2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems

PTPmesh: Data Center Network Latency Measurements Using PTP

Diana Andreea Popescu
Computer Laboratory, University of Cambridge, UK
diana.popescu@cl.cam.ac.uk

Andrew W. Moore
Computer Laboratory, University of Cambridge, UK
andrew.moore@cl.cam.ac.uk

***Abstract*—Many data center applications are latency-sensitive. Monitoring continuously the network latency and reacting to congestion on a network path is important to ensure that the applications performance does not suffer penalties. We show how to use the Precision Time Protocol (PTP) to infer network latency and packet loss in data centers, and we conduct network latency and packet loss measurements in data centers from different cloud providers, using PTPd, an open-source software implementation of PTP.**

I. Introduction

Data centers run a mix of applications, some of which are latency-sensitive, like web search, social networking or key-value stores. For latency-sensitive applications, increased network latency can lead to significant drops in application performance [1], [2]. In order to ensure the best application performance, we need to be able to measure continuously the network latency across paths in data centers. Having up-to-date latency values helps in tracking network service level agreements (SLAs) for applications and in quickly finding failures [3], [4]. These challenges led us to the idea of designing a lightweight network latency monitoring system for data centers. Additionally, the system would be able to determine packet losses, as these have a huge negative impact on application performance [3], [4].

In this paper, we investigate the use of the Precision Time Protocol (PTP), with its open source software implementation PTPd, to address these challenges. We leverage the statistics offered by PTPd to measure network latency. We define a metric to quantify packet loss ratio based on the number of messages sent between the PTP slave and the PTP master. We verify our metrics through performing experiments on small-scale testbeds and in data centers across the world from different cloud providers. Our approach is easy to deploy on either the hypervisor or on VMs, making it a feasible tool for cloud tenants for obtaining network performance statistics without significant overhead.

II. The Precision Time Protocol

The IEEE 1588 Precision Time Protocol (PTP) [5] is a protocol used to synchronize clocks over a network and it can achieve sub-microsecond precision. The master clock provides the reference time for the slave clocks by communicating over the network. A *grandmaster* is chosen from the available clocks in the network, using the best master clock algorithm (BMC). The grandmaster will be the root of a tree formed out of devices that are PTP-enabled. Each element of the tree is both a slave to its parent and a master for its children. There are several types of PTP clocks. The simplest type is the *ordinary clock*, which is an end device that has only one network connection, and can act as a master or a slave. A *boundary clock* has a slave port, receiving the time from the master clock, and master ports, disseminating the time to other slaves. Another type of clock is the *transparent clock*, which timestamps incoming and outgoing messages and updates the correction field in the messages to account for the delay across the device. The mechanisms used by the last two types of clocks ensure the scalability of PTP networks.

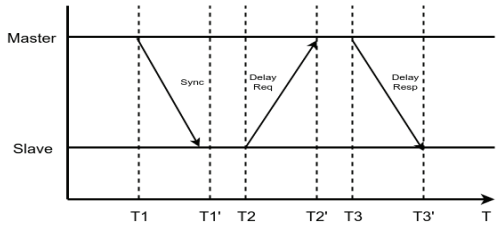


Fig. 1: PTP protocol

The PTP protocol message sequence is depicted in Figure 1. A PTP master sends a *Sync* message. The time when the Sync message was sent (T_1) is recorded at the master and sent to the slaves. A slave records the time when it received a Sync message (T_1'). The difference between the send and receipt times represents the *master-to-slave delay* ($T_1 - T_1'$). A PTP slave sends a *Delay Request* message. The slave records the time when the Delay Request message was sent (T_2), while the master records the receipt time (T_2'). The difference between the send and receipt times of the Delay Request messages represents the *slave-to-master delay* ($T_2 - T_2'$). By assuming the propagation delays master-to-slave and slave-to-master are symmetric, the *one-way delay* is computed as half of the sum of the two delays. The time difference between the master and slave clocks represents the *clock offset* from master and is computed as a difference between the master-to-slave delay and the one-way delay. If the master does not have the ability to embed T_1 in the Sync message, it sends an additional message after the Sync message, *Follow_Up*, that contains T_1. In the case that the master-to-slave and slave-to-master delays

2375-0227/17 $31.00 © 2017 IEEE
DOI 10.1109/MASCOTS.2017.30

are asymmetric (due to network congestion for example), the clock offset will suffer perturbations and the precision of the clock synchronization will be affected.

