Measuring Network Conditions in Data Centers Using the Precision Time Protocol

Diana Andreea Popescu, *University of Cambridge,* Andrew W. Moore, *University of Cambridge*

***Abstract*—Increased network latency and packets losses can affect substantially application performance. Due to the scale of data centers, custom network monitoring tools have been developed to measure network latency and packet loss. In this paper, we use the Precision Time Protocol (PTP) to estimate one-way delay and to quantify packet loss ratios. We propose PTPmesh as a cloud network monitoring tool which uses PTPd as a building block. We present an analysis of PTPmesh latency and packet loss measurements collected in ten data centers from three cloud providers. Our findings reveal different latency, latency variance, packet loss and path symmetry characteristics across data centers. To foster further research in this area, we make our dataset available at [3].**

***Index Terms*—Data Centres, Network Latency, Packet Loss, Precision Time Protocol.**

I. Introduction

Network latency matters for certain distributed applications even in small amounts, affecting their application performance. While some data center applications simply process and transfer data, many applications are latency-sensitive, such as Web search [4], [5], social networking [4], [5], ML frameworks [6] or key-value stores [7]. These applications have stringent latency requirements, due to being interactive (search engine, social network) or due to their synchronous communication. Changes in network latency can lead to significant drops in application performance for latency-sensitive applications [8]–[10]. Even though in recent years network performance has improved substantially in the cloud, network latency variability is still common in data centers [11]. In order to ensure the best application performance, one needs to be able to continuously measure the network latency across paths in data centers. Having up-to-date network latency values helps in tracking network SLAs for applications and in quickly finding failures [12], [13]. These monitoring challenges can be addressed using a network monitoring system for data centers (§II-B). The system should be able to measure network latency across network paths and to detect packet losses, as these have a huge negative impact on application performance [12], [13].

In this paper, we investigate the use of the Precision Time Protocol (PTP) through an open source software implementation PTPd [14], to measure network conditions in order to use it as a building block for a data center network monitoring system. We validate the use of the Precision Time Protocol (PTP) through small-scale experiments for estimating network latency and packet loss. We propose *PTPmesh* as a network monitoring tool for data centers, whose building block is PTPd [14]. To validate the use of PTPmesh under real data center network traffic, we carry out a measurement study in different cloud providers (Amazon AWS, Google Cloud Platform, and Microsoft Azure) in ten data centers in different regions across the world, highlighting their characteristics regarding latency magnitude, latency variance and packet loss. PTPmesh is easy to deploy on cloud tenants' VMs, making it a feasible tool for cloud tenants to obtain network performance statistics without significant overhead and without needing access to any custom hardware at the end-host or in the network from the cloud providers. The statistics offered by PTPmesh can be used to determine normal network conditions, and to detect anomalies after determining the baseline.

D. A. Popescu and A.W Moore are with the Department of Computer Science and Technology, University of Cambridge, UK. Correspoding author's e-mail: diana.popescu@cl.cam.ac.uk. A preliminary version of this work appeared at the IEEE MASCOTS 2017 [1] and at the IFIP/IEEE TMA 2018 [2] conferences

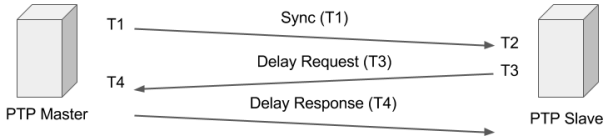


Fig. 1: PTP protocol.

II. Background and related work

A. The Precision Time Protocol (PTP)

The IEEE 1588 Precision Time Protocol (PTP) [15] is a standard protocol used to synchronise clocks over a network and it can achieve sub-microsecond precision. The master clock provides the reference time for the slave clocks. A *grandmaster* is chosen from the available clocks in the network. 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.

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. 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. A PTP *slave*, or *client*, records the time when it received a Sync message (T_2). The difference between the send and receipt times represents the *master-to-slave delay,* d_{m2s}:

$$d_{m2s} = T_1 - T_2 \tag{1}$$

A PTP slave sends a *Delay Request* message. The slave records the time when the Delay Request message was sent (T_3), while the master records the receipt time (T_4). The difference between the send and receipt times of the Delay Request messages represents the *slave-to-master delay,* d_{s2m}. The master will reply with a *Delay Response* message which contains the receipt time T_4, thus:

$$d_{s2m} = T_3 - T_4 \tag{2}$$

By assuming that the propagation delays master-to-slave and slave-to-master are symmetric, which usually translates to paths being symmetric, the *one-way delay* is computed as half of the sum of the two delays.

$$\text{OWD} = \frac{d_{m2s} + d_{s2m}}{2} \tag{3}$$

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.

$$\text{offset} = d_{m2s} - \text{OWD} = \frac{d_{m2s} - d_{s2m}}{2} \tag{4}$$

In the case that the master-to-slave and slave-to-master delays are asymmetric (due to network congestion for example), the clock offset will suffer perturbations and the precision of the clock synchronisation will be affected.

The messages sent by PTP fall in two categories: *event* and *general*. Messages like Sync and Delay Request message are event messages, whose send and receipt timestamps are used to compute the adjustment of the slave clocks, and thus the timestamps need to be accurate. Messages like Announce, Follow-up and Delay Response are general messages, and do not require accurate timestamps. Event messages are sent on port 319, while general messages are sent on port 320. PTP messages are sent using multicast messaging, but devices can negotiate unicast transmission if desired. PTP messages are usually sent over UDP. PTP supports two delay measurement mechanisms: peer-to-peer and end-to-end. In the peer-to-peer mechanism, each network device is PTP-aware, and the time synchronisation operates between the end-host and the network device. In the end-to-end mechanism, which we use in this work, only the end nodes need to be PTP-aware.

The send and receipt timestamps for the PTP packets can be generated either by the host operating system's kernel (software timestamping), or by a dedicated hardware unit (hardware timestamping). The first type of timestamping has the advantage of being widely available, but the timestamps generated are less precise due to variable interrupt servicing latencies [16]. The second type of timestamping is precise, but requires special hardware. For example, Solarflare network interface card (NIC) [17] generates hardware timestamps for PTP packets using a dedicated time stamping unit which is driven by an oscillator. On the arrival or departure of a PTP packet, the unit generates a hardware timestamp which is passed by the NIC to the network device driver. Additionally, a PTP stack enabled by the NIC is running on the server to discipline the NIC's precision oscillator. A user space application can access the hardware timestamps for the received packets using the SO_TIMESTAMPING socket option available in the Linux kernel.

PTP uses various mechanisms to ensure that there is no interference in the clock synchronisation. Firstly, PTP can use hardware timestamping to eliminate the end-host delay caused by the network stack and variance due to interrupt service latencies [18]. Secondly, PTP-enabled switches that run as transparent clocks can modify a field in the PTP messages to account for the delay incurred across the switches. In this 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.

Software implementations of the PTP protocol are PTPd [14] (open source) or TimeKeeper [19] (commercial). PTPd is a software-based system that uses software timestamps. It runs as a background user-space process. PTPd's precision is determined by the precision of sent and received messages timestamps. PTPd uses the Linux kernel's software clock. It adjusts the clock using the `adjtimex()` interface for clock tick-rate adjustment.

B. Data centre network monitoring systems

In this section, we review the most important tools to measure network latency and packet loss in data centers. Measuring network conditions within a data center is notoriously difficult, since the tools used need to satisfy several properties: be lightweight, always-on, not load the network, so as to not degrade users' application performance, offer information that can be quickly acted upon, and be easy to use and configure by network operators or users. Such a custom data center network monitoring tool can take advantage of the data center's known topology, and hardware and software configuration. Furthermore, the measurement techniques must be complemented by a highly scalable storage and analysis system that can alert operators about issues within the network, such as high latency, packet loss, but also to provide historical data to understand trends. The systems designed for data centers are based on active measurement, and can be complemented by passive techniques, such as exploiting the timestamps carried in the TCP headers when these are enabled [20]. Passive measurement of TCP RTT [20] is comparable in accuracy with ICMP measurement. If losses occur, the application will experience higher latencies due to TCP's in-order delivery semantics. Another way to monitor network latency in a data center, though costly, is to have each host equipped with a common clock, such as a GPS receiver, and run one-way delay measurements between hosts.

Table I compares the properties of systems used to measure network latency and packet loss in data centers, including PTPmesh. The comparison looks at aspects related to type of measurements taken, their frequency and coverage, availability, implementation, deployment, and data storage and analysis of collected measurements. A pair is defined by two hosts: one that sends a probe, and another that receives the probe and sends an answer. *Ping* and *traceroute* are the traditional tools to perform such measurements, however, these lack the precision, the flexibility and the scale of custom purpose built tools for data center monitoring. Cisco IP SLA [21] 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.

Large-scale monitoring systems, such as NetNORAD [13], Everflow [22], Pingmesh [12], or VNET Pingmesh [23], have originated from companies and cloud providers. NetNORAD [13] 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 [22] 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. Their behaviour is monitored through the network, and can be used to measure link RTTs. Pingmesh [12] is an always-on tool that runs 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. An extension to Pingmesh is VNET Pingmesh [23], which monitors latency for tenant virtual networks (VNETs), whereas Pingmesh performed bare-metal host monitoring. The TCP probes are sent from the virtual switch at the end-host. Unlike Pingmesh which measured latency from userspace, VNET Pingmesh measures the latency from the kernel, but the measured values can be negatively affected by increased CPU utilisation, disk I/O operations and caching effects. SLAM [24] is a latency monitoring framework for SDN-enabled data centers, which 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.

In a large-scale measurement system, probing is normally done between chosen pairs of servers at defined time intervals. Since a data center has tens of thousands of hosts, a server does not ping every other host, but instead a subset of servers is selected to ensure the best coverage, while minimising the number of redundant probes and reducing the network traffic incurred. Another challenge associated with probing is the multi-path nature of the data centers coupled with the use of ECMP, making it hard to know which network path the probes are taking, unless tracing the trajectories of packets through embedded identifiers is used [25], [26]. In Pingmesh [12], all of the servers under a ToR switch form a complete graph for pinging each other, and similarly all of the ToR switches form a complete graph through designated servers from all racks, and all of the data centers form a complete graph using the same procedure. Unlike Pingmesh, VNET Pingmesh [23] covers only the network paths between tenant VMs. NetNORAD [13] deploys a small number of pingers in each cluster and responders on all of the machines. All of the pingers use the same global target list, which contains at least two machines from every rack. deTector [27] uses an algorithm to minimise the number of probes sent for detecting and localising packet losses and latency spikes.

Programmable switches [28] enable more sophisticated operations for network monitoring, allowing the measurement of latency and packet loss directly in-network. Examples are Inband Network Telemetry (INT) [29], LossRadar [30], Marple [31]. The disadvantage of these frameworks is that programmable switches must be deployed in the network, with legacy networks not being able to run these frameworks. INT [29] measures 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. LossRadar [30] is a system that can detect packet losses in data centers within 10s of milliseconds, reporting their switch locations and the 5-tuple flow identifiers. It keeps specific data structures at switches, which are periodically exported to a remote collector and analyser. It does not perform latency measurements. Marple [31] uses programmable key-value stores on switches to compute different metrics, such as a moving exponentially weighted moving average (EWMA) over packet latencies per flow, packet loss rate per connection, or to capture packets experiencing high end-to-end queueing latency.

