Globally Synchronized Time via Datacenter Networks

Ki Suh Lee, Han Wang, Vishal Shrivastav, Hakim Weatherspoon
Computer Science Department
Cornell University
kslee,hwang,vishal,hweather@cs.cornell.edu

ABSTRACT

Synchronized time is critical to distributed systems and network applications in a datacenter network. Unfortunately, many clock synchronization protocols in datacenter networks such as NTP and PTP are fundamentally limited by the characteristics of packet switching networks. In particular, network jitter, packet buffering and scheduling in switches, and network stack overheads add non-deterministic variances to the round trip time, which must be accurately measured to synchronize clocks precisely.

In this paper, we present the Datacenter Time Protocol (DTP), a clock synchronization protocol that does not use packets at all, but is able to achieve nanosecond precision. In essence, DTP uses the physical layer of network devices to implement a decentralized clock synchronization protocol. By doing so, DTP eliminates most non-deterministic elements in clock synchronization protocols. Further, DTP uses control messages in the physical layer for communicating hundreds of thousands of protocol messages without interfering with higher layer packets. Thus, DTP has virtually zero overhead since it does not add load at layers 2 or higher at all. It does require replacing network devices, which can be done incrementally. We demonstrate that the precision provided by DTP in hardware is bounded by 25.6 nanoseconds for directly connected nodes, 153.6 nanoseconds for a datacenter with six hops, and in general, is bounded by $4TD$ where D is the longest distance between any two servers in a network in terms of number of hops and T is the period of the fastest clock ($\approx 6.4ns$). Moreover, in software, a DTP daemon can access the DTP clock with usually better than $4T$ ($\approx 25.6ns$) precision. As a result, the end-to-end precision can be better than $4TD + 8T$ nanoseconds. By contrast, the precision of the state of the art protocol (PTP) is not bounded: The precision is hundreds of nanoseconds when a network is idle and can decrease to hundreds of microseconds when a network is heavily congested.

SIGCOMM '16 August 22-26, 2016, Florianopolis , Brazil

ACM ISBN 978-1-4503-4193-6/16/08.
DOI: http://dx.doi.org/10.1145/2934872.2934885

CCS Concepts

•Networks → Time synchronization protocols; Data center networks; •Hardware → *Networking hardware;*

1. INTRODUCTION

Synchronized clocks are essential for many network and distributed applications. Importantly, an order of magnitude improvement in synchronized precision can improve performance. For instance, if no clock differs by more than 100 nanoseconds (ns) compared to 1 microsecond (us), one-way delay (OWD), which is an important metric for both network monitoring and research, can be measured precisely due to the tight synchronization. Synchronized clocks with 100 ns precision allow packet level scheduling of minimum sized packets at a finer granularity, which can minimize congestion in rack-scale systems [23] and in datacenter networks [47]. Moreover, taking a snapshot of forwarding tables in a network requires synchronized clocks [53]. In software-defined networks (SDN), synchronized clocks with microsecond level of precision can be used for coordinated network updates with less packet loss [42] and for real-time synchronous data streams [26]. In distributed systems, consensus protocols like Spanner can increase throughput with tighter synchronization precision bounds on TrueTime [22]. As the speeds of networks continue to increase, the demand for precisely synchronized clocks at nanosecond scale is necessary.

Synchronizing clocks with nanosecond level precision is a difficult problem. It is challenging due to the problem of measuring round trip times (RTT) accurately, which many clock synchronization protocols use to compute the time difference between a timeserver and a client. RTTs are prone to variation due to characteristics of packet switching networks: Network jitter, packet buffering and scheduling, asymmetric paths, and network stack overhead. As a result, any protocol that relies on RTTs must carefully handle measurement errors.

In this paper, we present the Datacenter Time Protocol (DTP) which provides nanosecond precision in hardware and tens of nanosecond precision in software, and at virtually no cost to the datacenter network (i.e. no protocol message overhead). DTP achieves better precision than other

protocols and provides strong bounds on precision: By running in the physical layer of a network stack, it eliminates non-determinism from measuring RTTs and it introduces zero Ethernet packets on the network. It is decentralized and synchronizes clocks of every network device in a network including network interfaces and switches.

In practice, in a 10 Gbps network, DTP achieves a bounded precision of 25.6 nanoseconds between any directly connected nodes, and 153.6 nanoseconds within an entire datacenter network with six hops at most between *any* two nodes, which is the longest distance in a Fat-tree [18] (i.e. no two nodes [clocks] will differ by more than 153.6 nanoseconds). In software, a DTP daemon can access its DTP clock with usually better than $4T$ nanosecond precision resulting in an end-to-end precision better than $4TD + 8T$ nanoseconds where D is the longest distance between any two servers in a network in terms of number of hops and T is the period of the fastest clock ($\approx$ 6.4ns). DTP's approach applies to full-duplex Ethernet standards such as 1, 10, 40, 100 Gigabit Ethernet (See Sections 2.5 and 7). It does require replacing network devices to support DTP running in the physical layer of the network. But, it can be incrementally deployed via DTP-enabled racks and switches. Further, incrementally deployed DTP-enabled racks and switches can work together and enhance other synchronization protocols such as Precise Time Protocol (PTP) [8] and Global Positioning System (GPS) by distributing time with bounded nanosecond precision within a rack or set of racks without any load on the network.

The contributions of our work are as follows:

- We present DTP that provides clock synchronization at nanosecond resolution with bounded precision in hardware and tens of nanosecond precision in software.
- We demonstrate that DTP works in practice. DTP can synchronize all devices in a datacenter network.
- We evaluate PTP as a comparison. PTP does not provide bounded precision and is affected by configuration, implementation, and network characteristics such as load and congestion.

2. TOWARDS PRECISE CLOCK SYNCHRONIZATION

In this paper, we show how to improve the precision and efficiency of clock synchronization by running a protocol in the *physical layer* of the network protocol stack. In fact, two machines physically connected by an Ethernet link are already synchronized: Synchronization is required to reliably transmit and receive bitstreams. The question, then, is how to use the bit-level synchronization of the physical layer to synchronize clocks of distributed systems in a datacenter, and how to scale the number of synchronized machines from two to a large number of machines in a datacenter? In this section, we state the problem of clock synchronization, why it is hard to achieve better precision and scalability with current approaches, and how synchronizing clocks in the physical layer can improve upon the state-of-the-art.

2.1 Terminology

A *clock* c of a process p[1] is a function that returns a local clock counter given a real time t, i.e. $c_p(t) =$ local clock counter. Note that a clock is a discrete function that returns an integer, which we call clock *counter* throughout the paper. A clock changes its counter at every clock *cycle* (or *tick*). If clocks c_i for all i are synchronized, they will satisfy

$$\forall i, j, t \; |c_i(t) - c_j(t)| \leq \epsilon \tag{1}$$

where ϵ is the level of *precision* to which clocks are synchronized. *Accuracy* refers to how close clock counters are to true time [48].

Each clock is driven by a quartz oscillator, which oscillates at a given frequency. *Oscillators* with the same nominal frequency may run at different rates due to frequency variations caused by external factors such as temperature. As a result, clocks that have been previously synchronized will have clock counters that differ more and more as time progresses. The difference between two clock counters is called the *offset*, which tends to increase over time, if not resynchronized. Therefore, the goal of clock synchronization is to periodically adjust offsets between clocks (offset synchronization) and/or frequencies of clocks so that they remain close to each other [48].

If a process attempts to synchronize its clock to true time by accessing an external clock source such as an atomic clock, or a satellite, it is called *external synchronization*. If a process attempts to synchronize with another (peer) process with or without regard to true time, it is called *internal synchronization*. Thus, externally synchronized clocks are also internally synchronized, but not vice versa [24]. In many cases, monotonically increasing and internally synchronized clocks are sufficient. For example, measuring one-way delay and processing time or ordering global events do not need true time. As a result, in this paper, we focus on how to achieve internal synchronization: We achieve clock synchronization of all clocks in a datacenter with high precision; however, their clock counters are not synchronized to an external source. We briefly discuss how to extend the protocol to support external synchronization in Section 5.

2.2 Clock Synchronization

Regardless of whether the goal is to achieve internal or external synchronization, the common mechanism of synchronizing two clocks is similar across different algorithms and protocols: A process *reads* a different process's current clock counter and computes an offset, adjusting its own clock frequency or clock counter by the offset.

In more detail, a process p sends a time request message with its current *local* clock counter (t_a in Figure 1) to a process q (q reads p's clock). Then, process q responds with a time response message with its local clock counter and p's original clock counter (p reads q's clock). Next, process p computes the offset between its local clock counter and the

[1]We will use the term *process* to denote not only a process running on a processor but also any system entities that can access a clock, e.g. a network interface card.

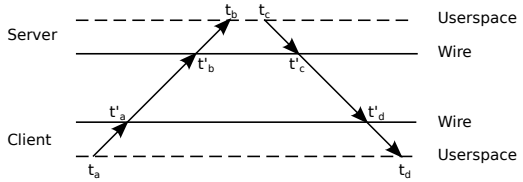


Figure 1: Common approach to measure offset and RTT.

remote clock counter (q) and round trip time (RTT) of the messages upon receiving the response at time t_d. Finally, p adjusts its clock counter or the rate of its clock to remain close to q's clock.