PTP uses various mechanisms to ensure that there is no interference in the clock synchronization. Firstly, PTP can use hardware timestamping to eliminate the end-host delay caused by the network stack [6]. PTP-enabled NICs have become common in recent years. Secondly, PTP-enabled switches that run as transparent clocks can eliminate the switching delays that can affect the clock synchronization. In our work, we do not use transparent clocks, as we want to leverage PTP's measurements to infer the actual network latency, which is affected by network conditions like congestion.

III. PTP Analysis

We use in our experiments the Precision Time Protocol daemon (PTPd) [7], which is an implementation of PTP. The PTPd logs statistics such as the clock offset, the master-to-slave delay, the slave-to-master delay, the one-way delay. The interval for sending the Sync and Delay request messages can be configured in PTPd, up to a rate of 128 messages per second, expressed as log 2 values.

A. PTP-enabled NICs

We first run experiments to see if PTPd is affected by network traffic that originates from the same host, since end-host packet processing might affect the measurements performed by PTPd under increased load. We use a testbed formed out of two hosts directly connected, running Ubuntu server 14.04 LTS, kernel version 4.4.0-62-generic. The host hardware is a single 3.5 GHz Intel Xeon E5-2637 v4 on a SuperMicro X10-DRG-Q motherboard, equipped with a Solarflare SFN8552 Network Interface Card (NIC) supporting PTP [8] with hardware timestamping. We compare the clock offset and one way delay obtained from *sfptpd*, Solarflare's PTP daemon which uses hardware timestamping (see Figure 2, note: ns y-scale), and from *PTPd* (see Figure 3, note: μs y-scale), which uses software timestamping, with and without running an *iperf* TCP stream between the hosts. One host is the PTP master, while the other acts as a PTP slave. We can see from the two figures that the clock offset reported by *sfptpd* is not affected by iperf. However, in the case of *PTPd*, the clock offset deviates when the iperf stream starts and ends. NICs supporting PTP are becoming increasingly available, which means that measurements performed by a PTP implementation that leverages this support will not suffer from end-host interference, and will also eliminate the end-host delay from measurements, reporting only the actual network latency.

B. Measuring Network Latency

Our testbed in Figure 4 consists of six servers Intel Xeon E5-2430L v2 running at 2.40GHz, with Ubuntu 16.04, kernel version 4.4.0.64-generic. The servers are connected using two Arista 7050Q switches and all network links are 10Gbps. In these experiments we use PTPd, with no NIC hardware timestamping support. The two hosts running PTPd do not send or receive any other network traffic. Our results include the end-host delay, but are thus not affected by any concurrent traffic originating from the host. Since our goal is that cloud tenants use PTP as a tool to measure network conditions, we do not use PTP-enabled NICs in the next experiments, since these may not be available in the cloud data centers.

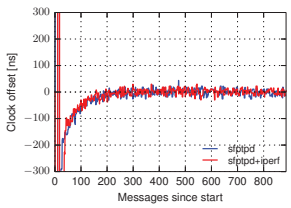


Fig. 2: The clock offset reported by *sfptpd* is not affected by iperf, since it uses NIC hardware timestamping.

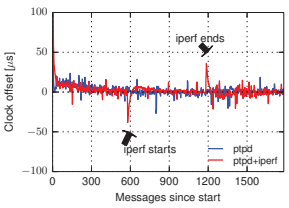


Fig. 3: The clock offset reported by PTPd is affected by iperf, due to end-host interference.

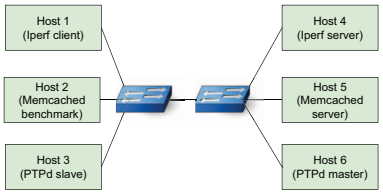


Fig. 4: Small-scale testbed to understand PTP's behaviour caused by congestion or packet loss.

