A Case for Feedforward Control with Feedback Trim to Mitigate Time Transfer Attacks

FATIMA M. ANWAR, University of Massachusetts Amherst
MANI SRIVASTAVA, University of California Los Angeles

We propose a new clock synchronization architecture for systems under time transfer attacks. Facilitated by a *feedforward control with feedback trim*–based clock adjustment, coupled with *packet filtering* and *frequency shaping* techniques, our proposed architecture bounds the clock errors in the presence of a powerful network attacker capable of attacking packets between a master and a client. A key advantage is consistent measurements, timely coordination, and synchronized actuation in distributed systems. In contrast, current time synchronization architectures behave poorly under attacks due to assumptions that the network is benign and delays are symmetric. The usage of feedback controllers aggravates poor performance. We provide an architecture that is indifferent to delays and eases the integration to traditional protocols. We implement a delay attack–resistant precision time protocol and validate the results on a hardware-supported testbed.

CCS Concepts: • **Security and privacy** → *Network security*; • **Computer systems organization** → *Embedded and cyber-physical systems*;

Additional Key Words and Phrases: Feedback control, two-way time transfer, precision time protocol

ACM Reference format:
Fatima M. Anwar and Mani Srivastava. 2020. A Case for Feedforward Control with Feedback Trim to Mitigate Time Transfer Attacks. *ACM Trans. Priv. Secur.* 23, 2, Article 11 (May 2020), 25 pages.
https://doi.org/10.1145/3382503

1 INTRODUCTION

Time Transfer is a way of sharing reference time among physically or geographically separated entities. A shared sense of time is critical for many applications such as to correlate observations in astronomical [2] and financial [37] systems, coordinate tasks between cell towers [23], and choreograph acts among autonomous agents [27]. Widely used time transfer systems either rely on *one-way* packets or a *two-way* packets exchange. In a one-way time transfer system, a server/master sends its current time over a network to multiple clients. This technique is simple, yet its accuracy suffers from uncompensated propagation delays. Global Positioning System (GPS) [7] is an

This material is based upon work supported in part by the National Science Foundation (NSF) under award # CNS-1329755 and CNS-1705135, and by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.
Authors' addresses: F. M. Anwar, 209E Knowles Engineering Building, University of Massachusetts, 151 Holdsworth Way, Amherst, MA 01003-9284; email: fanwar@umass.edu; M. Srivastava, University of California Los Angeles, Networked & Embedded Systems Lab (NESL), 1762 Boelter Hall, Los Angeles, California, 90095; email: mbs@ucla.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
2471-2566/2020/05-ART11 $15.00
https://doi.org/10.1145/3382503

example of one-way time transfer system with tightly calibrated delays. In a two-way time transfer systems, both master and a client exchange their current time with each other. The four timestamp measurements from two packets help calculate round-trip delays. A major drawback of two-way packet exchange is the assumption that delays are symmetric in both directions. In reality, these delays are asymmetric and translate to clock errors. Clock synchronization protocols make use of two-way time transfer to calculate delays and clock offsets. These offsets are fed to a feedback controller that adjusts a clock to compensate for the offset.

1.1 Time Transfer Attacks

Network elements that assist time transfer can be malicious. They can develop various Man in the middle (Mitm) capabilities such as dropping, replaying, preplaying, and delaying packets. Prior works [19, 35, 47] talk about attacks on time transfer packets and mitigating these attacks using cryptographic [8, 25] and network security [9, 24] mechanisms. Unfortunately, delay attacks are considered too strong and immune to proposed mitigations [31, 45]. Researchers have attempted to secure systems against delay attacks under many assumptions. Widely used assumptions relate to the availability of excess or redundant information in terms of (1) multiple masters providing reference time, (2) multiple communication paths to those masters, and (3) at least two thirds of those paths to not be compromised. These assumptions are not realistic in the presence of a malicious gateway router that can potentially delay "all" packets [19]. There exists numerous synchronization architectures where redundant resources come at a cost [1]. The ultimate goal of a Mitm attacker is to move the client's clock away from true time toward its malicious time for illegal activities in high frequency trading [37], digital rights violation [16], spoofing location [42], and so on.

Prior work has identified two key problems in current synchronization protocols that make them vulnerable to delay attacks. First, an attacker can induce large asymmetric delays by delaying one packet in a two-way time transfer [19, 31]. Second, a feedback control-based clock adjustment mechanism amplifies the affects of delay asymmetry by feeding the error back to the controller and steering the clock toward inaccurate time.

1.2 Contributions

It is already well known that one-way packets with a feedforward control can syntonize clocks; however, in this article, we make a case that *one-way* time transfer and a *feedforward* controller can achieve clock *syntonization*[1] even in the presence of "delay attacks." Syntonized clocks help systems perform various operations such as measuring precise inter-event times in localization [15], residency delays in high speed networks [21], and execution times for code profiling [18]. We highlight a key advantage of a feedforward controller to compensate for delay attacks, i.e., its ability to calculate relative frequency even in the presence of *constant delays*. Feedforward control leverages the property that *equally delayed* packets cannot give correct instantaneous offset, but they do preserve the rate of change in offset, i.e., the relative frequency of the two clocks.

In contrast to feedback control, *feedfoward* does not rely on the instantaneous time offset. Instead it monitors the "rate of change" in this offset over multiple one-way time transfer packets to calculate relative frequency error. A feedforward controller uses timestamps from a free running clock that is never adjusted. This is because relative frequency of two clocks is a property of their hardware oscillators. Comparing timestamps of free running clocks derived from these oscillators gives best estimate of relative clock frequency.

We make an observation that if two packets are received at the same interval at which they are transmitted, they either experience no delay or equal delays in the network. An adversary

[1]The process of aligning frequencies of different clocks is called *syntonization*.

may choose not to delay packets equally and degrade the feedforward controller's performance by injecting random delays. In this case, the received packets do not arrive at the same interval at which they are transmitted, and hence they experience unequal delays. Leveraging this observation and the fact that feedfoward control calculates accurate relative frequency only for equally delayed/periodic packets, we propose a technique that *restores periodicity* among unequally delayed packets. Given that a master transmits packets periodically—which is the generalized case—we are able to filter received packets and transform their timestamps to emulate received periodic packets under the right conditions. Once we *restore periodicity* of the delayed one-way packets, we feed the transformed timestamps of the emulated packets to a feedfoward controller to calculate the relative frequency error.

After aligning frequencies of different clocks in a secure manner, we present a secure clock *synchronization* approach in the presence of delay attacks. To synchronize two clocks, we cannot rely on a feedforward controller, as it only calculates relative frequency using raw timestamps. We need to calculate the accurate clock offset, i.e., the error between a reference time and the adjusted time. Unfortunately, delay attacks make it harder to calculate the exact offset. Therefore, we propose a packet filtering technique that looks for packet delay patterns to search for minimally delayed or, as we call them, *delay-free* packets. These delay-free packets provide a good estimate of the clock offset. Rather than directly adjusting a clock with this offset, we design a *frequency shaper* that uses this offset to trim the relative frequency calculated by the feedfoward controller. The resultant trimmed frequency is used to synchronize the clock. In short, we calculate the offset in a feedback loop and utilize it to shape the relative frequency for clock adjustment. We call this clock adjustment mechanism *feedfoward control with feedback trim*.

1.3 Advantages

Our proposed approach of feedforward control with feedback trim can syntonize clocks for constant delays and synchronize clocks in the presence of random delays and other delay patterns. It mostly relies on one-way time transfer with occasional reliance on two-way time transfer. Our standalone software implementation can easily be integrated to various synchronization protocols with minimal code changes. For proof of concept and to support one synchronization protocol, we implement our packet filtering techniques along with feedforward controller with feedback trim for Precision Time Protocol (PTP) [28] and evaluate our system on a real embedded platform [3] used in many clock synchronization applications. Our evaluation shows promise and a comparable performance in reference to systems that are not under attack.

To summarize our contributions,

- We present detailed experimental analysis of how different delay patterns affect feedback and feedforward control-based clock synchronization.
- We identify a key property of equally delayed packets. These packets preserve the relative frequency between two clocks. Hence, we leverage equally delayed packets to syntonize clocks with feedforward control.
- We propose two packets filtering techniques: one to *restore periodicity* of variably delayed packets to calculate relative frequency and the other to find *delay-free packets* to calculate offset.
- We design a secure clock synchronization architecture under delay attacks. It consists of a *feedforward control with feedback trim*–based clock adjustment mechanism and relies on a *frequency shaper* that trims the relative frequency based on the offset.
- Finally, we evaluate our proposed architecture on a hardware-supported testbed and support new feedforward controllers for widely used precision time protocol.