In order to improve precision, q can respond with two clock counters to remove the internal delay of processing the time request message: One upon receiving the time request (t_b), and the other before sending the time response (t_c). See Figure 1. For example, in NTP, the process p computes RTT δ and offset θ, as follows [41]:

$$\delta = (t_d - t_a) - (t_c - t_b)$$
$$\theta = \frac{(t_b + t_c)}{2} - \frac{(t_a + t_d)}{2}$$

Then, p applies these values to adjust its local clock.

2.3 Problems of Clock synchronization

Precision of a clock synchronization protocol is a function of clock skew, errors in reading remote clocks, and the interval between resynchronizations [24, 29, 33]. We discuss these factors in turn below and how they contribute to (reduced) precision in clock synchronization protocols.

2.3.1 Problems with Oscillator skew

Many factors such as temperature and quality of an oscillator can affect oscillator skew. Unfortunately, we often do not have control over these factors to the degree necessary to prevent reduced precision. As a result, even though oscillators may have been designed with the same nominal frequency, they may actually run at slightly different rates causing clock counters to diverge over time, requiring synchronization.

2.3.2 Problems with Reading Remote Clocks

There are many opportunities where reading clocks can be inaccurate and lead to reduced precision. In particular, reading remote clocks can be broken down into multiple steps (enumerated below) where each step can introduce random delay errors that can affect the precision of clock synchronization.

1. Preparing a time request (reply) message
2. Transmitting a time request (reply) message
3. Packet traversing time through a network
4. Receiving a time request (reply) message
5. Processing a time request (reply) message

Specifically, there are three points where precision is adversely affected: (a) accuracy of timestamping affects steps 1 and 5, (b) the software network stack can introduce errors in steps 2 and 4, and (c) network jitter can contribute errors in step 3. We discuss each one further.

Precision errors introduced by timestamps.

First, accurate timestamping is not trivial. Before transmitting a message, a process timestamps the message to embed its own local counter value. Similarly, after receiving a message, a process timestamps it for further processing (i.e. computing RTT). Timestamping is often inaccurate in commodity systems [36], which is a problem. It can add random delay errors which can prevent the nanosecond-level timestamping required for 10 Gigabit Ethernet (10 GbE) where minimum sized packets (64-byte) arriving at line speed can arrive every 68 nanoseconds. Improved timestamping with nanosecond resolution via new NICs are becoming more accessible [13]. However, random jitter can still be introduced due to the issues discussed below.

Precision errors introduced by network stack.

Second, transmitting and receiving messages involve a software network stack (e.g., between t_a and t'_a in Figure 1). Most clock synchronization protocols (e.g., NTP and PTP) run in a time daemon, which periodically sends and receives UDP packets between a remote process (or a time server). Unfortunately, the overhead of system calls, buffering in kernel and network interfaces, and direct memory access transactions can all contribute to errors in delay [25, 27, 36]. To minimize the impact of measurement errors, a daemon can run in kernel space, or kernel bypassing can be employed. Nonetheless, non-deterministic delay errors cannot be completely removed when a protocol involves a network stack.

Precision errors introduced by network jitter.

Third, packet propagation time can vary since it is prone to network jitter (e.g., between t'_a and t'_b or between t'_c and t'_d in Figure 1). Two processes are typically multiple hops away from each other and the delay between them can vary over time depending on network conditions and external traffic. Further, time requests and responses can be routed through asymmetric paths, or they may suffer different network conditions even when they are routed through symmetric paths. As a result, measured delay, which is often computed by dividing RTT by two, can be inaccurate.

2.3.3 Problems with Resynch Frequency

The more frequent resynchronizations, the more precise clocks can be synchronized to each other. However, frequent resynchronizations require increased message communication, which adds overhead to the network, especially in a datacenter network where hundreds of thousands of servers exist. The interval between resynchronizations can be configured. It is typically configured to resynchronize over a period of once per second [8], which will keep network overhead low, but on the flip side, will also adversely affect precision of clock synchronization.

2.4 NTP vs. PTP vs. GPS

In this section, we compare the most popular clock synchronization protocols, NTP, PTP, and GPS, in terms of the problems of clock synchronization discussed in Section 2.3. A summary of the comparison is in Table 1.

	Precision	Scalability	Overhead (pckts)	Extra hardware
NTP	us	Good	Moderate	None
PTP	sub-us	Good	Moderate	PTP-enabled devices
GPS	ns	Bad	None	Timing signal receivers, cables
DTP	ns	Good	None	DTP-enabled devices

Table 1: Comparison between NTP, PTP, GPS, and DTP

2.4.1 Network Time Protocol (NTP)

The most commonly used time synchronization protocol is the Network Time Protocol (NTP) [41]. NTP provides millisecond precision in a wide area network (WAN) and microsecond precision in a local area network (LAN). In NTP, time servers construct a tree, and top-level servers (or stratum 1) are connected to a reliable external time source (stratum 0) such as satellites through a GPS receiver, or atomic clocks. A client communicates with one of the time servers via UDP packets. As mentioned in Section 2.1, four timestamps are used to account for processing time in the time server.

NTP is not adequate for a datacenter. It is prone to errors that reduce precision in clock synchronization: Inaccurate timestamping, software network stack (UDP daemon), and network jitter. Furthermore, NTP assumes symmetric paths for time request and response messages, which is often not true in reality. NTP attempts to reduce precision errors via statistical approaches applied to network jitter and asymmetric paths. Nonetheless, the precision in NTP is still low.

2.4.2 Precise Time Protocol (PTP)

The IEEE 1588 Precise Time Protocol (PTP) [8][2] is an emerging time synchronization protocol that can provide tens to hundreds of nanosecond precision in a LAN when properly configured. PTP picks the most accurate clock in a network to be the *grandmaster* via the best master clock algorithm and others synchronize to it. The grandmaster could be connected to an external clock source such as a GPS receiver or an atomic clock. Network devices including PTP-enabled switches form a tree with the grandmaster as the root. Then, at each level of the tree, a server or switch behaves as a slave to its parent and a master to its children. When PTP is combined with Synchronous Ethernet, which syntonizes frequency of clocks (SyncE, See Section 8), PTP can achieve sub-nanosecond precision in a carefully configured environment [39], or hundreds of nanoseconds with tens of hops in back-haul networks [38].

The protocol normally runs as follows: The grandmaster periodically sends timing information (`Sync`) with IP multicast packets. Upon receiving a `Sync` message which contains time t_0, each client sends a `Delay_Req` message to the timeserver, which replies with a `Delay_Res` message. The mechanism of communicating `Delay_Req` and `Delay_Res` messages is similar to NTP, and Figure 1. Then, a client computes the offset and adjusts its clock or frequency. If the timeserver is not able to accurately embed t_0 in the `Sync` message, it emits a `Follow_Up` message with t_0, after the `Sync` message, to everyone.

To improve the precision, PTP employs a few techniques. First, PTP-enabled network switches can participate in the protocol as *Transparent clocks* or *Boundary clocks* in order to eliminate switching delays. Transparent clocks timestamp incoming and outgoing packets, and correct the time in `Sync` or `Follow_Up` to reflect switching delay. Boundary clocks are synchronized to the timeserver and work as masters to other PTP clients, and thus provide scalability to PTP networks. Second, PTP uses hardware timestamping in order to eliminate the overhead of network stack. Modern PTP-enabled NICs timestamp both incoming and outgoing PTP messages [13]. Third, a PTP-enabled NIC has a PTP hardware clock (PHC) in the NIC, which is synchronized to the timeserver. Then, a PTP-daemon is synchronized to the PHC [21, 45] to minimize network delays and jitter. Lastly, PTP uses smoothing and filtering algorithms to carefully measure one way delays.

As we demonstrate in Section 6.1, the precision provided by PTP is about few hundreds of nanoseconds at best in a 10 GbE environment, and it can change (decrease) over time even if the network is in an idle state. Moreover, the precision could be affected by the network condition, i.e. variable and/or asymmetric latency can significantly impact the precision of PTP, even when cut-through switches with priority flow control are employed [51, 52]. Lastly, it is not easy to scale the number of PTP clients. This is mainly due to the fact that a timeserver can only process a limited number of `Delay_Req` messages per second [8]. Boundary and Transparent clocks can potentially solve this scalability problem. However, precision errors from Boundary clocks can be cascaded to low-level components of the timing hierarchy tree, and can significantly impact the precision overall [30]. Further, it is shown that Transparent clocks often are not able to perform well under network congestion [52], although a correct implementation of Transparent clocks should not degrade the performance under network congestion.

2.4.3 Global Positioning System (GPS)

In order to achieve nanosecond-level precision, GPS can be employed [4, 22]. GPS provides about 100 nanosecond precision in practice [37]. Each server can have a dedicated GPS receiver or can be connected to a time signal distribution server through a dedicated link. As each device is directly synchronized to satellites (or atomic clocks) or is connected via a dedicated timing network, network jitter and software network stack is not an issue.

Unfortunately, GPS based solutions are not realistic for an entire datacenter. It is not cost effective and scalable because of extra cables and GPS receivers required for time signals. Further, GPS signals are not always available in a datacenter as GPS antennas must be installed on a roof with a clear view to the sky. However, GPS is often used in concert with other protocols such as NTP and PTP and also DTP.