Basic Latency. We measure the network latency in our setup between the PTP master and PTP slave hosts, using *ping* and the latency measurement UDP-based tool that uses the Time Stamp Counter (TSC) to measure the latency from [1], and we compare it with the one-way delay reported by PTPd. For PTPd, we set the interval for Sync and Delay Request messages to 1 per second, and we run the clock synchronization for 15 minutes. We run 1 million network latency measurements with the UDP-based tool and 30000 ping probes. Each test is conducted separately, and there is no other network traffic in our setup. The latency CDF is presented in Figure 5. Intuitively, we would have expected the one-way delay to be half of the values reported by the UDP-based tool, however this is not the case. We assume this is due to the fact that the one-way delay includes the computation and processing done by PTPd at the end-host. On the other hand, the one-way delay is approximately half of the median *ping* values. Another aspect worth noting is that once the two clocks synchronize (an initial starting period of 15 minutes is excluded from the plotted data for PTPd), the one-way delay reported is stable; there is no long tail for the reported one-way delay values, meaning the OS scheduling does not affect significantly PTPd [1].

Iperf. We study the effect of network congestion on the statistics reported by the PTPd slave using the testbed in Figure 4. In each test, we allow a clock synchronization phase of 5 minutes for PTPd, before starting concurrently the two other applications, *Memcached* [9] (with its corresponding

benchmark *memaslap* [10]) and *iperf* in TCP mode. We set the interval for the Sync and Delay Request messages to 0.25 seconds. We run two experiments: i) a 5 s iperf stream running (Figure 6) and ii) three 5 s iperf streams with a 5 s break between them (Figure 7). In the first experiment, we notice that the congestion episode determined by iperf leads to an increase in the slave-to-master delay (on the iperf stream direction), and consequently in the one-way delay, producing deviation to the clock offset. In the second experiment, the first congestion episode caused by iperf has the same effect as in the first experiment. The next two intervals of iperf traffic also produce deviations in the slave-to-master delay, however, before iperf runs again, the clock offset had not reached the normal operation values from before iperf started running. Thus, if there are several congestion episodes before the slave clock manages to resynchronize to the master clock, the one-way delay reported by the slave will not be indicative of the actual delay, although it will indicate that there is an event (congestion, failure) on that network path.

Changing the interval for the Sync and Delay request messages. We perform the same experiment with *iperf* and *memaslap* concurrently running with PTPd, but *iperf* runs for 1 s. We vary the interval for the Sync and Delay request messages, from 1 s down to 7.8125 ms. In Figure 8, we notice that *iperf* does not produce any change in the PTPd statistics, since the interval for messages is as long as iperf's run. However, when we increase the interval (Figures 9 and 10), the concurrent traffic leads to deviations in the PTPd statistics, and we see an increase in the one-way delay during iperf's run. On the other hand, the clock offset, master-to-slave and slave-to-master delays oscillate between larger values when the Sync and Delay Request interval is larger (compare the width of the lines in Figures). However, the one-way delay, the metric we are primarily interested in, does not exhibit such significant oscillations.

C. Detecting Packet Losses

PTPd records the number of messages sent and received (Announce, Sync, Followup, Delay Request, Delay Response), and it is possible to export them periodically. The counters can be reset after they are exported. On the slave side, a difference between the number of Delay Request and Delay Response messages would indicate packet loss, and the packet loss ratio can be approximated as $1 - \frac{\#Delay\_Response\_messages}{\#Delay\_Request\_messages}$. Normal operation should see the same number of Delay Request and Response messages or a difference of at most one. One disadvantage of computing the loss ratio in this way is that it does not account for the Announce, Sync and Followup messages that were potentially lost, as well as other types of packets that may be lost (ARP packets for example).

We verify whether the ratio can be used as a good indication for the packet loss ratio by artificially introducing packet loss in the network. We use *netem* [11], an enhancement of the Linux traffic control facilities, to emulate packet loss on the outgoing network interface of the host which runs the PTPd master. In this scenario, none of the Delay Request messages will be lost, although in practice this may happen. Since outgoing PTPd packets are looped back via the IP_MULTICAST_GROUP [12], loss conditions are applied on both the physical interface and loopback. We use the *loss random* option of netem, which adds an independent loss probability to the packets outgoing on the chosen network interface. We use as values for packet loss $1\%, 5\%$ and 10%, and we compute the ratio as described above to see if it matches the loss values. We run the clock synchronization for 50 minutes with a packet loss of 1%, 10 minutes with a packet loss of 5%, and 5 minutes with a packet loss of 10%, and for each loss value we perform 5 runs. The results are presented in Table I. We can see that the metric we defined can serve as a coarse estimate of the packet loss ratio over a defined interval of time.