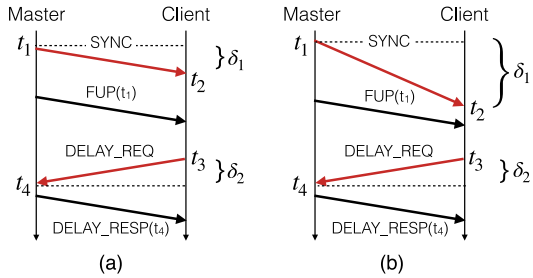


Fig. 1. (a) Symmetric delays in forward and reverse direction ($\delta_1 = \delta_2$); (b) asymmetric delays in both directions ($\delta_1 > \delta_2$).

After providing necessary background and related work in Section 2, we move on to explain the feedforward control in Section 3. Then we motivate our work in Section 4 by quantifying the delay attack damage in current synchronization protocols for both feedback and feedforward controllers. Section 5 states our threat model, gives an overview of our delay-tolerant clock synchronization architecture, and goes on to explain the details of our proposed techniques. The details of our testbed and evaluation of various test cases are in Section 6 followed by a discussion and conclusion in Section 7.

2 BACKGROUND

2.1 Time Synchronization Basics

In any time synchronization protocol, there is always a disciplinable clock and a `timestamping` mechanism that captures event times from that clock. Packets exchanged in the network act as events to be timestamped. These packets can be timestamped in software or hardware depending upon the desired synchronization accuracy and/or the availability of hardware timestamping in a given platform. A time synchronization protocol relies on one-way or two-way time transfer packets to align a clock with a reference time. To explain a time synchronization protocol, let us take an example of Precision Time Protocol (PTP) [28] that is widely used in high-precision measurement and control applications.

The packet exchange for PTP is shown in Figure 1. The master node sends the `SYNC` message at t_1, and clients receive it at t_2. The actual value of t_1 is sent to client in a follow up (`FUP`) message. The time offset of two clocks are, $offset = t_2 - t_1 - delay$. To determine this $delay$, the client sends a delay request (`DELAY_REQ`) message at t_3 that reaches the master at t_4. The master sends the timestamp t_4 to client in a response message (`DELAY_RESP`). In PTP, the timeliness of SYNC and DELAY_REQ message is important, as their timestamps are used to determine clock synchronization parameters. The calculated $delay = (t_2 - t_3 + t_4 - t_1)/2$ is round-trip delay divided into two symmetric delays. The delay in SYNC message to reach the client is termed as forward path delay (δ_1), whereas the delay that DELAY_REQ message incurs is the reverse path delay (δ_2). PTP like other protocols also assumes that both forward and reverse path delays are equal, i.e., symmetric with a very small error margin as shown in Figure 1(a). If an attacker launches a delay attack either on the forward or the reverse path, then the delays are no longer symmetric, as shown in Figure 1(b), and the quality of time synchronization is degraded.

2.2 Time Transfer Attacks and Mitigations

There are many ways for an adversary in a network to attack time transfer packets and degrade the accuracy of a time synchronization protocol [34]. It can launch a `removal attack` that selectively

drops the synchronization packets before they reach the receiver. Note that not all packets are dropped so as not to cause a DOS attack. `Replay attack` records previous synchronization packets and replay them at a later time providing inaccurate time information to the receiver. `Pre-play attack` injects new messages in the network and tricks the receiver into believing that the newly forged messages are from a legitimate sender. `Substitution attack` replaces time value inside a valid packet with a new value. Finally, `delay attack` simply holds the packet for some time and sends it later. Various cryptographic techniques protect against pre-play and substitution attacks by verifying that the messages have indeed been sent by a legitimate sender [8, 25]. A replay attack is thwarted by checking the freshness of a packet by observing a monotonically increasing sequence number [9, 24].

Delay attack, however, is considered too powerful to be completely mitigated [31, 45]. Cryptographic and network security mechanisms are insufficient to detect and estimate delay attacks. Some attempts have been made to bound these attacks [35, 47]. Most of these techniques rely on the diversity in servers that provide reference time [19], and the paths toward them [26, 33] to bound delay attacks, while assuming that not all paths to these servers are attacked. The assumption of a communication architecture with multiple servers per client is valid for a Network Time Protocol (NTP) [32]. However, all paths to those servers are compromised if a single gateway router is malicious and delays all NTP traffic. In PTP, however, the availability of multiple masters is not realistic. PTP networks rely on few grandmasters due to cost issues and utilize switches that contain boundary clocks and transparent clocks to scale the network [1]. In this scenario, a single compromised switch can delay all the packets to PTP clients.

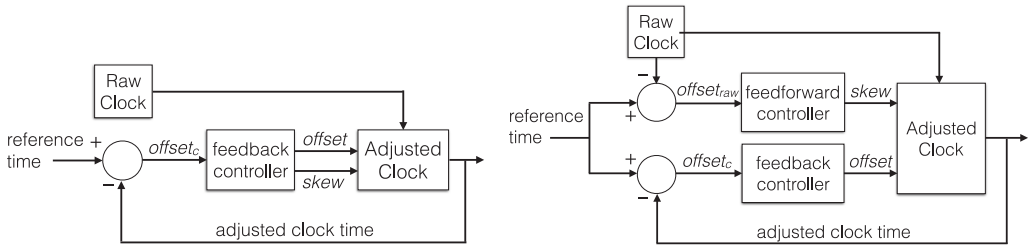
Prior works tried to calibrate round-trip delays of different paths in the network. During time synchronization, the two-way time transfer is not considered secure if the packets exceed the expected round-trip delay [43]. Annessi et al. have tried to give upper bounds on delay by modeling the clock drift [9]. The offline modeling-based approaches have conservative bounds and give a smart attacker sufficient slack to launch attacks and remain unnoticed. Narula et al. [36] presents a theory to assess security of protocols in a generic setting. Their necessary and sufficient conditions give an idea of how to prove a protocol's security. By doing so, they prove that PTP [28] is not secure. In securing PTP, their assumptions of communicating over line of sight channel or shortest possible path are not realistic. They also give a necessary condition of bounding delays by estimating round-trip delays *a priori*. Distance bounding protocols [42] leverage the same idea for bounding distances but there are numerous attacks [12, 22] possible on these protocols.

Researchers have also correlated different sensing modalities to timing signals. Dima et al. [38] secured NTP communication against delay attacks by detecting and estimating path asymmetries through the use of power grid voltages. Their solution requires specialized hardware and only works for grid-connected distributed systems [29], hence making it unsuitable for embedded and wireless sensor networks applications. Seismic deployments are also used to recover temporal integrity [30]. The availability of seismic modality is not widespread, and its accuracy is subseconds, which is insufficient for most applications.

2.3 Why Clock Synchronization Protocols Are Vulnerable to Delay Attacks

There are two key problems in a clock synchronization architecture that an attacker exploits to manipulate the time,

(1) Clock synchronization protocols calculate offset between two clocks by utilizing near-symmetric delays in a two-way time transfer. Both network variations and adversarial components in the network can violate the symmetric delay assumption in synchronization protocols. An attacker delays one of the two packets in a two-way time transfer to achieve desired delay asymmetry. This asymmetry directly translates to a clock error.



(a) Feedback control based clock adjustment (b) Feedforward and feedback control based adjustment

Fig. 2. Global clock synchronization based on different clock adjustment mechanisms.

(2) In a synchronization protocol, a controller adjusts a clock by aligning it to a reference clock both in time and frequency. Most protocols use a feedback controller as shown in Figure 2(a). This controller jumps time by a large *offset*, or it uses a small offset to tune the frequency of a clock called *skew*. In a closed loop feedback control, next measurements are affected by the previous adjustments. In case of an attack, adjustments are based on offsets calculated from asymmetric delays. These adjustments affect the next measurements, thus accumulating clock errors.

3 FEEDFORWARD CONTROL

In contrast to feedback control in current clock synchronization protocols, we propose a feedforward control for frequency correction in the presence of delay attacks. Unlike feedback, it decouples the time and frequency calculation so that the error in one does not affect the other. A comparison of feedback- and feedforward-based clock adjustment is shown in Figure 2(b). In a feedforward system, the control is not driven by an error in the output. Instead, it is based on the knowledge of the process. In context of clocks, the process knowledge we use is that all clocks are derived from hardware oscillators that tick at a nominal frequency. Due to manufacturing variations, the oscillators drift from their nominal frequency. These frequency drifts are unique to a hardware oscillator. A free running counter or a clock is directly impacted by this frequency drift. Comparing timestamps of two free running clocks gives an estimate of their relative frequency drift. This is termed as frequency error in context of a clock synchronization protocol. Feedfoward control directly compares the timestamps of two free running/raw clocks to get an accurate estimate of relative frequency error. It uses this relative frequency to adjust the clock. Thus a feedfoward control is able to *syntonize* clocks, i.e., aligns the frequency of two clocks. An accurate estimate of this frequency reduces the need to synchronize often thus saving network bandwidth. But the accuracy of feedfoward control depends on how accurate our process model is. In reference to Figure 2(b), a feedforward controller is represented as $t_{ref} = t_{raw_{current}} + offset_{raw} + skew * (t_{raw_{current}} - t_{raw_{previous}})$, where t_{ref} is the reference time estimated by calculated clock offset $offset_{raw}$ and frequency error $skew$ between master and slave and raw slave clock timestamps $t_{raw_{previous}}$ and $t_{raw_{current}}$ of two packets.

