Computer Communications 191 (2022) 26–35

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Review

Evolution of network time synchronization towards nanoseconds accuracy: A survey

Djalel Chefrour

Mathematics and Computer Science Laboratory, University of Souk-Ahras, BP 1553, 41000, Algeria

ARTICLE INFO

Keywords:
Network time protocols
Time synchronization
Clock skew estimation and removal
Network delay measurement

ABSTRACT

We expose the state of the art in the topic of network time synchronization. Many distributed applications require a common notion of time to function properly. Without time synchronization, the nodes clocks will drift and report different values for the same instant. This problem is exacerbated by varying network delays between the cooperating nodes. Our survey covers how this issue is tackled by standard time synchronization mechanisms and a representative range of recent research works. We expose how some of them achieve micro and nanoseconds accuracy in wired networks. The reviewed techniques are classified in two categories based on whether they change the hosts clocks or not. The latter category includes schemes that detect and remove clock skew from network traffic trace. We discuss the advantages and drawbacks of the techniques in each category; compare them according to their application environment, accuracy and cost; and conclude this survey with a summary of learned lessons and insights into future work.

Contents

E-mail address: djalel.chefrour@univ-soukahras.dz.

https://doi.org/10.1016/j.comcom.2022.04.023
Received 9 September 2021; Received in revised form 28 February 2022; Accepted 19 April 2022
Available online 26 April 2022

1. Introduction

The clocks of the collaborating nodes in a computer network are said to be synchronized if they all report the same value at the same instant in time. For that purpose, they need to agree on a particular starting value, which is often an epoch with respect to Coordinated Universal Time (UTC). Additionally, the clocks frequencies need to be adjusted, so they run at the same speed [1].

The advantages of using synchronized clocks in distributed systems have been identified as early as 1991 [2]. These include message ordering, authentication token expiration, replay attacks prevention, cache consistency and file system replication. Nowadays, many applications require the notion of synchronized time between networked hosts. Enumerating them is beyond the scope of this study. Nonetheless, we show hereafter the importance of this topic through some detailed examples:

- The measurement of time-dependent network metrics such as the packet one-way delay (OWD) [3]. We showed in a recent survey about OWD measurement that a lot of solutions use one mechanism of time synchronization or another between the sender and the receiver nodes [4].
- The scheduling of network configuration and maintenance activities. For example, in modern software defined networks (SDN), the update of multiple OpenFlow switches with new routing rules requires the synchronization of their clocks with the controller one [5].
- The analysis of network nodes logs after a service failure, a performance drop or a security breach. For instance, digital forensics experts compare the events timestamps in several hosts to trace back an attacker path in the network. Their findings would not be credible if the hosts and therefore their logs do not share the same notion of time.
- The decision making in distributed applications that rely on correct file timestamps in the network. That is the clock of the node running the application is synchronized with the one hosting the files it manipulates. Many programs such as the build tool `make`, the search utility `find` and the archive application `tar` belong to this category.
- Time-Sensitive Networking (TSN) which is a set of IEEE standards that aim to achieve real-time communication in Ethernet networks [6]. Particularly, time synchronization in TSN is described by the IEEE 802.1AS standard [7], which itself relies on a specific profile of the Precision Time Protocol (PTP) discussed in this survey. TSN is used in many applications such as audio–video bridging, safety-critical systems, Internet of Things (IoT) and Industry 4.0 networks [8].

This survey of network time synchronization is motivated by the recent advances in this mature topic, which achieve sub-microsecond accuracy in fast local area networks like datacenters an high performance computing clusters. It is also motivated by the lack of a literature review that covers such advances. Our main goal is to focus on time synchronization in wired networks. Hence, we do not address this topic in the context of wireless networks. This is due to the grand variety of existing wireless technologies (*e.g.,* Bluetooth, WLAN, wireless sensors and cellular networks). Covering all of them requires a separate survey, perhaps a dedicated one for each technology. For example, there are already surveys from Sivrikaya and Yener [9]; and Sundararaman et al. [10] devoted to the topic of time synchronization in sensor networks alone. However, we make an exception for the Global Positioning System (GPS), which uses wireless satellite communications to synchronize nodes clocks. The reason for this exception is that GPS can be used in wired networks as an external time reference, for instance, in hardware packet timestamping and in Network Time Protocol (NTP) Stratum 1 servers. Both use-cases are covered in this survey.

Many synchronization protocols exchange timestamped packets between the network nodes to estimate their clocks errors and correct them. As any network communication takes a certain delay, timing inaccuracies might occur in these protocols when the packet delay varies. Delay variation (*i.e.,* jitter) can be caused by many network effects such as nodes processing load, queuing, traffic congestion, packet loss, routing changes, etc. In addition to its variable part, the packet delay in a wired network is also composed of a deterministic part that is not null even if the network conditions are good. This is made of the transmission and the propagation times. The former corresponds to serialization of the packets bits by the Network Interface Card (NIC). The latter consists of the time needed for the bits to reach their destination, which happens at the speed of signal propagation through the network medium [4]. Overall, to tackle the challenge of network time synchronization, the proposed solutions must take such factors into account or circumvent them.

We divide the topic of network time synchronization in two categories based on whether the clocks of the nodes are modified or not. In the first case we talk about clock synchronization protocols. In the latter, we discuss skew detection and removal mechanisms, which rectify the timestamps of a traffic traces only. In addition to summarizing the important techniques in both topics, we analyze and compare them according to the following usage criteria: (i) **application environment**, that is in what type of network a technique can be used; (ii) **accuracy**, which is expressed in the two categories of the reviewed works, respectively, as the absolute and the relative error in measurement (the latter being: $|true\_value - estimated\_value|/true\_value$) and (iii) **cost**, which includes the network overhead and any special hardware required by a technique. The cost determines also the scalability.

The contributions of this paper include:

- A clarification of the clock nomenclature (Section 2) and the clock formal model (Section 3) used in the related literature.
- An analyses of where and when packet timestamps are generated and how they are encapsulated (Section 4).
- An up to date state of the art of clock synchronization protocols with a comparison based on their usage criteria (Section 5).
- A detailed review and comparison of clock skew estimation and removal algorithms (Section 6).
- A summary of learned lessons and a reflection on future research works (Section 8).

2. Clock nomenclature

A clock in our context is a device that keeps track of the time of day. It is a combination of a base value, which is often an epoch with respect to a time standard such as UTC; and a counting mechanism that advances this value to reflect the progress of physical time. Therefore, a clock can be implemented using a base register and a counter. Alternatively, a counter can act as both when it is set to the time of day and counts up from that point. In the clock model described below, we assume the counter plays both roles. Clock errors can occur when the

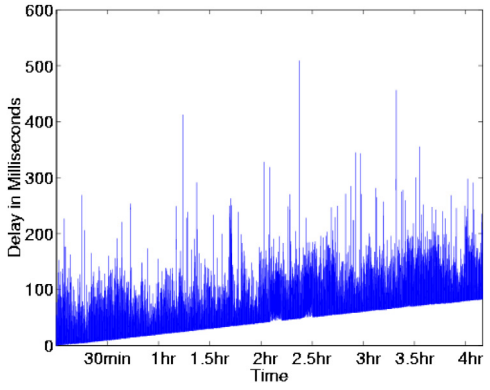

Fig. 1. Increasing network one-way delay caused by clock skew [12].