TABLE I: Packet loss ratio computed based on the number of *Delay Request* and *Delay Response* messages reported at the PTP slave

Netem Packet Loss	Max. sample size (Delay_Request messages)	Packet Loss Median	Packet Loss Std. deviation
1%	2961	1.08%	0.23%
5%	571	5.43%	0.74%
10%	285	9.47%	0.34%

IV. Cloud Deployment and Measurements

A. Deployment Scenarios

We look at two deployment scenarios for a system based on PTP in data centers [13]. In the first scenario, the cloud provider deploys PTPd (or a different software implementation for PTP) in the hypervisor, possibly alongside a separate clock synchronization mechanism. In the second scenario, the tenants themselves run PTPd inside their VMs and use the reported measurements to check the network conditions. In both scenarios, the PTP traffic should not prioritised and switches in the network should be PTP-aware, otherwise the measurements would not be indicative of the actual network latency. Since ECMP (Equal cost multi-path) is used in data centers to load balance the traffic across the available network paths between two servers, the PTP traffic between the servers may not follow the same network path as other network traffic that exists between the two servers. To mitigate this issue, we can change the ports on which PTPd is running, looping over a range of ports to cover all network paths between the two servers [4]. If the cloud provider knows how ECMP is implemented in their networks and they do not use randomness in the ECMP hash function [14], then they can define a list of ports for PTPd to ensure that all network paths between the two hosts are covered. Alternatively, if packet marking is available [15], then the path taken by the PTP packets will be known, and it can be verified that all the network paths are covered when using different ports. Another aspect that needs to be taken into account is the number of slaves a master can synchronize with before becoming overloaded due to the processing rate of the messages at the master.

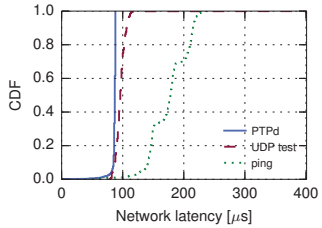


Fig. 5: RTT reported by *ping* and by the UDP-based tool [1] and the one-way delay reported by PTPd

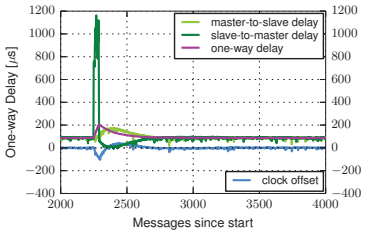


Fig. 6: A 5 s iperf stream starts running at second 300.

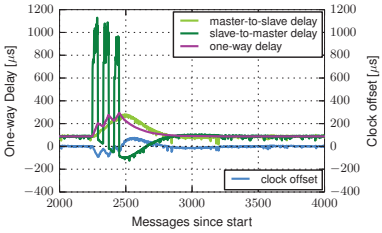


Fig. 7: 5 s iperf streams start running at second 300, 310 and 320, with 5 seconds break between streams.

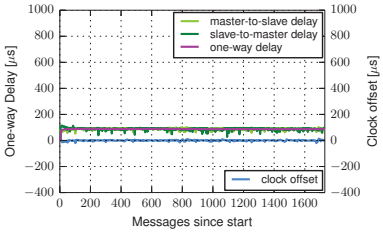


Fig. 8: 1 s interval for Sync and Delay request messages. A 1s iperf stream starts running at second 300, PTPd does not detect it.

Fig. 9: 125 ms interval for Sync and Delay request messages. A 1s iperf stream starts running at second 300, PTPd detects it.

Fig. 10: 7.8125 ms interval for Sync and Delay request messages. A 1s iperf stream starts running at second 300, PTPd detects it.