A feedback control continuously exchange messages to calculate and adjust the offsets. Frequently adjusting offsets indirectly compensates for the relative frequency error, but it comes at a cost of increased bandwidth. Any error in offset affects the relative frequency as well. Note that the feedback control sample timestamps from an adjusted clock to calculate the offset, whereas a feedforward control gathers timestamps from a free running clock to calculate relative frequency. Feedforward control also bypasses various sources of error in an adjusted clock due to noisy measurements and asymmetric delays.

RADclock [40] was the first to introduce a feedforward clock model. The difference between RADclock and our architecture is that RADclock provides a feedforward clock model [14], whereas we propose a feedforward controller to adjust a general clock model. RADclock derives frequency and time adjustments from two-way time transfer and feedback controllers to adjust their feedfoward clock. In our case, we use feedforward controllers to calculate clock adjustments for the regular clock models in the kernel. Unlike RADclock, our feedforward controller does not rely on a specific hardware functionality. It can also be integrated with any clock synchronization protocol and work with any clock model with minimal code changes.

We implement a feedforward controller for PTP and switch from the regular PTP feedback controller to this feedfoward controller. Using multiple timestamps of a local free running clock with respect to the reference clock over the passage of time, we calculate the rate of change in clock errors. This rate of change determines how fast or how slow the local clock should tick to be aligned with the reference clock. Once the relative frequency is known, feedforward control can switch to infrequent packets exchange. One disadvantage of feedforward control is that it cannot handle disturbances or transient events.

4 INADEQUACY OF THE STATUS QUO

Before designing a feedforward-based delay-tolerant architecture, an important step is to analyze the performance and compare the behaviors of feedforward and feedback controllers under attacks.

4.1 Experimental Methodology

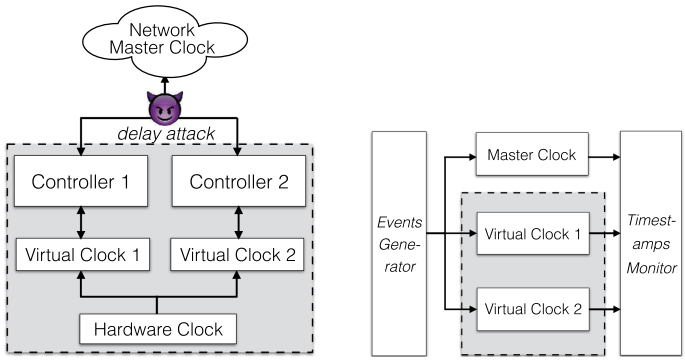
We introduce our experimental methodology early on to use it in our experimental analysis and later in evaluation. For experimental purpose, we use logical mappings derived from the same physical clock on a single platform. We term these logical mappings as virtual clocks, and they are no different than the actual physical clocks in real deployment. This concept of virtual clocks is not new; the need of multiple disciplinable clocks on a single platform has motivated previous architectures [10, 11]. Posix standard [5] virtual clocks derived from the same hardware timer are also part of the Linux kernel.

The basic setup is shown in Figure 3(a). Our testing set up uses virtual clocks to compare two controllers (feedforward and feedback) on a single hardware to alleviate network and hardware biases in testing. Time transfer packets flow between the master and the client for synchronization purpose, typically with a poll period of half a second. These packets are delayed in the network before reaching the client. Then the same network traffic is fed to all the controllers to remove the effects of network variations.

In these experiments, we intend to compare the performance of a feedback control with a feedforward control under delay attacks. Though our feedforward controller is generalized and applicable to a wide variety of synchronization protocols, we choose PTP for our experiments, as it is widely used in many applications. PTP provides a set of feedback controllers to choose from. We compare the commonly used Proportional Integral (PI) controller with our implementation of a linear regression-based feedforward controller. It is essential to note that we use hardware timestamping capability to timestamp incoming and outgoing packets in these experiments, but the results are generalized enough for any timestamping capability.

For evaluation purposes, a separate box shown in Figure 3(b) generates events at the rate of eight events per second. These events act as common timestamping opportunities for multiple virtual clocks under test. A monitor analyses the timestamps in the following two ways:

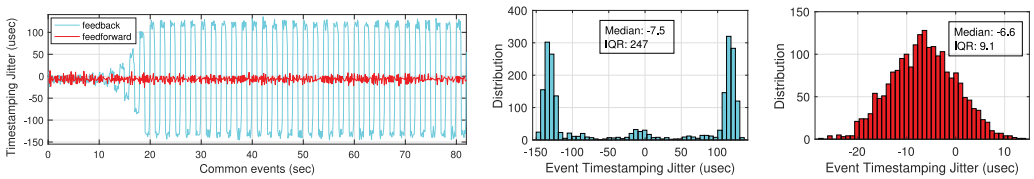
1. **Frequency stability:** Finds timestamping jitter in all clocks to compare relative frequency error of these clocks with respect to the master clock.



(a) Testbed to synchronize multiple clocks on a single platform

(b) Testbed to verify synchronization accuracy of multiple clocks on a single platform

Fig. 3. (a) A single platform testbed to synchronize virtual clocks to a master in the network. Each virtual clock is disciplined by a separate controller. (b) Once the virtual clocks are disciplined, we test their accuracy with respect to master by supplying common events to all clocks with an event generator. All clocks timestamp the same events, and a monitor analyses these timestamps.



(a) Feedback versus Feedforward clock jitter

(b) Feedback controller

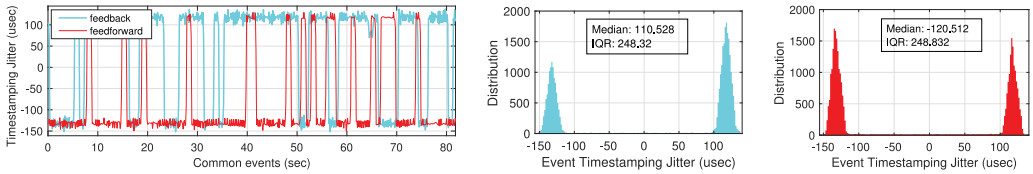
(c) Feedforward controller

Fig. 4. Periodic events occurring every 125 ms are timestamped by two clocks, where one clock is adjusted by feedback control and the other clock with feedforward control. Both of these clocks experience forward path delays of 1 s. (a) Feedback clock jitter gradually oscillates to a steady state, whereas feedfoward clock jitter is smooth from the start. (b) Timestamping jitter distribution for feedback-based clock sync varies in the order of 247 μs, whereas (c) jitter distribution for feedfoward-based clock is only 9 μs.

2. **Time accuracy:** Finds the absolute time accuracy of virtual clocks with respect to the master.

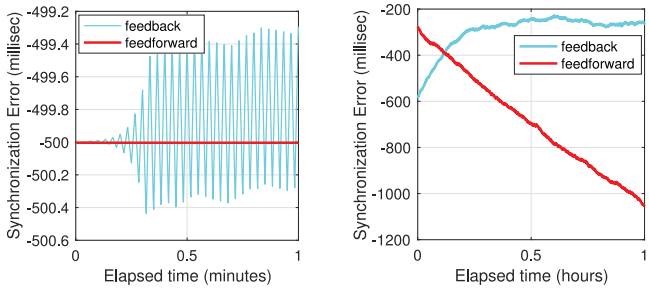
All the experiments for synchronization and validation are run concurrently so that clocks to be compared experience identical conditions and show a real-time performance.

Timestamping Jitter under constant delay attack: We first run an experiment where we constantly add 1-s delays to all one-way SYNC packets in PTP. This induces asymmetry of 1 s and an error of 0.5 s in offset estimation. We synchronize two virtual clocks on the same platform, one with feedback and the other with feedforward control. Then we generate periodic events separated by 125 ms and use both clocks to timestamp these periodic events. Using these timestamps, we calculate the interval between consecutive events. Note that this interval should ideally be 125 ms, but, due to errors in relative frequency calculation, there is a jitter in interval measurements. We plot the timestamping jitter for both clocks. Nodes only need to align their frequencies to measure fine inter-event times. Our results show that the frequency error calculation for our feedforward clock has less variations under constant delay attack as shown in Figure 4(c), because feedfoward control is indifferent to asymmetric delay, whereas feedback clock utilizes asymmetric delays to



(a) Feedback versus Feedforward clock jitter (b) Feedback controller (c) Feedforward controller

Fig. 5. Clock under random delay attack from 0 to 1 s. (a) Timestamping jitter of both feedback and feedforward clocks have large variations in the range of −150 μs to 150 μs. The jitter distributions are shown in (b) and (c) with approximately 250-μs IQR.



