach has features that can explain its superior performance. Vivaldi aims to explicitly determine a low-dimensional embedding of the network that agrees with the measured latencies; we target distance estimation directly. In this sense, our approach is similar to DMF, although we use a centralized global optimization procedure and do not require regularization beyond that imparted by the low-rank constraint. IDES is also similar, but is landmark-based and assumes few if any missing measurements. In contrast, our approach is designed to handle cases in which most of the data are missing. The results in this section show that even when prior methods use minOWD, predictions using our approach are more accurate, especially for the situation in which we have incomplete data.

Self-consistency checks. In this analysis, we randomly hold out available OWD values from matrix X and compare them against the predicted values. Figure 5 shows the CDF of distances held-out and the corresponding distances predicted by our algorithm. For all these different held-out client groups, our approach produced highly accurate estimates of OWDs with an average error rate on the order of less than 2%.

E. Applicability to non-US regions

The gains that we see in distance estimation accuracy are fundamentally based on TimeWeaver's ability to accurately filter and extract OWD measurements from NTP logs, making it robust against the effects of asymmetry or NTP errors. Hence, even for other non-US regions that are susceptible to circuitous routing [34], we argue that our methods are resistant to routing changes as long as accurate *minimum* OWD measurement is available despite the presence of other large and varying OWD measurements. We plan to investigate this in detail as part of future work.

VI. Related Work

Internet path characteristics. Empirical measurement of Internet path properties (*e.g.*, latency, loss, etc.) has been an

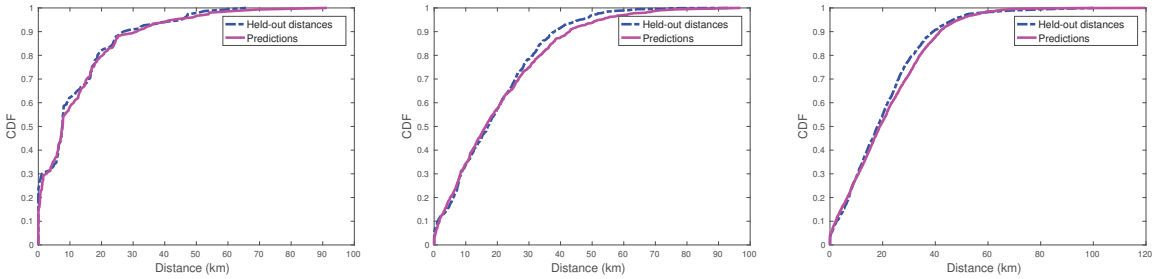

Fig. 5. **CDF of average relative prediction error (hold-out distances vs. predicted distances) for 100 (left), 1000 (center) and 3668 (right) clients.**

active area of research for many years. Early studies of path latencies include Mills's report on RTTs collected using ICMP echo requests [35] and Bolot's work, which also examined packet loss and queuing [36]. The landmark work of Paxson used a set of specially-deployed systems to measure packet loss and latency [11], and has informed much of the ongoing work in this area. There are a number of efforts today that take a similar approach of having specially-deployed systems to collect an essentially continuous stream of measurements such as latency, loss, and routing [37]–[39].

There are a number of specific efforts that have focused on accurate estimation of OWD to account for asymmetry. A common approach is to assume host clocks have been synchronized (*e.g.*, [40]–[42]) and to accept OWD measurements at face value. Other work has explicitly addressed correction for clock offset, drift, and skew, *e.g.*, [11], [43], [44]. Yet other works have attempted to estimate OWD using timestamps in flow records [45] or through analysis of multiple one-way measurements collected from a group of unsynchronized hosts [46], [47]. An extensive analysis of path asymmetric delays and their prevalence was done by Pathak *et al.* [48] using the `owping` tool [49], emphasizing the need for accurate OWDs.

Our work generalizes and extends prior work that identifies NTP as a source of latency measurement [15], [26], [27]. In particular, we develop a comprehensive filtering algorithm based on detailed examination of the NTP codebase that enables accurate OWDs to be identified, and we develop and propose new methods for distance estimation based on the availability of these measurements.

Internet distance estimation. Apart from measuring latencies, there have been a variety of techniques developed to estimate latencies between arbitrary nodes in the Internet. IDMaps [17] examined network distance prediction from a topological perspective and influenced later work on King [50], which expands on the IDMaps technique but uses DNS servers as landmarks, and Meridian [51] which probes landmarks on demand to predict network distances. The work by Ng and Zhang on GNP [18] uses a low-dimensional Euclidean space to embed the nodes by relying on well-known pivots (or landmarks). Similar to GNP, Lighthouse [52] uses a transition matrix to achieve embedding with reference to any pivots. Tang *et al.* propose a virtual landmark-based embedding scheme [53] which is computationally efficient and is independent of landmark positions. Subsequent efforts used different embedding systems, resulting in different performance and accuracy characteristics [1], [54], and the work by Mao *et al.* [55] proposes matrix factorization techniques to determine network distances. One of the interesting questions raised by Madhyastha *et al.* [56] regarding matrix factorization is how OWDs from landmarks to arbitrary clients might be measured. In light of our work, NTP servers naturally become the landmarks and the OWDs to a large and distributed set of clients are easily obtained.