B. Network Latency Measurements

We measure the one-way delay between multiple virtual machines (VMs) from different cloud providers. For each of the two cloud providers, A (Google Compute Engine) and B (Amazon EC2), we chose one data center from West USA and one from West Europe, and we rented 4 VMs in each data center. We run the PTPd master on one VM, while the other three VMs act as slaves. The VMs' type is the default type recommended by each cloud provider, running Ubuntu 16.04. Since the VMs' performance may be affected by other collocated VMs, we ran the measurements over 6 hours between each VM pair. We use PTPDv2 2.3.1, using the latest source code from the public repository. We run PTPd in unicast mode, using unicast negociation, and using end-to-end delay measurement. We set the number of Sync and Delay Request messages to 1 per second. The results can be seen in Table II and Figure 11. We also patched PTPd in order to be able to specify the port on which the PTP event and general messages are sent and received. We run PTPd using different ports to see if the one-way delay reported by PTPd changes significantly between runs, possibly signaling that a different network path was used due to ECMP hashing on a different header. While the one-way delay CDFs show differences between ports in some cases, we cannot be sure that these differences are not caused by an increase in network traffic or a change in network configuration.

C. Path Symmetry

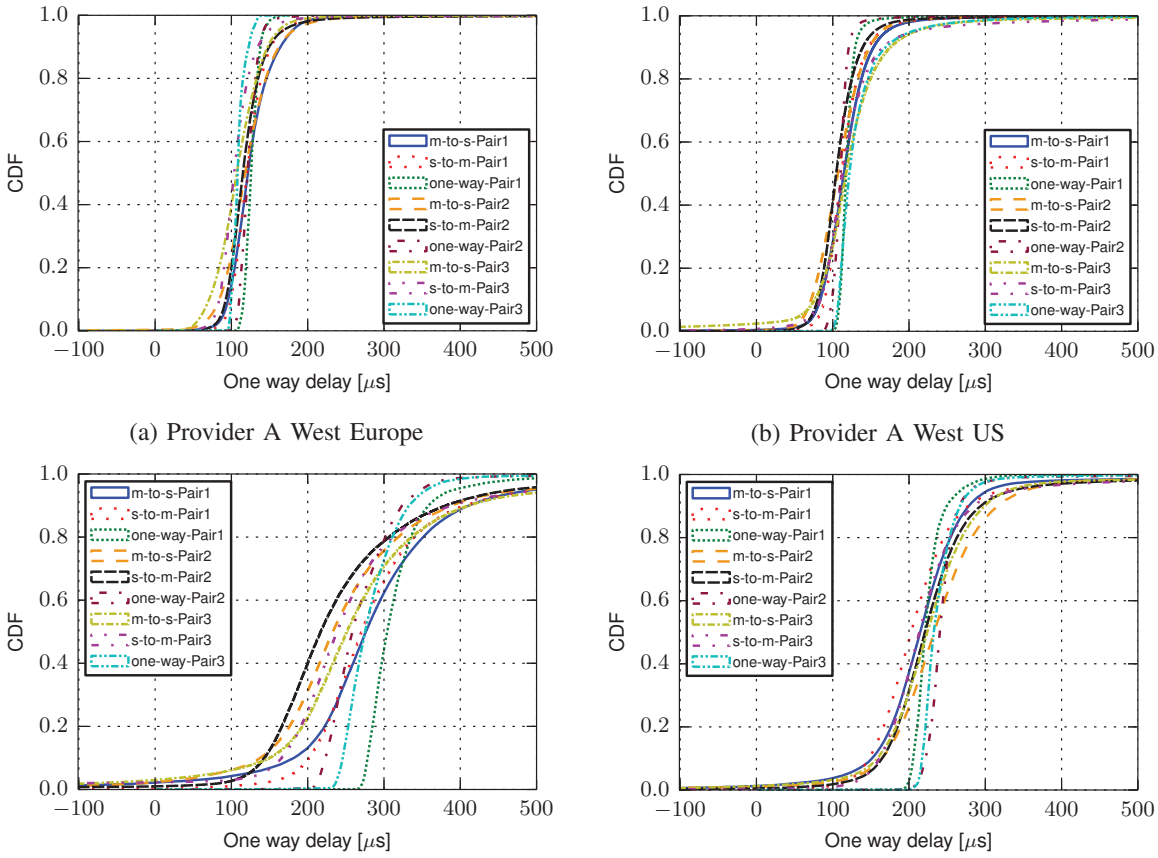
PTPd reports the *master-to-slave* and *slave-to-master* measured delays. We use these two statistics to determine if the packets are sent on the same network path from the master to the slave and from the slave to the master. We are interested in seeing whether the simplifying assumption that the one-way delay can be computed as half of the measured round-trip time holds true in cloud data centers. We plot the master-to-slave and slave-to-master CDF for 4 data centers from 2 cloud providers, A and B, in Figure 11, for each of the three VM pairs. Note here that the values can be negative, due to the differences between the clocks of the master and that of the slave. We can compare the quantiles of the two distributions to determine if one network path is different than the other. Table II lists the median values of the distributions. For cloud provider A, the quantiles of the two distributions are equal or close to each other, thus we can conclude that the two paths (master to slave and slave to master) are symmetric. For cloud provider B, we notice differences of at least 5μs between the median values. For both cloud providers, we observe that the master-to-slave median delay values are larger than the median slave-to-master delay values. We checked that this is not due to an issue with the VMs by reversing the role of the master and slave between the two VMs, and we got consistent results with what we have already observed.