III. Measuring network conditions with the Precision Time Protocol (PTP)

A. Experimental setup and methodology

We use two testbeds for the experiments in this paper. The first testbed in Figure 2 consists of six servers Intel Xeon E5-2430L v2 running at 2.40GHz, with Ubuntu 16.04, kernel version 4.4.0.64-generic, equipped with 10Gb/s Intel X520 NICs with two SFP+ ports. The servers are connected using two Arista 7050Q switches, and all network links are 10Gbps. This testbed does not have PTP-enabled NICs with hardware timestamping (§II-A). The experiments in this paper use the default NIC settings. The two hosts running PTPd do not send or receive any other network traffic, thus the PTPd measurements can be affected only by the traffic originating from the four other hosts in the testbed (*Memcached* and *iperf* traffic).

For most of the experiments we do not use PTP-enabled NICs, because this type of NIC is not available to the tenants to access in cloud data centers. However, we additionally run

	Measurement	Probe Type	Probe Frequency	Availability	Coverage	Deployment	Data Storage and Analysis
Ping	RTT; packet loss ratio	ICMP	-	single measurement	targeted pair	Hypervisor or VM	locally; analyse independently
Traceroute	RTT	ICMP ECHO/ TCP SYN	-	single measurement	targeted pair	Hypervisor or VM	locally; analyse independently
Cisco IP SLA [21]	RTT; one-way delay (requires synchronised clocks); packet loss	ICMP/ UDP/ TCP/ HTTP/ DNS	between 1 and 604800 seconds	always-on	targeted path	CISCO switches	locally; analyse independently
Pingmesh [12]	RTT; packet loss ratio	TCP/HTTP	minimum 10s seconds	always-on	inter-servers in a rack, inter-ToRs, inter-data center	Hypervisor	Cosmos and SCOPE [32]
NetNORAD [13]	RTT; packet loss ratio	UDP	configurable	always-on	all pairs	Hypervisor or VM	Scribe and Scuba [13]
Everflow [22]	link RTT	packet marked with debug bit	-	single measurement	targeted path	switches and controller	custom analyser and SCOPE [32]
SLAM [24]	network path latency distribution	crafted probe	-	single measurement	targeted path	OpenFlow switches	controller
INT [29]	end-to-end latency	crafted probe	-	single measurement	targeted path	programmable switches	last switch on path; analyse independently
LossRadar [30]	packet losses at switches	no probes	10 ms	always-on	cover all paths	programmable switches	custom collector and analyzer
deTector [27]	packet drop	UDP	10 packets/sec	always-on	selected paths	end-host, central controller	custom analyser
007 [33]	packet drop	TCP and traceroute	per flow	always-on	all paths	end-host	custom analysis agent at end-host
PathDump [34]	packet drop	no probes	per packet	always-on	all paths	end-host and switches	custom server stack and controller
Marple [31]	packet drop	no probes	per flow	always-on	all paths	end-host, programmable switches	programmable key-value store
VNET Pingmesh [23]	RTT; packet loss ratio	TCP	minimum 10s seconds	always-on	VNET full mesh	virtual switch at end-host	Cosmos and SCOPE [32]
PTPmesh [1]	one-way delay (estimate); packet loss ratio	UDP	up to 128 probes per second	always-on	multiple pairs	Hypervisor or VM	locally; analyse independently

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

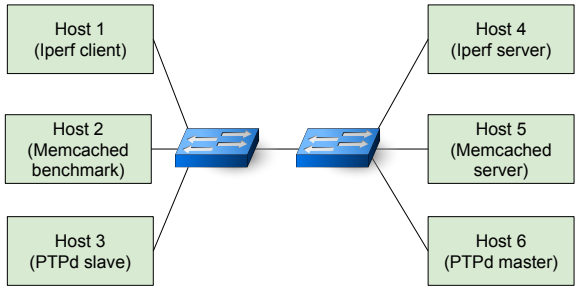


Fig. 2: Testbed to analyse PTPd's behaviour under different network conditions.

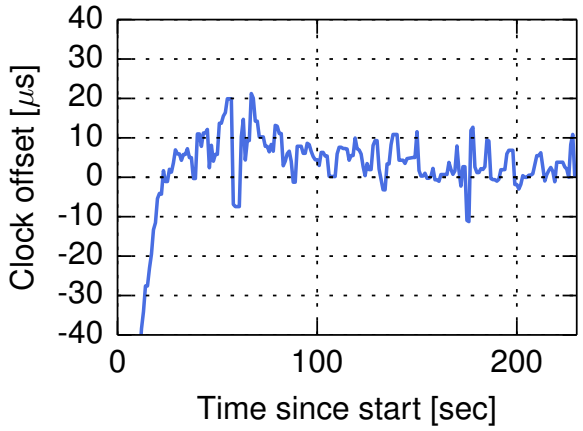


Fig. 3: The slave's clock offset is within 20μs of the master's clock after less than five minutes after PTPd's start-up.

experiments using PTP-enabled NICs to compare the results obtained in this second approach to the ones obtained using the testbed without PTP-enabled NICs. The second testbed is 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 [17] with hardware timestamping (§II-A).

PTPd logs measurements such as the clock offset, the master-to-slave delay, the slave-to-master delay, and the one-way delay. The interval for sending Sync and Delay Request messages can be configured in PTPd, up to 128 messages per second for each, expressed as $\log_2$ values between -7 and 7. The default setting is 0, which means sending 1 message per second of both *Sync* and *Delay Request* message types, and 2^{-7} means 7.8125ms between messages, with 128 messages per second. We call the number of messages sent per second *message frequency* in order to distinguish the name from message rate, since in the case of PTP, there is a set time interval between the messages exchanged, the messages being sent with a given frequency.

In all of the experiments, we wait for an initial period to reach a stable state before making changes to the system, *e.g.*, starting other applications that send traffic. After PTPd

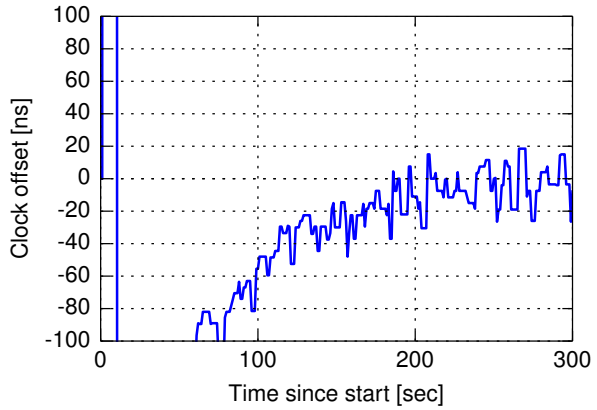


Fig. 4: The slave's clock offset is within 40ns of the master's clock after less than five minutes after Solarflare's PTP daemon start-up.

starts up, it performs an initial clock reset if the clock is off by one second. Then the slave clock gradually synchronises with the master clock. Thus, before this convergence period ends, the system is not in a stable state, and this may distort the results of the experiments. Next, we measure on my first testbed the convergence period when using a message frequency of 1 message per second. We verify that five minutes are sufficient for the PTPd master and PTPd client to synchronise to within 20μs of each other (Figure 3). Allowing more than five minutes for convergence did not decrease the margin between the two clocks' values, with the clocks remaining within 20μs of each other. For the second testbed which uses PTP-enabled NICs, the clocks are synchronised within 40ns in less than five minutes (Figure 4). Hence, we wait for five minutes before running any intended experiment. The results in the following subsections represent sample runs of the experiments.

B. Measuring network latency

We measure the RTT in the first testbed between the PTPd master and PTPd slave hosts, using *ping* and the UDP-based tool [8] that uses the Time Stamp Counter (TSC), and we compare the values obtained divided by two with the one-way delay reported by PTPd. For PTPd, we set the message frequency for Sync and Delay Request messages to 1 per second, and we run the clock synchronisation for 15 minutes. For the two other experiments, we run 1 million RTT measurements with the UDP-based tool, and $30,000$ ping probes. There is no other network traffic in the testbed, and each test is conducted separately.

The network latency CDF is presented in Figure 5. Intuitively, one would have expected the one-way delay to be half of the values reported by the UDP-based tool, however this was not the case in the default configuration (Figure 5b). We investigated why this happened, and we found that changing the interrupt rate of the NIC at both the master and the slave by setting to zero the number of microseconds to wait before raising an RX interrupt after a packet has been received gives the expected results for the one-way delay reported by PTPd (Figure 5a). This experiment shows the drawbacks of software timestamping, and reinforces the importance of using hardware timestamping for obtaining precise measurements. However, most cloud operators do not allow changing the default network stack settings.

Once the two clocks are synchronised, the one-way delay reported by PTPd is stable; there is no long tail for the reported one-way delay, due to filtering of abnormal values. On the other hand, the RTT CDFs produced by the two other tools exhibit a long tail due to OS scheduling artefacts [8]. The one-way delay reported by PTPd and the RTT/2 reported by the UDP-basel tool is approximately half of the median *ping* RTT/2 values. This difference may be due to how the ICMP traffic is treated in the network and at the end-host network stack [35]. Furthermore, *ping* may not be appropriate as a measurement tool for network latency in the cloud, because of the possibility that ICMP packets are treated differently, *e.g.*, being redirected for security checks [36].

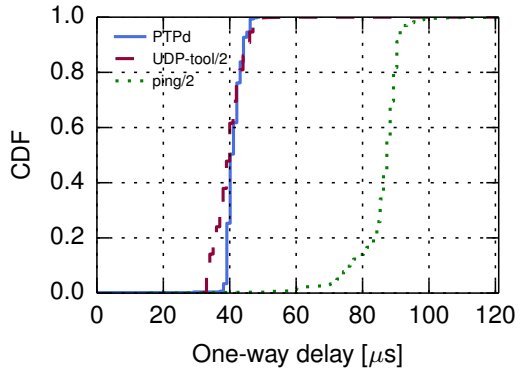
C. The effect of network congestion on PTPd measurements

The experiments in this section aim to answer the second question posed in the introduction, namely how network congestion affects the PTPd measurements.

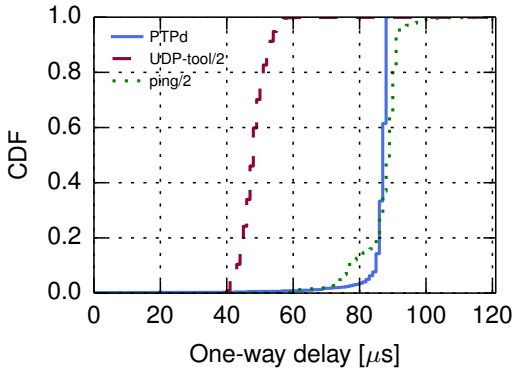
1) Concurrent network traffic: We study the effect of network congestion on the measurements reported by the PTPd slave using the testbed in Figure 2. In each test, we allow a clock synchronisation phase of 5 minutes for PTPd, before starting concurrently the two other applications, *Memcached* [37] (with its corresponding benchmark *memaslap* [38]) and *iperf* in TCP mode. *Memcached* is a latency-sensitive application, for which increases in network latency lead to significant performance loss [8], [9]. In this specific experiment, we set the interval for the Sync and Delay Request messages to 0.25 seconds (message frequency of 4 messages per second), but the results are similar for different message intervals. We run two experiments: i) a 5s iperf stream running (Figure 6a) and ii) three 5s iperf streams with 5s breaks between them (Figure 6b).