2.5 Datacenter Time Protocol (DTP): Why the PHY?

Our goal is to achieve nanosecond-level precision as in GPS, with scalability in a datacenter network, and without

[2]We use PTPv2 in this discussion.

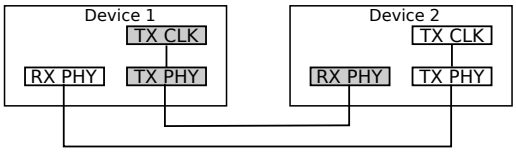


Figure 2: Clock domains of two peers. The same color represents the the same clock domain.

any network overhead. We achieve this goal by running a decentralized protocol in the physical layer (PHY).

DTP exploits the fact that two *peers*[3] are already synchronized in the PHY in order to transmit and receive bitstreams reliably and robustly. In particular, the receive path (RX) of a peer physical layer recovers the clock from the physical medium signal generated by the transmit path (TX) of the sending peer's PHY. As a result, although there are two physical clocks in two network devices, they are virtually in the same circuit (Figure 2; What each rectangle means is explained in Section 4.1).

Further, a commodity switch often uses one clock oscillator to feed the sole switching chip in a switch [2], i.e. all TX paths of a switch use the same clock source. Given a switch and N network devices connected to it, there are $N+1$ physical oscillators to synchronize, and all of them are virtually in the same circuit.

As delay errors from network jitter and a software network stack can be minimized by running the protocol in the lowest level of a system [48], the PHY is the best place to reduce those sources of errors. In particular, we give three reasons why clock synchronization in the PHY addresses the problems in Section 2.3.

First, the PHY allows accurate timestamping at sub-nanosecond scale, which can provide enough fidelity for nanosecond-level precision. Timestamping [27, 36] in the PHY achieves high precision by counting the number of bits between and within packets. Timestamping in the PHY relies on the clock oscillator that generates bits in the PHY, and, as a result, it is possible to read and embed clock counters with a deterministic number of clock cycles in the PHY.

Second, a software network stack is not involved in the protocol. As the physical layer is the lowest layer of a network protocol stack, there is always a deterministic delay between timestamping a packet and transmitting it. In addition, it is always possible to avoid buffering in a network device because protocol messages can always be transmitted when there is no other packet to send.

Lastly, there is little to no variation in delay between two peers in the PHY. The only element in the middle of two physically communicating devices is a wire that connects them. As a result, when there is no packet in transit, the delay in the PHY measured between two physically connected devices will be the time to transmit bits over the wire (propagation delay, which is always constant with our assumptions in Section 3.1), a few clock cycles required to process bits in the PHY (which can be deterministic), and a clock domain crossing (CDC) which can add additional random delay. A CDC is necessary for passing data between two clock domains, namely between the TX and RX paths. Synchronization FIFOs are commonly used for a CDC. In a synchronization FIFO, a signal from one clock domain goes through multiple flip-flops in order to avoid metastability from the other clock domain. As a result, one random delay could be added until the signal is stable to read.

Operating a clock synchronization protocol in the physical layer not only provides the benefits of zero to little delay errors, but also zero overhead to a network: There is no need for injection of packets to implement a clock synchronization protocol. A network interface continuously generates either Ethernet frames or special characters (Idle characters) to maintain a link connection to its peer. We can exploit those special characters in the physical layer to transmit messages (We will discuss this in detail in Section 4). The Ethernet standard [9] requires at least twelve idle characters (`/I/`) between any two Ethernet frames regardless of link speed to allow the receiving MAC layer to prepare for the next packet. As a result, if we use these idle characters to deliver protocol messages (and revert them back to idle characters), no additional packets will be required. Further, we can send protocol messages between every Ethernet frame without degrading the bandwidth of Ethernet and for different Ethernet speeds (See Section 7).

3. DATACENTER TIME PROTOCOL

In this section, we present the Datacenter Time Protocol (DTP): Assumptions, protocol, and analysis. The design goals for the protocol are the following:

- Internal synchronization with nanosecond precision.
- No network overhead: No packets are required for the synchronization protocol.

3.1 Assumptions

We assume, in a 10 Gigabit Ethernet (10 GbE) network, all network devices are driven by oscillators that run at slightly different rates due to oscillator skew, but operate within a range defined by the IEEE 802.3 standard. The standard requires that the clock frequency f_p be in the range of $[f - 0.0001f, f + 0.0001f]$[4] where f is 156.25 MHz in 10 GbE (See Section 4.1).

We assume that there are no "two-faced" clocks [34] or Byzantine failures which can report different clock counters to different peers.

We further assume that the length of Ethernet cables is bounded and, thus, network propagation delay is bounded. The propagation delay of optic fiber is about 5 nanoseconds per meter ($2/3 \times$ the speed of light, which is 3.3 nanoseconds per meter in a vacuum) [31]. In particular, we assume the longest optic fiber inside a datacenter is 1000 meters, and as a result the maximum propagation delay is at most 5 us. Most cables inside a datacenter are 1 to 10 meters as they are typically used to connect rack servers to a Top-of-Rack (ToR) switch; 5 to 50 nanoseconds would be the more common delay.

[3]two peers are two physically connected ports via a cable.

[4]This is ± 100 parts per million (ppm).

Algorithm 1 DTP inside a network port

STATE:
gc : global counter, from Algorithm 2
$lc \leftarrow 0$: local counter, increments at every clock tick
$d \leftarrow 0$: measured one-way delay to **peer** p
TRANSITION:
T0: After the link is established with p
$lc \leftarrow gc$
Send (*Init*, lc)
T1: After receiving (*Init*, c) from p
Send (*Init-Ack*, c)
T2: After receiving (*Init-Ack*, c) from p
$d \leftarrow (lc - c - \alpha)/2$
T3: After a timeout
Send (*Beacon*, gc)
T4: After receiving (*Beacon*, c) from p
$lc \leftarrow \max(lc, c + d)$

3.2 Protocol

In DTP, every network port (of a network interface or a switch) has a local counter in the physical layer that increments at every clock tick. DTP operates via protocol messages between peer network ports: A network port sends a DTP message timestamped with its current *local* counter to its peer and adjusts its local clock upon receiving a *remote* counter value from its peer. We show that given the bounded delay and frequent resynchronizations, local counters of two peers can be precisely synchronized in Section 3.3.

Since DTP operates and maintains local counters in the physical layer, switches play an important role in scaling up the number of network devices synchronized by the protocol. As a result, synchronizing across all the network ports of a switch (or a network device with a multi-port network interface) requires an extra step: DTP needs to synchronize the local counters of all local ports. Specifically, DTP maintains a *global* counter that increments every clock tick, but also always picks the *maximum* counter value between it and all of the local counters.

DTP follows Algorithm 1 to synchronize the local counters between two peers. The protocol runs in two phases: `INIT` and `BEACON` phases.

INIT phase The purpose of the `INIT` phase is to measure the one-way delay between two peers. The phase begins when two ports are physically connected and start communicating, i.e. when the link between them is established. Each peer measures the one-way delay by measuring the time between sending an `INIT` message and receiving an associated `INIT-ACK` message, i.e. measure RTT, then divide the measured RTT by two (T0, T1, and T2 in Algorithm 1).

As the delay measurement is processed in the physical layer, the RTT consists of a few clock cycles to send / receive the message, the propagation delays of the wire, and the clock domain crossing (CDC) delays between the receive and transmit paths. Given the clock frequency assumption, and the length of the wire, the only non-deterministic part is the CDC. We analyze how they affect the accuracy of the measured delay in Section 3.3. Note that α in Transition 2 in Algorithm 1 is there to control the non-deterministic variance added by the CDC (See Section 3.3).

Algorithm 2 DTP inside a network device / switch

STATE:
gc: global counter
$\{lc_i\}$: local counters
TRANSITION:
T5: at every clock tick
$gc \leftarrow \max(gc + 1, \{lc_i\})$

BEACON phase During the `BEACON` phase, two ports periodically exchange their local counters for resynchronization (T3 and T4 in Algorithm 1). Due to oscillator skew, the offset between two local counters will increase over time. A port adjusts its local counter by selecting the maximum of the local and remote counters upon receiving a `BEACON` message from its peer. Since `BEACON` messages are exchanged frequently, hundreds of thousands of times a second (every few microseconds), the offset can be kept to a minimum.

Scalability and multi hops Switches and multi-port network interfaces have two to ninety-six ports in a single device that need to be synchronized within the device[5]. As a result, DTP always picks the maximum of all local counters $\{lc_i\}$ as the value for a global counter gc (T5 in Algorithm 2). Then, each port transmits the global counter gc in a `BEACON` message (T3 in Algorithm 1).

Choosing the maximum allows any counter to increase monotonically at the same rate and allows DTP to scale: The maximum counter value propagates to all network devices via `BEACON` messages, and frequent `BEACON` messages keep global counters closely synchronized (Section 3.3).

