

Clockscalpel: Understanding root causes of Internet clock synchronization inaccuracy

Chi-Yao Hong, Chia-Chi Lin, and Matthew Caesar

University of Illinois at Urbana-Champaign

Abstract. Synchronizing clocks is an integral part of modern network and security architectures. However, the ability to synchronize clocks in modern networks is not well-understood. In this work, we use testbeds equipped with a high-accuracy GPS receiver to acquire ground truth, to study the accuracy of probe-based synchronization techniques to over 1861 public time servers. We find that existing synchronization protocols provide a median error of 2 – 5 ms, but suffer from a long-tail. We analyze sources of inaccuracy by decoupling and quantifying different network factors. We found that most inaccuracies stem from asymmetry of propagation delay and queuing delay. We discuss possible schemes to compensate these errors to improve synchronization accuracy.

1 Introduction

Nearly every activity that involves multiple participants requires synchronized time to operate at peak levels, from plane departures and sporting events, to financial transactions, to business, power control, and industrial processes. In networks, technologies such as directory services, collaboration, authentication, monitoring, diagnosis, and coordination are highly dependent on accurate, synchronized time to operate effectively. Synchronizing time is known to simplify a number of distributed algorithms by removing uncertainty of message delivery times, can simplify network monitoring and forensics by establishing ordering and timing of events across distributed vantage points, and has been used as a core building block in a number of Internet measurement studies [1, 2].

To support these functions, probe-based synchronization protocols such as *Network Time Protocol* (NTP) and the *Simple Network Time Protocol* (SNTP) are widely used. These protocols comprise one of the largest Internet systems, with hundreds of thousands of NTP servers providing service to tens of millions of clients. Understanding the performance of applications that require synchronized time requires understanding the accuracy achievable by these underlying protocols. However, despite the high degree of reliance that many networked systems have on probe-based synchronization protocols, we lack an understanding of how these protocols behave in modern, wide-area networks.

There exists relatively little attention on the performance of probe-based time synchronization in the wide area over the past decade – most recent surveys date back to 1999 [3] and 1990 [4]. Recently, novel frameworks [5, 6] are

⁰ The authors wish the published data set (§3) to be considered for the award.

proposed to further improve the synchronization accuracy. While these studies showed that some NTP servers are inaccurate, the underlying cause of inaccuracies remain unclear. However, with the advent of modern network applications that have stringent end-to-end latency requirements on the order of milliseconds or microseconds [2], such as VoIP, interactive video conferencing, automated trading, and high performance computing, understanding *network factors* that affect time synchronization performance becomes increasingly important.

In this work, we take some preliminary steps towards understanding the ability to synchronize time in the wide-area Internet, leading up to potential frameworks that effectively improve synchronization inaccuracy. In particular, we study the precision of existing time protocols and characterize their performance in terms of the level of synchronization accuracy it provides to hosts. We leverage GPS hardware (which has greatly reduced in cost over the past decade) to provide “ground truth” clock information to parts of our measurement infrastructure. We use this information to directly study the underlying sources of inaccuracies, assess their impact, and evaluate the potential to compensate for or remove these sources.

We have three key findings. First, we find that existing synchronization protocols work well in the common case (with median accuracy of 2-5 ms), but suffer from a long-tail with a few servers/probes incurring high error rates. Second, we found the main sources of this inaccuracy come from propagation delay asymmetry and queueing delay asymmetry in the wide area. Third, although the path delay asymmetry is considered not measurable [6]¹, we find that synchronization accuracy is well-correlated with some path properties that can be probed by end-hosts. To address this, we evaluate several heuristics that compensate for this error (by estimating the error and correcting for it).

We hope our results may enable designers of measurement experiments and network applications to better understand effects synchronization protocols may have on their results, and may assist system operators and protocol designers for tuning their configurations and extending synchronization protocols to improve performance.