VII. Summary and Future Work

In this paper, we consider the problem of gathering OWD measurements in the Internet, at scale. Our approach is based on passive measurement and analysis of NTP traffic. Based on detailed analysis of the protocol, of the codebase, and of NTP traces, we develop a new method and tool called TimeWeaver for correcting and filtering OWD measurements extracted from NTP packets. We apply TimeWeaver to a $\sim$1TB corpus of NTP trace data collected over a period of 30 days from 19 servers in the US. We find that TimeWeaver offers much greater accuracy for its resulting top-tier OWD estimates than prior work. We also find that TimeWeaver's filtering and precision classifying approach results in a much broader set of OWD measurements that can be extracted from raw NTP traffic. To illustrate the utility of having accurate OWD measurements, we approach the problem of distance estimation under the assumption that accurate minOWD data is available and that measurement data can be missing or incomplete. We use minOWD estimates from a subset of NTP clients for which we have the highest tier (tier 3) estimates, and which contact multiple servers. We apply iterative hard-thresholded SVD to complete the matrix of inter-host delays. We find that the resulting estimates are highly accurate with relative errors are on the order of 2%.

Currently, we are extending TimeWeaver for real-time OWD estimation. This requires a source for real-time NTP measurements at strategic locations (`pool.ntp.org` is a simple and cost effective option), and adaptation of TimeWeaver to operate on streams of NTP data. We plan to consider three application areas: IP geolocation, census and survey of active Internet addresses, and network operations and management. Finally, we believe that TimeWeaver-based OWD measurements from various tiers have significant potential for continuous assessment of network availability, performance

and fault monitoring. We intend to consider each of these areas in our future work.

ACKNOWLEDGMENTS

We thank the NTP operators for providing server logs. We thank our shepherd, Kenjiro Cho, and the reviewers for their insightful comments. This work is supported by NSF grants CNS-1703592, DHS BAA 11-01, AFRL FA8750-12-2-0328. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, DHS, AFRL or the U.S. Government.