Network dynamics When a device is turned on, the local and global counters of a network device are set to zero. The global counter starts incrementing when one of the local counters starts incrementing (i.e., a peer is connected), and continuously increments as long as one of the local counters is incrementing. However, the global counter is set to zero when all ports become inactive. Thus, the local and global counters of a newly joining device are always less than those of other network devices in a DTP network. We use a special `BEACON_JOIN` message in order to make large adjustments to a local counter. This message is communicated after `INIT_ACK` message in order for peers to agree on the maximum counter value between two local counters. When a network device with multiple ports receives a `BEACON_JOIN` message from one of its ports, it adjusts its global clock and propagates `BEACON_JOIN` messages with its new global counter to other ports. Similarly, if a network is partitioned and later restored, two subnets will have different global counters. When the link between them is re-established, `BEACON_JOIN` messages allow the two subnets to agree on the same (maximum) clock counter.

Handling failures There are mainly two types of failures that need to be handled appropriately: Bit errors and faulty devices. IEEE 802.3 standard supports a Bit Error Rate (BER) objective of 10^{-12} [9], which means one bit error

[5]Local counters of a multi-port device will not always be the same because remote clocks run at different rates. As a result, a multi-port device must synchronize local counters.

could happen every 100 seconds in 10 GbE. However, it is possible that a corrupted bit coincides with a DTP message and could result in a big difference between local and remote counters. As a result, DTP ignores messages that contain remote counters off by more than eight (See Section 3.3), or bit errors not in the three least significant bits (LSB). Further, in order to prevent bit errors in LSBs, each message could include a parity bit that is computed using three LSBs. As BEACON messages are communicated very frequently, ignoring messages with bit errors does not affect the precision.

Similarly, if one node makes too many *jumps* (i.e. adjusting local counters upon receiving BEACON messages) in a short period of time, it assumes the connected peer is faulty. Given the latency, the interval of BEACON messages, and maximum oscillator skew between two peers, one can estimate the maximum offset between two clocks and the maximum number of jumps. If a port receives a remote counter outside the estimated offset too often, it considers the peer to be faulty and stops synchronizing with the faulty device.

3.3 Analysis

As discussed in Section 2.1, the precision of clock synchronization is determined by oscillator skew, interval between resynchronizations, and errors in reading remote clocks [24, 29, 33]. In this section, we analyze DTP to understand its precision in regards to the above factors. In particular, we analyze the bounds on precision (clock offsets) and show the following:

- Bound of two tick errors due to measuring the one-way delay (OWD) during the INIT phase.
- Bound of two tick errors due to the BEACON interval. The offset of two synchronized peers can be up to two clock ticks if the interval of BEACON messages is less than 5000 ticks.
- As a result, the offset of two peers is bound by four clock ticks or $4T$ where T is 6.4ns. In 10 GbE the offset of two peers is bound by 25.6ns.
- Multi hop precision. As each link can add up to four tick errors, the precision is bounded by $4TD$ where 4 is the bound for the clock offset between directly connected peers, T is the clock period and D is the longest distance in terms of the number of hops.

For simplicity, we use two peers p and q, and use T_p (f_p) and T_q (f_q) to denote the period (frequency) of p and q's oscillator. We assume for analysis p's oscillator runs faster than q's oscillator, i.e. $T_p < T_q$ (or $f_p > f_q$).

Two tick errors due to OWD. In DTP, the one-way delay (OWD) between two peers, measured during the INIT phase, is assumed to be stable, constant, and symmetric in both directions. In practice, however, the delay can be measured differently depending on *when* it is measured due to oscillator skew and *how* the synchronization FIFO between the receive and transmit paths interact. Further, the OWD of one path (from p to q) and that of the other (from q to p) might not be symmetric due to the same reasons.We show that DTP still works with very good precision despite any errors introduced by measuring the OWD.

Suppose p sends an INIT message to q at time t, and the delay between p and q is d clock cycles. Given the assumption that the length of cables is bounded, and that oscillator skew is bounded, the delay is d cycles for both directions. The message arrives at q at $t + T_p d$ (i.e. the elapsed time is $T_p d$). Since the message can arrive in the middle of a clock cycle of q's clock, it can wait up to T_q before q processes it. Further, passing data from the receipt path to the transmit path requires a synchronization FIFO between two clock domains, which can add one more cycle randomly, i.e. the message could spend an additional T_q before it is received. Then, the INIT-ACK message from q takes $T_q d$ time to arrive at p, and it could wait up to $2T_p$ before p processes it. As a result, it takes up to a total of $T_p d + 2T_q + T_q d + 2T_p$ time to receive the INIT-ACK message after sending an INIT message. Thus, the measured OWD, d_p, at p is,

$$d_p \leq \lfloor \frac{T_p d + 2T_q + T_q d + 2T_p}{T_p} \rfloor / 2 = d + 2$$

In other words, d_p could be one of d, $d+1$, or $d+2$ clock cycles depending on when it is measured. As q's clock is slower than p, the clock counter of q cannot be larger than p. However, if the measured OWD, d_p, is larger than the actual OWD, d, then p will think q is faster and adjust its offset more frequently than necessary (See Transition $T4$ in Algorithm 1). This, in consequence, causes the global counter of the network to go faster than necessary. As a result, α in T2 of Algorithm 1 is introduced.

$\alpha = 3$ allows d_p to always be less than d. In particular, d_p will be $d-1$ or d; however, d_q will be $d-2$ or $d-1$. Fortunately, a measured delay of $d-2$ at q does not make the global counter go faster, but it can increase the offset between p and q to be two clock ticks most of the time, which will result in q adjusting its counter by one only when the actual offset is two.

Two tick errors due to the BEACON interval. The BEACON interval, period of resynchronization, plays a significant role in bounding the precision. We show that a BEACON interval of less than 5000 clock ticks can bound the clock offset to two ticks between peers.

Let $C_p(X)$ be a clock that returns a real time t at which $c_p(t)$ changes to X. Note that the clock is a discrete function. Then, $c_p(t) = X$ means, the value of the clock is stably X at least after $t - T_p$, i.e. $t - T_p < C_p(X) \leq t$.

Suppose p and q are synchronized at time t_1, i.e. $c_p(t_1) = c_q(t_1) = X$. Also suppose $c_p(t_2) = X + \Delta P$, and $c_q(t_2) = X + \Delta Q$ at time t_2, where ΔP is the difference between two counter values of clock p at time t_1 and t_2. Then,

$$t_2 - T_p < C_p(X + \Delta P) = C_p(X) + \Delta P T_p \leq t_2$$
$$t_2 - T_q < C_q(X + \Delta Q) = C_q(X) + \Delta Q T_q \leq t_2$$

Then, the offset between two clocks at t_2 is,

$$\Delta t(f_p - f_q) - 2 < \Delta P - \Delta Q < \Delta t(f_p - f_q) + 2$$

where $\Delta t = t_2 - t_1$.

Since the maximum frequency of a NIC clock oscillator is $1.0001f$, and the minimum frequency is $0.9999f$, $\Delta t(f_p - f_q)$ is always smaller than 1 if Δt is less than 32 us. As a

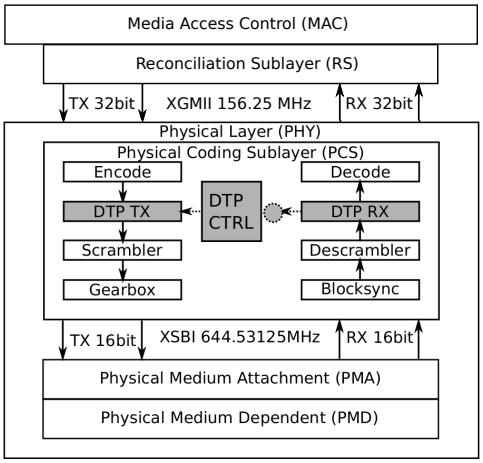


Figure 3: Low layers of a 10 GbE network stack. Grayed rectangles are DTP sublayers, and the circle represents a synchronization FIFO.

result, $\Delta P - \Delta Q$ can be always less than or equal to 2, if the interval of resynchronization (Δt) is less than 32 us ($\approx$ 5000 ticks). Considering the maximum latency of the cable is less than 5 us ($\approx$ 800 ticks), a beacon interval less than 25 us ($\approx$ 4000 ticks) is sufficient for any two peers to synchronize with 12.8 ns ($=$ 2 ticks) precision.

Multi hop Precision. Note that DTP always picks the maximum clock counter of all nodes as the global counter. All clocks will always be synchronized to the fastest clock in the network, and the global counter always increases monotonically. Then, the maximum offset between any two clocks in a network is between the fastest and the slowest. As discussed above, any link between them can add at most two offset errors from the measured delay and two offset errors from BEACON interval. Therefore, the maximum offset within a DTP-enabled network is bounded by $4TD$ where D is the longest distance between any two nodes in a network in terms of number of hops, and T is the period of the clock as defined in the IEEE 802.3 standard ($\approx 6.4ns$).

4. IMPLEMENTATION

In this section, we briefly discuss the IEEE 802.3ae 10 Gigabit Ethernet standard before presenting how we modify the physical layer to support DTP.

4.1 IEEE 802.3 Standard

According to the IEEE 802.3ae, the physical layer (PHY) of 10 GbE consists of three sublayers (Figure 3): The Physical Coding Sublayer (PCS), the Physical Medium Attachment (PMA), and the Physical Medium Dependent (PMD). The PMD is responsible for transmitting the outgoing symbolstream over the physical medium and receiving the incoming symbolstream from the medium. The PMA is responsible for clock recovery and (de-)serializing the bitstream. The PCS performs 64b/66b encoding / decoding.