(a) Constant delay attacks (b) Random delay attacks

Fig. 6. Clock errors of feedback- and feedfoward-based clock synchronization under different attacks.

calculate the offset and then uses this offset to calculate frequency error. This results in high jitter, as shown in Figure 4(b). Figure 4(a) shows how the jitter in both clocks varies over time.

Timestamping Jitter under random delay attacks: In reality, an adversary can delay synchronization packets both in forward and reverse paths by any value. To illustrate the effect of a random attacker, we imitate an adversary that delays packets by a random value from a uniform distribution of 0 to 1 s. We synchronize one clock with feedback control and the other with feedforward control using SYNC packets delayed randomly in the network. In a separate process, we repeat the same validation mechanism; the two clocks timestamp 125 ms periodic events from an event generator. We plot the clock jitter in Figure 5(a). There is a jitter in the order of sub-milliseconds both for feedback and feedforward clock. The jitter distributions in Figure 5(b) and (c) are comparable. The Inter Quartile Range (IQR) for both clock's jitter is around 250 μs. We conclude that the feedback has similar clock jitter behavior for constant and random delays, whereas the feedforward clock performs worse in the presence of random attacks.

Synchronization error under constant/random delay attacks: We also compare the synchronization error of the feedback and feedforward clocks. Again the master and the client clocks timestamp periodic events. The errors are shown in Figure 6. The synchronization errors in Figure 6(a) are influenced by constant delay attacks. Note that the feedforward clock error is almost constant at −0.5 s with little fluctuations that are hard to visualize in comparison to large error variations for feedback clock. The variations in feedback clock are in the range of 800 μs. We conclude that a constant delay attack only adds a large offset to a feedforward clock with no effect on the clock's frequency, whereas the feedback clock is affected both in terms of an offset and large peak oscillations due to frequency error. In Figure 6(b), however, we see that random delay attack largely influences the open loop relative frequency calculation in a feedforward controller. The random delay gradually steers the frequency in one direction, and the error of feedforward clock

never converges. However, a feedback clock gradually accumulates error and reaches a steady-state value of one-fourth of the maximum random attacks. It oscillates in the order of milliseconds after reaching the steady state. This gives us an idea that feedforward works well for constant attacks, but it performs far worse under random attacks.

Delaying both forward and reverse path packets with equal delays in both directions is not an attack, because in this case the delay is symmetric. Feedback clock error, however, takes longer to converge in the presence of high symmetric delays, while feedforward clock error quickly converges. Another delay attack model is to gradually increase delay to a maximum attack value and then gradually decrease the delay to a minimum value. The behavior of both feedback and feedforward control is quite predictable in this scenario. For both controllers, the clock error gradually increases to a maximum offset and then decreases to zero. They both exhibit oscillating behavior with a period that depends on the rate of change of delay.

Through our experimental analysis, we deduce that an attacker can cause significant damage by first launching a constant delay attack and then switching to random delays. With this attack, both feedback and feedforward controllers jump to a clock error that equals half the initial delay value and then oscillates at a maximum frequency due to random delays. In case of feedforward clock, the error never converges.

Observations: We make three key observations after analyzing the behavior of both controllers in the presence of different attacks,

(1) The frequency error calculated by the feedforward controller is unaffected by constant delay attacks. This gives us an intuition that we can *syntonize* clocks with a feedforward controller if subsequent packets are equally delayed.
(2) The problem of delay asymmetry does not affect one-way clock syntonization. But delay asymmetry greatly influences two-way clock synchronization.
(3) Both feedback and feedforward controllers accumulate clock errors under different attacks. The only way to reliably calculate offset between two clocks is through one-way packets that do not experience delays or through two-way packets that are delayed by the same value in both directions.

5 DELAY ATTACK-TOLERANT ARCHITECTURE FOR CLOCK SYNCHRONIZATION

After observing the behavior of controllers under delay attacks, we first state our threat model. Then leverage some of the observations we made in our experimental analysis to put forward a delay attack-tolerant clock synchronization architecture for systems under time transfer attacks.

5.1 Threat Model

Delay attacks are considered too strong to protect against [31]. Prior work [19] offered delay attacks mitigations for NTP with a weak assumption that only one-third of the time servers and network links are compromised. We argue that an adversary sitting on a gateway router can easily delay or attack all the packets for a client. In our work, we consider a much stronger threat model, where all the links between clients and master can be compromised. In other words, an attacker sitting on a network element is capable of delaying all the packets by any value. A practical assumption, however, is the attacker delays packets by a finite value to avoid detectable denial of service.

Clock synchronization protocols employ sanity checks through prior knowledge of the environment, network, and physical clock characteristics to reject outliers that exceed the delay bounds. These bounds are conservative and gives attacker enough slack to launch delay attacks. A smart attacker can significantly alter the time value within the specified bounds. In our threat model, an attacker can delay packets by a value smaller or greater than the synchronization period. Small,

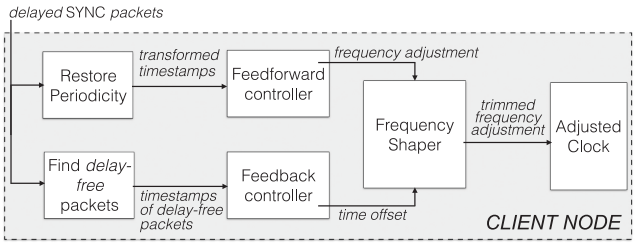


Fig. 7. Delay attack-tolerant clock synchronization architecture.

large, incremental, or distribution-based delay variations with prior knowledge of the network delay are in scope.

In this work, we trust the master or any entity that provides the reference time to the system. In many industrial systems, the nodes that provide reference time are mostly considered secure. Most of the servers are equipped with GPS modules that provide reference time in many clock synchronization protocols such as NTP, and PTP. Any GPS spoofing attack is not in our scope. We do not consider any hardware clock manipulation at the master or the clients. Prior works have addressed hardware-based protections [24, 39]. Our threat model also excludes packet replay, pre-play, substitution, and removal attacks that are mitigated by cryptographic techniques [8, 9, 25]. We are interested in addressing the most powerful attack considered in clock synchronization literature: a delay attack. We assume different cryptographic and network security mechanisms in place that protect both master and slave packet timestamps [19].

In our threat model, a delay attack corresponds to a value by which an attacker delays a packet in the network and induces clock error greater than the system can tolerate. A packet is considered "delay-free" if it experiences lesser delay than the one classified as an attack. Although numerous delay attack distributions are possible, we only consider and evaluate for three major categories of delay attacks: constant delay, random delay, and incremental delay. As most synchronization protocols have filters discarding packets that experience delays larger than a threshold, we assume that the attacker will not launch a delay attack greater than this threshold to avoid detection. Therefore, our architecture builds upon the information of maximum delay attack value.

5.2 Overview

The delay attack-tolerant clock synchronization architecture along with its major components is shown in Figure 7. Our architecture is designed for a *feedforward* controller that aligns clock frequencies for equally delayed packets. As not all packets are equally delayed, we put forward a packet processing technique that *restores periodicity* of variably delayed packets such that the feedforward controller can better estimate the relative frequency error. As our focus is also to synchronize clocks in the presence of attacks, we search for a property in *subsequent one-way* packets that can safely declare them *delay-free*. These delay-free packets provide good offset estimates that are used by a *frequency shaper* to trim the relative frequency. We refer to it as a *feedback trim* to the feedforward calculated frequency. Finally, the trimmed frequency is used to steer the clock toward true time.

We make the following key contributions to bound the effect of delay attacks on clock synchronization protocols,

- **Restore periodicity of delayed one-way packets** under certain conditions with a frequency tolerance. These transformed periodic packets at the client are assumed to be equally delayed in the network.

ALGORITHM 1: Restore Periodicity

```
procedure TRANSFORM TIMESTAMPS (sync_period, tolerance)
    s ← sync_period
    ϵ ← tolerance
sample new packet:
    t_a ← t_epoch_pkt                          ▷ Timestamp of epoch packet
    t_b ← t_new_pkt                            ▷ Timestamp of latest packet
    inter_pkt_dist ← t_b − t_a
    if inter_pkt_dist > 0 then                 ▷ In order packets
        if inter_pkt_dist = s ± ϵ then         ▷ Condition I
            t_epoch_pkt ← t_b
            t_a ← t_a
            t_b ← t_b
        else if inter_pkt_dist ≤ ϵ then        ▷ Condition II
            t_epoch_pkt ← t_b
            t_a ← t_a
            t_b ← t_b + s
        else if inter_pkt_dist = ns ± ϵ then   ▷ Condition III
            t_epoch_pkt ← t_b
            t_a ← t_a + (n − 1)s
            t_b ← t_b
        else                                   ▷ No Condition Met
            t_epoch_pkt ← t_a
    goto sample new packet.
```

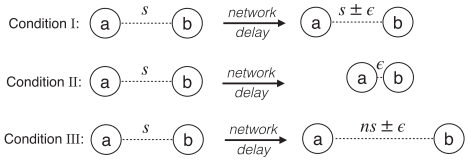


Fig. 8. Two packets sent at a predefined interval of s seconds experience different delays in the network. Packet a and packet b may incur same delay with a small tolerance ϵ as shown in Condition I. Packet a could be delayed s seconds more than packet b such that both packets are only ϵ seconds apart from each other as shown in Condition II. Packet b could be delayed s times more than packet a such that they have large delays between them as shown in Condition III.