REFERENCES

[1] F. Dabek and R. Cox and F. Kaashoek and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," in *ACM SIGCOMM*, 2004.
[2] Y. Mao and L. K. Saul and J. M. Smith, "IDES: An Internet Distance Estimation Service for Large networks," in *IEEE JSAC*, 2006.
[3] Y. Chen, X. Wang, X. Song, E. Lua, C. Shi, X. Zhao, B. Deng, and X. Li, "Phoenix: Towards an Accurate, Practical and Decentralized Network Coordinate System," in *Networking*, 2009.
[4] Y. Liao and P. Geurts and G. Leduc, "Network Distance Prediction based on Decentralized Matrix Factorization," in *Networking*, 2010.
[5] M. Jain and C. Dovrolis, "End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput," in *ACM SIGCOMM*, 2002.
[6] J. Sommers, P. Barford, and W. Willinger, "A Proposed Framework for Calibration of Available Bandwidth Estimation Tools," in *IEEE ISCC*, 2006.
[7] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida, "Constraint-Based Geolocation of Internet Hosts," *IEEE/ACM TON*, 2006.
[8] B. Wong, I. Stoyanov, and E. Sirer, "Octant: A Comprehensive Framework for the Geolocation of Internet Hosts," in *USENIX NSDI*, 2007.
[9] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao, "Moving Beyond End-to-end Path Information to Optimize CDN Performance," in *ACM IMC*, 2009.
[10] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-peer Systems," in *IFIP/ACM Middleware*, 2001.
[11] V. Paxson, "Measurements and Analysis of End-to-end Internet Dynamics," Ph.D. dissertation, University of California, Berkeley, 1997.
[12] M. Sánchez, J. Otto, Z. Bischof, D. Choffnes, F. Bustamante, B. Krishnamurthy, and W. Willinger, "Dasu: Pushing Experiments to the Internet's Edge." in *Usenix NSDI*, 2013.
[13] Y. Zhang and N. Duffield, "On the Constancy of Internet Path Properties," in *ACM IMW*, 2001.
[14] R. Durairajan, S. K. Mani, P. Barford, R. Nowak, and J. Sommers, "TimeWeaver: Opportunistic One Way Delay Measurement via NTP," https://arxiv.org/abs/1801.02123, 2018.
[15] R. Durairajan and S. Mani and J. Sommers and P. Barford, "Time's Forgotten: Using NTP to Understand Internet Latency," in *ACM HotNets*, 2015.
[16] "NTP Codebase." https://github.com/ntp-project/ntp, 2016.
[17] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service," *IEEE/ACM TON*, 2001.
[18] E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-based Approaches," in *IEEE INFOCOM*, 2002.
[19] E. Chunikhina and R. Raich and T. Nguyen, "Performance Analysis for Matrix Completion via Iterative Hard-thresholded SVD," in *IEEE SSP Workshop*, 2014.
[20] "NTP Clock Discipline Algorithm." https://www.eecis.udel.edu/~mills/ntp/html/discipline.html.
[21] "NTP Polling Interval." http://www.eecis.udel.edu/~mills/ntp/html/poll.html.
[22] D. Mills, "Simple Network Time Protocol (SNTP)," https://tools.ietf.org/html/rfc1769, March 1995.
[23] S. K. Mani, R. Durairajan, P. Barford, and J. Sommers, "MNTP: enhancing time synchronization for mobile devices," in *ACM IMC*, 2016.
[24] "TCPDUMP," https://github.com/the-tcpdump-group/tcpdump.
[25] "NTP Pool Servers," http://pool.ntp.org.
[26] "NTP Clock Filter Algorithm." https://www.eecis.udel.edu/~mills/ntp/html/filter.html.
[27] J. Ridoux and D. Veitch, "Principles of Robust Timing over the Internet," *Queue*, 2010.
[28] V. Paxson, "Strategies for Sound Internet Measurement," in *ACM IMC*, 2004.
[29] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification," https://tools.ietf.org/html/rfc5905, June 2010.
[30] "World Geodetic System 1984," http://www.dtic.mil/docs/citations/ADA167570.
[31] "Vincenty Formula." https://en.wikipedia.org/wiki/Vincenty\%27s_formulae.
[32] A. Singla and B. Chandrasekaran and B. Godfrey and B. Maggs, "The Internet at the Speed of Light," in *ACM HotNets*, 2014.
[33] "Team Cymru whois lookup service." http://www.team-cymru.org/IP-ASN-mapping.html\#whois.
[34] A. Gupta, M. Calder, N. Feamster, M. Chetty, E. Calandro, and E. Katz-Bassett, "Peering at the Internet's Frontier: A First Look at ISP Interconnectivity in Africa," in *PAM*, 2014.
[35] D. Mills, "Internet Delay Experiments," https://tools.ietf.org/html/rfc889, December 1983.
[36] J. Bolot, "End-to-end Packet Delay and Loss Behavior in the Internet," in *SIGCOMM CCR*, 1993.
[37] "CAIDA's Ark Project." http://www.caida.org.
[38] "PingER: Ping End-to-end Reporting," http://www-iepm.slac.stanford.edu/pinger/, 2015.
[39] "RIPE Atlas," https://atlas.ripe.net, 2015.
[40] A. Hernandez and E. Magana, "One-way Delay Measurement and Characterization," in *IEEE ICNS*, 2007.
[41] L. D. Vito, S. Rapuano, and L. Tomaciello, "One-way Delay Measurement: State of the Art," *IEEE TIM*, 2008.
[42] M. Shin, M. Park, D. Oh, B. Kim, and J. Lee, "Clock Synchronization for One-way Delay Measurement: A Survey," in *Advanced Communication and Networking*, 2011.
[43] A. Pasztor and D. Veitch, "A Precision Infrastructure for Active Probing," in *PAM*, 2001.
[44] ——, "PC-based Precision Timing Without GPS," in *ACM SIGMETRICS*, 2002.
[45] J. Kögel, "One-way Delay Measurement Based on Flow Data: Quantification and Compensation of Errors by Exporter Profiling," in *ICOIN*, 2011.
[46] O. Gurewitz, I. Cidon, and M. Sidi, "One-way Delay Estimation using Network-wide Measurements," *IEEE/ACM TON*, vol. 14, no. SI, 2006.
[47] A. Vakili and J.-C. Gregoire, "Accurate One-way Delay Estimation: Limitations and Improvements," *IEEE TIM*, 2012.
[48] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao, "A Measurement Study of Internet Delay Asymmetry," in *PAM*, 2008.
[49] S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, and M. Zekauskas, "RFC 4656: A One-way Active Measurement Protocol (OWAMP)," September 2006.
[50] K. Gummadi, S. Saroiu, and S. Gribble, "King: Estimating Latency between Arbitrary Internet End Hosts," in *ACM IMW*, 2002.
[51] B. Wong, A. Slivkins, and E. Sirer, "Meridian: A Lightweight Network Location Service Without Virtual Coordinates," in *ACM SIGCOMM*, 2005.
[52] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for Scalable Distributed Location," in *USENIX IPTPS*, 2003.
[53] L. Tang and M. Crovella, "Virtual Landmarks for the Internet," in *ACM IMC*, 2003.
[54] Y. Shavitt and T. Tankel, "On the Curvature of the Internet and its Usage for Overlay Construction and Distance Estimation," in *IEEE INFOCOM*, 2004.
[55] Y. Mao and L. K. Saul, "Modeling Distances in Large-scale Networks by Matrix Factorization," in *ACM IMC*, 2004.
[56] H. V. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani, "A Structural Approach to Latency Prediction," in *ACM IMC*, 2006.