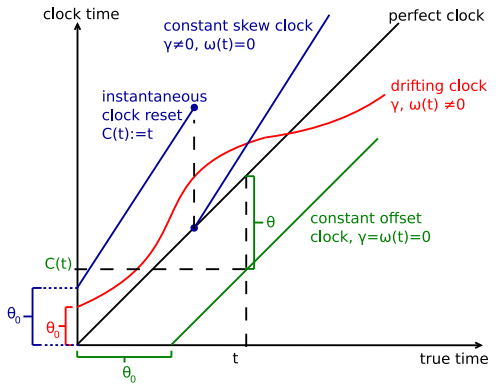

Fig. 2. Simple clock skew model. *Source:* Adapted from Ridoux and Veitch [15].

base value is initially set and/or when the counter is not 100% precise (and no counter is).

In a computer, several clocks can be present in different places to serve diverse purposes, such as timestamp generation. Clocks can be embedded in software, for example, to maintain time in virtual machines (VMs). They can also be embedded in hardware, for instance, NICs can have their own clocks separate from those used by the CPUs. Additionally, various types of counters (*e.g.,* High Precision Event Timer (HPET), Real Time Clock (RTC) and Time Stamp Counter (TSC)) can be used to count the passage of time, but some might not themselves know the time of day.

Throughout this article we use the clock nomenclature defined in Mills's work about the Network Time Protocol [11]. Specifically, a clock's *resolution* is the minimum unit by which its time is changed (a *tick*). A clock's *offset* at a particular moment (*i.e.,* its error) is the difference between the time reported by the clock and the "true" time as defined by UTC. A clock's *accuracy* is how close the absolute value of its offset is to zero. A clock's *frequency* is the rate at which it progresses.

Two clocks that are not running at the same frequency exhibit a relative *skew*, which is represented as the ratio or the difference between their frequencies. Skew is measured in seconds per real second or in the dimensionless unit: parts per million (ppm). For instance, if a crystal clock has a skew of 20 ppm, then it will register an error of ±20 seconds after one million real seconds. In other words, its absolute error per day will be $(24 * 60 * 60) * (20 * 10^{-6}) = 1.728$ seconds.

A clock's *drift* indicates the variation in its skew due to fluctuations of the clock's frequency itself in time. It is closely related to environmental conditions such as temperature variations and other factors (*e.g.,* quality of oscillator, aging and line voltage). The skew and the drift of a clock are respectively the first and the second derivative of its offset with respect to true time.

If the clocks in a network are not synchronized then time-dependent tasks will have issues. For instance, the OWD measured between a sender and a receiver will contain the offset of their clocks even if they run at the same rate. Moreover, if the clocks frequencies are different (*i.e.,* their skew is not null), this OWD will vary in time. It will do so in a way that cannot be distinguished from legitimate network effects like queuing [12]. This problem can even happen for clocks that have the same nominal frequency, because the actual frequency might also vary in time (*i.e.,* their drift is not null).

In the particular case where the clock skew is constant, the minimum OWD of a fixed path and packet size will appear to increase (as in Fig. 1) or decrease over time. In such a case, the linear coefficient of the skew corresponds to the slope of line beneath the OWD points. The intercept of this line corresponds to the sum of the clock offset and the minimum OWD at the beginning of the measurement.

3. Clock model

Veitch et al. propose a formal model of simple clock skew which is adopted by several research works [13]. In this model, the skew is considered as the deterministic linear component of the drift. The latter contains also a random component, which is less important than the skew for small time scales (*i.e.,* below 1000 s).

$$\theta(t) = C(t) - t = \theta_0 + \gamma t + \omega(t) \tag{1}$$

where $C(t)$ is the clock value at instant t of true time, $\theta(t)$ is the offset, θ_0 is the initial value of the offset, γ is the simple skew and $\omega(t)$ is a remainder that encapsulates nonlinear deviations. For ordinary quartz-based clocks, the coefficient γ is usually given by the quartz oscillator data-sheet in the range of 10 to 100 ppm [14]. The relation between a clock's offset, skew, drift and true time is depicted in Fig. 2. When a clock is synchronized with another reference clock, its value is *adjusted/reset* instantaneously or smoothly by varying its frequency over a small period of time.

This simple skew model corroborates earlier observations from Paxson about TCP connections that span 120 s [16]. He found that the relative skew is nearly linear between clock adjustments. Nonetheless, the assumption of linear skew does not hold for small time scales in the presence of extreme temperature variations. Such would be the case with sensor networks deployed in harsh conditions. Sugihara and Gupta [17] conducted experiments to measure quartz oscillators frequencies in varying temperatures. They reported an increase of 0.6 kHz at 20°C from a nominal frequency of 32 768.5 kHz at 0°C. That is a ratio of 0.6/32768.5 or 18.31 ppm, which is not far from the ±25 ppm shift reported earlier by Vig [18] for a temperature variation in a wider range (−55°C to 85°C).

For larger time scales, the drift is also important and must be tracked, at least by bounding it. We note that in traditional distributed systems literature, the terms skew and drift are used in a slightly different way. Namely, the term skew is used to denote the offset as defined above and the term drift is employed in place of skew [19–21]. In our opinion, this difference is due to a more general model of drift, which is used in the larger time scales of the distributed systems. In this model the drift is considered as an arbitrary variable with bounded values. The bound is a small constant denoted ρ, which delimits the tolerance of the clock frequency $F(t)$ around its nominal value F_0. That is:

$$\forall t : (1-\rho) \leq \frac{F(t)}{F_0} \leq (1+\rho), \quad \frac{\partial C(t)}{\partial t} = \frac{F(t)}{F_0} \tag{2}$$

For this survey we adopt the first nomenclature as it was specifically coined for network time synchronization and is the most used in the related literature.

4. Network packet timestamping

Network packet timestamping refers to the operation of capturing the transmission or reception time of a packet and piggybacking this information on the packet itself in a special field called *timestamp*. We focus on this operation because, in addition to recording network events, it is useful for synchronizing clocks. Packet timestamping techniques have been classified by Orgerie et al. into three categories based on their location: hardware, firmware or software [22].

Hardware timestamping is performed by special NICs like the DAG cards, which rely on GPS to synchronize their internal clocks with nanosecond level accuracy [23]. They give the most accurate timestamps for packet departure and arrival events, which are called *wire-times* in the IETF definition of OWD [3]. However, they are the most expensive.

The firmware techniques try to obtain a good accuracy while using commodity hardware. The idea is to let the firmware of a programmable NIC stamp the packets using a good reference time such as PTP.

Software timestamps are sometimes generated in user space. These are called *host-times* in IETF parlance. They have the worst accuracy because of the latency of context switching, system scheduling and packet copying between the application and the kernel space. Donnelly et al. compared user space timestamps which are based on a system clock synchronized with GPS to DAG wire-times. They observed that the time error is within $50\,\mu s$ at a 95% confidence level [24].