- **Implement a feedforward controller** that leverages the transformed one-way packets to calculate the relative frequency error with respect to the master. This frequency error is used to adjust the client clock's frequency.
- **Find delay-free packets** or packets that experience less delays in the network to calculate the time offset between the clocks. The occurrence of these packets depend upon the maximum delay an attacker is capable of launching.
- **Design a feedback trim** to synchronize clocks. The feedforward calculated frequency is trimmed based on clock offset to compensate for the accumulated error due to inaccuracies of feedforward controller and the error in finding delay-free packets.

Restore one-way packets periodicity: Periodic one-way packets from the master are delayed in the network by an adversary. Our approach transforms receive timestamps of selected packets that satisfy certain conditions in a way that restores their original periodicity. Maintaining original period of one-way packets does not reflect zero delays in the network; rather, it emulates equal delays for all the packets.

A master sends one-way packets with a predefined period. If two packets are sent with a time interval of s seconds, then upon receiving the packets we mark those packets valid for our clock synchronization if they satisfy one of the three conditions shown in Figure 8. Packet a is sent before packet b, and both are s seconds apart.

(1) *Condition* I: If the received packets arrive at the client in order and they are s seconds apart within some tolerance ϵ, then they hold the periodicity property and do not need to be transformed.
(2) *Condition* II: If the received packets arrive in order and the time interval between them is negligibly small, then packet a is delayed s seconds more than packet b. In this case, we transform the receive timestamp of packet b such that it reflects the same delay as packet a and both packets remain periodic.
(3) *Condition* III: If the received packets are in order and their inter-packet time is much higher than the actual period, then it can be safely said that packet b is delayed multiples of s times

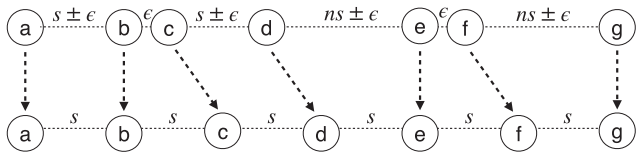


Fig. 9. An illustrated example of receive timestamps transformation to keep the packets periodic.

the packet a. We add the same amount of delay to packet a timestamp to align it with the desired period.

Refer to Algorithm 1 for the steps to restore periodicity among received packets by transforming their timestamps. The *sync_period* and *tolerance* are the input parameters that can be tuned to achieve desired synchronization performance. When a new packet is received, its receive timestamp is compared with the previous packet's timestamp. Note that these timestamps are sampled from the raw client clock. If their difference is greater than zero, then we received in order packets. If the inter-packet time difference is equal to the *sync_period* with a small tolerance, then the timestamps are not changed (Condition I). If the packets arrive at a very small time difference, then we move the second packet's timestamp ahead by the period (Condition II). If the packets have big enough of a difference, then we transform the first packet's timestamp so as to maintain the *sync_period* (Condition III). We demonstrate an example in Figure 9 to show how the transformation works. Packets a and b maintains the same period s hence condition I is applied and their timestamps are used as it is. Packets b and c are very close hence condition II dictates that packet c be delayed by s seconds. Finally, packet f is farther away from g, and condition III transforms timestamp of f to be only s seconds away from packet g.

It is to be noted that packets satisfying the above conditions may or may not be consecutive packets. We define *epoch packet* to be the first packet that did not satisfy any of the mentioned conditions. The epoch packet remains the same until a condition is met and then the latest packet is assigned as the new epoch. If any of the conditions are not met, then it remains the same. As illustrated in Algorithm 1, an epoch packet is compared with every latest packet to capture different delay patterns satisfying our conditions over various intervals. For example, consecutive packets that follow this delay pattern (0, 2ϵ, 0, 2ϵ) do not satisfy our conditions, however, every second packet in this pattern does satisfy our conditions giving rise to opportunities to compensate for accumulated error. Epoch packets provide more data points to calculate frequency error and avoid high error accumulation due to frequent opportunities of syntonization.

The key to restore periodicity is the knowledge of the one-way packets period. We assume that the master is trustworthy and sends one-way packets at a known rate. Current feedback-based clock adjustment in PTP also makes use of the knowledge of packets' period. If the received packets at the client adhere to the same known period, then they are assumed to be equally delayed and do not alter frequency error calculation in a feedforward controller.

Implement a feedforward controller: We filter out the packets satisfying the three conditions mentioned above and transform their timestamps such that the filtered packets are periodic with an error tolerance. Note that these timestamps come from a free running client clock that is not adjusted. This free running clock is labeled "Raw Clock" in Figure 2(b). After transformation, these timestamps emulate packets that experience same delay in the network. The rate of change of these timestamps w.r.t master timestamps over the course of one period determines the frequency error of client clock w.r.t master clock. For example, given two sets of transformed timestamps ($t_{1previous}, t_{2previous}$) and ($t_{1current}, t_{2current}$) for two one-way packets, where t_1 are

ALGORITHM 2: Find Delay Free Packets

```
procedure PACKET FILTERING(sync_period, offset_tolerance, max_delayattack)
    s ← sync_period
    γ ← offset_tolerance
    x ← max_delayattack
sample new packet:
    t_a ← t_epoch_pkt                    ▷ Timestamp of previous packet
    t_b ← t_new_pkt                      ▷ Timestamp of epoch packet
    inter_pkt_dist ← t_b − t_a
    if inter_pkt_dist > 0 then           ▷ In order packets
        if inter_pkt_dist > x + s − γ then   ▷ Condition IV
            (t_1, t_2) ← t_a
        t_epoch_pkt ← t_a
    if inter_pkt_dist < 0 then           ▷ Out of order packets
        if inter_pkt_dist > x − s − γ then   ▷ Condition V
            (t_1, t_2) ← t_b
        t_epoch_pkt ← t_b
    goto sample new packet.
```

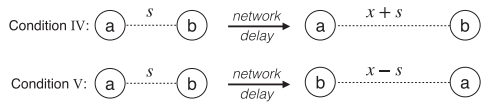


Fig. 10. To calculate clock offset, only those packets are chosen that experience very small delay. The adversary delays packets by no more than x seconds. If two packets sent at a predefined interval s seconds experience delays such that packet a and packet b are $x + s$ apart, then packet a incurs negligible delay (Condition IV), or if packet a gets ahead of packet b by $x - s$ then packet b incurs negligible delay (Condition v).

client timestamps and t_2 are master timestamps, the frequency error is calculated as

$$skew = \frac{(t_{2current} - t_{1current}) - (t_{2previous} - t_{1previous})}{t_{2current} - t_{2previous}}.$$

Note that this equation to calculate frequency error ($skew$) based on previous and current packets with equal delays in the network only holds true for a feedforward controller. The benefit of feedforward over feedback under delay attacks is its ability to calculate frequency error in the presence of large and varying delays. The transformed timestamps are not advantageous to the feedback controller, as it cannot make use of timestamps of equally delayed packets.

We know that feedforward controller is only capable of aligning frequencies, i.e., syntonization between a master and a client, because we gather timestamps from a free running undisciplined clock. However, the feedforward controller gradually builds up error in its $skew$ calculation due to environmental disturbances, error tolerance, path noise, and inaccuracies in timestamping. The accumulation of error due to inaccuracies in $skew$ estimation results in increased time error between the two clocks. To overcome this error and also to synchronize two clocks, we propose a feedback trim to our feedforward controller.

Search for delay-free packets: There are two components in a clock adjustment mechanism: One is the frequency and the other is a time component. Two clocks are *syntonized* if we only adjust the frequency component of a clock w.r.t a reference clock. However, two clocks are *synchronized* when we adjust the time component of a clock. Syntonized clock is enough for applications interested in precisely measuring small inter-event times, whereas synchronized clocks are necessary for applications that coordinate tasks or choreograph acts.

We already provide syntonization to clocks under delay attacks through a feedforward controller. Now we attempt to synchronize two clocks in the presence of delay attacks. To synchronize two clocks, we have to find an offset between them. As the offsets are affected by adversarial network delays, we first filter the packets that are least affected by network delays. In other words, we search for *delay-free* packets. To find packets with less delays, we again rely on periodic one-way packets. Except now our conditions to filter packets are different. In Figure 10, a master sends two packets at a known interval s. We assume that an adversary is able to delay packets by at most x seconds. This assumption is realistic as it can easily be checked by a sanity check. We assert that one of the two packets experience small delay if both packets satisfy the following two conditions:

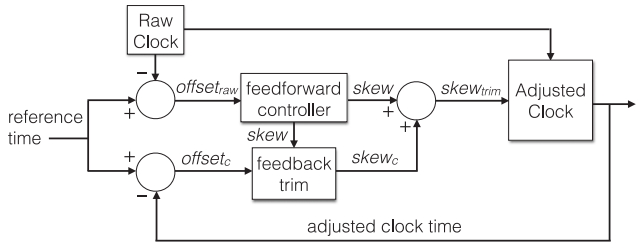


Fig. 11. Feedforward control with feedback trim-based clock adjustment.