In the PHY, there is a 66-bit Control block (/E/), which encodes eight seven-bit idle characters (/I/). As the standard requires at least twelve /I/s in an interpacket gap, it is *guaranteed* to have at least one /E/ block preceding any

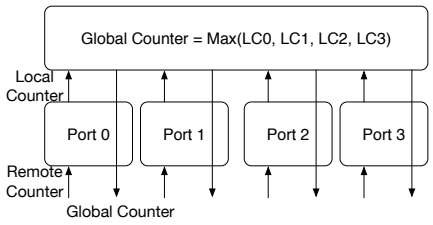


Figure 4: DTP enabled four-port device.

Ethernet frame[6]. Moreover, when there is no Ethernet frame, there are always /E/ blocks: 10 GbE is always sending at 10 Gbps and sends /E/ blocks continuously if there are no Ethernet frames to send.

As briefly mentioned in Section 2, the PCS of the transmit path is driven by the local oscillator, and the PCS of the receive path is driven by the recovered clock from the incoming bitstream. See Figure 2.

4.2 DTP-enabled PHY

The control logic of DTP in a network port consists of Algorithm 1 from Section 3 and a local counter. The local counter is a 106-bit integer (2 $\times$ 53 bits) that increments at every clock tick (6.4 ns = 1/156.25 MHz), or is adjusted based on received BEACON messages. Note that the same oscillator drives all modules in the PCS sublayer on the transmit path and the control logic that increments the local counter. i.e. they are in the same clock domain. As a result, the DTP sublayer can easily insert the local clock counter into a protocol message with no delay.

The DTP-enabled PHY is illustrated in Figure 3. Figure 3 is exactly the same as the PCS from the standard, except that Figure 3 has DTP control, TX DTP, and RX DTP sublayers shaded in gray. Specifically, on the transmit path, the TX DTP sublayer inserts protocol messages, while, on the receive path, the RX DTP sublayer processes incoming protocol messages and forwards them to the control logic through a synchronization FIFO. After the RX DTP sublayer receives and uses a DTP protocol message from the Control block (/E/), it replaces the DTP message with idle characters (/I/s, all 0's) as required by the standard such that higher network layers do not know about the existence of the DTP sublayer. Lastly, when an Ethernet frame is being processed in the PCS sublayer in general, DTP simply forwards blocks of the Ethernet frame unaltered between the PCS sublayers.

4.3 DTP-enabled network device

A DTP-enabled device (Figure 4) can be implemented with additional logic on top of the DTP-enabled ports. The logic maintains the 106-bit global counter as shown in Algorithm 2, which computes the maximum of the local counters of all ports in the device. The computation can be optimized with a tree-structured circuit to reduce latency, and can be performed in a deterministic number of cycles. When

[6]Full-duplex Ethernet standards such as 1, 10, 40, 100 GbE send at least twelve /I/s (at least one /E/) between every Ethernet frame.

a switch port tries to send a BEACON message, it inserts the global counter into the message, instead of the local counter. Consequently, all switch ports are synchronized to the same global counter value.

4.4 Protocol messages

DTP uses /I/s in the /E/ control block to deliver protocol messages. There are eight seven-bit /I/s in an /E/ control block, and, as a result, 56 bits total are available for a DTP protocol message per /E/ control block. Modifying control blocks to deliver DTP messages does not affect the physics of a network interface since the bits are scrambled to maintain DC balance before sending on the wire (See the scrambler/descrambler in Figure 3). Moreover, using /E/ blocks do not affect higher layers since DTP replaces /E/ blocks with required /I/s (zeros) upon processing them.

A DTP message consists of a three-bit message type, and a 53-bit payload. There are five different message types in DTP: INIT, INIT−ACK, BEACON, BEACON−JOIN, and BEACON−MSB. As a result, three bits are sufficient to encode all possible message types. The payload of a DTP message contains the local (global) counter of the sender. Since the local counter is a 106-bit integer and there are only 53 bits available in the payload, each DTP message carries the 53 least significant bits of the counter. In 10 GbE, a clock counter increments at every 6.4 ns (=1/156.25MHz), and it takes about 667 days to overflow 53 bits. DTP occasionally transmits the 53 most significant bits in a BEACON−MSB message in order to prevent overflow.

As mentioned in Section 4.1, it is always possible to transmit one protocol message after/before an Ethernet frame is transmitted. This means that when the link is fully saturated with Ethernet frames DTP can send a BEACON message every 200 clock cycles ($\approx$ 1280 ns) for MTU-sized (1522B) frames[7] and 1200 clock cycles ($\approx$ 7680 ns) at worst for jumbo-sized ($\approx$9kB) frames. The PHY requires about 191 66-bit blocks and 1,129 66-bit blocks to transmit a MTU-sized or jumbo-sized frame, respectively. This is more than sufficient to precisely synchronize clocks as analyzed in Section 3.3 and evaluated in Section 6. Further, DTP communicates frequently when there are no Ethernet frames, e.g every 200 clock cycles, or 1280 ns: The PHY continuously sends /E/ when there are no Ethernet frames to send.

5. PRACTICAL CONSIDERATIONS

5.1 Accessing DTP counters

Applications access the DTP counter via a *DTP daemon* that runs in each server. A DTP daemon regularly (e.g., once per second) reads the DTP counter of a network interface card via a memory-mapped IO in order to minimize errors in reading the counter. Further, TSC counters are employed to estimate the frequency of the DTP counter. A TSC counter is a reliable and stable source to implement software clocks [46, 50, 25]. Modern systems support *invariant* TSC counters that are not affected by CPU power states [10]. Applications can accurately estimate DTP counters via a get_DTP_counter API that interpolates the DTP counter at any moment using TSC counters and the estimated DTP clock frequency. Similar techniques are used to implement gettimeofday(). The details of how a DTP daemon works and how the API is implemented is standard. Note that DTP counters of each NIC are running at the same rate on every server in a DTP-enabled network and, as a result, software clocks that DTP daemons implement are also tightly synchronized.

[7]It includes 8-byte preambles, an Ethernet header, 1500-byte payload and a checksum value.

5.2 External Synchronization

We discuss one simple approach that extends DTP to support external synchronization, although there could be many other approaches. One server (either a timeserver or a commodity server that uses PTP or NTP) periodically (e.g., once per second) broadcasts a pair, DTP counter and universal time (UTC), to other servers. Upon receiving consecutive broadcast messages, each DTP daemon estimates the frequency ratio between the received DTP counters and UTC values. Next, applications can read UTC by interpolating the current DTP counter with the frequency ratio in a similar fashion as the method discussed in Section 5.1. Again, DTP counters of each NIC are running at the same rate, and as a result, UTC at each server can also be tightly synchronized with some loss of precision due to errors in reading system clocks. It is also possible to combine DTP and PTP to improve the precision of external synchronization further: A timeserver timestamps sync messages with DTP counters, and delays between the timeserver and clients are measured using DTP counters.

5.3 Incremental Deployment

DTP requires the physical layer to be modified. As a result, in order to deploy DTP, network devices must be modified. As there is usually a single switching chip inside a network device [2], the best strategy to deploy DTP is to implement it inside the switching chip. Then network devices with DTP-enabled switching chips can create a DTP-enabled network. This would require updating the firmware, or possibly replacing the switching chip. PTP uses a similar approach in order to improve precision: PTP-enabled switches have a dedicated logic inside the switching chip for processing PTP packets and PTP-enabled NICs have hardware timestamping capabilities and PTP hardware clocks (PHC). Therefore, the cost of achieving the best configuration of PTP is essentially the same as the cost of deploying DTP, as both require replacing NICs and switches.

An alternative way to deploy DTP is to use FPGA-based devices. FPGA-based NICs and switches [5, 43] have more flexibility of updating firmware. Further, customized PHYs can be easily implemented and deployed with modern FPGAs that are equipped with high-speed transceivers.

One of the limitations of DTP is that it is not possible to deploy DTP on routers or network devices with multiple line cards without sacrificing precision. Network ports on separate line cards typically communicate via a bus inter-

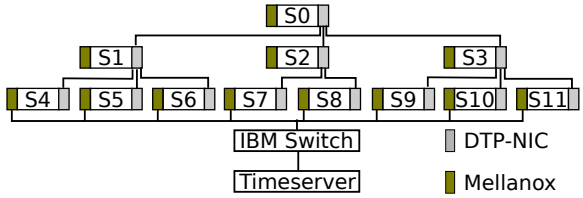


Figure 5: Evaluation Setup

face. As a result, it is not possible to maintain a single global counter with high precision over a shared bus, although each line card can have its own separate global counter. Fortunately, as long as all switches and line cards form a connected graph, synchronization can be maintained.

Replacing or updating switches and NICs in a datacenter at once is not possible due to both cost and availability. Importantly, DTP can be incrementally deployed: NICs and a ToR switch within the same rack are updated at the same time, and aggregate and core switches are updated incrementally from the lower levels of a network topology. Each DTP-enabled rack elects one server to work as a master for PTP / NTP. Then, servers within the same rack will be tightly synchronized, but servers from different racks are less tightly synchronized depending on the performance of PTP / NTP. When two independently DTP-enabled racks start communicating via a DTP-enabled switch, servers from two racks will be tightly synchronized both internally and externally after communicating `BEACON_JOIN` messages.