The first experiment (Figure 6a) shows that the congestion episode determined by iperf leads to an increase in the slave-to-master delay (on the iperf stream's direction). PTPd packets are queued in switches behind iperf's packets, thus it takes longer for the packets from the slave to reach the master, hence the increase in the slave-to-master delay. Consequently, the one-way delay increases. The PTPd slave interprets these changes as clock offset from the master clock, then corrects its clock accordingly, and as a result changes also appear in the master-to-slave delay. After TCP exits the startup phase and reaches the steady state, and assuming that the iperf stream continues to run after this state is reached, the slave's clock will gradually reconverge, with the clock offset nearing zero. However, the one-way delay will still reflect an increased delay determined by the iperf traffic.

The second experiment (Figure 6b) shows that if there are several congestion episodes before the slave clock manages to resynchronise with the master clock, the one-way delay reported by the slave is not indicative of the actual delay, but it still indicates that there is an event (network congestion,

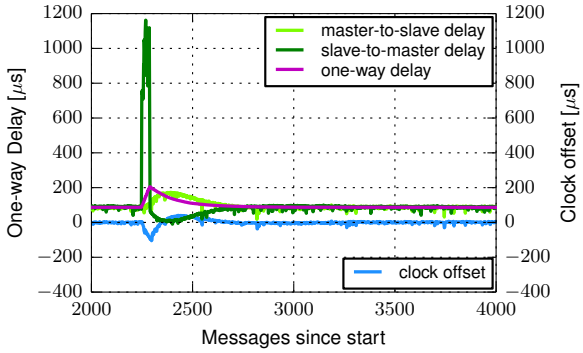


(a) Setting `rx-usecs` to 0

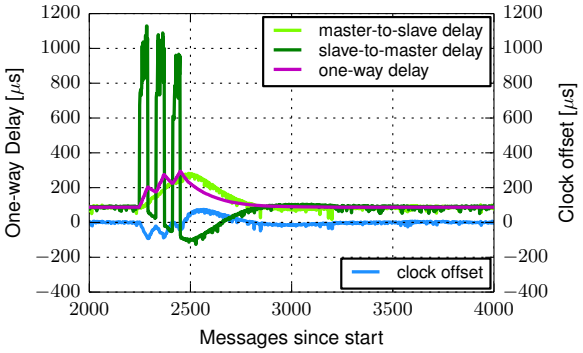


(b) Setting `rx-usecs` to 1

Fig. 5: RTT/2 reported by *ping* and the UDP-based tool that uses the TSC [8], and one-way delay reported by PTPd



(a) A 5s iperf stream starts running at second 300.



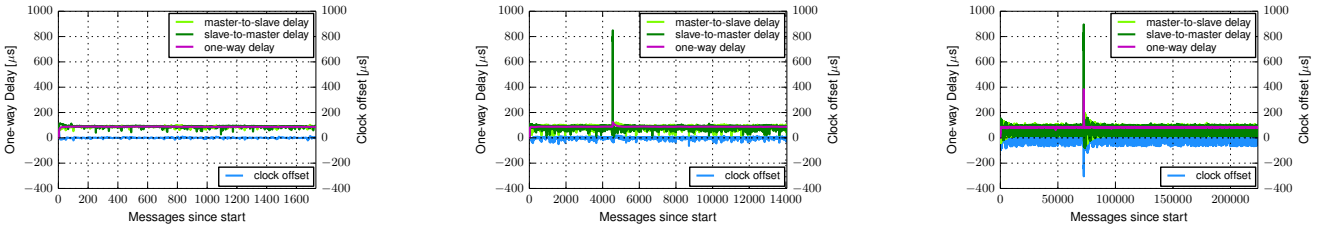
(b) Three 5s iperf streams start running at second 300s, 310s and 320s, respectively, with 5 seconds breaks between streams.

Fig. 6: Network congestion effect on PTPd measurements.

link failure) on that network path. In this experiment, the first congestion episode caused by iperf has the same effect as in the first experiment. The next two intervals of iperf traffic produce further deviations to the slave-to-master delay, because the two clocks did not have time to resynchronise before the start of the next iperf stream. The figure illustrates how the delays are gradually going back to the baseline values, but before this can fully happen, a new iperf stream starts. While this experiment shows that the one-way delay does not provide the true value of the latency between hosts at all times, the approach can be used to understand long-term trends, and with appropriate data post-processing the accuracy of the OWD measurement could be increased. There are two ways to deal with the periods of increased latency during the reconvergence period. One is to increase the message frequency, which would reduce the period needed for convergence. The second one is knowing how quickly the clocks reconverge under normal circumstances after a period of congestion, which gives the time interval during which the results are distorted. Then, for this time period the additional latency observed compared to the baseline latency can be subtracted from the OWD returned by PTPd. A shown in the experiment, if the congestion caused by network traffic resumes, the OWD becomes even higher than the previous peak value (since the OWD did not get back to the baseline value), so this process has to be continued accounting the value to which the OWD had time to reconverge to. If the network utilization increased for a longer period of time (minutes or hours), the OWD will reflect this increased utilization over the respective period of time, but when the utilization drops, the OWD will not go back instantaneously to the value before the congestion.

In both experiments, Memcached shows increased request-response latencies due to the iperf traffic, in the second experiment this increase being less than in the first experiment, as iperf runs for a total shorter period of time. These results show that the changes observed in the PTPd OWD measurements can serve as an indicator that the performance of other applications running on the same network may suffer.

2) Changing the message frequency of the Sync and Delay Request messages: This experiment explores the time resolution at which network congestion affects the PTPd measurements by changing the interval at which messages are exchanged between the PTPd master and PTPd client. We perform the same experiment with *iperf* and *memaslap* concurrently running with PTPd, but *iperf* runs for a duration of 1s. We vary the interval at which the Sync and Delay Request messages are sent, from 1s down to 7.8125ms, meaning from 1 messager per second to 128 messages per second. This allows



(a) 1s interval for Sync and Delay request messages. A 1s iperf stream starts running at second 300, PTPd measurements do not detect it.

(b) 125ms interval for Sync and Delay request messages. A 1s iperf stream starts running at second 300, PTPd measurements detect it.

(c) 7.8125ms interval for Sync and Delay request messages. A 1s iperf stream starts running at second 300, PTPd measurements detect it.

Fig. 7: Changing the interval for the Sync and Delay Request messages.

detection of congestion periods at milliseconds resolution. Increasing the message frequency beyond 128 messages per second would allow detection at an even higher resolution. In Figure 7a, it can be seen that iperf does not produce any change in the PTPd measurements, since the interval between messages is the same as iperf's runtime, hence it is not running long enough to delay the PTP packets. However, when the interval is decreased (Figures 7b and 7c), the iperf traffic leads to deviations in the PTPd measurements, and an increase in the one-way delay, similar to Figure 6a. The clock offset, master-to-slave and slave-to-master delays oscillate between larger values when the Sync and Delay Request interval is smaller (comparing the width of the lines in Figure 7b and Figure 7c). This may happen because of software timestamping, or because of PTPd clock servo algorithm's settings. The one-way delay does not exhibit such significant oscillations, as it is computed as the average of the master-to-slave and slave-to-master delays.

3) Convergence period for the PTPd measurements after network congestion: We perform an experiment to explore how long it takes for the PTPd measurements to return to the same values they had before network traffic that caused congestion was injected in the network using a stream of iperf of 1s, and is performed for each message frequency between 1 and 128 messages. We count the number of samples greater than the baseline OWD value for the first testbed in Section III-B to determine how many messages are needed before the OWD returns to this baseline value after iperf finished running. The results show that the higher the message frequency is, the shorter the reconvergence time will be. An exact relationship between the message frequency and the convergence time cannot be derived from the results, but the converge time in most cases approximately halves as the message frequency doubles.

D. Measuring network latency in virtualised environments

All of the previous experiments were performed without virtualisation, on bare-metal hosts. In order to be able to interpret the measurement data collected in the cloud, where virtualisation is the norm, we run an experiment on the first local testbed to quantify the overhead of virtualisation on PTPd measurements, more specifically on the OWD, with the PTPd master and PTPd client running in VMs. The hypervisor used is Oracle VM VirtualBox version 5.0.40 Ubuntu r115130. The results of the two experiments are shown in Table II, and show that virtualisation adds almost 200μs of overhead and causes increased jitter to the OWD compared to a non-virtualised setting. Even so, the standard deviation of the OWD is not significant. These issues can be solved by using PTP-enabled NICs which provide hardware timestamping. Additionally, OS bypass through a custom software packet processing path [39] or through custom hardware [40] alleviates these issues. Also, some cloud providers (Amazon AWS) offer bare metal instances, thus the virtual switch overhead does not exist in this case. These results show that the OWD increases due to virtualisation, but the results are demonstrative of the *OWD's stability*, the OWD having low standard deviation even in the presence of virtualisation, making PTPd a convenient way to estimate network latency.

E. Estimating packet loss ratio

Packet loss increases the latency perceived by the user, since dropped packets need to be retransmitted [12]. It is thus important to keep track of the packet loss ratios, as these can be correlated with the observed application performance. Additionally, tracking packet loss helps to uncover software or hardware faults in the network.

PTPd records the number of messages sent and received (Announce, Sync, Followup, Delay Request, Delay Response), and it is possible to export these numbers 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. The packet loss ratio over a defined interval of time can be approximated as:

$$pkt\_loss\_ratio = 1 - \frac{\#Delay\_Response\_messages}{\#Delay\_Request\_messages} \quad (5)$$

Normal operation should see the same number of Delay Request and Response messages or a difference of at most one message. One disadvantage of computing the packet loss ratio in this way is that it does not account for the Announce, Sync and Followup messages that were potentially lost.

We verify if the proposed metric can be used as a coarse estimation for the packet loss ratio by artificially introducing packet loss in the network. We use *NetEm* [41], an enhancement of the Linux traffic control facilities, to emulate packet

	Min	Average	Median	99^{th}	99.9^{th}	Max	Std. dev.
Bare metal 1msg/s	$80.85\mu s$	$82.59\mu s$	$82.68\mu s$	$84.06\mu s$	$84.15\mu s$	$84.15\mu s$	$0.8\mu s$
Virt. 1msg/s	$273.96\mu s$	$286.7\mu s$	$285.98\mu s$	$295.72\mu s$	$295.9\mu s$	$295.92\mu s$	$5.27\mu s$
Bare metal 128msg/s	$51.28\mu s$	$64.54\mu s$	$64.7\mu s$	$69.97\mu s$	$71.12\mu s$	$72.1\mu s$	$2.65\mu s$
Virt. 128msg/s	$201.69\mu s$	$253.33\mu s$	$253.59\mu s$	$271.21\mu s$	$284.86\mu s$	$327.25\mu s$	$7.62\mu s$

TABLE II: One-way delay reported by a PTPd client on a bare metal host and with virtualisation for different message frequencies.