Software timestamping is usually done in the kernel by a packet filtering component like the Berkeley Packet Filter (BPF) [25]. In-kernel timestamps include a latency of few tens of microseconds induced by the operating system load and the NIC driver latency [26,27]. In detail, there is a transmission latency between the moment where the stamp is generated in the networking stack and the real departure of the packet on the network link. NTPv4 accounts for this latency in interleaved mode by generating the stamp – which is called *softstamp* – right before the packet is buffered in the NIC output queue [28]. As it cannot be added to this packet, it is sent with the immediately following one. Similarly, there is a reception latency between the arrival of the (last bit of the) packet and the moment when it is stamped by the network stack. This stamp – which is named *drivestamp* – is generated in NTPv4 interleaved mode shortly after the input device interrupt and before the packet is buffered in the input queue.

Regarding encapsulation, timestamps appear at different network layers depending on the protocol that uses them. They can be part of the packet payload or its header as in P4 In-band Network Telemetry [29] and NTP. More specifically, the NTP header (which is carried over UDP) uses timestamps coded in 64 bits. The highest 32 bits represent the seconds since the first of January 1990, while the other 32 bits contain fractions of seconds [11].

The layer where timestamps are encoded is very relevant for their usage. One the one hand, link layer timestamps limit their usage to local networks only. On the other hand, timestamps encapsulated in the payload of IP packets might be impacted by fragmentation. Indeed, if the nodes that fragment and reassemble packets are interleaved with the measurement points, the latter will neither see the same number of packets, nor be able to find the timestamps. For this reason, RFC 2679 (about OWD measurement) considers fragmented packets as lost [3]. Alternatively, we suggest to set the Do not Fragment (DF) bit of probe packets to avoid this issue or limit their size according to their path maximum transmission unit (MTU).

5. Clock synchronization mechanisms

In the following two subsections, we review respectively the common standard synchronization protocols: GPS, NTP and PTP; and a representative selection among the more recent clock synchronization mechanisms: Alg1, TSCclock, CCT, DPT and Huygens. They all have in common the fact they modify the network nodes clocks to keep them synchronized, so applications hosted on these nodes read the same time with a certain accuracy. We then compare these mechanisms according to the retained criteria.

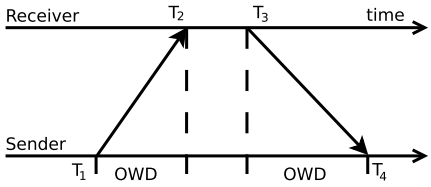

Fig. 3. Using timestamped messages for the delay and clock offset estimation (NTP and PTP).

5.1. Standard clock synchronization protocols

5.1.1. Global positioning system

The Global Positioning System can provide sub microsecond accuracy by relying on the atomic clocks in its satellites. Each GPS satellite broadcasts radio signals repeatedly every 1 millisecond. These signals carry navigation messages that provide position and time information. The latter includes the offset between GPS time and UTC, which is specified to be ≤ 40 nanosecond 95% of the time [30]. The total accuracy of the UTC offset at the receiver depends on this time information and on other factors like its antenna location, the number of satellites in view and the atmospheric interference. It is in the order of hundreds of nanoseconds [31].

A GPS node needs to receive the signals of at least four satellites to derive the GPS time and three position coordinates by triangulation. However, using GPS to synchronize network clocks requires a hardware receiver per node which is not a scalable solution. In addition to its cost at large scale, GPS might not be affordable too for certain applications due to node size and power consumption (*e.g.,* sensor networks). Furthermore, GPS requires an unobstructed line of sight, so it may be unusable if the satellite signals are not accessible (*e.g.,* indoors) or are disrupted (*e.g.,* during solar flares) [32].

5.1.2. Network time protocol

The Network Time Protocol is the most deployed solution. It distributes time over the internet using a hierarchical client–server model [11]. The client, which is the initial sender, exchanges periodic messages with the NTP server using a poll interval that increases gradually. Each NTP message contains the latest three timestamps of the exchange between the two nodes (see Fig. 3). NTP uses software, firmware or hardware timestamping based on the available equipment and the administrator configuration. When an NTP message returns to its sender, the fourth timestamp becomes known, so the round trip time (RTT) and the clock offset can be measured respectively as $(T_4 - T_1) - (T_3 - T_2)$ and $[(T_2 - T_1) + (T_3 - T_4)]/2$. Only the smallest observed RTTs are considered to filter out the effect of queuing in the network. However, path delay asymmetry is not taken into account. The measured offset is used to adjust the client clock smoothly in a control loop. This loop varies the clock value and/or the frequency of its oscillator, while maintaining its skew below 0.01 ppm. Additionally, the client can check for packet loss and out of order delivery by verifying the first timestamp in each incoming response, which should match the transmission timestamp of its previous request.

Each level of the NTP hierarchy is called a *Stratum*. At the top level, NTP servers (from Stratum 1) use precise external time sources such as atomic clocks or GPS (*i.e.,* Stratum 0). But due to variable network latency the precision degrades from few microseconds at the top nodes, as achieved by NTPv4 in a local area network (LAN), to the order of tens of milliseconds in the lower hierarchies, as in wide area networks (WAN) [28].

5.1.3. Precision Time Protocol

Similar to NTP, the IEEE 1588 Precision Time Protocol (PTP) calculates periodically the relative offset between a master clock and slaves' clocks by exchanging timestamped messages [33]. However, PTP usually uses Ethernet PHY hardware clocks (PHC) in the NICs of the master, the slaves and the intermediate switches to timestamp its messages. In one-step PTP, a message about to leave the device carries its own timestamp. Conversely, in two-step PTP, a follow-up packet is used to communicate the exit time of the previous message. PTP is best suited for LANs and is often used in industrial automation. Its accuracy varies from tens to hundreds of nanoseconds depending on LAN traffic load and more importantly on the quality of its PHC oscillators. PTP has the advantage of electing the best master automatically. This election is based on the accuracy of the participating nodes clocks, their variance and user assigned priorities.

5.2. Non standard clock synchronization mechanisms

5.2.1. TSCclock

The Time Stamp Counter clock (TSCclock), also known as RADclock, is proposed as a middle ground between the accurate but expensive PTP; and the cheap and widely available but less accurate NTP [13]. This solution provides two clocks named *difference* and *absolute* that are based on the Time Stamp Counter (TSC) register present in commodity hardware. TSC is a PC CPU cycle counter that has a rate error of 0.1 ppm and is therefore highly stable. TSCclock relies also on NTP exchanges between the host to synchronize and an NTP Stratum 1 server.

On the one hand, the *difference clock* is derived from the TSC register by multiplying its value with its frequency. This clock is used to measure the offset between the host local events within time scales below 1000 s. In particular, it is used to detect the minimum RTT between the host and the NTP Stratum 1 server. Furthermore, the NTP timestamped packets that have the minimum RTT (*i.e.,* quality packets) are used to synchronize the frequency of the TSC register.

On the other hand, the *absolute clock* is defined off the difference clock by subtracting from it its own error. This error is estimated from the timestamps of the packets exchanged with the NTP server. However, contrary to NTP which modifies the client clock to filter out this error, TSCclock does not adjust the rate of its absolute clock. Instead, it subtracts the error from the value of this clock when it reads it. The absolute clock is used for timestamping events with sub-millisecond accuracy.