5.4 Following The Fastest Clock

DTP assumes that oscillators of DTP-enabled devices operate within a range defined by IEEE 802.3 standard (Section 3.1). However, in practice, this assumption can be broken, and an oscillator in a network could run at a frequency outside the range specified in the standard. This could lead to many jumps from devices with slower oscillators. More importantly, the maximum offset between two devices could be larger than 4TD. One approach to address the problem is to choose a network device with a reliable and stable oscillator as a master node. Then, through DTP daemons, it is possible to construct a DTP spanning tree using the master node as a root. This is similar to PTP's best master clock algorithm. Next, at each level of the tree, a node uses the remote counter of its parent node as the global counter. If an oscillator of a child node runs faster than its parent node, the local counter of a child should *stall* occasionally in order to keep the local counter monotonically increasing. We leave this design as a future work.

6. EVALUATION

In this section, we attempt to answer following questions:

- *Precision:* In Section 3.3, we showed that the precision of DTP is bounded by $4TD$ where D is the longest distance between any two nodes in terms of number of hops. In this section, we demonstrate and measure that precision is indeed within the $4TD$ bound via a prototype and deployed system.
- *Scalability:* We demonstrate that DTP scales as the number of hops of a network increases.

Further, we measured the precision of accessing DTP from software and compared DTP against PTP.

6.1 Evaluation Setup

For the DTP prototype and deployment, we used programmable NICs plugged into commodity servers: We used DE5-Net boards from Terasaic [3]. A DE5-Net board is an FPGA development board with an Altera Stratix V [15] and four Small Form-factor Pluggable (SFP+) modules. We implemented the DTP sublayer and the 10 GbE PHY using the Bluespec language [1] and Connectal framework [32]. We deployed DE5-Net boards on a cluster of twelve Dell R720 servers. Each server was equipped with two Xeon E5-2690 processors and 96 GB of memory. All servers were in the same rack in a datacenter. The temperature of the datacenter was stable and cool.

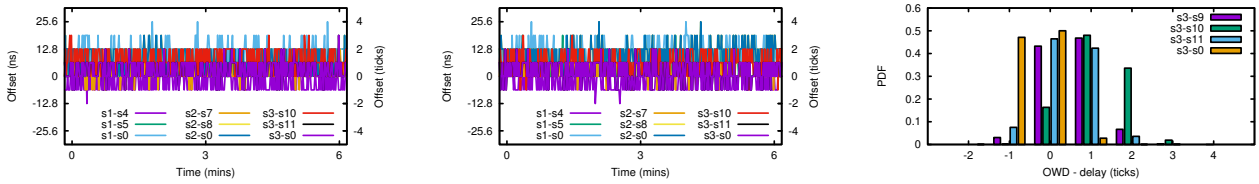
We created a DTP network as shown in Figure 5: A tree topology with the height of two, i.e. the maximum number of hops between any two leaf servers was four. DE5-Net boards of the root node, $S0$, and intermediate nodes, $S1 \sim S3$, were configured as DTP switches, and those of the leaves ($S4 \sim S11$) were configured as DTP NICs. We used 10-meter Cisco copper twinax cables to a DE5-Net board's SFP+ modules. The measured one-way delay (OWD) between any two DTP devices was 43 to 45 cycles ($\approx$ 280 ns).

We also created a PTP network with the same servers as shown in Figure 5 (PTP used Mellanox NICs). Each Mellanox NIC was a Mellanox ConnectX-3 MCX312A 10G NIC. The Mellanox NICs supported hardware timestamping for incoming and outgoing packets which was crucial for achieving high precision in PTP. A VelaSync timeserver from Spectracom was deployed as a PTP grandmaster clock. An IBM G8264 cut-through switch was used to connect the servers including the timeserver. As a result, the number of hops between any two servers in the PTP network was always two. Cut-through switches are known to work well in PTP networks [52]. We deployed a commercial PTP solution (Timekeeper [16]) in order to achieve the best precision in 10 Gigabit Ethernet. Note that the IBM switch was configured as a transparent clock.

The timeserver multicasted PTP timing information every second, i.e. the synchronization rate was once per second, which was the recommended sync rate by the provider. Note that each `sync` message was followed by `Follow_Up` and `Announce` messages. Further, we enabled PTP `UNICAST` capability, which allowed the server to send unicast `sync` messages to individual PTP clients once per second in addition to multicast `sync` messages. In our configuration, a client sent two `Delay_Req` messages per 1.5 seconds.

6.2 Methodology

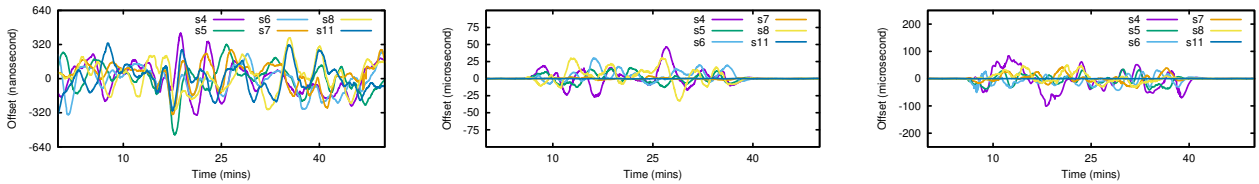
Measuring offsets at nanosecond scale is a very challenging problem. One approach is to let hardware generate pulse per second (PPS) signals and compare them using an oscilloscope. Another approach, which we use, is to measure the precision directly in the PHY. Since we are mainly interested in the clock counters of network devices, we developed a *logging* mechanism in the PHY.



(a) DTP: BEACON interval = 200. Heavily loaded with MTU packets.

(b) DTP: BEACON interval = 1200. Heavily loaded with Jumbo packets.

(c) DTP: Offset distribution from S3. (BEACON interval = 1200 cycles)

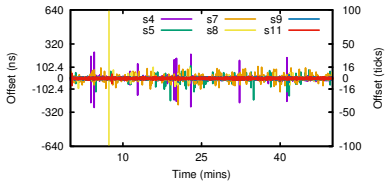


(d) PTP: Idle network

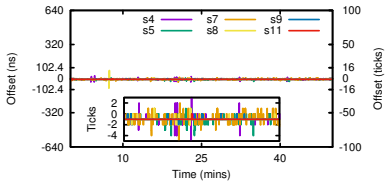
(e) PTP: Medium loaded

(f) PTP: Heavily loaded

Figure 6: Precision of DTP and PTP. A *tick* is 6.4 nanoseconds.



(a) Before smoothing: Raw offset_{sw}



(b) After smoothing: Window size = 10

Figure 7: Precision of DTP daemon.

Each leaf node generates and sends a 106-bit log message twice per second to its peer, a DTP switch. DTP switches also generate log messages between each other twice per second. A log message contains a 53-bit estimate of the DTP counter generated by the DTP daemon, t_0 (See Section 5), which is then timestamped in the DTP layer with the lower 53-bits of the global counter (or the local counter if it is a NIC). The 53-bit timestamp, t_1, is appended to the original message generated by the DTP daemon, and, as a result, a 106-bit message is generated by the sender. Upon arriving at an intermediate DTP switch, the log message is timestamped again, t_2, in the DTP layer with the receiver's global counter. Then, the original 53-bit log message (t_0) and two timestamps (t_1 from the sender and t_2 from the receiver) are delivered to a DTP daemon running on the receiver. By computing $\text{offset}_{hw} = t_2 - t_1 - \text{OWD}$ where OWD is the one-way delay measured in the INIT phase, we can estimate the precision between two peers. Similarly, by computing $\text{offset}_{sw} = t_1 - t_0$, we can estimate the precision of a DTP daemon. Note that offset_{hw} includes the non-deterministic variance from the synchronization FIFO and offset_{sw} includes the non-deterministic variance from the PCIe bus. We can accurately approximate both the offset_{hw} and offset_{sw} with this method.

For PTP, the Timekeeper provides a tool that reports measured offsets between the timeserver and all PTP clients. Note that our Mellanox NICs have PTP hardware clocks (PHC). For a fair comparison against DTP that synchronizes clocks of NICs, we use the precision numbers measured from a PHC. Also, note that a Mellanox NIC timestamps PTP packets in the NIC for both incoming and outgoing packets.

The PTP network was mostly idle except when we introduced network congestion. Since PTP uses UDP datagrams for time synchronization, the precision of PTP can vary relying on network workloads. As a result, we introduced network workloads between servers using `iperf` [11]. Each server occasionally generated MTU-sized UDP packets destined for other servers so that PTP messages could be dropped or arbitrarily delayed.

To measure how DTP responds to varying network conditions, we used the same `heavy load` that we used for PTP and also changed the BEACON interval during experiments from 200 to 1200 cycles, which changed the Ethernet frame size from 1.5kB to 9kB. Recall that when a link is fully saturated with MTU-sized (Jumbo) packets, the minimum BEACON interval possible is 200 (1200) cycles.

6.3 Results

Figure 6 and 7 show the results: We measured precision of DTP in Figure 6a-c, PTP in Figure 6d-f, and the DTP daemon in Figure 7. For all results, we continuously synchronized clocks and measured the precision (clock offsets) over at least a two-day period in Figure 6 and at least a few-hour period in Figure 7.