(1) *Condition* IV: Given two packets sent at an interval of s seconds, capable of being delayed in the network by at most x seconds, the *first packet* is said to be minimally delayed if the two packets are *in order* and the received interval between these packets is at least $x + s - \gamma$, where γ is the error tolerance in offset calculation.
(2) *Condition* V: Given two packets sent at an interval of s seconds, capable of being delayed by at most x seconds, we consider the *second packet* delay-free if the two packets are *out of order* and the resulting interval between the packets is at least $x - s - \gamma$.

Refer to Algorithm 2 for details.

There must be enough delay variation among two packets to satisfy the above two strict conditions. We do not put the expectation of providing delay variation on the attacker. Instead, we argue that less variations in delay attack would take longer for Conditions IV and V to be met and the clocks to be synchronized. Epoch packets play a key role in finding delay free packets in the presence of small variations by looking for Conditions IV and V over longer intervals. These less variations in delay also supplements our approach to restore periodicity and the ability of our feedforward controller to precisely calculate frequency adjustment. However, huge variations in delay speed up our efforts to synchronize clocks. Nonetheless, our clock syntonization and synchronization architecture works with a variety of delay attacks.

Design a feedback trim for the feedforward controller: Feedback control-based clock adjustment is only capable of providing synchronized time, whereas our implementation of *feedforward control with feedback trim* (feedforward w/feedback trim) shown in Figure 11 is capable of providing both syntonized frequency and synchronized time. Note that Figure 7 gives a complete picture of the proposed architecture while Figure 11 emphasizes only on the control part of the architecture. After identifying the delay-free packets, we record their timestamps (t_1, t_2). In Figure 11, reference time represents t_1, and the timestamp from the client's adjusted clock is t_2. The $offset_c = t_2 - t_1$ is calculated from the timestamps of delay-free packets. A clock's time can be adjusted in two ways: Either add the time error, $offset_c$, to the clock or increase/decrease the clock frequency to gradually compensate for the clock error. We refrain from adding $offset_c$ to adjust the clock, because it would cause discontinuities in time, and a clock should not have time discontinuities greater than it can tolerate. Instead, we feed $offset_c$ and the frequency error ($skew$) from the feedforward controller to the *feedback trim* block. This block transforms the time error to a frequency error $f = offset_c/sync_period$ and trims the frequency $skew$ of feedforward controller. Details of the feedback trim algorithm are found in Algorithm 3.

Putting it all together: An adversary in the network delaying packets can cause a lot of damage to the clock synchronization accuracy. We propose a delay attack-tolerant clock synchronization architecture that bounds the effect of delay attacks on clock errors. Our clock synchronization architecture consists of four major blocks as shown in Figure 7. The timestamps of delayed one-way packets are transformed in an effort to restore their periodicity such that a feedforward controller

ALGORITHM 3: Feedback Trim

```
procedure TRIM FREQUENCY ADJUSTMENT (skew, offset_c, sync_period, max_adj)
delay-free packet:
    f ← offset_c / sync_period                          ▷ time error converted to frequency error
loop:
    skew_adj ← skew + f
    if skew_adj < −max_adj then                         ▷ lower frequency adjustment limit
        skew_c = −max_adj − skew
        f = skew_adj + max_adj                          ▷ residual error
    else if skew_adj > max_adj then                     ▷ upper frequency adjustment limit
        skew_c = max_adj − skew
        f = skew_adj − max_adj                          ▷ residual error
    else
        skew_c = f
        f = 0
    if f! = 0 then
        goto loop                                       ▷ loop again for residual error
    else
        goto delay-free packet                          ▷ no residual error
```

utilizes them to calculate frequency adjustment of a clock. Our architecture is also capable of finding delay-free packets to assist in offset calculation. The feedback control utilizes timestamps of delay-free packets to calculate the clock offset. Using this offset, we trim the adjusted frequency and provide the final trimmed frequency to discipline the clock. Thus presenting our architecture that uses feedforward control w/feedback trim-based clock adjustment.

6 IMPLEMENTATION AND EVALUATION

We implement a testbed to evaluate the clock synchronization architecture. Our testbed consists of Beaglebone Black (BBB) [3] embedded Linux platform used in many applications in industry [20], fog computing [46], smart grids [41], financial market [37], and autonomous agents [27]. The ethernet interface on BBB is IEEE 1588 standard compliant and provides hardware timestamping capability essential for PTP protocol. Our testbed consists of two BBB nodes, one serving as the master and the other as a client. These nodes are connected via an IEEE 1588 compliant switch [4].

We do not necessitate the use of one synchronization protocol for our approach. However, we choose PTP [28] for prototyping because of its demand in various high-precision applications. We run a modified version of linuxPTP [6] on BBB nodes. One modification in PTP is we do not discipline the PTP clock. Instead, we modify virtual clocks derived from that PTP clock. This concept of virtual clocks is not new; the need of multiple disciplinable clocks on a single platform has motivated this architecture [10, 11]. Posix standard [5] virtual clocks derived from the same hardware timer are also part of the Linux kernel. Our approach does not depend on any single architecture. It can work with POSIX clocks in the kernel as well as *timeline* in QoT stack [10]. The only assumption we make is the access to a free running raw clock from which we derive a disciplinable clock. In the context of Linux kernel, CLOCK_MONOTONIC_RAW serves as the free running clock, whereas CLOCK_REALTIME is a disciplinable clock derived from this raw clock. Our software implementation can be plugged into any synchronization protocol as a standalone combination of a filter and a controller. We already provide controllers for PTP that work alongside PTP's traditional PI and Linear regression controllers. A user can switch between any controller of its choice.

We simulate delay attacks by adding delay to the received PTP SYNC packets at the client as shown in Figure 1(b). Note that our approach utilizes only one-way SYNC packets and does not require DELAY_REQ packets as the feedforward controller does not rely on calculated delay. We established in Section 4 that feedforward calculated frequency error from one-way packets that

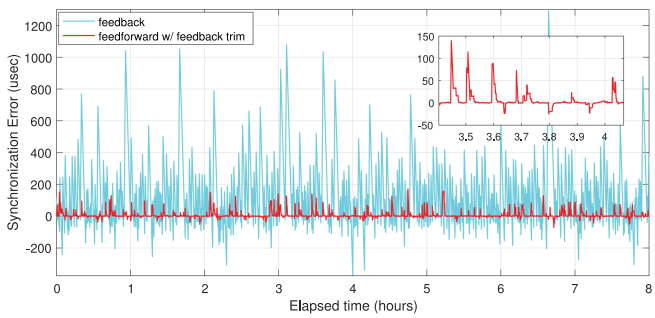
are delayed equally is valid. Therefore, a constant delay attack is not considered an attack for a feedforward controller. Instead it complements frequency error estimation for the controller. But our results for random delay injection show that feedforward-based frequency estimation never converges. Therefore, we evaluate our architecture for various random attack distributions throughout this section and discuss countermeasures for a combination of attacks.

Having established in Section 4 that an attacker is capable of increasing synchronization error by launching random delay attacks within 0 to 1 s, we evaluate our approach on higher delay values to show that our proposed architecture is also capable of overcoming large delay attack values. Though there are filters that raise an exception or time-out when a certain duration elapses, that duration is set according to the communication medium and network conditions and may vary from one medium to another. Hence, we evaluate our architecture for both small and large delay values to support all time synchronization protocols.

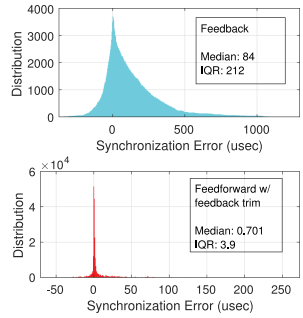
Feedback-only versus Feedforward w/feedback trim: We compare a *Feedback-only* controller used in all synchronization protocols with our proposed *Feedforward with feedback trim* controller in the presence of delay attacks. In this test case, a master periodically sends one-way packets at 2 Hz (0.5 s). An adversary is delaying these packets by randomly choosing a delay value from a uniform distribution of 0 to 2 s. Both controllers are fed the same delayed packets. The adjustable clocks for the controllers are derived from the same timing hardware ticking at 24 MHz. This test scenario is depicted in Figure 3(a), where we alleviate the bias caused by network and hardware variations on our results. In this test case, both the controllers process filtered one-way packets using our *Restore Periodicity* and *Delay-Free packets* techniques explained in Section 5. We choose the frequency error tolerance $\epsilon = 10$ for restoring the periodicity of delayed packets, whereas offset error tolerance $\gamma = 100$ to find delay-free packets. Note that we choose these values of ϵ and γ because they result in minimum error for this experimental scenario. For a smaller γ value, it would take longer for synchronization Conditions IV and V to be met. The longer it takes to synchronize, the more error gets accumulated due to frequency errors, and hence synchronization error increases. Achieved accuracy doe not depend upon individual ϵ and γ values but a combination of these. While both controllers discipline their respective clocks, we continuously measure the time error of the adjustable clocks w.r.t the master clock using the setup shown in Figure 3(b). We run this experiment for 8 hours, and the results are shown in Figure 12.