2 Background

NTP uses the *Intersection Algorithm* to estimate accurate time from several noisy time sources. The algorithm works by having the local host periodically probe remote clock sources. The local host acquires (a) the clock value of the remote host when the probe was received and sent (b) an estimate of the RTT to the remote clock source. In particular, client i periodically sends a probe to the server j , and the server replies with a timestamp collected from its local clock. Let the sending times of client i and server j be $t_{i,TX}$ and $t_{j,TX}$, and let their receiving times be $t_{i,RX}$ and $t_{j,RX}$. The round-trip time $RTT_{i,j}$ can be derived

¹ A recent study [7] also indicated that router-level asymmetry does not necessarily imply delay asymmetry.

by

$$RTT_{i,j} = (t_{i,RX} - t_{i,TX}) - (t_{j,TX} - t_{j,RX}) \quad (1)$$

NTP assumes symmetric delay, allowing it to measure the one-way delay OWD

$$OWD_{i,j} = OWD_{j,i} = RTT_{i,j}/2 \quad (2)$$

The clock offset of the server j relative to client i is then derived as

$$\theta_{i,j} = (t_{j,TX} + OWD_{j,i}) - (t_{i,RX}) = \frac{1}{2}[(t_{j,RX} - t_{i,TX}) + (t_{j,TX} - t_{i,RX})] \quad (3)$$

In reality, the one way delay might not be symmetric, i.e.,

$$OWD_{i,j}^* - OWD_{j,i}^* = \Delta_{i,j} \quad (4)$$

where the difference $\Delta_{i,j}$ is bounded by $\pm RTT_{i,j}$. By (4) and (3), the error of the clock offset θ is equal to $|\Delta|/2$, i.e., *delay asymmetry Δ governs the synchronization accuracy*. Although protocols such as One-Way Active Measurement Protocol (OWAMP) can be used to estimate delay asymmetry on network paths [7], OWAMP requires NTP to synchronize the system clock, which can lead to biased measurements. Since the synchronization inaccuracy is unknown, it can be hard to accurately infer measured properties.

3 Methodology and Data Sets

Ideally, we would like to compare clock values across hosts against an Internet-wide global time. However, this is hard due to lack of “ground truth” on every device. Previous work on studying NTP performance [3, 4] focused on computing the difference between the client’s clock time and its *estimation* of the NTP server’s clock. This metric is useful to indicate how well NTP hosts converge over time, i.e., the value of θ . Unfortunately, this does not give us the actual synchronization error ($|\Delta|/2$), which arguably impossible to derive without the “ground truth” at the client end.

To provide ground truth, we instrumented our local machine with custom hardware to synchronize with GPS time signals. We used a Garmin 18x LVC GPS receiver, which provides a *pulse-per-second* (electrical signal that precisely indicates the start of a second) aligned to within 1 microsecond of UTC time. We then constructed a simple custom circuit to serve as an interface between the GPS receiver and the local machine. The power supply of GPS receiver comes from the PC through a type-B USB connector, while the *GPS signals*, including NMEA sentences and pulse-per-second signal, are transmitted to the local machine over the RS-232 serial port. GPS receiver is positioned at proper place such that the received SNR is above a certain threshold.

To measure accuracy of clock synchronization, we ran a Linux machine with NTP v4.2.6, and varied the remote server peer NTP would synchronize with. We modified the NTP source code to print out detailed probe information (e.g., NTP Timestamps). To make the pulse-per-second signal accessible to our measurement tools, we then patched the Linux kernel 2.6.32-rc10 with Linux’s Pulse

Table 1. GPS-instrumented testbed set. For each set we show: a) the deployment locations, b) the type of location, c) the type of connection, and d) the upstream network.

Location name	Location type	Connection type	Upstream network
Cornell Univ., NY	school	1 Gbps LAN	Cornell Univ.
UIUC, IL	school	1 Gbps LAN	Univ. Illinois
Chicago, IL	home	cable modem	Comcast
Green Bay, WI	business	1 Gbps LAN	Road Runner Holding Co.
Taipei, Taiwan	school	100 Mbps LAN	Taiwan Academic Network

Table 2. Synchronization error data set. For each set we show: a) the name, b) the number of clients and servers (and the measured server set), c) the duration of the experiment, d) the starting date of the measurement, and e) the average interval between consecutive measurement of the same pair.