Figures 6a-b demonstrate that the clock offsets between any two directly connected nodes in DTP never differed by more than four clock ticks; i.e. offsets never differed by more than 25.6 nanoseconds ($4TD = 4 \times 6.4 \times 1 = 25.6$): Figures 6a and b show three minutes out of a two-day measurement period and Figure 6c shows the distribution of the

measured offsets with node S3 for the entire two-day period. The network was always under `heavy load` and we varied the Ethernet frame size by varying the `BEACON` interval between 200 cycles in Figure 6a and 1200 cycles in Figure 6b. DTP performed similarly under `idle` and `medium load`. Since we measured all pairs of nodes and no offset was ever greater than four, the results support that precision was bounded by $4TD$ for nodes D hops away from each other. Figure 7 shows the precision of accessing a DTP counter via a DTP daemon: Figure 7a shows the raw offset$_{sw}$ and Figure 7b shows the offset$_{sw}$ after applying a moving average algorithm with a window size of 10. We applied the moving average algorithm to smooth the effect of the non-determinism from the PCIe bus, which is shown as occasional spikes. The offset between a DTP daemon in software and the DTP counter in hardware was usually no more than 16 clock ticks ($\approx 102.4ns$) before smoothing, and was usually no more than 4 clock ticks ($\approx 25.6ns$) after smoothing.

Figures 6d-f show the measured clock offsets between each node and the grandmaster timeserver using PTP. Each figure shows minutes to hours of a multi-day measurement period, enough to illustrate the precision trends. We varied the load of the network from `idle` (Figure 6d), to `medium load` where five nodes transmitted and received at 4 Gbps (Figure 6e), to `heavy load` where the receive and transmit paths of all links except S11 were fully saturated at 9 Gbps (Figure 6f). When the network was `idle`, Figure 6d showed that PTP often provided hundreds of nanoseconds of precision, which matches literature [7, 17]. When the network was under `medium load`, Figure 6e showed the offsets of $S4 \sim S8$ became unstable and reached up to 50 microseconds. Finally, when the network was under `heavy load`, Figure 6f showed that the maximum offset degraded to hundreds of microseconds. Note that we measured, but do not report the numbers from the PTP daemon, `ptpd`, because the precision with the daemon was the same as the precision with the hardware clock, PHC. Also, note that all reported PTP measurements include smoothing and filtering algorithms.

There are multiple takeaways from these results.

1. DTP more tightly synchronized clocks than PTP.
2. The precision of DTP was not affected by network workloads. The maximum offset observed in DTP did not change either when load or Ethernet frame size (the `BEACON` interval) changed. PTP, on the other hand, was greatly affected by network workloads and the precision varied from hundreds of nanoseconds to hundreds of microseconds depending on the network load.
3. DTP scales. The precision of DTP only depends on the number of hops between any two nodes in the network. The results show that precision (clock offsets) were always bounded by $4TD$ nanoseconds.
4. DTP daemons can access DTP counters with tens of nanosecond precision.
5. DTP synchronizes clocks in a short period of time, within two `BEACON` intervals. PTP, however, took about 10 minutes for a client to have an offset below one microsecond. This was likely because PTP needs history to apply filtering and smoothing effectively. We omitted these results due to limited space.
6. PTP's performance was dependent upon network conditions, configuration such as transparent clocks, and implementation.

Data Rate	Encoding	Data Width	Frequency	Period	Δ
1G	8b/10b	8 bit	125 MHz	8 ns	25
10G	64b/66b	32 bit	156.25 MHz	6.4 ns	20
40G	64b/66b	64 bit	625 MHz	1.6 ns	5
100G	64b/66b	64 bit	1562.5 MHz	0.64 ns	2

Table 2: Specifications of the PHY at different speeds

7. DISCUSSION

What about 1G, 40G or 100G? In this paper we discussed and demonstrated how we can implement and deploy DTP over a datacenter focusing on 10 GbE links. However, the capacity of links in a datacenter is not homogeneous. Servers can be connected to Top-of-Rack switches via 1 Gbps links, and uplinks between switches and routers can be 40 or 100 Gbps. Nonetheless, DTP is still applicable to these cases because the fundamental fact still holds: Two physically connected devices in high-speed Ethernet (1G and beyond) are already synchronized to transmit and receive bitstreams. The question is how to modify DTP to support thousands of thousands of devices with different link capacities.

DTP can be extended to support 40 GbE and 100 GbE in a straight forward manner. The clock frequency required to operate 40 or 100 Gbps is multiple of that of 10 Gbps (Table 2). In fact, switches that support 10 Gbps and beyond normally use a clock oscillator running at 156.25 MHz to support all ports [14]. As a result, incrementing clock counters by different values depending on the link speed is sufficient. In particular, see the last column of Table 2, if a counter tick represents 0.32 nanoseconds, then DTP will work at 10, 40, and 10GbE by adjusting a counter value to match the corresponding clock period (i.e. $20 \times 0.32 = 6.4$ ns, $5 \times 0.32 = 1.6$ ns, and $2 \times 0.32 = 0.64$ ns, respectively).

Similarly, DTP can be made to work with 1 GbE by incrementing the counter of a 1 GbE port by 25 at every tick (see the last column of Table 2). However, the PHY of 1 Gbps is different, it uses a 8b/10b encoding instead of a 64b/66b encoding, and we need to adapt DTP to send clock counter values with the different encoding.

8. RELATED WORK

Clock synchronization is critical to systems and has been extensively studied from different areas. As we discussed NTP [41], PTP [8], and GPS [37] in Section 2, we briefly discuss other clock synchronization protocols.

Because NTP normally does not provide precise clock synchronization in a local area network (LAN), much of the literature has focused improving NTP without extra hardware. One line of work was to use `TSC` instructions to implement precise software clocks called TSCclock, and later called RADclock [25, 46, 50]. It was designed to replace `ntpd` and `ptpd` (daemons that run NTP or PTP) and pro-

vide sub-microsecond precision without any extra hardware support. Other software clocks include Server Time Protocol (STP) [44], Coordinated Cluster Time (CCT) [28], AP2P [49], and skewless clock synchronization [40], which provide microsecond precision.

Implementing clock synchronization in hardware has been demonstrated by Fiber Channel (FC) [6] and discussed by Kopetz and Ochsenreiter [33]. FC embeds protocol messages into interpacket gaps similar to DTP. However, it is not a decentralized protocol and the network fabric simply forwards protocol messages between a server and a client using physical layer encodings. As a result, it does not eliminate non-deterministic delays in delivering protocol messages.

Synchronous optical networks (SONET/SDH) is a standard that transmits multiple bitstreams (such as Voice, Ethernet, TCP/IP) over an optical fiber. In order to reduce buffering of data between network elements, SONET requires precise frequency synchronization (i.e., *syntonization*). An atomic clock is commonly deployed as a Primary Reference Clock (PRC), and other network elements are synchronized to it either by external timing signals or by recovering clock signals from incoming data. DTP does not synchronize frequency of clocks, but values of clock counters.

Synchronous Ethernet (SyncE) [12] was introduced for reliable data transfer between synchronous networks (e.g. SONET/SDH) and asynchronous networks (e.g. Ethernet). Like SONET, it synchronizes the frequency of nodes in a network, not clocks (i.e. *syntonization*). It aims to provide a synchronization signal to all Ethernet network devices. The idea is to use the recovered clock from the receive (RX) path to drive the transmit (TX) path such that both the RX and TX paths run at the same clock frequency. As a result, each Ethernet device uses a phase locked loop to regenerate the synchronous signal. As SyncE itself does not synchronize clocks in a network, PTP is often employed along with SyncE to provide tight clock synchronization. One such example is White Rabbit which we discuss below.

White Rabbit [43, 35, 39] has by far the best precision in packet-based networks. The goal of White Rabbit (WR) [43] was to synchronize up to 1000 nodes with sub-nanosecond precision. It uses SyncE to syntonize the frequency of clocks of network devices, and WR-enabled PTP [35] to embed the phase difference between a master and a slave into PTP packets. WR demonstrated that the precision of a non-disturbed system was 0.517ns [39]. WR also requires WR-enabled switches, and synchronizes slaves that are up to four-hops apart from the timeserver. WR works on a network with a tree topology and with a limited number of levels and servers. Furthermore, it currently supports 1 Gigabit Ethernet only, and it is not clear how WR behaves under heavy network loads as it uses PTP packets. DTP does not rely on any specific network topology, and can be extended to protocols with higher speeds.

Similarly, BroadSync [19] and ChinaMobile [38] also combine SyncE and PTP to provide hundreds of nanosecond precision. The Data Over Cable Service Interface Specification (DOCSIS) is a frequency synchronized network designed to time divide data transfers between multiple cable modems (CM) and a cable modem termination system (CMTS). The DOCSIS time protocol [20] extends DOCSIS to synchronize time by approximating the internal delay from the PHY and asymmetrical path delays between a reference CM and the CMTS. We expect that combining DTP with frequency synchronization, SyncE, will also improve the precision of DTP to sub-nanosecond precision as it becomes possible to minimize or remove the variance of the synchronization FIFO between the DTP TX and RX paths.