5.2.2. Coordinated Cluster Time

The Coordinated Cluster Time (CCT) mechanism uses the Timestamp Exchange Protocol (XTP) to synchronize the clocks within computer clusters [34]. The client nodes exchange their timestamps with a reference node (called the XTP server) four to ten times every two seconds using low-latency messages. The implementation of XTP relies on Cisco's InfiniBand switch, which is an Input–Output (IO) fabric that supports remote Direct Memory Access (DMA). InfiniBand is characterized by its high speed and low latency. The implementation of CCT was evaluated with InfiniBand connections at a speed above 2.5 Gbps, an RTT below few microseconds and a jitter less than 100 nanosecond. Recently, InfiniBand networks reached speeds up to 400 Gbps and latencies in the range of hundreds of nanoseconds [35].

On the one hand, the multiple exchange of XTP messages warms up lookaside structures such as the cache and the translation lookaside buffer (TLB). On the other hand, remote DMA avoids system calls and their added latency during IO. Therefore, XTP is freed from system noise. Additionally, CCT subjects the captured one-way delays to convex hull filtering to estimate the clock offset and skew at each client. These estimations are used continuously to update a piecewise linear mapping of the TSC counter to the reference time. Hence, CCT can converge to one microsecond accuracy after around 85 s of operation. Nonetheless, its usage is limited to high-performance clusters.

5.2.3. Alg1

Alg1 is another algorithm that shares CCT code base and achieves microsecond level accuracy within IBM BladeCenter clusters [36]. It is qualified as a skewless algorithm because it does not perform clock skew estimation and compensation. Alg1 rather corrects the skew smoothly by considering both the current offset and the exponential weighted average of the past offsets. The current offset is measured using the TSC register and the XTP packet exchanges between neighbors.

Alg1 is robust to clock jitter because it does not allow abrupt offset changes. Additionally, it can be responsive to clock drift if the network contains a leader node. However, Alg1 convergence requires fine parameter tuning and is dependent on the number of neighboring nodes. Moreover, as with CCT, this algorithm is limited to high end cluster environments.

5.2.4. Huygens

Huygens is a clock synchronization protocol for datacenters with tens of nanoseconds accuracy [37]. Each Huygens node probes 10 to 20 other nodes using closely spaced pairs of packets. This probing happens in intervals of two seconds where the clock skew is assumed linear. The probes transmit and reception timestamps are generated by the NICs hardware. Comparing them gives an upper and a lower bound on the offset between the clocks. Only the pairs that have an inter-arrival time very close to the inter-departure one are kept for farther filtering. The latter is performed using the Support Vector Machines (SVM) classifier, which is usually used in supervised learning.

In general, SVM separates the points in a data set with a hyperplane according to a binary label such as "upper bound point" or "lower bound point". The hyperplane is at a maximum distance from the closest point of either label in the data. In Huygens, SVM returns a line which slope is the skew between a pair of clocks and its intercept is the initial relative offset.

Huygens applies further corrections to the nodes clock offsets at the midpoint of the next two seconds interval. These corrections are based on the network effect where comparing many clocks allows the detection and correction of noise, which is caused by asymmetric delay paths or congestion under high load. To alleviate the processing overhead and allow scalability, the calculations are distributed between slaves that perform pair-wise SVM filtering and a master that performs the network effect corrections. Nonetheless, the bandwidth overhead of the probes remains important.

5.2.5. Datacenter Time Protocol

Datacenter Time Protocol (DTP) synchronizes datacenter clocks with nano-seconds bounded accuracy by using a decentralized scheme in the physical layer [38]. DTP exploits the fact that two Ethernet ports physically connected by a cable are already synchronized in the PHY to exchange bit-streams reliably. When there is no traffic, the delay between these ports is only the propagation delay which is constant and bounded by the cable length. DTP enabled servers and LAN switches exchange their PHY oscillators counters frequently and then use the maximum observed one as a global synchronized clock that increments monotonically.

The DTP messages (of 53 bits) are carried in modified Ethernet Idle characters which are inter-packet gaps required by the Ethernet standard between any two frames. That is at a frequency of 1.28 per microsecond in each direction for a 10 Gbps link saturated by MTU-sized frames. Hence, unlike the other protocols, DTP provides bounded accuracy without the overhead and non-determinism of the higher network layers. However, it requires changes in the network hardware and is limited to full-duplex datacenter Gigabit Ethernet.

Table 1
Clock synchronization protocols environment and accuracy.

Protocols	Environment	Accuracy
NTP	Any network	10s of µs to 100s of ms
TSCclock	LAN	few 10s of µs
CCT	Cluster	few µs
Alg1	Cluster	few µs
GPS	Unobstructed Line of Sight	100s of ns
PTP	Fast LAN	10s to 100s of ns
Huygens	Datacenter	few 10s of ns
DPT	Datacenter	few 10s of ns

Table 2
Clock synchronization protocols cost.

Protocols	Special hardware	Sync. interval
GPS	GPS receiver	1 ms
NTPv3	No	64 s to 17 m
NTPv4	No	64 s to 36 h
PTP	NIC hardware clocks	0.5 to 2 s
TSCclock	TSC register	256 s
CCT	TSC register + InfiniBand	0.2 to 0.5 s
Alg1	TSC register + InfiniBand	1 s
Huygens	NIC hardware clocks	0.1 to 0.2 s
DPT	Modified Ethernet PHY	1.28 µs

5.3. Comparison and discussion

Tables 1 and 2 summarize and compare the previous clock synchronization protocols with respect to their most relevant usage criteria. That is the application environment, the accuracy expressed as the time error between clocks and the cost defined in terms of special hardware and network overhead. The latter depends on the synchronization interval, which determines the probing rate. This parameter is configurable by the user in most cases. All these protocols work online to correct the clocks of the involved network nodes and can therefore provide a near real-time estimation of the network OWD.

Table 1 is sorted with respect to the increasing accuracy of the reviewed protocols, whereas Table 2 is ordered chronologically within each sub-category (*i.e.,* standard and non-standard protocols, which are separated by a horizontal line). We note that the accuracy became finer over time along with the advent of faster networks and application environments. It evolved towards microseconds scale with TSCclock and Alg1, down to few tens of nanoseconds with Huygens and DTP. Though, this improvement is made possible with an additional hardware cost, which is nowadays acceptable for datacenters.

6. Skew estimation and removal techniques

We review in this section a broad multiplicity of skew estimation and removal techniques, which synchronize the timestamps of traffic traces without adjusting the network nodes clocks. Removing the estimated skew from the trace yields a better approximation of the OWD. This is sometimes called frequency synchronization as it eliminates the effects of the difference in the clock frequency between measurement points. However, it does not estimate the initial clock offset (θ_0), unless two way measurements are available from symmetric paths. We label the techniques below for which the authors did not give a name based on their most important idea.

6.1. Cumulative minima

Paxson's heuristics can detect the relative offset of a clock in addition to its skew and resets. They do so offline by analyzing timestamped bi-directional Internet traffic traces [16]. First, the clock relative offset can be approximated by comparing the overall minimum OWD of the forward path versus the reverse one, assuming both paths are delay symmetric. Second, the N OWD measurements in each direction are de-noised by partitioning them into $\sqrt{N}$ segments and keeping the minimum OWD in each one. This allows the complexity of the detection of clock resets and skew to remain linear ($O(N)$) even when using sub-algorithms that run at $O(N^2)$. The clock resets are identified by looking for pivot points in the de-noised OWDs where there is a shift of the delay in one path, which coincides with a similar shift in the reverse path that is equal in amplitude and opposite in direction.