Data set	#Clients x #Servers	Duration	Starting Date	Interval
SE-24st1	3 x 63 (only stratum-1 servers)	24h	April 18th, 2010	90s
SE-115allsvrs	4 x 1861 (all servers)	115h	May 5th, 2010	90m
SE-403allsvrs	5 x 1861 (all servers)	403h	August 30th, 2010	90m

Per Second support. In addition, some of our experiments required setting up `ntpd`s using the pulse-per-second signal. We used `GPSd` to translate between the GPS signal and the data to NTP protocol. To provide additional vantage points to cross-check the results collected in our local testbed, we deployed GPS-instrumented machines in other locations (Table 2). Finally, some of our experiments synchronize our testbed to public NTP servers. To acquire IP addresses of these servers, we start with public NTP time server lists [8], and exploit a spider program [3] which uses `xntpd` to crawl the NTP hierarchy, resulting in 67,782 servers.

We collect three synchronization error data sets of various sizes (Table 2). For the SE-24st1 data set, clients (Table 2) measure the synchronization error to 63 stratum-1 NTP servers with an average interval of 90 seconds between consecutive measurements of the same client/server pair. More accurate snapshot of synchronization error can be collected using smaller timescale. However, it comes with a cost that we can only measure a limited number of servers accurately (63 servers in the first data set). We collect two much larger data set (1861 servers) when the time interval is increased to 90 minutes. For the SE-115allsvrs and SE-403allsvrs data sets, we randomly choose 361 stratum-1 NTP servers and 500 stratum- i NTP servers, $2 \leq i \leq 4$. We ensure that all client-server pairs are measured at least once within a certain interval.

4 Synchronization Accuracy in the Internet

Accuracy across servers: Fig. 1(a) shows the CDF of synchronization error (i.e., $|\Delta/2|$) from our testbeds to 406 NTP stratum-1 servers. As expected, we observed a long-tailed distribution because some stratum-1 servers may have an inaccurate source clock or misconfigured daemon [3, 4]. How to choose a right server to synchronize with is important as only 10% – 20% of servers can provide sub-millisecond accuracy. Fig. 1(b) shows the CDF of “signed” synchronization error. We observe that the distribution is skewed, i.e., the median error is not

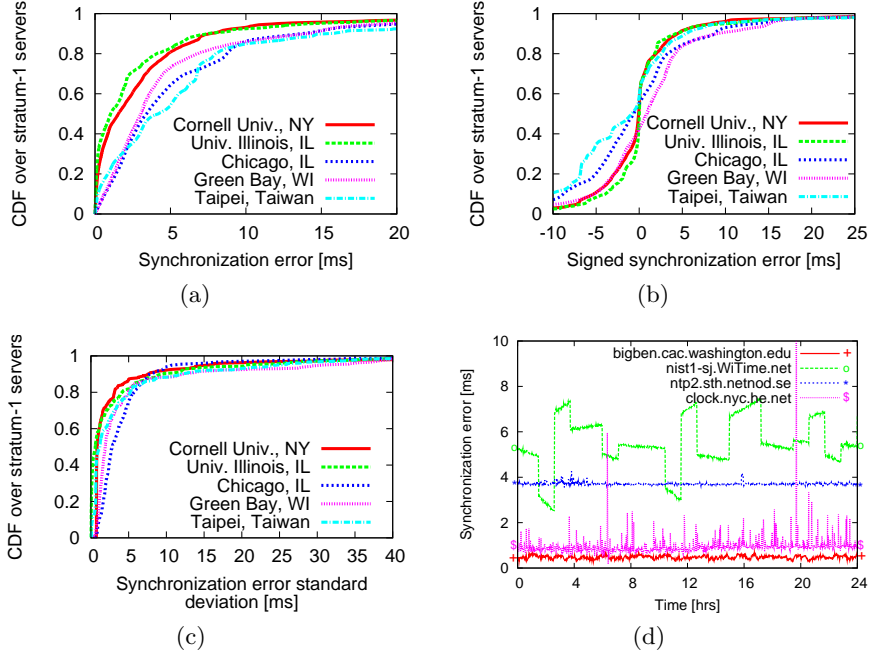


Fig. 1. The CDF of synchronization error for (a) Average error, (b) Average of “signed” error, (c) Standard deviation of “signed” error. (d) The time series of errors from Univ. Illinois to four representative public stratum-1 servers