9. CONCLUSION

Synchronizing clocks with bounded and high precision is not trivial, but can improve measurements (e.g. one-way delay) and performance (e.g. Spanner TrueTime). In this paper, we presented DTP that tightly synchronizes clocks with zero network overhead (no Ethernet packets). It exploits the fundamental fact that two physically connected devices are already synchronized to transmit and receive bitstreams. We demonstrated that DTP can synchronize clocks of network components at tens of nanoseconds of precision, can scale up to synchronize an entire datacenter network, and can be accessed from software with usually better than twenty five nanosecond precision. As a result, the end-to-end precision is the precision from DTP in the network (i.e. 25.6 nanoseconds for directly connected nodes and 153.6 nanoseconds for a datacenter with six hops) plus fifty nanosecond precision from software.

10. ACKNOWLEDGMENTS

This work was partially funded and supported by a SLOAN Research Fellowship received by Hakim Weatherspoon, DARPA MRC, DARPA CSSG (D11AP00266), NSF CAREER (1053757), NSF TRUST (0424422), Cisco, and Intel. We would like to thank our shepherd, Alex Snoeren, and the anonymous reviewers for their comments.

11. REFERENCES

[1] Bluespec. www.bluespec.com.
[2] Broadcom. http://http://www.broadcom.com/products/Switching/Data-Center.
[3] DE5-Net FPGA development kit. http://de5-net.terasic.com.tw.
[4] Endace DAG network cards. http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html.
[5] Exablaze. https://exablaze.com/.
[6] Fibre channel. http://fibrechannel.org.
[7] Highly accurate time synchronization with ConnectX-3 and Timekeeper. http://www.mellanox.com/pdf/whitepapers/WP_Highly_Accurate_Time_Synchronization.pdf.
[8] IEEE Standard 1588-2008. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4579757.
[9] IEEE Standard 802.3-2008. http://standards.ieee.org/about/get/802/802.3.html.
[10] Intel 64 and IA-32 architectures software developer manuals. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.
[11] iperf. https://iperf.fr.
[12] ITU-T Rec. G.8262. http://www.itu.int/rec/T-REC-G.8262.
[13] Mellanox. www.mellanox.com.
[14] Open compute project. http://www.opencompute.org.
[15] Stratix V FPGA. http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp.

[16] Timekeeper. http://www.fsmlabs.com/timekeeper.

[17] IEEE 1588 PTP and Analytics on the Cisco Nexus 3548 Switch. http://www.cisco.com/c/en/us/products/collateral/switches/nexus-3000-series-switches/white-paper-c11-731501.html, 2014.

[18] AL-FARES, M., LOUKISSAS, A., AND VAHDAT, A. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM Conference on Data Communication* (2008).

[19] BROADCOM. Ethernet time synchronization. http://www.broadcom.com/collateral/wp/StrataXGSIV-WP100-R.pdf.

[20] CHAPMAN, J. T., CHOPRA, R., AND MONTINI, L. The DOCSIS timing protocol (DTP) generating precision timing services from a DOCSIS system. In *Proceedings of the Spring Technical Forum* (2011).

[21] COCHRAN, R., MARINESCU, C., AND RIESCH, C. Synchronizing the Linux System Time to a PTP Hardware Clock. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication* (2011).

[22] CORBETT, J. C., DEAN, J., EPSTEIN, M., FIKES, A., FROST, C., FURMAN, J. J., GHEMAWAT, S., GUBAREV, A., HEISER, C., HOCHSCHILD, P., HSIEH, W., KANTHAK, S., KOGAN, E., LI, H., LLOYD, A., MELNIK, S., MWAURA, D., NAGLE, D., QUINLAN, S., RAO, R., ROLIG, L., SAITO, Y., SZYMANIAK, M., TAYLOR, C., WANG, R., AND WOODFORD, D. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation* (2012).

[23] COSTA, P., BALLANI, H., RAZAVI, K., AND KASH, I. R2C2: A network stack for rack-scale computers. In *Proceedings of the ACM Conference on SIGCOMM* (2015).

[24] CRISTIAN, F. Probabilistic clock synchronization. *Distributed Computing 3* (September 1989), 146–158.

[25] DAVIS, M., VILLAIN, B., RIDOUX, J., ORGERIE, A.-C., AND VEITCH, D. An IEEE-1588 Compatible RADclock. In *Proceedings of International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication* (2012).

[26] EDWARDS, T. G., AND BELKIN, W. Using SDN to Facilitate Precisely Timed Actions on Real-time Data Streams. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (2014).

[27] FREEDMAN, D. A., MARIAN, T., LEE, J. H., BIRMAN, K., WEATHERSPOON, H., AND XU, C. Exact temporal characterization of 10 Gbps optical wide-area network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet measurement* (2010).

[28] FROEHLICH, S., HACK, M., MENG, X., AND ZHANG, L. Achieving precise coordinated cluster time in a cluster environment. In *Proceedings of International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication* (2008).

[29] GUSELLA, R., AND ZATTI, S. The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD. *IEEE Transactions on Software Engineering 15*, 7 (July 1989), 847–853.

[30] JASPERNEITE, J., SHEHAB, K., AND WEBER, K. Enhancements to the time synchronization standard IEEE-1588 for a system of cascaded bridges. In *Proceedings of the IEEE International Workshop in Factory Communication Systems* (2004).

[31] KACHRIS, C., BERGMAN, K., AND TOMKOS, I. *Optical Interconnects for Future Data Center Networks*. Springer, 2013.

[32] KING, M., HICKS, J., AND ANKCORN, J. Software-driven hardware development. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2015).

[33] KOPETZ, H., AND OCHSENREITER, W. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers C-36* (Aug 1987), 933–940.

[34] LAMPORT, L., AND MELLIAR-SMITH, P. M. Byzantine Clock Synchronization. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing* (1984).

[35] LAPINSKI, M., WLOSTOWKI, T., SERRANO, J., AND ALVAREZ, P. White Rabbit: a PTP Application for Robust Sub-nanosecond Synchronization. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication* (2011).

[36] LEE, K. S., WANG, H., AND WEATHERSPOON, H. SoNIC: Precise Realtime Software Access and Control of Wired Networks. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation* (2013).

[37] LEWANDOWSKI, W., AZOUBIB, J., AND KLEPCZYNSKI, W. J. GPS: primary tool for time transfer. *Proceedings of the IEEE 87* (January 1999), 163–172.

[38] LI, H. IEEE 1588 time synchronization deployment for mobile backhaul in China Mobile, 2014. Keynote speech in the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication.

[39] LIPINSKI, M., WLOSTOWSKI, T., SERRANO, J., ALVAREZ, P., COBAS, J. D. G., RUBINI, A., AND MOREIRA, P. Performance results of the first White Rabbit installation for CNGS time transfer. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication* (2012).

[40] MALLADA, E., MENG, X., HACK, M., ZHANG, L., AND TANG, A. Skewless Network Clock Synchronization. In *Proceedings of the 21st IEEE International Conference on Network Protocols* (2013).

[41] MILLS, D. L. Internet time synchronization: the network time protocol. *IEEE transactions on Communications 39* (October 1991), 1482–1493.

[42] MIZRAHI, T., AND MOSES, Y. Software Defined Networks: It's about time. In *Proceedings of the IEEE International Conference on Computer Communications* (2016).

[43] MOREIRA, P., SERRANO, J., WLOSTOWSKI, T., LOSCHMIDT, P., AND GADERER, G. White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication* (2009).

[44] OGDEN, B., FADEL, J., AND WHITE, B. IBM system z9 109 technical introduction.

[45] OHLY, P., LOMBARD, D. N., AND STANTON, K. B. Hardware assisted precision time protocol. design and case study. In *Proceedings of the 9th LCI International Conference on High-Performance Clustered Computing* (2008).

[46] PÁSZTOR, A., AND VEITCH, D. PC Based Precision Timing Without GPS. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (2002).

[47] PERRY, J., OUSTERHOUT, A., BALAKRISHNAN, H., SHAH, D., AND FUGAL, H. Fastpass: A centralized "zero-queue" datacenter network. In *Proceedings of the ACM Conference on SIGCOMM* (2014).

[48] SCHNEIDER, F. B. Understanding Protocols for Byzantine Clock Synchronization. Tech. Rep. TR87-859, Cornell University, August 1987.

[49] SOBEIH, A., HACK, M., LIU, Z., AND ZHANG, L. Almost Peer-to-Peer Clock Synchronization. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium* (2007).

[50] VEITCH, D., BABU, S., AND PÀSZTOR, A. Robust Synchronization of Software Clocks Across the Internet. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement* (2004).

[51] ZARICK, R., HAGEN, M., AND BARTOS, R. The impact of network latency on the synchronization of real-world IEEE 1588-2008 devices. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication* (2010).

[52] ZARICK, R., HAGEN, M., AND BARTOS, R. Transparent clocks vs. enterprise ethernet switches. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication* (2011).

[53] ZENG, H., ZHANG, S., YE, F., JEYAKUMAR, V., JU, M., LIU, J., MCKEOWN, N., AND VAHDAT, A. Libra: Divide and conquer to verify forwarding tables in huge networks. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation* (2014).