D. Packet Loss

We investigate possible packet loss in the four data centers over a 24 hours run for each of the VM pair. Using the metric we defined for packet loss in Section III, we get packet loss ratios of 10^{-5} to 10^{-4} (last line of Table II), similar to the ranges reported in [3].

TABLE II: Median values for the master-to-slave, slave-to-master and one-way delays across the four data centers for the 12 VM pairs. The last line indicates the packet loss ratios over a period of 24 hours.

	A EU 1	A EU 2	A EU 3	A US 1	A US 2	A US 3	B EU 1	B EU 2	B EU 3	B US 1	B US 2	B US 3
m-to-s (μs)	121.6	118	107	114.9	107	117.8	276.3	233.1	252.1	214.6	233	224
s-to-m (μs)	120.5	115	103.6	111.7	104.7	112.9	271.1	217.5	238.4	202.8	221	213.7
one-way (μs)	125.8	121.1	108.1	117.5	110.3	121.8	304.2	258.6	275.2	221	240.3	232.5
pkt loss $\times 10^{-5}$	8.11	13.98	9.37	9.26	6.96	6.96	15	6.97	10.4	0	1.27	2.41



(a) Provider A West Europe

(b) Provider A West US

(c) Provider B West Europe

(d) Provider B West US

Fig. 11: Master-to-slave, slave-to-master and one-way delay within data centers for cloud providers A and B.

V. Limitations

Our measurement study in the cloud uses a total of 12 VMs across four data centers from two cloud providers. In the future, we would like to expand our study across a wider range of VMs and data centers. We do not focus on builing a complete system to process the data collected by PTPd, but our work can serve as a starting point for a system that includes additional components for data storage and analysis.

VI. Related Work

Network Latency and Packet Loss Monitoring Systems. Table III compares the properties of systems used to measure network latency and packet loss in data centers, including our approach. Our comparison looks at aspects related to type of measurements taken, implementation, deployment, and data storage and analysis of collected measurements. Traditional tools are *ping* and *traceroute*, however these lack the precision and flexibility of custom purpose built tools. Cisco IP SLA [16] monitors network performance by sending probe packets. It runs on Cisco switches and it can collect data about one-way latency, jitter, packet loss and other metrics. The measurements can be accessed through SNMP or command-line interface, being stored in the switches. NetNORAD [4] is a system used in Facebook's data centers to measure RTT and packet loss ratio by making servers ping each other, for different Quality-of-Service (QoS) classes of traffic. The system runs measurements at data center, region and global level. Everflow [18] is a system that monitors all control packets and special TCP packets for all flows (TCP SYN, FIN, RST), and supports guided probing by injecting crafted packets and monitoring their behaviour through the network, which can be used to measure link RTTs. Pingmesh [3] is an always-

TABLE III: Comparison between systems used to measure network latency and packet loss in data centers