The skew is detected by a robust line fitting algorithm based on statistical tests that distinguish real skew from network effects like queuing fluctuations. The main test identifies negative skew slopes by calculating the number of cumulative minima in the de-noised OWDs over a period of time. It is based on the fact that the smallest OWD continues to decrease or increase because of the skew. The number of found minima should be close to the number of de-noised measurements observed in that period. A similar test is applied to detect positive skew slopes by counting the minima working in reverse from the last measurement to the first one. These heuristics are limited to small time scales (*i.e.,* below 120 s) where the skew remains linear.

6.2. Linear programming

Moon et al. formulate clock skew estimation as an optimization problem and solve it using linear programming [12]. They propose an algorithm that fits a line, the closest possible, under the OWD measurement points (see Fig. 1). That is to minimize the sum of vertical distances between the line and these points. The slope of this line gives an estimate of the sender's clock skew relative to the receiver. However, the clock offset cannot be distinguished from the fixed portion of the network delay by using one way measurements only [32]. Nonetheless, this linear algorithm is robust against queuing fluctuations and changes in the skew magnitude. Its relative error is less than 4% and does not increase with the skew magnitude, which is not the case of Paxson's method [12].

6.3. Convex hull

Zhang et al. adopt an approach similar to the previous one, but based on finding the convex hull of the observed OWD points [39]. They propose an objective function that minimizes the surface between the lower boundary of this convex hull and the line supporting the skew. In addition, they employ an algorithm that detects both types of clock resets: instantaneous (abrupt) changes and clock frequency adjustments. This algorithm works by finding a piecewise linear function where each piece has the same slope. However, it can give false positives in the case of congestion and large delay jitter. Both algorithms are linear and can be applied online incrementally. This is possible because the convex hull is built by parsing the measurement points in increasing order while using a stack to memorize its segments.

6.4. Simple average

Khlifi and Grégoire present two offline and two online methods that deal with clock skew in OWD traces [40]. The first method estimates the skew as the ratio of the difference between packet delays to the inter-departure time of these packets. However, only two OWD points are considered from the trace. These are the first and the last minimum OWDs observed in one minute intervals at the beginning and the end of the trace, respectively.

The complexity of this method is constant relative to the whole trace ($O(1)$). Nevertheless, it requires that the two points of interest be sufficiently away of each other, so the effect of the skew remains apparent. Thus, it works only for traces with a very large number of samples. Moreover, it is not robust to queuing that coincides with at least one of the extremities of the trace. If this queuing lasts longer than the one minute interval, this method will select a false minimum and therefore give an invalid skew estimation.

6.5. Resolution based correction

The second offline technique from [40] tracks the increasing effect of the clock skew on the OWD in terms of the clock resolution, that is the unit of time measurement. This unit is set to one millisecond, which the authors consider as the clock resolution of common computers. The traffic trace is subdivided into successive intervals where the OWD increases or decreases by one millisecond, while remaining above the overall minimum measured value. The advantage of this method is the correction of the OWD in linear time by removing the skew effect without estimating it. Therefore, without yielding delays with fractions that are below the clock resolution, as it can happen with other methods. However, this technique does not account for the extra variable delay induced by queuing. It will rather interpret the latter as an increased skew trend.

This skew correction method is augmented with a clock reset detection scheme, which identifies resets by looking for drops of the OWD that happen in constant periods of times. Nevertheless, this scheme will miss the clock resets that coincide with large queuing in the network.

6.6. Sliding window

The first online method from [40] is very similar to the previous one. However, it tracks the changes in the minimum observed OWD using a sliding window mechanism. The window size is assumed to correspond to one millisecond error in the OWD caused by the clock skew. As the latter is unknown in the first place, it is hard to choose the right window size which maintains a high probability of observing a minimum OWD in each cycle. This task becomes impossible if queuing is considered as its effect on the OWD can be highly variable, contrary to skew.

The authors propose a second online method as an improvement of the sliding window mechanism by combining it with the previous convex hull algorithm. The idea is to start with the sliding window to yield a quick estimation of the skew, even if it is less accurate, then repeat the convex hull method periodically. This interesting combination covers for the slow convergence of the convex hull scheme which is common to all the line fitting techniques above. Indeed, a common trait of these techniques is the need for more and more data points to increase their accuracy.

6.7. Piece-wise reliable clock skew estimation algorithm

Bi et al. propose the Piece-wise Reliable Clock Skew Estimation Algorithm (PRCSEA) [41]. It uses piece-wise linear equations to model clock resets and to approximate drift over long OWD measurement periods. The idea is to identify the intervals where the skew remains constant and the points when it changes because of clock resets and drift. For that purpose, PRCSEA partitions the OWD trace recursively into smaller intervals and uses convex hull line fitting to estimate the skew in each interval. The reliability of this estimation is tested based on the probability that at least a certain number of points in each interval belong to its skew supporting line (*i.e.,* their packets experienced the minimum OWD). This probability is fixed empirically to 5%. If the test fails, the concerned interval is subdivided further until it passes the test or the minimum allowed interval is hit. The latter is fixed to 100 packets or 20 s span.

The reliability test of PRCSEA is an addition compared to the previous methods. Combining this test with piecewise linear functions yields a good fit for the skew of a clock that is adjusted at fixed periods. The OWDs in this case will have the shape of saw-tooth function. However, if the network experiences queuing periods that are longer than the partitioned intervals, then PRCSEA might confuse the inflection point at the peak of the queue with a clock reset. Therefore, it will validate wrong estimations of the skew. In the general case, this drawback makes the choice of the appropriate length for the minimum allowed interval difficult.

6.8. Linear regression

Aoki et al. proposal for skew estimation is based on the measurement of OWD variations (OWDV) [42]. It is limited to traffic flows with fixed packet size and constant inter departure time (like voice over IP). The receiver subtracts the packet inter departure time from the observed inter arrival time to obtain the OWDV data points. This act cancels out the deterministic part of the OWD to leave the variable part that is caused by skew and queuing. The OWDV points are segmented over time into periods of 2 s and the minimum point of each segment is selected. Then, linear regression is applied on these points to fit the line supporting the clock skew. The simulation of this technique with an inter departure time of two milliseconds estimates the skew with a relative error of 3.7%. However, as emphasized by Paxson [16], least-squares fitting is not robust to queuing. Elevated queuing in only one period of observation will through off the fit. In other terms, the skew will be mistakenly amplified by the queuing effect.

6.9. Timestamp based incremental clock synchronization

Harrison and Newman present Timestamp based Incremental Clock Synchronization (TICSync) for the estimation of the clock offset and skew [43]. TICSync uses a client–server model with request–response messages. It is based on the observation that the negation of the minimum request OWD constitutes a lower bound for the clock offset. In the same way, the minimum response OWD represents its upper bound. Hence, the parallel lines that fit these bounds form a corridor around the true offset. TICSync assumes that the clock skew is constant for periods of ten minutes. It aims at finding a pair of maximally separated parallel lines which pass between the corridor bounds. TICSync achieves this by using the convex hull algorithm to find a lower and an upper hull. Then, it seeks the instant of time of minimum vertical distance between the two hulls. The slope of the hull segment at this instant gives the slope of the maximum separation lines and therefore the skew.