NetEm packet loss ratio	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%

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

loss on the outgoing network interface of the host which runs the PTPd master. In this scenario, none of the Delay Request messages are lost, although in practice this may happen. Since outgoing PTPd packets are looped back via the IP_MULTICAST_GROUP [18], loss conditions are applied on both the physical interface and the loopback interface. 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 packet loss ratios of $1\%, 5\%$ and 10%, and we compute the packet loss ratio as described above to see if it matches the induced loss ratios. We run the clock synchronisation 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 ratio we perform 5 runs. The results are presented in Table III. It can be seen that the metric we defined can serve as a coarse estimate of the packet loss ratio over a defined interval of time.

F. PTP-enabled NICs

NICs supporting PTP are becoming increasingly available. The measurements performed by a PTP implementation that leverages this support do not suffer from end-host interference caused by other network traffic that originates from the same host. Furthermore, this type of NIC also removes the delay associated with the end-host network stack or virtualisation layer from measurements, the measurements thus reporting only the actual network latency.

We run experiments to see if PTPd measurements are adversely affected by other network traffic that originates from the same host to answer the fifth question posed in the introduction. This might happen because of end-host packet processing delays under increased load. On the second testbed described in Section III-A, We compare the clock offset and one-way delay obtained from *sfptpd*, Solarflare NIC [17] PTP daemon which uses hardware timestamping (see Figure 8, note: ns y-scale), and from *PTPd* (see Figure 9, 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. It can be seen from the two figures that the clock offset reported by

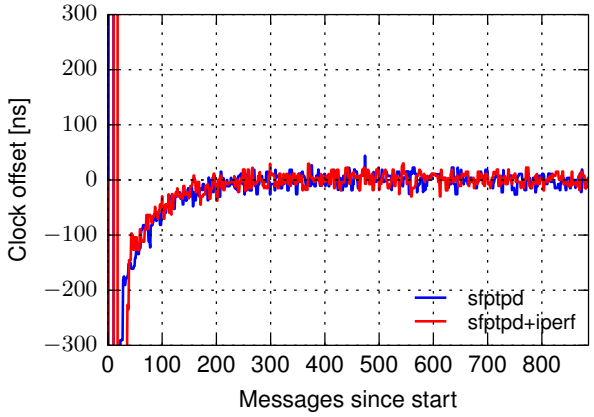


Fig. 8: The clock offset reported by *sfptpd* is not affected by the iperf traffic, since it uses NIC hardware timestamping.

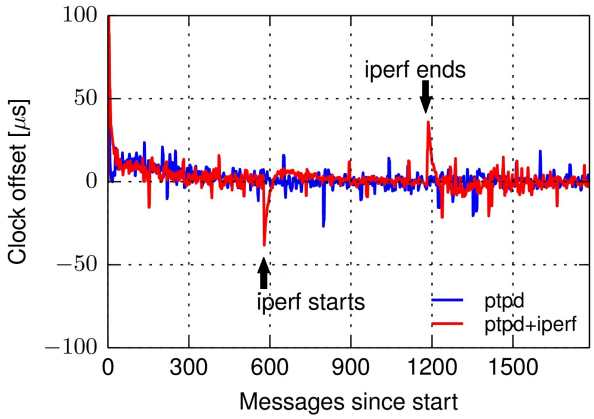


Fig. 9: The clock offset reported by PTPd is adversely affected by the iperf traffic because of end-host interference.

sfptpd is not affected by the iperf traffic. However, in the case of *PTPd*, the clock offset deviates when the iperf stream starts and ends. Furthermore, it should be noted that for hardware timestamping the clock offset's magnitude is nanoseconds, while for software timestamping it is microseconds.

IV. Measuring the cloud network with PTPmesh

A. Deployment scenarios

We consider two possible deployment scenarios for a system based on PTP in data centers [42]. In the first scenario, the cloud provider deploys PTPd [14] (or a different software implementation for PTP) in the hypervisor, possibly alongside a separate clock synchronisation mechanism. Several PTPd clients can run on the same machine in different PTP domains, and thus they do not interfere with each other. In the second

scenario, the tenants themselves run PTPd inside their VMs and use the reported measurements to check the network conditions. PTPmesh's design follows the second scenario. In both scenarios, the PTP traffic should not be prioritised, and switches in the network should not be PTP-aware, otherwise the measurements would not be indicative of the actual network latency.

Since ECMP 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 these two servers. To mitigate this issue, a similar approach to the one used in Pingmesh [12] and NetNORAD [13] can be used, specifically changing the port numbers on which PTPd is running. Since the port number is part of the ECMP hash computation, for every port number a different ECMP hash value is obtained. As a result, ECMP may select different network paths to route the PTP packets for different port numbers. Looping over a range of port numbers would ensure that the PTP traffic is sent over each distinct network path between two servers [12], [13]. Moreover, if the cloud operator knows how ECMP is implemented on their switches and they do not use randomness in the ECMP hash function [43], then the cloud operator can define a list of port numbers for PTPd to ensure that each distinct network path between the two hosts is covered. Alternatively, if the trajectory of the packets can be traced using techniques such as the ones described in [26], [44], then it would be straightforward to verify whether all network paths between two servers are covered when using a range of port numbers for PTPd.

B. Measurement methodology

We use PTPd v2 2.3.1 [14], in unicast mode for the cloud deployment, with unicast negotiation and end-to-end delay measurement (§II-A). We measure the one-way delay between multiple VMs from different cloud providers. In cloud computing terminology, a *region* is a geographical location where compute resources can be deployed, and it comprises one or more *zones*. Usually, a region has three or more zones. For each of the three cloud platforms, Amazon AWS EC2, Google Cloud Platform - Compute Engine, and Microsoft Azure, we choose several zones. We deploy two, four or ten VMs in each zone. The VMs run Ubuntu 16.04. We run the PTPd master on one VM, while the other VMs act as PTPd slaves, running simultaneously. We used the following VM types: AWS - t2.micro, GCE - n1-standard-1, Azure - Standard D1 v2, Standard D2s v3, Standard_E16s_v3 and Standard_E32s_v3. It should be noted that the latency measurements collected may be influenced by the underlying server hardware, as shown by previous research [45].

We list the zones along with an assigned name to identify the traces collected. For Amazon AWS EC2, we run measurements in regions Ireland zone eu-west-1a (EC2-EUW), Northern California zone us-west-1b (EC2-USW) and Ohio zone us-east-2a (EC2-USE). For Google Compute Engine, the measurements are run in us-west1-b (GCE-USW) and europe-west1-b (GCE-EUW), and between us-west1-b (GCE-USW) and us-west1-d (GCE-USW2). For Microsoft Azure,

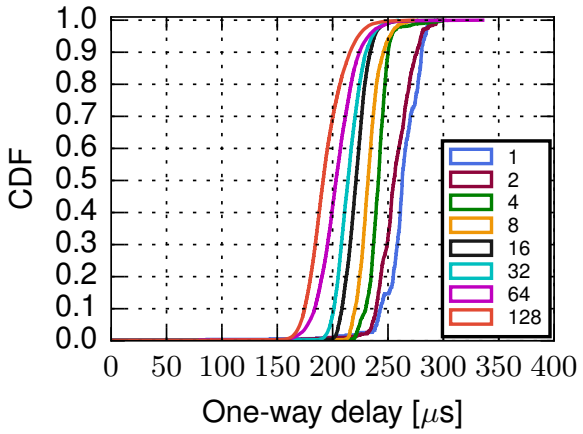


Fig. 10: OWD measured using PTPd for periods of 15 minutes between two VMs in Azure-KS.

We use UK West (Azure-UKW), UK South (Azure-UKS), US West (Azure-USW) and Korea South (Azure-KS). In the UK South, the VM type we use is the Standard D2s v3 and Standard_E16s_v3 or Standard_E32s_v3 (with Azure Accelerated Networking [40] enabled), while in the other zones we use the Standard D1 v2. We will refer to a zone as a data center in the rest of the paper, but the underlying network topology and configuration of a zone is not disclosed by the cloud providers.

C. Measurement calibration

1) Message frequency impact: We perform several experiments where we vary the number of messages between 1 to 2^7 per second to determine whether a different message interval yields different one-way delay values. We vary the message frequencies of both the Sync and Delay Request messages exchanged between the master and the slave, and we use the same message frequency for both.

We perform an experiment with a PTPd master and a single PTPd client running in the Azure-KS data center with different message frequencies. Figure 10 shows the OWD CDF for different message frequencies. As the message frequency increases, the OWD decreases, going from median 262.92μs and 99^{th}percentile 286.6μs for 1 message per second, to 191.49μs and 99^{th}percentile 237.85μs for 128 messages per second. It is speculated that the cause of this behaviour is that, when the message frequency increases, the code that performs the timestamping remains in the cache, leading to smaller OWD values.Another cause might be due to the way the interrupts are coalesced at the NIC, since messages are not timestamped by the NIC, but by the kernel.

While increasing the message frequency leads to better accuracy for the one-way delay measurements, the CPU utilisation and network bandwidth consumption increase. Since the initial goal was to have a low-overhead measurement system that runs as a service in a VM or in the hypervisor, choosing the message frequency implies a tradeoff between host and network resources consumption and measurement accuracy. We run measurements using the same setup in the Azure-KS

# msg/s	CPU utilisation [%]	Avg. net. bw. rx [Kb/s]	Avg net. bw. tx [Kb/s]	Avg [μs]	Std.dev. [μs]
1	0	0.65	2.13	262.11	21.54
2	0	1.12	3.88	255.88	19.15
4	0	2.21	7.38	240.63	14.1
8	0	4.35	14.38	232.42	13.19
16	0.1	8.68	28.44	221.02	13.21
32	0.2	17.35	56.53	214.7	13.56
64	0.4	34.43	112.12	203.67	18.4
128	0.7	68.77	223.7	193.37	17.56

TABLE IV: The setup has one PTPd master and one PTPd client. CPU utilisation and network bandwidth double as the message frequency doubles. OWD average goes down, while standard deviation is roughly the same.

data center for each message frequency from 1 message per second to 2^7 messages per second for 15 minutes to monitor the average CPU utilisation (using *top*), memory, and send and receive network bandwidth (using *iftop*).

Table IV presents the CPU utilisation and network bandwidth for different message frequencies. The results show that the accuracy of the OWD measurements improves as the message frequency increases with the average OWD decreasing, while the precision of the measurements stays roughly the same, the standard deviation for different message frequencies being almost the same. It should be noted that the OWD measurements may have been affected by concurrent network traffic within the data center. On the other hand, the CPU and network resources double as the message frequency doubles. The experiment shows that it takes approximately 100 seconds to reach a value close to the maximum CPU utilisation of the PTPd master. For a frequency of 1 message per second, the average CPU utilisation is almost 0% and the average network bandwidth consumption is 0.65Kb/s (receive) and 2.13Kb/s (send). For a frequency of 2^7 messages per second, the average CPU utilisation is 0.7% and the average network bandwidth consumption is 68.77Kb/s (receive) and 223.7Kb/s (send). Since these values are reported for a single slave, when synchronising with multiple slaves, it is expected the network bandwidth will increase proportionally. For example, if using $1,000$ slaves, the average network bandwidth at the PTPd master would be 6.87 Mb/s (receive) and 22.37 Mb/s (send). The memory usage for the PTPd master is the same regardless of the message frequency, being 0.1% when using a VM with 1.6 GB RAM. However, the number of PTPd clients has an impact on the amount of memory used by the master [14], with the maximum number of clients (unicast destinations) supported being 2048.