close to zero and the error distribution is not symmetric about zero error. Hence even if a host synchronizes across many servers and uses averaging/intersection techniques, it will still suffer from bias. The distribution in other strata is similar, but the average error is a few milliseconds higher.

Accuracy across time: Fig. 1(c) shows the CDF of standard deviation error. In general, synchronization error is stable over time (i.e., median standard deviation ranges from 0.4 to 2.2 ms), while we observed that some servers have undesirable variation (i.e., the maximal standard deviation is about 120 ms). Fig. 1(d) shows synchronization error from Univ. Illinois to four representative public stratum-1 servers. For server in Univ. Washington (`bigben.cac.washington.edu`), we observe an average error 0.5 ms, while the error oscillates with a magnitude of ± 0.2 ms. Since a GPS clock is directly attached to this server, the accuracy of the server should be within sub-microsecond, i.e., most of inaccuracies come from network factors. We also synchronize with a public server (`nist1-sj.WiTime.net`) maintained by National Institute of Standards and Technology (NIST) Internet time services. NIST time servers use dial-up Automated Computer Time Service [9] to synchronize server clock to global time. Again, we observe that the error oscillates in a range of ± 0.2 ms. However, the overall error of the NIST server is up to 7 ms, which is significantly more inaccurate than that of GPS-equipped time servers. Also, we notice that the error abruptly changed around every 4000 seconds, as these are intervals between ACTS calibrations (we confirmed this with the NIST Time and Frequency Division). For another GPS-instrumented time server in Stockholm, Sweden (`ntp2.sth.netnod.se`), the

average error over time is about 3.7 ms, which is much higher than that in Univ. Washington. Fig. 1(d) shows the error to a time server (`clock.nyc.he.net`) in New York City using CDMA signals from cellular networks as time synchronization source. Generally, we found the accuracy of CDMA-based time servers is comparable to GPS-based servers. For CDMA-based and GPS-based time servers, the error is mostly stable over time. However, some sharp spikes happened occasionally are observed. This might come from transient instability of Internet routing (from our testbeds to stratum-1 servers). Other network factors such as Internet path asymmetry and bandwidth asymmetry are possible root causes to the synchronization errors.

5 Understanding Underlying Factors

Although we observe that the typical synchronization error is around 2 – 5 ms, it can undergo very large variations (Fig. 1(a)). Although we observe the characteristics of synchronization error in Fig. 1, but the root of inaccuracies is unknown. In this section, we analyze the impact of underlying networking factors that affect the synchronization inaccuracies. As shown in §2 the synchronization error $|\Delta|/2$ is determined by delay asymmetry Δ , which comprises four independent factors. To understand the impacts of these factors, we propose schemes to decouple and measure these factors separately as follows.

5.1 Software Stack Delay Asymmetry

To measure the software stack delay asymmetry, we conducted an isolated network with only two machines communicating directly through a 1-Gbps Ethernet switch. In this setting, both the transmission delay and propagation delay are perfectly symmetric. The switch is dedicated to this experiment to ensure probes have low queuing delay asymmetry. Under this scenario, we believe the software stack delay asymmetry is the predominant factor of synchronization error. To measure the synchronization error, GPS receivers are attached to the machines as the ground truth as described in §3.

We observed that the software stack delay asymmetry is approximately a uniform distribution with range $(0, 165) \mu s$ when the CPU utilization is low (Fig. 2(a)). We use a synthetic load generator to inject predictable loads on a server. With heavy system load, we observed a long-tail distribution (Fig. 2(a)) where the maximal error (over 30,000 trials) is 289.2 μs . We also leverage SystemTap to trace the delay in the Linux kernel, and capture the kernel delay in each directions. The asymmetry of kernel delay is much smaller, with a median of 3.58 ns. This indirectly shows that the software stack delay asymmetry is dominated by the factors that SystemTap missed. For example, SystemTap cannot measure the latency between packets arriving at the system and when the OS interrupt handler is called, which may be the main source of software stack delay asymmetry.