The authors propose an online incremental implementation of TICSync that adds each new OWD measurement to the hulls in a constant time, then repeats the last step. Furthermore, by assuming a Weibull distribution of the delay, TICSync provides probabilistic bounds on the estimation errors of the offset and the skew. Conversely, it can also determine the minimum number of required measurements to stay below a particular error level.

6.10. Weighted average of slopes

The technique proposed by Eylen and Bazlamaçcı performs clock skew estimation for the purpose of correcting OWD measurements [44]. First, the receiver monitors a window of N probe packets sent with a fixed inter-departure time. It selects the m probes that did not suffer queuing. These are called non delayed packets. Their selection assumes maximum values for the clock skew and drift based on the characteristics of the operating environment (*i.e.,* bounded temperature changes with a bounded rate of change). It also assumes that at least some non delayed packets exist in each observation window. Second, the skew is estimated from the m probes using a weighted average of the slopes of the lines formed by all possible OWD pairs. A bigger weight is given to the pairs that are close in time and the ones which are more recent. Third, the subsequent packets OWDs are corrected by removing the estimated skew.

Finally, an error analyses based on the previous assumptions is performed to bound the OWD measurement error. The skew bound itself is re-estimated for every window to accommodate for drift, which is another advantage. However, this technique does not consider clock

Table 3
Clock skew estimation and removal methods usage environment.

Methods	Network	Online	Queuing
Cumulative minima	wan		✓
Linear programming	wan		✓
Convex hull	wan	✓	✓
Simple average	man		
Resolution based	man		
Sliding window	man	✓	
PRCSEA	wan		✓
Linear regression	wan		
TICSync	wan	✓	✓
Weighted average	lan	✓	✓

Table 4
Detection of clock effects in skew estimation and removal methods.

Methods	Offset	Skew	Reset	Drift
Cumulative minima	✓	✓	✓	
Linear programming		✓		
Convex hull		✓	✓	✓
Simple average		✓		
Resolution based		✓	✓	
Sliding window		✓	✓	
PRCSEA		✓	✓	✓
Linear regression		✓		
TICSync	✓	✓		
Weighted average		✓		✓

Table 5
Accuracy and complexity of clock skew estimation and removal methods.

Methods	Accuracy		Complexity
	Bounded error	Relative error	
Cumulative minima			$O(N)$
Linear programming		4%	$O(N)$
Convex hull			$O(N)$
Simple average			$O(1)$
Resolution based			$O(N)$
Sliding window			$O(1)$
PRCSEA			$O(Nlog(N))$
Linear regression		3.7%	$O(N)$
TICSync	✓		$O(N)$
Weighted average	✓		$O(m^2)$

Legend: N=# OWD measurements, m=# non delayed packets, $m < N$.

resets. As it has a complexity of $O(m^2)$, it cannot be applied online except for a small m. Additionally, its measurement of the OWD can be biased by network synchronization effects because its probing interval is fixed. That is the probes might coincide with another periodic network activity, so they will always suffer some queuing. Moreover, as this interval is small (20 ms), the number of probes will constitute an overhead on the network.

6.11. Comparison and discussion

The previous skew estimation schemes are summarized in Tables 3, 4 and 5, where they are sorted in a chronological order. It is interesting to see how they evolved over time since the ground breaking heuristics of Paxson. This evolution aimed to either: improve the estimation robustness (*e.g.,* Linear programming), alleviate the required computations (*e.g.,* Simple average and Convex hull), account for drift and resets (*e.g.,* PRCSEA), provide an online result incrementally (*e.g.,* Sliding window, TICSync and Weighted average) or bound the estimation error of the skew (*e.g.,* Weighted average) and the offset (*e.g.,* TICSync). They can be used with the clock synchronization protocols that do not have a high accuracy (like NTP in WAN), to calibrate the OWD measurements as recommended by Paxson [16] and Zhang et al. [39].

A common trait of the previous methods is the usage of different line fitting techniques, which they apply on the minimum observed OWDs. In other terms, they exclude or ignore the variable part of the delay and track the evolution of its deterministic part over time. Hence, filtering network effects like queuing from the OWD measurements is very important to reduce the error in the skew estimation. However, on the one hand, many of these skew estimation techniques are biased by long queuing episodes. In such a case, assuming that the minimum observed delay in a certain time interval is the deterministic part of the OWD is not true. Moreover, if a long-lasting queue resolves gradually but slowly, the OWD observed during this period will always be decreasing without hitting the real minimum. This case directly affects a heuristic as robust as the Cumulative minima. On the other hand, some of the reviewed techniques (*e.g.,* simple averaging and linear regression) are also biased by short queuing due, for instance, to temporary congestion (*cf.*, column four of Table 3).

Some of the skew estimation schemes take a certain time to converge to an acceptable estimation of the clock skew as they require many calculations (*e.g.,* Paxson's heuristics and Linear programming) or need to wait for points at the end of the measurement period (*e.g.,* Simple average). Therefore, they can only be used offline for tasks such as network optimization and SLA validation. The complexity of a skew estimation method can be amortized if it can be implemented incrementally (*e.g.,* convex hull, TICSync and Weighted average). In addition, even if some methods deal with the clock drift over relative long time periods (*e.g.,* PRCSEA and Weighted average), it is not clear how they will react to fast skew changes due, for instance, to rapid temperature variations.

Providing a bound on the error of the skew (and therefore the OWD) estimate is important for many use-cases like real-time applications and QoS provisioning. Only few methods achieve this goal (*e.g.,* TICSync and Weighted average). Additionally, even though the relative error of the skew estimation can be evaluated with simulation, it was only reported by Aoki et al. and Moon et al.. Moreover, none of the discussed works reported the relative error in field tests. This is most likely due to the fact that the ground truth (*i.e.,* the real skew) is hard to establish without using a high accuracy independent reference time between the measurement points (like GPS and DAG cards).

Finally, only few works provided information about where timestamping takes place (*e.g.,* Cumulative minima and Convex hull used `tcpdump`). Timestamping uncertainty is a main contributor to the time error in skew estimation. Nonetheless, some of these works evaluated the absolute error of OWD measurement after skew removal (*e.g.,* simple average, TICSync and Weighted average). We did not include it in Table 5 as absolute error values from different setups cannot be compared directly.

7. Related work

There is an abundant literature about the topic of network time synchronization including some review papers. In the following, we investigate the works that are close to ours and emphasize the difference with them.

The survey by Shin et al. categorizes time synchronization mechanisms in two classes: external source based protocols and end-to-end measurement based methods [45]. They include in the former class GPS, NTP and PTP. The latter is further subdivided into online and offline algorithms for skew detection and removal. We adopt a similar division with a slightly different naming based on the action taken by these mechanisms instead of their location.

DeVito et al. provide a detailed comparison between the three popular synchronization protocols GPS, NTP and NTP [46]. Likewise, Orgerie et al. cover these protocols and add TSCclock [22]. Orgerie et al. gave also a differentiation of packet timestamping based on where it is performed. By contrast, we cover more synchronization mechanisms and also analyze in which network layer/header are the timestamps encapsulated and what impact this has on their usage.