To sum up, depending on the available resource budget, more accurate OWD measurements can be obtained, but at the expense of more CPU and network resources. Memory requirements for running the PTPd master are consistently low. There is a tradeoff between measurement accuracy and resource consumption that should be considered when choosing the message frequency. A *lightweight network monitoring system* should not incur significant overhead on the end-host or the network. For example, Pingmesh [12] uses less than 45MB memory, the average CPU usage is less than 0.26%, and it sends only tens of Kb/s. Thus, based on the Pingmesh resource usage and depending on the number of PTPd clients that synchronise with one PTPd master, the message frequency for PTPd, and thus for PTPmesh, should be chosen between 1 and 32.

In Section V, we conduct most of the latency measurements using a message frequency of 1 message per second on a setup with one PTPd master and three PTPd clients as part of PTPmesh in order to have similar CPU utilisation and network bandwidth consumption as Pingmesh. We additionally perform measurements with a message frequency of 128 messages per second for higher OWD measurements accuracy.

2) Number of concurrent PTPd clients: Another aspect that needs to be taken into account is the number of slaves a master can synchronise with before the end-host becomes overloaded because of processing too many messages, which may affect the measurement accuracy. To see if the number of clients affects significantly the OWD values, we perform a suite of experiments with PTPmesh on a local testbed with ten bare-metal hosts, using one PTPd master and a maximum of nine PTPd clients, and a similar experiment in EC2-USE (one VM PTPd master and up to nine VM PTPd clients), using 128 messages per second. Using a local testbed with nine clients and one master, we found that the reported OWD is not affected by the number of clients, with median values between 18.5μs and 19μs, with standard deviations less than 1μs and a maximum value of 20μs across runs with different numbers of clients. In contrast, Figure 11 shows the OWD between one pair of VMs when varying the number of concurrent PTPd clients synchronising with the same PTPd master. The variations in the OWD when having up to four clients are not related to the number of clients. However, adding another client leads to an increase in the median latency of 7 μs. Having six or seven clients leads to increases in the OWD by approximately 10 μs. For eight clients, the median OWD is larger by approximately further 7μs. For nine clients, the median OWD is approximately larger by 20μs. In data centers from the two other cloud providers (GCE and Azure) We did not see any noticeable impact when using a maximum of four VM clients. This result is probably due to the instance type used in this experiment (t2.micro which uses a CPU credits mechanism to account for CPU utilization), since the ten hosts testbed is composed of bare metal hosts. The results can also be influenced by competing network traffic in the cloud, whereas in the local testbed there was no other traffic. Given that all measurements were taken using a maximum of three simultaneous VM clients, our measurements were not affected by this behaviour. To mitigate this potential issue for a large number of clients, the current infrastructure of PTPmesh can be extended to perform measurements between VMs independently in pairs. Alternatively, all VMs can be visited in a round robin manner with measurements running for several minutes per VM client to allow the VM client's clock to synchronise with the VM master.

D. Datasets

We collect several datasets whose characteristics are presented in Table V. A trace represents the measurements taken

Trace	#Msgs	Start time	Duration	Avg.(μs)	50th(μs)	99th(μs)	99.9th(μs)	Max(μs)	Std.dev.(μs)	#L.s.
EC2-EUW-1	1204053	2017-11-06 14:43:20	7d00h03m10s	304.87μs	289.57μs	415.18μs	516.5μs	2.69ms	49.71μs	19
EC2-EUW-2	879099	2017-11-06 14:43:22	5d15h12m50s	352.77μs	345.17μs	481.53μs	2.32ms	14.36ms	192.57μs	90
EC2-EUW-3	906750	2017-11-06 14:43:24	5d17h28m36s	352.17μs	350.97μs	459.5μs	616.3μs	3.16ms	50.68μs	48
EC2-USW-1	934978	2017-11-07 12:56:48	5d10h08m30s	259.63μs	256.93μs	335.08μs	486.3μs	1.73ms	33.18μs	7
EC2-USW-2	953109	2017-11-07 12:56:50	5d12h50m25s	279.22μs	275.86μs	361.42μs	474.35μs	1.25ms	29.08μs	12
EC2-USW-3	870190	2017-11-07 12:56:52	5d01h04m16s	287.82μs	283.44μs	363.88μs	429.06μs	686μs	24.15μs	5
GCE-EUW-1	1208861	2017-10-16 14:11:06	7d00h00m00s	138.32μs	97.1μs	1.2ms	2.74ms	7.72ms	223.32μs	237
GCE-EUW-2	1069898	2017-10-16 14:10:59	6d04h45m41s	138.29μs	87.34μs	1.44ms	3.48ms	6.58ms	268.65μs	216
GCE-EUW-3	1208306	2017-10-16 14:10:51	7d00h03m09s	132.78μs	82.97μs	1.38ms	3.6ms	8.83ms	275.73μs	243
GCE-USW-1	1210156	2017-10-16 16:38:32	7d00h02m57s	81.05μs	76.9μs	120.34μs	981.94μs	4.57ms	60.64μs	16
GCE-USW-2	1210507	2017-10-16 16:39:15	7d00h02m14s	72.07μs	71.35μs	92.24μs	119.22μs	531μs	8.65μs	1
GCE-USW-3	1209171	2017-10-16 16:41:33	6d23h59m56s	79.7μs	78.79μs	104.34μs	128.65μs	396μs	8.03μs	1
GCE-USW2-1	42907	2017-04-07 23:58:42	0d05h59m28s	191.65μs	180.66μs	526.84μs	908.27μs	1.21ms	63.04μs	5
Azure-UKW-1	1206919	2017-09-13 15:51:29	6d23h45m10s	441.37μs	447μs	529.27μs	570.62μs	1.38ms	47.4μs	2380
Azure-UKW-2	1160593	2017-09-13 15:51:33	6d17h11m17s	432.95μs	441.3μs	522.88μs	565.55μs	1.01ms	48.7μs	3979
Azure-UKW-3	1208739	2017-09-13 15:51:40	6d23h59m59s	412.59μs	419.62μs	483.48μs	521.42μs	827μs	39.96μs	134
Azure-USW-1	1203300	2017-09-13 15:26:09	6d23h08m41s	313.42μs	315.14μs	357.72μs	379.86μs	549μs	22.78μs	1
Azure-USW-2	1208955	2017-09-13 15:26:11	7d00h00m00s	282.46μs	281.21μs	330.64μs	362.23μs	717μs	15.31μs	3
Azure-USW-3	1208849	2017-09-13 15:26:16	6d23h59m59s	357.83μs	358.46μs	415.68μs	449.11μs	732μs	22.22μs	4
Azure-UKS-N1	108073	2018-02-22 20:11:00	0d16h37m07s	268.49μs	261.31μs	363.22μs	481.65μs	598μs	25.16μs	2
Azure-UKS-A1	96635	2018-02-22 22:18:24	0d13h54m09s	95.7μs	94.54μs	139.79μs	212.75μs	268μs	11.08μs	0
EC2-USE-1	21864606	2018-02-19 17:27:23	0d23h59m48s	181.96μs	172.05μs	291.55μs	411.82μs	2.04ms	30.71μs	139
EC2-USE-2	21864606	2018-02-19 17:27:23	0d23h59m49s	197.33μs	190.08μs	293.74μs	390.46μs	1.77ms	27.14μs	79
EC2-USE-3	21891242	2018-02-19 17:27:23	0d23h59m49s	188.53μs	196.83μs	301.86μs	406.26μs	1.48ms	30.65μs	86
GCE-USW-1	19378393	2017-12-22 22:31:59	1d10h04m00s	65.48μs	64.33μs	89.08μs	106.15μs	451μs	7.14μs	0
GCE-USW-2	21161197	2017-12-22 22:32:17	1d09h17m34s	70.4μs	69.47μs	92.8μs	106.11μs	295μs	8.35μs	0
GCE-USW-3	20854143	2017-12-22 22:32:29	1d07h52m56s	58.47μs	57.94μs	76.02μs	86.28μs	286μs	6.33μs	0
Azure-UKS-1	20164042	2018-02-17 22:12:21	0d23h59m48s	286.58μs	269.34μs	684.45μs	884.02μs	1.22ms	74μs	2235
Azure-UKS-2	21111652	2018-02-17 22:12:21	0d23h59m48s	271.41μs	249.55μs	724.43μs	907.24μs	1.37ms	84.65μs	4747
Azure-UKS-3	17427943	2018-02-17 22:12:21	0d23h59m48s	340.02μs	322.17μs	760.13μs	949.7μs	1.29ms	81.7μs	2793
Azure-UKS-A2	5043445	2018-02-23 12:47:02	0d05h54m20s	83.23μs	82.28μs	118.92μs	178.79μs	459μs	9.11μs	0

TABLE V: Traces collected in data centers across the world from three cloud providers. The last column represents the number of latency spikes (l.s.) ($> 500\mu$s) observed throughout the trace.

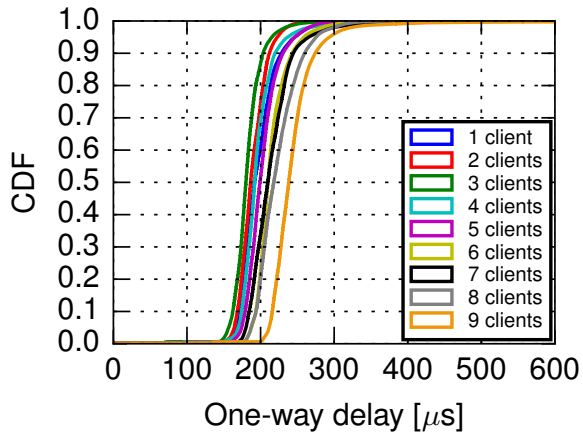


Fig. 11: Varying the number of PTPd clients that synchronise with the PTPd master in EC2-USE.

between two VMs (master and client). The trace is the log of a PTPd client running in a VM. We list the start time and duration of the trace. Each trace is identified by the assigned name of the data center and a number. For the first part of the table, the low message frequency was used (see Section V-A), while in the second part of the table, the high message frequency was used (see Section V-B). These traces are indicative for the temporal perspective of network conditions in data centers, as they have been captured for periods of up to a week. The spatial perspective is limited, since we use a maximum of three VM PTPd clients at the same time, hence we do not capture the full scale of conditions in the studied data center. All datasets, except one, contain measurements taken between VMs that are located within the same data center (zone). One dataset (GCE-USW-2) contains measurements taken between VMs that are located in different data centers (zones) within the same region (§IV-B).

V. One-way delay (OWD) measurements

In the first instance, we set the number of Sync and Delay Request messages to 1 per second, since this is the default value configured in PTPd, which will be named in the rest of the paper as the low message frequency. We run a full week of measurements in six data centers using the low message frequency. Additionally, we perform measurements in three data centers for one day using a higher message frequency of 2^7 messages per second, which will be named in the rest of the paper the high message frequency. The challenge with using a higher message frequency is, on one hand, the increased CPU utilisation and network bandwidth at the end-host, while on the other, the amount of data collected for which additional storage is needed if measurements are performed for an extended period of time. The advantages of using a higher frequency are better OWD accuracy (§IV-C1) and detection of possible network congestion events with a higher resolution. Regardless of the message frequency used, the OWD values offered by PTPmesh can serve as reference for normal network conditions

and can be used to detect anomalies. Additionally, as shown in Section III-A, an initial period of five minutes should be used to wait for the clocks to get synchronized before relying on the OWD results.