5.2 Queuing Delay Asymmetry

As Internet paths are mostly stable over a few hours [10], their propagation delay and transmission delay are likely to be similar on short time scales. Therefore, if

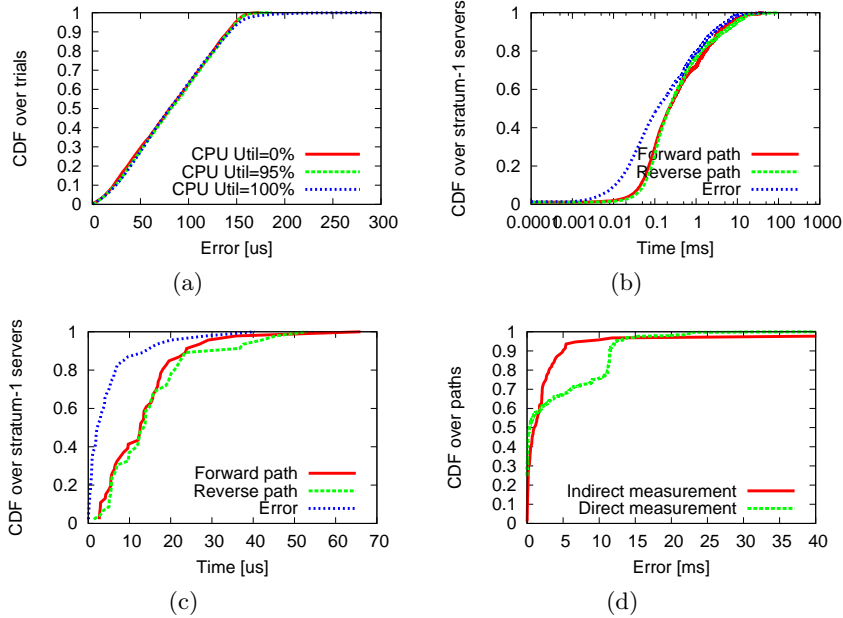


Fig. 2. (a) Synchronization error induced by software stack delay (§5.1) by varying CPU utilization. (b) Queueing delay (§5.2) in both directions, and the induced error for synchronization. Note that the x-axis is log scale. (c) Transmission delay (§5.3) in both directions, and the induced error for synchronization. (d) Comparison between the indirect (§5.4) and the direct (§6) measurement of the error induced by propagation delay asymmetry.

server’s clock is accurate, the *variation* of synchronization error (e.g., oscillation in Fig. 1(d)) should come from asymmetries of software stack delay and queueing delay. As software stack usually contributes less than $150 \mu\text{s}$ error, the variation (e.g., spikes in Fig. 1(d)) should be dominated by queueing delay asymmetry.

To quantify the queueing delay asymmetry, we measure the synchronization error from our testbed nodes to GPS-equipped stratum-1 time servers over intervals of 15 minutes. For each measured interval, we assume that the synchronization error derived from probes with *minimal* one-way delay is composed of asymmetries of the propagation and transmission delay. After subtracting the minimal one-way delay in both directions, the remaining synchronization error will be vastly dominated by packet queueing. We observed that the median error is around $150 \mu\text{s}$ (Fig. 2(b)). However, the last 25% of the path has $\geq 1 \text{ ms}$ error induced by queueing delay asymmetry. To evaluate this in the larger scale, we conduct the same experiment over paths between 300 lightly loaded PlanetLab nodes, and we obtained a very similar error distribution.