Wang et al. classify skew estimation and removal algorithms in two subcategories [47]. The first contains mono-segment algorithms

that assume a constant clock skew. The second includes multi-segment models that consider in addition clock adjustments. The term multi-segment refers to the divide and conquer approach of these algorithms, which detect the clock resets that delimit each time segment, then estimate its clock skew.

Lévesque and Tipper survey the requirements of different applications, such as smart power grids and industrial Internet of Things, in terms of time synchronization [14]. In addition to NTP and PTP, they review link layer standards such as IEEE 802 Time Sensitive Networking (TSN) and Synchronous Ethernet (SyncE); and wireless sensor networks solutions like Reference Broadcast Synchronization (RBS) and Timing-Sync Protocol for Sensor Networks (TPSN). Furthermore, they put emphasize on the need to address security threats by protecting timing messages using authentication and integrity checking. Comparatively, we cover other schemes including the ones dedicated to clusters and datacenter networks.

Mallada et al. propose a double classification of clock synchronization protocols based on: whether they use offset corrections (*e.g.,* NTP), skew corrections (*e.g.,* CCT and Alg1) or both (*e.g.,* TSCclock); and whether they update the clock using offset values only, relative frequency error or both [36].

More recently, Karthik and Blum concentrated on synchronization protocols that are based on two-way message exchange, to combat the effect of stochastic network delays on the estimation of clock skew [48]. In particular, using invariant estimation theory, the authors developed optimum joint clock skew and offset estimators for the PTP standard. These estimators guarantee the lowest skew error independently of the probability density function used to model the queuing delay. Furthermore, using other schemes (*e.g.,* multiple master clocks), they proposed estimators that are robust against unknown asymmetry between the path delays.

Our survey of time synchronization complements the existing ones in several ways. We address both categories of time synchronization in one place, *i.e.,* direct clock modification; and skew estimation and removal from traffic traces. We also cover more recent protocols (*e.g.,* Alg1, CCT, Huygens and DTP). In addition, we analyze and compare the surveyed works thoroughly based on their relevant usage criteria.

8. Conclusion and future work

We set out to review the problem of network time synchronization and covered a representative set of established standards (GPS, NTP and PTP) and recent research works (*e.g.,* TSCclock, Huygens and DTP). Each mechanism was summarized succinctly; and its merits and limitations discussed. Using rich table formats, we compared techniques from the same category with respect to their application environment, accuracy and cost. We showed that clock synchronization protocols evolved over time towards finer accuracy, down to the sub-microsecond realm. Likewise, skew estimation and removal mechanisms progressed to account for drift and adopt incremental implementations suitable for online deployment.

An important lesson we learned from this survey is that there is no one size fits all solution. Hence, it is crucial to the choose the right synchronization tool for the task at hand based on the relevant usage criteria. Many of these tools have parameters (*e.g.,* probing rates) that require careful tuning to strike a balance between the overhead on the network and the accuracy.

The development of new techniques for time synchronization (*e.g.,* CCT, Huygens and DTP) was possible thanks to the advent of high speed gigabit per second networks. Additionally, it was in big part pushed by new time critical applications that require finer clock resolution and better accuracy. We think that these recent techniques have not yet shown their full potential. We leave the discussion for their use in applications and network management for future work. Other novel schemes will most likely be developed to address new networks with variable characteristics and different usage scenarios.

Concerning the evaluation of time synchronization methods, we notice that obtaining the ground truth (*i.e.,* the real clock skew) is a challenging task. We think that modifying network simulators to introduce an artificial clock skew could make this evaluation easier. Such a straightforward modification could be extended to adapt a complex skew model and therefore simulate clock drift. This would allow a direct thorough comparison of recent time synchronization mechanisms.

With respect to timestamping, it would be interesting to see how user-space fast packet IO frameworks like the Data Plane Development Kit (DPDK) [49]; and zero copy techniques in the socket APIs would perform in this regard. One would expect that running the full stack (down to NIC access) in user space and avoiding packet processing in interrupts should yield a better accuracy. We have no knowledge of research works that rely on these ideas, so this question is for now unanswered.

Finally, most of the schemes we surveyed did not address the issue of security. RFC 7384 discusses the threats that NTP and PTP can suffer in more details (*e.g.,* Denial of Service and packet manipulation) [50]. It proposes prevention mechanisms like authentication, encryption and integrity checking. The identification of other threats, either general or specific to a certain technique, would be a valuable addition. Moreover, a rigorous security analysis of the reviewed techniques is essential for a wide and safe deployment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The author would like to thank the anonymous reviewers for their constructive feedback.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] D. Mills, On the accuracy and stability of clocks synchronized by the network time protocol in the internet system, ACM SIGCOMM Comput. Commun. Rev. 20 (1) (1989) 65–75.

[2] B. Liskov, Practical uses of synchronized clocks in distributed systems, Distrib. Comput. 6 (4) (1993) 211–219.

[3] S. Kalidindi, M. Zekauskas, G. Almes, A one-way delay metric for IPPM, in: Request for Comments, (2679) 1999, http://dx.doi.org/10.17487/RFC2679, RFC 2679.

[4] D. Chefrour, One-way delay measurement from traditional networks to SDN: A survey, ACM Comput. Surv. 54 (7) (2021) 1–35.

[5] The Open Networking Foundation, OpenFlow switch specification, version 1.5.1, 2015, URL https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[6] J. Farkas, L.L. Bello, C. Gunther, Time-sensitive networking standards, IEEE Commun. Stand. Mag. 2 (2) (2018) 20–21, http://dx.doi.org/10.1109/MCOMSTD.2018.8412457.

[7] IEEE standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications, IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011) (2020) 1–421., http://dx.doi.org/10.1109/IEEESTD.2020.9121845.

[8] F. Zezulka, P. Marcon, Z. Bradac, J. Arm, T. Benesl, Time-sensitive networking as the communication future of industry 4.0, IFAC-PapersOnLine 52 (27) (2019) 133–138, http://dx.doi.org/10.1016/j.ifacol.2019.12.745, 16th IFAC Conference on Programmable Devices and Embedded Systems PDES 2019.

[9] F. Sivrikaya, B. Yener, Time synchronization in sensor networks: A survey, IEEE Netw. 18 (4) (2004) 45–50.

[10] B. Sundararaman, U. Buy, A.D. Kshemkalyani, Clock synchronization for wireless sensor networks: A survey, Ad Hoc Netw. 3 (3) (2005) 281–323.

[11] D. Mills, Network time protocol (version 3) specification, implementation and analysis, in: Request for Comments, (1305) 1992, http://dx.doi.org/10.17487/RFC1305, RFC 1305.