A. Low message frequency measurements

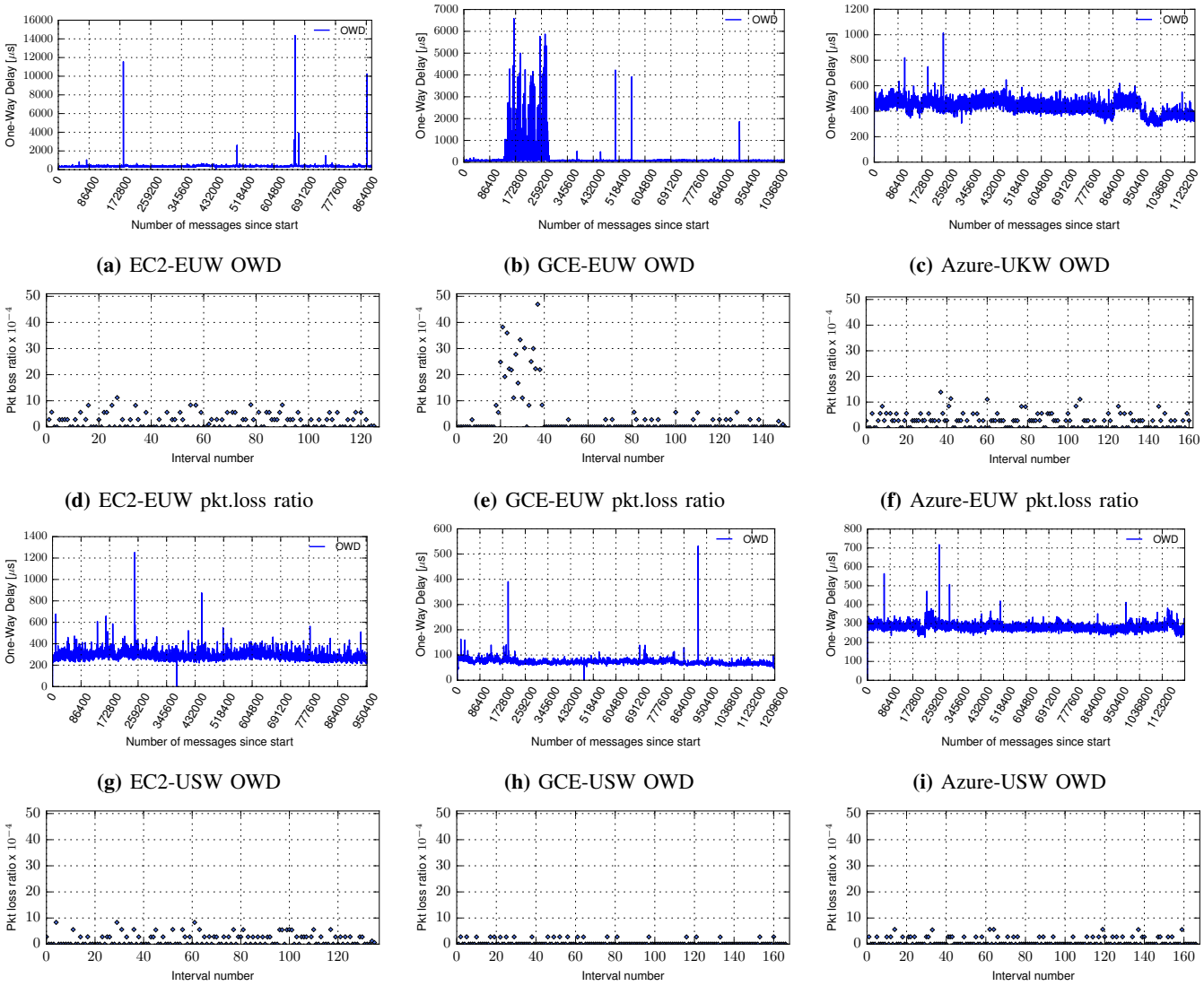
Latency magnitude. Table V lists the average, median, $99^{\text{th}}$and 99.9^{th}percentiles, maximum, standard deviation for the OWD values, and the number of latency spikes (a sudden increase in latency to values over 500μs) for the trace. OWD values are higher in the EU data centers than the US data centers for EC2 and Azure. The GCE-EUW data center OWD values are similar to the ones in the GCE-USW data center, the difference coming from the extended period of increased latency that can be seen in Figure 12b. Most of the traces have maximum observed OWD values in the order of milliseconds.

In Figure 12a, in the EC2-EUW-2 trace multiple latency spikes can be observed, with a maximum of 14.364ms. In the GCE-EUW traces, the OWD values are less or slightly higher than 100μs up to the 90^{th}percentile, with a maximum 99^{th}percentile of 1.44ms and maximum 99.9^{th}percentile of 3.6ms amongst the three VM pairs. In contrast, for GCE-USW data center, the maximum $99^{\text{th}}$is 120.34μs, and only in the case of the trace between VM1 and VM2 the 99.9^{th}percentile is higher, 981.945μs, compared to the traces for the two other VM pairs. The traces captured in the GCE-EUW data center stand out in comparison to the other traces collected, since they contain major disruptions for latency values over a prolonged period, accompanied by a high packet loss ratio. Between 2017-10-17 09:25 and 2017-10-18 05:16 the OWD reported varied greatly, reaching a maximum value of 8.83ms, with a significant part of the latency spikes of over 500μs taking place during this interval. These millisecond-scale latencies indicate switch queueing and packet loss [23]. These events can be noticed for all three VM pairs, which can lead to the hypothesis that these events were data center-wide, or that the VMs were placed within the same rack or on the same host. While the median latencies within the same data center are between 71 and 97 μs, the median latency between two data centers in the same region is 180 μs (GCE-USW2-1), almost double compared to the one ones within a data center.

The Azure-UKW data center traces show a decrease of the OWD values of approximately 100μs towards the end of the trace (Figure 12c), which corresponds to the network traffic for Sunday. The last part of the trace was captured on Monday, showing an increase in the OWD values back to the values before Sunday, except for the VM1-VM3 pair. In the case of the Azure-USW data center (Figure 12i) in the second day of measurements (after 172800 messages), a sudden decrease by approximately 50μs in OWD can be noticed for all three pairs for a period of time, followed by an increase for the OWD to values higher by approximately 50μs than the ones before the dip. It is interesting to see that the traces share similar characteristics for certain changes in the OWD values, meaning that the events were data center-wide or that the VMs were placed within the same rack or the same host. We also perform experiments using more powerful machines in Azure-UKS (Azure-UKS-N1), the median latency is in similar ranges to the ones obtained using slower machines. We additionally perform experiments with VMs with the Azure Accelerated Networking feature [40] enabled, which removes most of the software-based networking stack into FPGA-based smartNICs, and found that the one-way delay reported is significantly lower than the other reported values, with median values of 94.54μs with low message frequency (Azure-UKS-A1) and 82.28μs with high message frequency (Azure-UKS-A2). EC2 offers a similar option using SR-IOV [46], but we have not performed measurements using it for this study.

Latency variance. An important aspect of network latency is latency variance [47], as it can cause a decrease in application performance. If the variance is low, then the application performance will be determined by the median latency observed, essentially reducing the problem to improving the static component of latency in data centers. To this end, we compute the standard deviations of the OWD measurements over different intervals of time. We compute histograms for the standard deviations of the OWD for intervals of 1 minute, 10 minutes and 1 hour, binned in bins of size 1. The distributions for the standard deviations OWD are skewed towards small values, with a few values that are larger than the rest. The histograms for the two AWS EC2 data centers are similar. When looking at periods of 1 minute, the standard deviations fall mostly between 0μs and 10μs (medians 5.44μs and 4.59μs). When looking at periods of 10 minutes, the standard deviations fall between 10μs and 20μs (medians 15.34μs and 12.42μs), while when looking at periods of 1 hour, the standard deviations fall between 10μs and 30μs (medians 22.67μs and 18.31μs). The histograms for the two GCE data centers are similar, but they are different from the two other cloud providers, in that the standard deviations of the OWD values are smaller. For the GCE-USW trace, for 1 minute intervals, most of the values are between 0μs and 1μs (median 0.759μs). When looking at periods of 10 minutes, the standard deviations are between 0μs and 5μs (median 2.84μs). For 1 hour intervals, most of the values are between 1μs and 10μs (median 4.63μs). The median values for the GCE-EUW trace are slightly higher, due to the increase in the OWD for a long period of time (1.5 days). On the other hand, there are differences between the two Azure data centers. In the case of the Azure-UKW trace, for 1 minute intervals, most of the values are between 1μs and 15μs (median 7.42μs). When looking at periods of 10 minutes, the standard deviations are between 10μs and 20μs (median 16.6μs). For 1 hour intervals, most of the values are between 10μs and 30μs (median 20.29μs). In the case of the Azure-USW data center, the values are slightly lower. For 1 minute intervals, most of the values are between 0μs and 10μs (median 4.49μs). When looking at periods of 10 minutes, the standard deviations are between 5μs and 15μs (median 9.63μs). For 1 hour intervals, most of the values are between 10μs and 20μs (median 12.17μs).

The results show that the OWD in GCE has the lowest variance. EC2 and Azure are similar, with more variance seen for EC2. When enabling [40], the Azure latency variance profile becomes similar to the GCE one. Having less variance for OWD is better, since tail latencies can lead to a decrease in application performance [47].



(a) EC2-EUW OWD **(b)** GCE-EUW OWD **(c)** Azure-UKW OWD

(d) EC2-EUW pkt.loss ratio **(e)** GCE-EUW pkt.loss ratio **(f)** Azure-EUW pkt.loss ratio

(g) EC2-USW OWD **(h)** GCE-USW OWD **(i)** Azure-USW OWD

(j) EC2-USW pkt.loss ratio **(k)** GCE-USW pkt.loss ratio **(l)** Azure-USW pkt.loss ratio

Fig. 12: OWD and packet loss ratios over 1-hour intervals between VM1-VM3 in EU and US data centers over one week.

B. High message frequency measurements

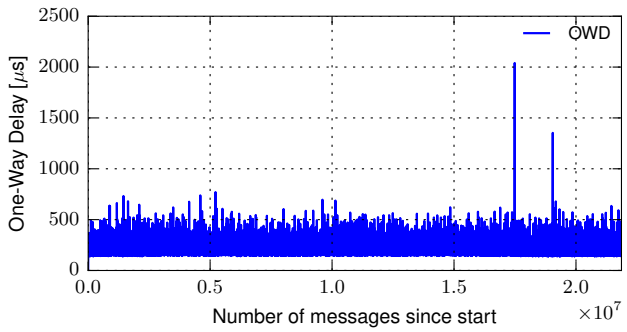
Latency magnitude. The OWD values measured using high message frequency are lower than the ones measured using the low message frequency (§IV-C1). The EC2-USE OWD median values are between 172μs and 196μs (Figure 13a). The GCE-USW OWD values have medians between 58μs and 69μs (Figure 13b). In GCE, the low message frequency measurements may have been redirected through switch gateways (due to the low throughput of the measurements run, less than 20kbps), whereas the high message frequency ones may have been sent host-to-host [39]. The median OWD values for Azure-UKS are between 271μs and 340μs (Figure 14). The three traces are correlated, displaying periods of increased latency at the same time and having the same shape.