	Measurement	Probe Type	Probe Frequency	Availability	Scalability	Coverage	Deployment	Data Storage and Analysis
Ping	RTT; packet loss ratio	ICMP	-	single measurement	intra-DC	targeted pair	Hypervisor or VM	locally; analyze independently
Traceroute	RTT	ICMP ECHO/ TCP SYN	-	single measurement	intra-DC	targeted pair	Hypervisor or VM	locally; analyze independently
Cisco IP SLA [16]	RTT (average); one-way delay; packet loss	ICMP/ UDP/ TCP/ HTTP/ DNS	between 1 and 604800 seconds	always-on	intra-DC	targeted path	CISCO switches	locally; analyze independently
Pingmesh [3]	RTT; packet loss ratio	TCP/HTTP	minimum 10s seconds	always-on	inter-DC	all pairs	Hypervisor	Cosmos and SCOPE [17]
NetNORAD [4]	RTT; packet loss ratio	UDP	configurable	always-on	inter-DC	all pairs	Hypervisor or VM	Scribe and Scuba [4]
Everflow [18]	link RTT	packet marked with debug bit	-	single measurement	intra-DC	targeted path	switches and controller	custom analyzer and SCOPE [17]
SLAM [19]	network path latency distribution	crafted probe	-	single measurement	intra-DC	targeted path	OpenFlow switches	controller
INT [20]	end-to-end latency	crafted probe	-	single measurement	intra-DC	targeted path	P4 switches	last switch on path; analyze independently
LossRadar [21]	packet losses at switches	no probes	10 ms	always-on	intra-DC	cover all paths	P4 switches	custom collector and analyzer
TIMELY [20]	RTT	TCP	per flow	always-on	intra-DC	all pairs	end hosts with special NICs	locally; analyze independently
PTPmesh (this work)	one-way delay (average); packet loss ratio	UDP	up to 128 probes per second	always-on	inter-DC	multiple pairs	Hypervisor or VM	locally; analyze independently

on tool that runs round-trip time (RTT) measurements between every two servers in data centers. The system measures inter-server latencies at three levels, Top-of-Rack switch, intra data center and inter data center. Pingmesh also reports the packet drop rate, which is inferred based on the TCP connection setup time. SLAM [19] is a latency monitoring framework for SDN-enabled data center that sends probe packets in order to trigger control messages from the first and last switches of a network path. SLAM uses the arrival times at the controller of the control messages to compute a latency distribution for that network path and is able to detect increases in latency of tens of milliseconds on a path. INT [20] leverages the P4 programming language [22] to measure the end-to-end latency between virtual switches. Each network element on the path appends their per-hop latency to a packet that flows between the two virtual switches located at the ends of the path. The end-to-end latency is computed by adding the per-hop latencies, and it assumes that switching and queueing delays dominate, while the propagation delays are negligible. However, this solution requires a network which uses only P4-compatible switches, making it difficult to deploy in legacy networks. LossRadar [21] is a system that can detect packet losses in data centers within 10s of milliseconds, reporting their locations and the 5-tuple flow identifiers. It keeps specific data structures at switches, which are periodically exported to a remote collector and analyzer. It does not offer any latency measurements. TIMELY [23] uses the NIC hardware timestamps and NIC-based ACK generation to compute the RTT between two servers. Another way to monitor network latency in a data center, though costly, is to have each host equipped with a GPS receiver (the host clocks will thus be synchronized) and run one-way delay measurements between any pair. In comparison, our work, *PTPmesh*, uses PTP statistics to report one-way delay and compute a packet loss ratio. The number of probes is configurable, it can provide continuous measurements, and does not have much overhead. Futhermore, it is easy to deploy on either hypervisor or VM, and tenants themselves can deploy it on their VMs.

Time Synchronization Protocols. NTP (Network Time Protocol) [24] is a network protocol for clock synchronization, that achieves microseconds precision. The recent DTP (Datacentre Time Protocol) [25] is a protocol that uses the physical layer to synchronize the hosts' clocks, achieving sub-ns clock precision. However it is not immediately deployable, since it requires hardware modifications to switches.

VII. Conclusion

In this paper, we studied how the Precision Time Protocol statistics (one-way delay, master-to-slave delay, slave-to-master and clock offset) can be used to measure network latency and detect packet loss. We used the PTPd open-source implementation for our study. We determined that PTP one-way delay measurements can be used to infer network latency. We defined a metric for computing packet loss ratio based on the counters exported by the PTPd daemon. Finally, we performed a small scale study in four data centers across the world in which we deployed a modified version of PTPd to perform network latency measurements. Our reproduction environment and results are available at [26].

Acknowledgments. Diana Andreea Popescu is supported by the EU FP7 ITN METRICS (grant agreement no. 607728) project. The authors would like to thank George Neville-Neil and the reviewers for useful comments.