[12] S.B. Moon, P. Skelly, D. Towsley, Estimation and removal of clock skew from network delay measurements, in: IEEE International Conference on Computer Communications, vol. 1, INFOCOM, IEEE, New York, NY, USA, 1999, pp. 227–234.
[13] D. Veitch, J. Ridoux, S.B. Korada, Robust synchronization of absolute and difference clocks over networks, IEEE/ACM Trans. Netw. 17 (2) (2008) 417–430.
[14] M. Lévesque, D. Tipper, A survey of clock synchronization over packet-switched networks, IEEE Commun. Surv. Tutor. 18 (4) (2016) 2926–2947.
[15] J. Ridoux, D. Veitch, Principles of robust timing over the internet, ACM Queue 8 (4) (2010) 30–43.
[16] V. Paxson, On calibrating measurements of packet transit times, in: SIGMETRICS Conference, ACM, New York, NY, USA, 1998, pp. 11–21.
[17] R. Sugihara, R.K. Gupta, Clock synchronization with deterministic accuracy guarantee, in: European Conference on Wireless Sensor Networks, Springer-Verlag, Berlin, 2011, pp. 130–146.
[18] J.R. Vig, Introduction to Quartz Frequency Standards, Tech. rep., U. S. Army Electronics Technology and Devices Laboratory, 1992.
[19] A.S. Tanenbaum, M.R. van Steen, Distributed Systems, 3rd ed., 2017, pp. 298–303, URL https://www.distributed-systems.net/index.php/books/ds3/.
[20] G.F. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems: Concepts and Design, 5th ed., Pearson Education, 2012, p. 598.
[21] E. Anceaume, I. Puaut, Performance Evaluation of Clock Synchronization Algorithms, Research report, INRIA, France, 1998.
[22] A.-C. Orgerie, P. Gonçalves, M. Imbert, J. Ridoux, D. Veitch, Survey of network metrology platforms, in: IEEE/IPSJ 12th International Symposium on Applications and the Internet, IEEE, Izmir, Turkey, 2012, pp. 220–225.
[23] S. Donnelly, High Precision Timing in Passive Measurements of Data Networks (Ph.D. thesis), The University of Waikato, 2002, pp. 71–82.
[24] S. Donnelly, I. Graham, R. Wilhelm, Passive calibration of an active measurement system, in: Passive and Active Measurement Conference (PAM), Springer-Verlag, Berlin, 2001, pp. 1–8.
[25] S. McCanne, V. Jacobson, The BSD packet filter: A new architecture for user-level packet capture, in: USENIX Winter, vol. 46, USENIX association, San Diego, CA, USA, 1993.
[26] B. Villain, M. Davis, J. Ridoux, D. Veitch, N. Normand, Probing the latencies of software timestamping, in: IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings, IEEE, San Francisco, CA, USA, 2012, pp. 1–6.
[27] K.M. Salehin, R. Rojas-Cessa, S.G. Ziavras, A method to measure packet processing time of hosts using high-speed transmission lines, IEEE Syst. J. 9 (4) (2014) 1248–1251.
[28] J. Martin, J. Burbank, W. Kasch, D. Mills, Network time protocol version 4: Protocol and algorithms specification, in: Request for Comments, (5905) 2010, http://dx.doi.org/10.17487/RFC5905, RFC 5905.
[29] M. Hira, L. Wobker, Improving network monitoring and management with programmable data planes, 2015, URL https://p4.org/p4/inband-network-telemetry/.
[30] NAVSTAR, Global Positioning System Standard Positioning Service (SPS) Performance Standard, Technical documentation, USA Department of Defence, 2008.
[31] P.H. Dana, Global positioning system (GPS) time dissemination for real-time applications, Real-Time Syst. 12 (1) (1997) 9–40.
[32] N.M. Freris, S.R. Graham, P. Kumar, Fundamental limits on synchronizing clocks over networks, IEEE Trans. Automat. Control 56 (6) (2010) 1352–1364.
[33] J. Eidson, K. Lee, IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems, in: 2nd ISA/IEEE Sensors for Industry Conference, vol. 10, IEEE, Houston, TX, USA, 2002, pp. 98–105.
[34] S. Froehlich, M. Hack, X. Meng, L. Zhang, Achieving precise coordinated cluster time in a cluster environment, in: IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, IEEE, Ann Arbor, MI, USA, 2008, pp. 54–58.
[35] InfiniBand Trade Association (IBTA), The IBTA expands testing to include non-member ethernet switches and validates speeds up to 400G, 2021, URL https://www.infinibandta.org/ibta-expands-testing-to-include-non-member-ethernet-switches-and-validates-speeds-up-to-400g.
[36] E. Mallada, X. Meng, M. Hack, L. Zhang, A. Tang, Skewless network clock synchronization without discontinuity: Convergence and performance, IEEE/ACM Trans. Netw. 23 (5) (2014) 1619–1633.
[37] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, A. Vahdat, Exploiting a natural network effect for scalable, fine-grained clock synchronization, in: 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI, USENIX Association, USA, 2018, pp. 81–94.
[38] V. Shrivastav, K.S. Lee, H. Wang, H. Weatherspoon, Globally synchronized time via datacenter networks, IEEE/ACM Trans. Netw. 27 (4) (2019) 1401–1416.
[39] L. Zhang, Z. Liu, C.H. Xia, Clock synchronization algorithms for network measurements, in: IEEE International Conference on Computer Communications, vol. 1, INFOCOM, IEEE, New York, NY, USA, 2002, pp. 160–169.
[40] H. Khlifi, J.-C. Grégoire, Low-complexity offline and online clock skew estimation and removal, Comput. Netw. 50 (11) (2006) 1872–1884.
[41] J. Bi, Q. Wu, Z. Li, On estimating clock skew for one-way measurements, Comput. Commun. 29 (8) (2006) 1213–1225.
[42] M. Aoki, E. Oki, R. Rojas-Cessa, Measurement scheme for one-way delay variation with detection and removal of clock skew, ETRI J. 32 (6) (2010) 854–862.
[43] A. Harrison, P. Newman, TICSync: Knowing when things happened, in: IEEE International Conference on Robotics and Automation, ICRA, IEEE, Shanghai, China, 2011, pp. 356–363.
[44] T. Eylen, C.F. Bazlamaçcı, One-way active delay measurement with error bounds, IEEE Trans. Instrum. Meas. 64 (12) (2015) 3476–3489.
[45] M. Shin, M. Park, D. Oh, B. Kim, J. Lee, Clock synchronization for one-way delay measurement: A survey, in: International Conference on Advanced Communication and Networking, Springer-Verlag, Berlin, 2011, pp. 1–10.
[46] L. De Vito, S. Rapuano, L. Tomaciello, One-way delay measurement: State of the art, IEEE Trans. Instrum. Meas. 57 (12) (2008) 2742–2750.
[47] J. Wang, M. Zhou, Y. Li, Survey on the end-to-end internet delay measurements, in: International Conference on High Speed Networks and Multimedia Communications, HSNMC, Springer-Verlag, Berlin, 2004, pp. 155–166.
[48] A.K. Karthik, R.S. Blum, Recent advances in clock synchronization for packet-switched networks, Found. Trends (R) Signal Process. 13 (4) (2020) 360–443, http://dx.doi.org/10.1561/2000000108.
[49] 6WIND, Data plane development kit (DPDK), DPDK Project, 2013, URL https://www.dpdk.org/.
[50] T. Mizrahi, Security requirements of time protocols in packet switched networks, in: Request for Comments, (7384) 2014, http://dx.doi.org/10.17487/RFC7384, RFC 7384.