Latency variance. In the case of GCE-USW, the latency variance profile is similar to the one obtained using the low message frequency. The EC2-USE latency variance profile is similar to the EC2-USW one, and the Azure-UKS one is similar to the Azure-UKW one, even if EC2-USW and Azure-UKW latency variance profiles have been obtained using the low message frequency.

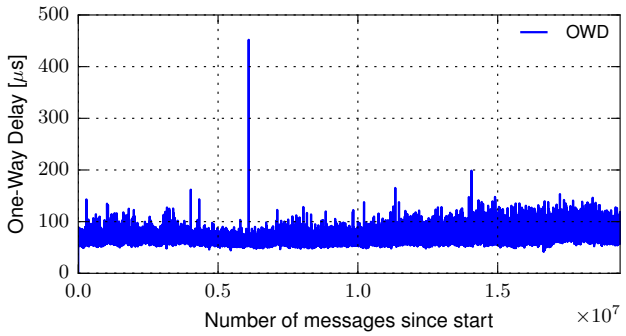
C. High value OWD events timescale

After analysing the general characteristics of the traces, we take a closer look at the timescale of high OWD events, as these are important in the context of cluster scheduling of latency-sensitive applications. If the OWD is stable, then the VM placement decision will have a lasting effect throughout the execution of the application. On the other hand, admitting more applications into the data center can lead to increased network utilisation, and hence increased network latency. If the OWD is not stable, it might be better for certain applications to be preempted and migrated to a different placement.

Long timescale events are considerable changes in latency during several hours or days. These types of events are evident in the week-long Azure traces, where the latency

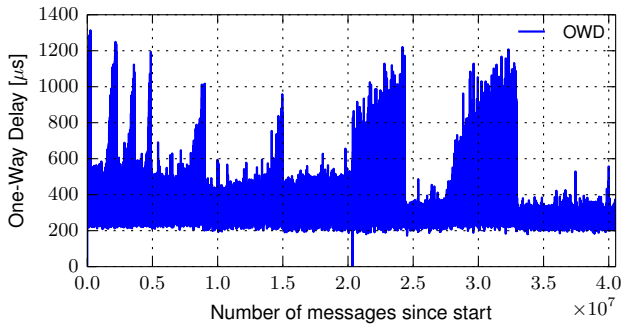


(a) VM1-VM2 timeline EC2-USE

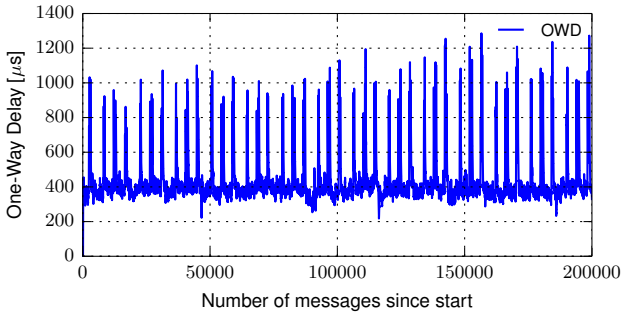


(b) VM1-VM2 timeline GCE-USW

Fig. 13: Measured OWD between VM1 and VM2 using the high message frequency.



(a) Timeline



(b) First 15 minutes of timeline

Fig. 14: Measured OWD between VM1 and VM2 in Azure-UKS using the high message frequency.

decreases during the weekend. Similarly, the GCE EU traces display significant increases in latency for more than one day. Also, in the Azure-KS data center, after restarting the VMs, we consistently got substantial latencies (median 1.391ms) compared to previous values (median 191.486μs), that we kept on measuring even after several VM restarts, and across almost one month of measurements. The first time we observed these large latency values was on the 29th of December 2017, and the last time we performed measurements in this data center was 23rd of January 2018. In this case, it might be better to migrate the application to a different data center.

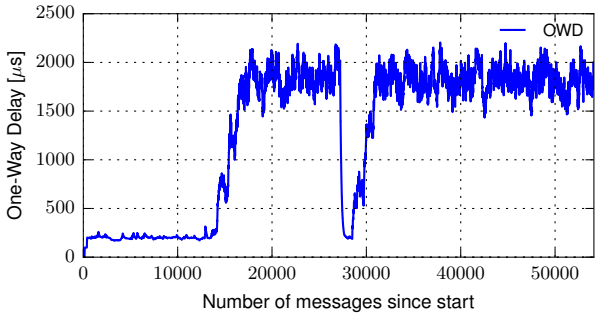


Fig. 15: Measured OWD between VM1 and VM10 in EC2-USE data center.

Short timescale events are transient latency spikes. The difference between the measured median latency and the maximum latency observed during the spike should be substantial (*e.g.*, more than 500μs). For example, while performing the EC2-USE measurements, the latency has suddenly increased substantially from median 200μs to median 1.75ms, as seen in Figure 15. It can be noticed that the latency values return for a brief period of time (2s, with high message frequency) to the previous values, but then the latency increases again. Similarly, the Azure-UKS traces (Figure 14) display several short latency spikes. In this case, if the OWD is not stable and suffers from frequent changes, it may be better for the application to be migrated to a different placement.

VI. Packet Loss Ratio Measurements

We investigate packet losses in six data centers over a week for each of the VM pair. we log the number of *Delay Request* and *Delay Response* messages exchanged between the clients and the master in PTPmesh for the measurements with the low message frequency. Using the metric we defined for computing packet loss ratio in Section III-E, we compute the packet loss ratio over intervals of 1 hour and 1 day over one week. In Table VI, we show the minimum, average, median, maximum and standard deviation for all the 1-hour and 1-day intervals across all pairs. Interestingly, all EU data centers have higher packet loss ratios than the US data centers across all cloud

providers. Figure 12 presents timelines over one week for packet loss ratios computed over 1 hour intervals for one VM pair in EU and US data centers, respectively. The ratios computed depend on the message frequency, but they can serve as baseline for normal conditions, and to determine anomalies when deviating from these baseline values.

In general, the packet loss ratios have low values for all data centers, with most of the 1-hour intervals having no loss or having 1-4 messages lost per hour (out of 3600), which is at most approximately 11.1×10^{-4}. For AWS EC2, the number of messages lost per hour is at most four (Figure 12d and Figure 12j), with more losses observed in the EU data center. High packet loss values of up to 46.96×10^{-4} appear in the first part of the GCE-EUW data center traces (Figure 12e), and significant increases in network latency can be seen in Figure 12b, but later in the trace the values are normal, with at most three messages lost per hour. In the GCE-USW data center (Figure 12k), the number of messages lost per hour is at most two, being the data center with the smallest packet loss ratio. For Azure-EUW (Figure 12f), slightly higher packet loss ratios can be observed, while for Azure-USW (Figure 12l) the maximum number of messages lost per hour is four.

These results lead to the hypothesis that the EU data centers use older hardware, or they run at higher network utilisation.

VII. Path symmetry

PTPd reports the *master-to-slave* and *slave-to-master* measured delays. These two measurements can be used to determine if the paths from the master to the slave and from the slave to the master are symmetric, but with some caveats, as these two metrics incorporate the offset between the two slave and master clocks and possible congestion effects. We am interested in determining whether the simplifying assumption that the OWD can be computed as half of the measured RTT holds true in data centers. Note here that the values of the two delays can be negative, due to the differences between the master and slave clocks.

Figure 16 shows the master-to-slave and slave-to-master delays for the VM1-VM3 pair in six data centers. The plots for the other pairs and data centers are similar. Based on the collected data, EC2 and GCE forward and reverse paths are symmetrical, while the paths in Azure are not. Azure data centers (Figures 16f and 16c) display significant differences between the master-to-slave and slave-to master delay CDFs (different curve shapes), leading to the conclusion that the forward and reverse paths between the VMs are not symmetrical. In the case of the GCE-EUW trace (Figure 16b), the long congestion period is reflected in a vertical translation of the master-to-slave and slave-to-master delay CDFs, but this does not mean that the paths are asymmetrical.

VIII. Discussion

PTPmesh offers end-to-end measurements, including the intermediate virtualisation layer. The one-way delay latency values offer insights with respect to end-host overhead, in-network congestion and data center network architecture. When combining these measurement results in data centers and the virtualisation overhead measurements from Section III-D with the network latency contributions percentages from [8], we arrive at the same conclusion as prior research: the end-host, with the hypervisor, is a significant contributor to the overall measured latency from within the VM. The other significant contributor is network queueing at switches. When looking at the network latency contributors in [8], it can be noticed that switching in the data center fat tree topology takes up almost 75%, which is approximately 15μs in this analysis, while the rest is taken up by NICs and fibre length, with an estimate of 20μs. This analysis represents baseline contributions. On top of this, the hypervisor's overhead can be added, which based on my measurements from Section III-D is 200μs when using a low message frequency, and 190μs when using a high message frequency, giving a median OWD baseline of 220μs and 210μs, respectively. This back-of-the-envelope calculation shows that the remaining latency may come from in-network congestion, traffic bursts from other colocated VMs, transparent VM live migration, when it is observed for short periods, or from sustained increased network utilisation (whose cause may be bulk network transfers across the data center, competing traffic from other colocated VMs, cluster drains), when it is observed over longer periods of time. Smaller values than this baseline may mean that the OS is bypassed [39], [40], or that the VMs may be colocated on the same host, or that there is a shorter network path between VMs.