The time series plot of the synchronization errors in Figure 12(a) shows large fluctuations in feedback clock error. While the clock error for feedforward w/feedback trim has comparatively small variations. We zoomed into this result to show the range of clock error fluctuations for feedforward w/feedback trim control. The feedforward controller calculates the frequency adjustment/skew from raw timestamps of a free running clock. The property of skew is that it does not vary much over time. Hence, there are regions in Figure 12(a) for feedforward control that do not have much variation. As a result, the median of clock error distribution for feedforward w/feedback trim control in Figure 12(b)(2) is almost 0.7 μs with an IQR of only 4 μs. However, the clock error distribution for feedback control has a median of 84 μs with and IQR of 212 μs. These results show 100 times synchronization improvement of feedforward w/feedback trim control over feedback-only control in the presence of random delay attacks sampled from a uniform distribution of 0 to 2 s.

Figure 13 shows a zoomed-in 6-s duration in Figure 12(a). This result gives the reason behind large clock errors in feedback than feedfoward control. As shown, the clock error for feedback control continues to increase while the error for feedforward control is quite consistent. We know that feedback control only relies on the clock offset. This offset is calculated from delay-free packets, and the duration between two delay-free packets is not predictable. The feedback controller adjusts the frequency of the clock calculated from the clock offset and waits for the next delay-free



(a) Time series plot of feedback versus feedforward control w/ feedack trim for a total of 8 hours duration. Feedback clock error has large and continuous fluctuations in the range of -300 to 1300μseconds, while feedforward w/ feedback trim clock error ranges from -50 to 150μseconds with large durations of stable clock error.



(b) Distribution of clock errors shown in part (a). Feedback gives hundred times high error median and IQR as compared to feedforward w/feedback trim.

Fig. 12. Synchronization error of two client clocks with respect to a master clock. One clock is disciplined by a feedback controller, and the other is adjusted by a feedforward controller with feedback trim. For a fair comparison, both controllers process transformed timestamps of periodic one-way packets, and timestamps of delay-free packets. Note that these packets are delayed in the network randomly by 0 to 2 s.

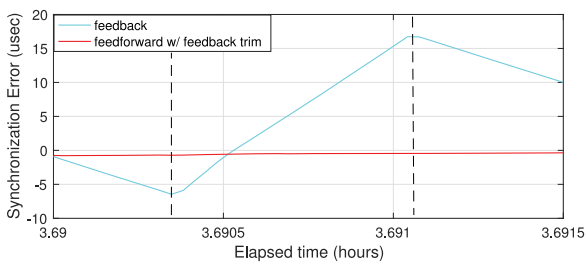
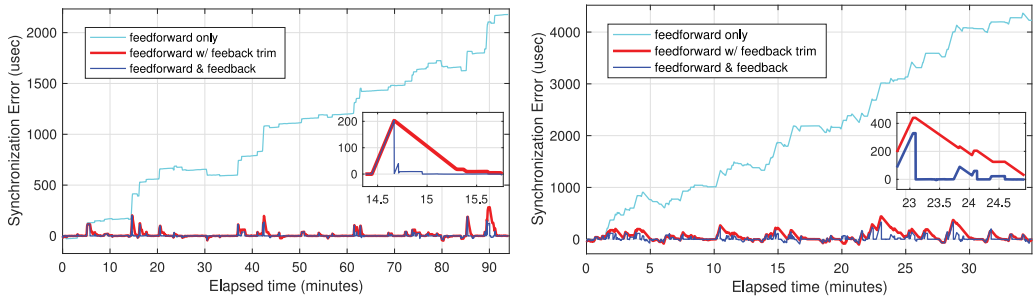


Fig. 13. Zoomed-in plot of 6-s duration from Figure 12(a). Feedback control steers clock toward high errors with high frequencies, whereas feedforward control maintains clock error with consistent frequency.

offset calculation. In the meantime, it updates the clock using the previous frequency adjustment, and the error continues to accumulate as shown in Figure 13 between two dashed lines. In this particular case, the delay-free packet comes after few seconds but it can take longer as well. When the new delay-free packet comes, feedback control identifies the large clock error and steers the error in the other direction. However, feedforward w/feedback trim control does not solely rely on the offset value from delay-free packets. For our controller, the time and frequency components are independent of each other. While waiting for a delay-free offset, it continuously adjusts the clock frequency calculated from transformed timestamps of periodic one-way packets. Hence, feedforward w/feedback trim maintains clock synchronization through syntonization.

As we establish that our approach of feedforward w/feedback trim performs better than feedback-only approach, we now present other controller combinations and find the optimal combination among all. Other controller combinations are *feedforward-only* and independent *feedforward & feedback* under delay attacks. It is to be noted that none of these combinations exist in the current synchronization architectures. The only purpose of this comparison is to validate our choice of feedforward w/feedback trim over other combinations using our filtering and frequency shaping techniques.



(a) Clock synchronization performance of three controllers under random 0 to 2 seconds delay attack. This experiment runs for 95 minutes (x-axis), and the maximum accumulated error is approximately 2500μseconds (y-axis)

(b) Clock synchronization performance of three controllers under random 0 to 4 seconds delay attack. This experiment runs for 35 minutes (x-axis), and the maximum accumulated error is approximately 5000μseconds (y-axis)

Fig. 14. Clock errors for three different controllers. Less error fluctuations for feedforward & feedback control as well as feedforward control w/feedback trim. An added advantage of smooth error adjustment by the latter controller shown in zoomed-in subplots in (a) and (b). Feedfoward-only control never compensates for clock offset and accumulates error at a small rate. Note the change in x-axis and y-axis for both (a) and (b).

(1) A *feedforward-only* approach only calculates the frequency adjustment using timestamps sampled from a free running raw clock. It does not account for clock offset and accumulated clock error due to inaccuracies on feedfoward frequency calculation. There is no loop to feed the error back for compensation. This control can syntonize but cannot synchronize the clocks.
(2) A *feedforward & feedback* approach uses two independent controllers: one feedforward controller to adjust the clock frequency and one feedback controller to fix the clock offset. The feedforward component syntonizes the clock, and the feedback component synchronizes the clock.
(3) A *feedforward w/feedback trim* approach is similar to feedforward & feedback, except here the two controllers are not independent. The feedback calculated offset is used to trim the feedforward calculated frequency.

Comparison of feedforward-only, feedforward & feedback, and feedforward w/feedback trim: Now we compare the three possible controller combinations to find the optimum one in the presence of delay attacks. In this test case, we try to synchronize three virtual clocks on one client to a master. The reason of putting all clocks on a single client is to remove hardware and network variations related biases in results. Each of these clocks is disciplined form one of the above three controllers. For a fair comparison, all the controllers process the same filtered packets after restoring packets periodicity and finding delay-free packets using approaches in Section 5. We plot the clock errors in Figure 14.

In the presence of random 0 to 2 s delay attacks, Figure 14(a) shows that the performance of feedfoward & feedback and feedforward w/feedback trim clock errors is comparable. Their clock errors fluctuate in the range of −50 to 200 μs with large durations of stable error. The feedfoward & feedback median error is 0.27 μs and 0.93 μs IQR. While feedforward w/ feedback trim median error is slightly larger as 1 μs with 11 μs IQR. The slightly better results of feedfoward & feedback control comes at a benefit of increased clock discontinuities as shown in the zoomed plot inside Figure 14(a). This 1-minute zoomed plot shows how both controllers adjust their clock offsets. As feedforward & feedback adjusts offset independently, it has no choice but to jump the time

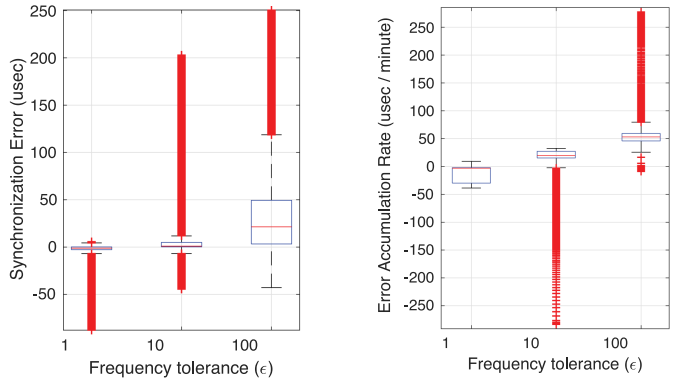
to adjust the clock. These jumps cause discontinuities in time that are not desirable. However, feedforward w/ feedback trim transforms the calculated offset to a frequency adjustment and then gradually adjust the frequency with maximum and minimum bounds to compensate for the offset. Thus avoiding time discontinuities at the cost of increased clock error.