5.3 Transmission Delay Asymmetry

To decouple the transmission delay in the two directions, we send packets with *asymmetric size*. For example, to measure the transmission delay in the forward path, we only vary the packet size in the forward direction while the size in

the reverse path is fixed. As the transmission delay is exactly proportional to the packet size, we sent probes with different packet sizes to measure the transmission delay. Specifically, to measure the transmission delay of a 76-byte NTP packet (including UDP and IP header size) *in the forward direction*, we send two probes with different sizes, p_1 and $p_2 < p_1$ in bytes in the forward direction, and received the destination reply as equal-size probes in the reverse path. Then the transmission delay of NTP packet is measured by $76 \times (t_1 - t_2)/(p_1 - p_2)$, where t_i represents the RTT of i -th probe.

This measurement scheme assumes that the delay contributed by other factors would remain the same for these two packets, which might hold for propagation delay because Internet paths are stable within a short period. However, software stack delay and queueing delay could vary over consecutive packets. To curtail the variance of queueing delay, we repeat the probes and select the one with minimal delay. The intuition is that the packet with minimal delay would suffer from smaller software stack and queueing asymmetry [5]. We measure the transmission delay asymmetry of NTP packets from our testbed nodes to stratum-1 time servers. We observe the median error is $2.2 \mu\text{s}$ (Fig. 2(c)). We repeated these experiments on PlanetLab and we got similar results.

5.4 Propagation Delay Asymmetry

Propagation delay asymmetry is hard to acquire accurately without knowing the length of the cables that carried probes. Hence, we determine this information indirectly by simply subtracting the errors induced by other factors from the overall synchronization error. In particular, we use the probes with minimal delay to exclude the synchronization error induced by queueing delay and software stack delay asymmetry. Similarly, as we are able to measure the error induced by transmission delay asymmetry (§5.3), the remaining error after subtraction should come from propagation delay asymmetry. Fig. 2(d) shows that the median error induced by propagation delay asymmetry is 2 ms, while the last 5% paths have error ≥ 10 ms.

Table 3. Error sources and quantity in typical case and worst 5% case. Note the percentage in the worst 5% case column is derived by assuming typical error for the other sources.

Asymmetry source	Typical error (percentage)	worst 5% case (percentage)
Software stack delay	$85 \mu\text{s}$ (1.6%)	$150 \mu\text{s}$ (3.5%)
Queueing delay	$150 \mu\text{s}$ (2.9%)	2 ms (49%)
Transmission delay	$2 \mu\text{s}$ (0.03%)	$15 \mu\text{s}$ (0.7%)
Propagation delay	2 ms (95%)	7 ms (97%)

6 Discussion

For systems and measurements that rely on high degrees of clock accuracy, it may be desirable to reduce inter-host synchronization error over wide-area networks. From our results, several techniques may show promise in reducing these errors.

Pinning network paths: Based on our findings (Table 3), propagation delay asymmetry dominates synchronization error. Because of this, symmetric physical paths may be preferable for clock synchronization. Unfortunately, Internet paths are inherently asymmetric [10, 11]. As a result, techniques that can “pin” routes to symmetric paths may provide the largest gains for network synchronization. For example, configuring routing protocols to prefer symmetric paths, or using tunneling protocols [12, 13] to assign symmetric paths between devices that need good synchronization (e.g., between NTP servers) may provide benefits. While these techniques require network changes, our results also indicate that gains can be realized by preferring *existing* symmetric paths. To illustrate this, we build a support vector classification model using LIBSVM [14]. By probing a set of NTP servers, end-hosts simply collect multiple network factors such as the maximal and the minimal round-trip time over multiple probes, forwarding and reverse hop counts². Given the sampled instances, the classifier is able to choose the top 5% servers with smallest synchronization error with a cross-validation accuracy of 91.8%. Our model uses the radial basis function (RBF) kernel where the samples are non-linearly mapping to a higher dimensional space. The training data (probing 47 NTP stratum-1 servers for each client) and the detailed results are available at [15].