REFERENCES

[1] N. Zilberman, M. Grosvenor, D. A. Popescu, N. Manihatty-Bojan, G. Antichi, M. Wójcik, and A. W. Moore, "Where has my time gone?" in *Passive and Active Measurement: 18th International Conference, PAM 2017, Sydney, NSW, Australia, March 30-31, 2017, Proceedings*. Cham: Springer International Publishing, 2017, pp. 201–214. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-54328-4_15

[2] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, no. 4, pp. 48–54, Mar. 2017. [Online]. Available: http://doi.acm.org/10.1145/3015146

[3] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 139–152. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787496

[4] A. Adams, P. Lapukhov, and J. H. Zeng, "NetNORAD: Troubleshooting networks via end-to-end probing," https://code.facebook.com/posts/1534350660228025/ netnorad-troubleshooting-networks-via-end-to-end-probing/, 2016, online; accessed February 2017.

[5] "IEEE 1588-2008 Precision Time Protocol," https://www.nist.gov/el/intelligent-systems-division-73500/introduction-ieee-1588, 2017, online; accessed March 2017.

[6] P. Ohly, D. N. Lombard, and K. B. Stanton, "Hardware Assisted Precision Time Protocol. Design and case study," in *In Proc. of the 9th LCI International Conference on High-Performance Clustered Computing. Intel GmbH*, 2008.

[7] "PTP daemon," https://github.com/ptpd/ptpd, 2017, online; accessed March 2017.

[8] "Solarflare PTP Adapters," http://www.solarflare.com/ptp-adapters, 2017, online; accessed March 2017.

[9] "Memcached," https://memcached.org/, 2017, online; accessed April 2017.

[10] "Memaslap," http://docs.libmemcached.org/bin/memaslap.html, 2017, online; accessed April 2017.

[11] "Netem," http://man7.org/linux/man-pages/man8/tc-netem.8.html, 2017, online; accessed March 2017.

[12] P. Ohly, D. N. Lombard, and K. B. Stanton, "Hardware assisted precision time protocol. design and case study," in *Proc. of the 9th LCI International Conference on High-Performance Clustered Computing. Intel GmbH*, 2008.

[13] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967

[14] N. Guilbaud and R. Cartlidge, "Localizing packet loss in a large complex network," https://www.nanog.org/meetings/nanog57/presentations/Tuesday/ tues.general.GuilbaudCartlidge.Topology.7.pdf, 2013, online; accessed February 2017.

[15] A. Roy, H. Zeng, J. Bagga, and A. C. Snoeren, "Passive realtime datacenter fault detection and localization," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 595–612. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/roy

[16] Cisco, "Cisco ios ip slas configuration guide," http://www.cisco.com/c/en/us/td/docs/ios/12_4/ip_sla/ configuration/guide/hsla_c/hsoverv.html, online; accessed April 2017.

[17] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "Scope: Easy and efficient parallel processing of massive data sets," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1265–1276, Aug. 2008. [Online]. Available: http://dx.doi.org/10.14778/1454159.1454166

[18] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng, "Packet-level telemetry in large datacenter networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 479–491. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787483

[19] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, *Software-Defined Latency Monitoring in Data Center Networks*. Cham: Springer International Publishing, 2015, pp. 360–372. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-15509-8_27

[20] M. Hira and L. Wobker, "Improving Network Monitoring and Management with Programmable Data Planes," http://p4.org/p4/inband-network-telemetry/, 2016, online; accessed September 2016.

[21] Y. Li, R. Miao, C. Kim, and M. Yu, "Lossradar: Fast detection of lost packets in data center networks," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: ACM, 2016, pp. 481–495. [Online]. Available: http://doi.acm.org/10.1145/2999572.2999609

[22] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[23] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 537–550. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787510

[24] N. T. Fundation, "Network time protocol," http://www.ntp.org/, online, accessed April 2017.

[25] K. S. Lee, H. Wang, V. Shrivastav, and H. Weatherspoon, "Globally synchronized time via datacenter networks," in *Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016.

[26] D. A. Popescu and A. W. Moore, "PTPmesh: Data Center Network Latency Measurements Using PTP," http://www.cl.cam.ac.uk/research/srg/netos/projects/latency/mascots2017/, 2017.