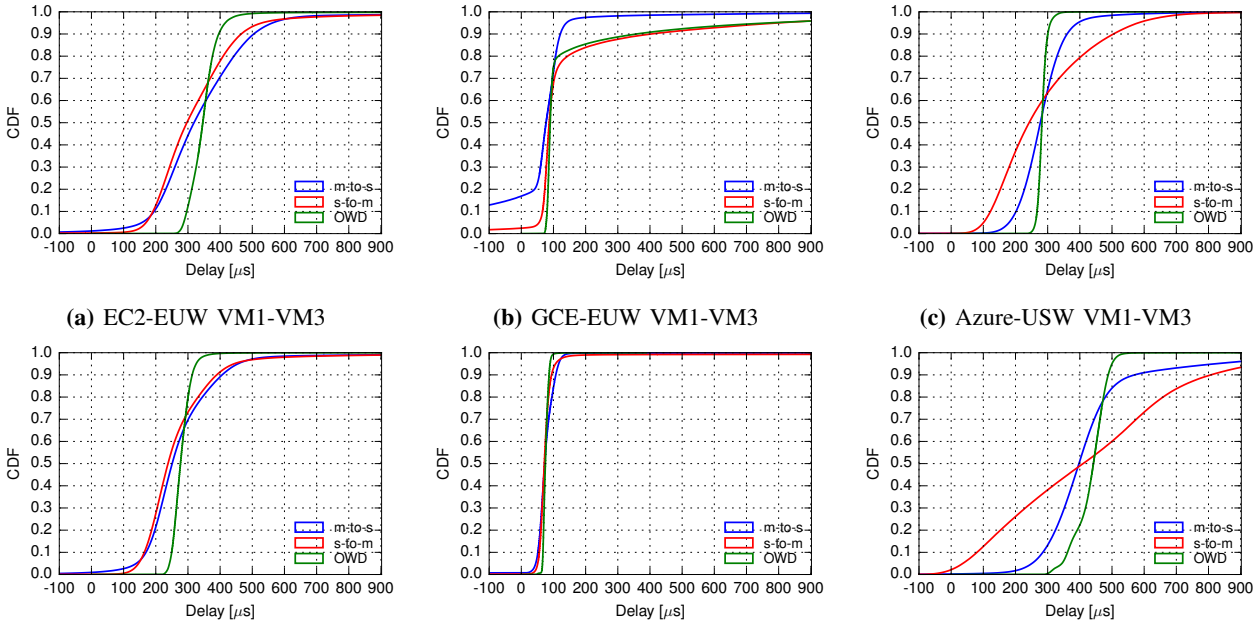
IX. Limitations

End-to-end measurements The network latency measurements presented in this chapter represent end-to-end measurements from within the VMs, which include the virtualisation layer. Cloud tenants usually do not have access to advanced features of the underlying hardware, *e.g.*, hardware timestamping. If access were provided, the precision of the network latency measurements would be improved due to the removal of the end-host network stack latency contribution from the measured network latency [8], [23]. On the other hand, end-to-end measurements offer a more accurate value of the latency that the application experiences, encompassing also the end-host network stack latency contribution. Determining the cause of latency spikes (end-host issue or network fault) may be difficult without access to the internal cloud infrastructure, and even then, finding the root cause may still prove tedious [23].

Spatial analysis The measurement study conducted in this paper does not cover the spatial analysis of the data centers. However, our results show that, in some cases, small groups of VMs are clustered together, observing the same network conditions. This means that, to measure the data center network across the core and aggregation switches, one would have to rent a substantial number of VMs to ensure they are not placed within the same rack or on the same machine. On the other hand, this might not hold true for all cloud providers. For example, Microsoft Azure's tenant VMs are placed randomly within a cluster, or they can even be spread across data centers in a region [23].

Data center	1 hour					1 day				
	min	*average*	*median*	*max*	*stddev*	*min*	*average*	*median*	*max*	*stddev*
AWS-EUW	0.0	2.028	0.161	11.198	2.511	0.886	1.737	1.679	2.548	0.461
AWS-USW	0.0	1.06	0.0	8.324	1.74	0.116	0.779	0.777	1.617	0.373
GCE-EUW	0.0	2.96	0.0	46.961	7.081	0.109	2.95	0.463	14.082	4.56
GCE-USW	0.0	0.476	0.0	8.373	1.158	0.0	0.154	0.0	0.81	0.256
Azure-UKW	0.0	2.45	2.758	16.533	2.806	1.273	2.405	2.197	3.707	0.633
Azure-USW	0.0	1.244	0.0	11.123	1.9	0.116	0.843	0.753	1.618	0.417

TABLE VI: Packet loss ratio $\times 10^{-4}$ over one week.



(a) EC2-EUW VM1-VM3 **(b)** GCE-EUW VM1-VM3 **(c)** Azure-USW VM1-VM3

(d) EC2-USW VM1-VM3 **(e)** GCE-USW VM1-VM3 **(f)** Azure-EUW VM1-VM3

Fig. 16: CDF for the master-to-slave (m-to-s) delay, slave-to-master (s-to-m) delay and OWD in different data centers.

Scalability PTPmesh's design implies that O(n^2) measurements are taken for n VMs. This can quickly become a bottleneck both at the end-host (in terms of CPU resource consumption) and in the network (in terms of network bandwidth taken up by the messages exchanged between VMs). Thus, it is important to choose an appropriate message frequency, as discussed in Section IV-C1. However, even if a tenant has a small number of VMs, as it is often the case [48], the combined resource usage of PTPmesh across different tenant networks can be a burden to the data center infrastructure. To mitigate part of this issue, the PTPd master could be consolidated to run in the hypervisor or in the virtual switch, similar to VNET-Pingmesh [23]. For example, if the hypervisor used is Xen [49], the PTPd master should run in `Dom0`. This design decision has some tradeoffs. While the CPU and network resources used are smaller than in the current design, the network latency measured does not express the latency experienced by the user applications within a VM [8], [23].

X. Conclusion

We showed that PTPmesh provides a majority of the features needed by a data center network monitoring system. PTPmesh is a lightweight data center network monitoring tool is lightweight, easy to configure and deploy, highly available, and offers sufficient coverage of the network. PTPmesh uses PTP measurements to estimate one-way delay and packet loss ratio. The number of probes sent is configurable, it can provide continuous measurements, and does not have significant overhead. Furthermore, it is easy to deploy within VMs by tenants themselves. PTPmesh can infer network conditions for tenant deployments in the cloud. It can keep track of the latency within the data center and inter-data center, and can help in detecting network congestion and packet loss.

References

[1] D. A. Popescu and A. W. Moore, "PTPmesh: Data Center Network Latency Measurements Using PTP," in *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sept 2017, pp. 73–79.

[2] D. A. Popescu and A. W. Moore, "A First Look at Data Center Network Condition Through The Eyes of PTPmesh," in *2018 Network Traffic Measurement and Analysis Conf. (TMA)*, June 2018, pp. 1–8.

[3] D. A. Popescu and A. W. Moore, "Dataset and Reproduction Environment for paper "A First Look At Network Conditions Through The Eyes of PTPmesh"," https://doi.org/10.17863/CAM.23126, June 2018.

[4] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conf.*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74.

[5] R. Kapoor *et al.*, "Chronos: Predictable low latency for data center applications," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 9:1–9:14.

[6] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conf. on Operating Systems Design and Implementation*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283.

[7] B. Atikoglu *et al.*, "Workload analysis of a large-scale key-value store," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conf. on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '12. New York, NY, USA: ACM, 2012, pp. 53–64.

[8] N. Zilberman *et al.*, "Where has my time gone?" in *Proceedings of the 18th International Conf. on Passive and Active Measurement*, ser. PAM 2017, 2017.

[9] D. A. Popescu, N. Zilberman, and A. W. Moore, "Characterizing the impact of network latency on cloud-based applications' performance," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-914, Nov. 2017.

[10] L. Barroso *et al.*, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, no. 4, pp. 48–54, Mar. 2017.

[11] J. C. Mogul and R. R. Kompella, "Inferring the network latency requirements of cloud tenants," in *Proceedings of the 15th USENIX Conf. on Hot Topics in Operating Systems*, ser. HOTOS'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 24–24.

[12] C. Guo *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceedings of the 2015 ACM Conf. on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 139–152.

[13] 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 December 2018].

[14] PTPd, "PTP daemon," https://github.com/ptpd/ptpd, 2018, [Online; accessed December 2018].

[15] IEEE, "IEEE 1588-2008 Precision Time Protocol," https://www.nist.gov/el/intelligent-systems-division-73500/introduction-ieee-1588, 2008, [Online; accessed December 2018].

[16] K. Correll and N. Barendt, "Design considerations for software only implementations of the ieee 1588 precision time protocol," in *In Conf. on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2006.

[17] Solarflare, "Solarflare PTP Adapters," http://www.solarflare.com/ptp-adapters, [Online; accessed December 2018].

[18] 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 Conf. on High-Performance Clustered Computing. Intel GmbH*, 2008.

[19] Timekeeper, "TimeKeeper," http://www.fsmlabs.com/timekeeper, [Online; accessed December 2018].

[20] S. D. Strowes, "Passively Measuring TCP Round-trip Times," https://queue.acm.org/detail.cfm?id=2539132, 2013.

[21] 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 December 2018].

[22] Y. Zhu *et al.*, "Packet-level telemetry in large datacenter networks," in *Proceedings of the 2015 ACM Conf. on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 479–491.

[23] A. Roy *et al.*, "Cloud datacenter sdn monitoring: Experiences and challenges," in *Proceedings of the Internet Measurement Conf. 2018*, ser. IMC '18. New York, NY, USA: ACM, 2018, pp. 464–470.

[24] C. Yu *et al.*, *Software-Defined Latency Monitoring in Data Center Networks*. Cham: Springer International Publishing, 2015, pp. 360–372.

[25] D. A. Popescu and A. W. Moore, "Omniscient: Towards realizing near real-time data center network traffic maps," in *Proceedings of the ACM International Conf. on emerging Networking EXperiments and Technologies*, ser. CoNEXT Student Workshop 15. New York, NY, USA: ACM, 2015.

[26] P. Tammana, R. Agarwal, and M. Lee, "Cherrypick: Tracing packet trajectory in software-defined datacenter networks," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. New York, NY, USA: ACM, 2015, pp. 23:1–23:7.

[27] Y. Peng *et al.*, "detector: a topology-aware monitoring system for data center networks," in *2017 USENIX Annual Technical Conf. (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 55–68.

[28] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conf. on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 99–110.

[29] M. Hira and L. Wobker, "Improving Network Monitoring and Management with Programmable Data Planes," http://p4.org/p4/inband-network-telemetry/, 2016, [Online; accessed December 2018].

[30] Y. Li *et al.*, "Lossradar: Fast detection of lost packets in data center networks," in *Proceedings of the 12th International on Conf. on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: ACM, 2016, pp. 481–495.

[31] S. Narayana *et al.*, "Language-directed hardware design for network performance monitoring," in *Proceedings of the Conf. of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 85–98.

[32] R. Chaiken *et al.*, "Scope: Easy and efficient parallel processing of massive data sets," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1265–1276, Aug. 2008.

[33] B. Arzani *et al.*, "007: Democratically finding the cause of packet drops," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, 2018, pp. 419–435.

[34] P. Tammana, R. Agarwal, and M. Lee, "Simplifying datacenter network debugging with pathdump," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 233–248.

[35] C. Pelsser *et al.*, "From paris to tokyo: On the suitability of ping to measure latency," in *Proceedings of the 2013 Conf. on Internet Measurement Conf.*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 427–432.

[36] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *Proceedings of the 29th Conf. on Information Communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 1163–1171.

[37] Memcached, "Memcached," https://memcached.org/, 2018, [Online; accessed December 2018].

[38] libMemcached Project, "memaslap - Load testing and benchmarking a server," http://docs.libmemcached.org/bin/memaslap.html, [Online; accessed December 2018].

[39] M. Dalton *et al.*, "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, 2018, pp. 373–387.

[40] D. Firestone *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, 2018, pp. 51–66.

[41] S. Hemminger, "NetEm - Network Emulator," http://man7.org/linux/man-pages/man8/tc-netem.8.html, [Online; accessed December 2018].

[42] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conf. on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74.

[43] 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 December 2018].

[44] A. Roy *et al.*, "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.

[45] R. Neugebauer *et al.*, "Understanding pcie performance for end host networking," in *Proceedings of the 2018 Conf. of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: ACM, 2018, pp. 327–341.

[46] Amazon, "Amazon Enhanced Networking," https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html, 2018, [Online; accessed December 2018].

[47] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[48] E. Cortez *et al.*, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 153–167.

[49] P. Barham *et al.*, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 164–177.