Compensating at endpoints: Our results indicate that some aspects of delay asymmetry can be estimated. Because of this, it may be possible for end hosts to subtract this estimate to compensate. While we presented an indirect scheme to measure propagation delay asymmetry (§5.4), it requires a GPS receiver in the end host. Instead of using GPS receiver, we could use a geolocation-based service to approximate propagation delay. One could use traceroute (and reverse traceroute [16]) to find IP addresses of intermediate routers in both directions. Then the propagation delay can be estimated based on the geographical distance between intermediate routers, which is given by the IP geolocation services. We implement this by using a commercial IP geolocation service [17]. To reduce inaccuracy, we also inspect DNS names of ISP routers as a hint to infer their geographic location, as done in [18]. We observe that the *round-trip* propagation delay (measured by the geolocation-based service) and actual RTT are correlated with a coefficient of 0.56 (with a p-value $< 10^{-7}$) over PlanetLab nodes. We also compared the error with that measured by indirect measurement (Fig. 2(d)). They are not perfectly matched because the geolocation service may be inaccurate in some cases. The accuracy is likely to be improved by implementing a better IP geolocation optimization model [18], we leave this for future work. To discard inaccurate results in the indirect scheme, we performed a sanity check to select only paths satisfying $0.9 \leq RTT/(P_f + P_r) \leq 2.5$, where P_f and P_r are the forward and reverse propagation delay as estimated by geolocation-based service. With the limit lookups of reverse traceroute for security and load reasons, we limit our evaluation to only 8 public time servers, and the average synchronization error is reduced from 2.87 ms to 1.07 ms. The program and the detailed results are available at [15].

² The reverse hop counts can be derived by guessing the initial TTL of the time server.

7 Conclusions

In this work we study the ability to synchronize clocks between network devices over the modern Internet. We find that while traditional synchronization protocols have only moderate errors (5 - 10 ms) in the common case, they can suffer from large “bursty” error in some cases. Our work may motivate future work on improving existing synchronization algorithms such as NTP to perform more efficiently in the wide area. Finally, we are continuing to grow our testbed infrastructure, which we hope will be useful to other experimenters who wish to conduct Internet measurements that require fine-grained synchronization of time across endpoints (users who wish accounts may find contact information at [15]).

References

1. V. Paxson, “Strategies for sound internet measurement,” in *IMC*, 2004.
2. R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese, “Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator,” in *SIGCOMM*, 2009.
3. N. Minar, “A survey of the NTP network,” December 1999.
4. D. L. Mills, “On the accuracy and stability of clocks synchronized by the network time protocol in the Internet system,” *SIGCOMM Comp. Comm. Rev.*, vol. 20, no. 1, pp. 65–75, 1990.
5. J. Ridoux and D. Veitch, “Principles of robust timing over the internet,” *Commun. ACM*, vol. 53, no. 5, pp. 54–61, 2010.
6. D. Veitch, J. Ridoux, and S. B. Korada, “Robust synchronization of absolute and difference clocks over networks,” *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 417–430, 2009.
7. A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao, “A measurement study of Internet delay asymmetry,” in *Proc. Passive and Active Measurement*, April 2008.
8. “The NTP Public Services Project.” <http://support.ntp.org/>.
9. “NIST Automated Computer Time Service (ACTS).” <http://www.nist.gov/physlab/div847/grp40/acts.cfm>.
10. V. Paxson, “End-to-end routing behavior in the internet,” *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, pp. 41–56, 2006.
11. Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker, “On routing asymmetry in the internet,” in *GLOBECOM*, 2005.
12. E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture.” RFC 3031, January 2001.
13. S. Hanks, T. Li, D. Farinacci, and P. Traina, “Generic Routing Encapsulation over IPv4 networks.” RFC 1702, October 1994.
14. C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001.
15. <http://netfpga.cs.uiuc.edu/proj/doku.php?id=sync>.
16. E. Katz-Bassett, H. Madhyastha, V. Adhikari, C. Scott, J. Sherry, P. van Wesep, A. Krishnamurthy, and T. Anderson, “Reverse traceroute,” in *NSDI*, April 2010.
17. “IPInfoDB.” <http://www.ipinfodb.com/>.
18. E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe, “Towards IP geolocation using delay and topology measurements,” in *IMC*, 2006.