We have yet to explain the performance of feedforward-only approach. We plot the normalized clock error in Figure 14(a) by subtracting all the errors from the first error. Thus, the plot represents the rate of increase in error, because feedforward control does not adjust clock offsets. The increased clock error is due to inaccuracies in calculated frequency adjustment. Note the sudden jumps in error. These jumps occur when a wrong feedforward-based frequency adjustment makes the clock progress at the maximum frequency. We have set the limits of frequency adjustment to not exceed 10 μs/s. In our test duration of almost 90 minutes, the clock error increased to approximately 2 ms at the rate of 22 μs/minute median and 15 μs/minute IQR. Even though this error accumulation rate is low, an adversary may end up accumulating large errors over large durations. Therefore, it is necessary to compensate for these errors. Hence, the feedforward-only approach could only work for measuring small durations inter-event times where the 20 μs/minute error accumulation can be tolerated. In cases where better syntonization or where synchronized clocks are required, this approach would not work.

Performance under large delay attacks: So far our experiments had a random delay injection in the range of 0 to 2 s. We run the same set of experiments of comparing different controller under large delay attacks of 0 to 4 s. The clock errors of feedforward & feedback control and feedforward w/feedback trim control in Figure 14(b) are not as stable as errors for these controllers in Figure 14(a). The median error for feedforward & feedback control is -0.4 μs with 38 μs IQR, while feedforward w/feedback trim control provides 47 μs median error with 124 μs IQR. Even though the high IQR of clock errors shows less error stability, we are still able to provide synchronization accuracy that many applications require. For example, financial sector is a big target for attackers to cause maximum damage by manipulating the causality of financial transactions [37]. Many applications that rely on location/proximity [42] to authenticate users also require a secure sense of time. Attackers may also disrupt temporal forensic analysis [17]. These and many more application areas have traditionally been a target of time delay attacks and require sub-millisecond accuracy. Our proposed approach is targeted toward these applications. With different delay distributions, choosing the right frequency and offset tolerance to filter delayed packets greatly affects the synchronization performance.

Frequency tolerance (ϵ) and offset tolerance (γ): Frequency tolerance ϵ is the upper error limit our system can ignore in calculating frequency adjustment when checking for different conditions to restore delayed packets' periodicity as explained in Section 5. Offset tolerance γ is the system's upper error limit in its offset calculation when finding delay-free packets. Filtering delayed packets that satisfy conditions in Figure 8 and 10 is affected by ϵ and γ value. Too-high tolerance values result in high false positives. Fewer packets are filtered out, which increases the probability of accepting those delayed packets that do not necessarily satisfy the desired conditions. Too-low tolerance produces false negatives, and valid syntonization and synchronization opportunities are missed. Therefore, it is necessary to find the right tolerance values when filtering packets.

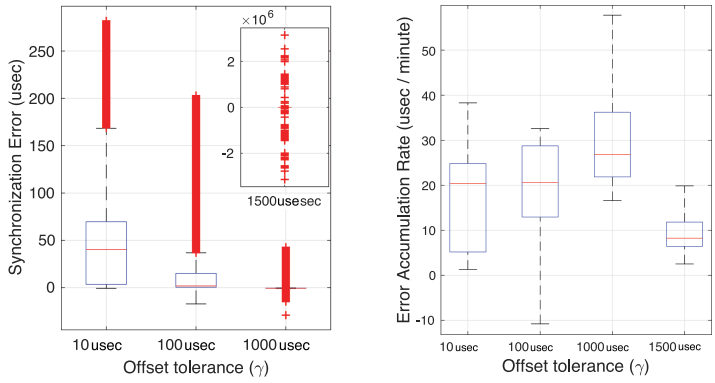
We evaluate our delay-tolerant synchronization approach with three different ϵ values. The result in Figure 15(a) shows the clock error for different ϵ values. For $\epsilon = 1$ μs, the clock error is small with a relatively smaller number of high outliers. For the same ϵ value, the conditions to restore periodicity do not occur often, and hence there are fewer chances for the feedforward controller to calculate frequency adjustment. That is why a wrong frequency adjustment keeps on doing the damage before the next frequency adjustment can be calculated. The results are a few



(a) Affect of frequency tolerance on clock errors

(b) Affect of frequency tolerance on clock errors accumulation rate

Fig. 15. Showcasing the relationship between frequency tolerance and clock error rate. The higher the tolerance, the higher the rate of error accumulation raising the instantaneous clock error.



(a) Affect of offset tolerance on clock error

(b) Affect of offset tolerance on error accumulation rate

Fig. 16. The relationship between offset tolerance and clock error. The higher the tolerance, the smaller the clock error but only to a certain limit beyond which the errors are marked as attacked. Offset tolerance does not affect error accumulation rate.

outliers in Figure 15(a) for 1 μs. On the flip side, the small tolerance range reduces false positives, resulting in valid frequency calculation most of the time and hence small clock error. The error accumulated rate due to wrong frequency calculation is also less for $\epsilon = 1$ μs in Figure 15(b). The high IQR with respect to median error accumulation rate signifies occasional wrong frequency calculations. The clock error and the rate at which this error gets accumulated keep on increasing with the rise in ϵ value. As ϵ increases, the high chances of calculating wrong frequency on false positives increases clock error as well as error accumulation rate.

The clock errors are also dependent on the offset tolerance γ. Note in Figure 16(a) that the clock error keeps on decreasing with an increase in γ. A controller with small γ value takes more time to synchronize, because high tolerance puts strict bounds on offset error and reduces the chance

of getting a delay-free packet. This results in high clock error accumulated due to the inaccuracies in frequency adjustment. A large γ value means getting delay-free packets more frequently, and hence increased opportunities to fix the clock offset and overcome accumulated error due to wrong frequency. But there is a limit to this increase, as too-high a value increases false positives, and the clock error has large outliers and huge discontinuities as shown for $\gamma = 1500$ in the subplot in Figure 16(a). There are huge jumps on the order of seconds that are heavily influenced by delay attacks. Thus, a large γ value is useful but only up to a certain limit. The error accumulation rate is only a function of ϵ, and hence it remains unaffected by γ, as shown in Figure 16(b).

For given deployment conditions, our architecture selects tolerance values by keeping track of error accumulation rate, time taken to synchronize, and calculated offset outliers. High error accumulation rate gives an indication to decrease ϵ, while frequent and large offset outliers is an indication to decrease γ. Reducing γ also speeds up synchronization. The architecture goes through an initial calibration phase to determine ϵ and γ values for a specific deployment. For a given system, we exhaustively search for those ϵ and γ values that provide lower error accumulation rate with a given time to synchronize and a fixed number of offset outliers. Ultimately, a particular combination of frequency and offset tolerance gives small and stable error.

Incremental delay attacks and countermeasures: There are three major kinds of delay attacks: constant delay, random delay, and incremental delay. Most of the other attacks are a combination of these attacks. We already analyzed and evaluated constant and random delay attacks. Indeed, our approach works best under large delay variations, but it can also detect attacks with small delay variations over a longer time. Over small duration, an incremental attacker injects small delays. If these increments are smaller than the frequency tolerance (*epsilon*), then our approach can potentially make use of them to calculate the adjustment frequency. Therefore, consecutive packets undergoing incremental attacks are used to calculate frequency error as long as they remain under the frequency tolerance. Hence, our architecture does not wait longer to get a frequency fix. However, if left unchecked, then a patient adversary can accumulate large error over a long period by inducing small errors in adjusted frequency. To overcome the effects of accumulating errors over long durations, our system compares the timestamp of an epoch packet with the latest packet to look for potential conditions. Our system does not need to know the value of incremental attack; rather, by comparing every arriving packet with an epoch packet, our system can find potential subsequent packets that match our conditions to restore periodicity and to find delay-free packets. Depending upon the value of ϵ and γ, the conditions may occur sooner or later, but it is to be noted that the clocks are syntonized nonetheless and the accumulated error does not exceed the tolerance values. Detailed performance analysis of our architecture in the presence of incremental attacks is omitted because of its equivalent performance under random attacks over long durations (as shown in Figure 12).

Our proposed packet filtering techniques work for various period and tolerance combinations. In benign environments, where there are no delay attacks, our conditions are still met that correspond to frequency and time error calculations. The idea presented in this article is, in the presence of delay attacks, our approach reduces the error by 1,000 times as compared to traditional approaches. Therefore, our architecture works in nonmalicious networks. The idea of restoring packets periodicity in the presence of attacks is equally valid for packets that are not delayed. In a benign network, the packets reach at almost the same interval, and hence it can be used to calculate frequency adjustment. Clock frequency also varies over time depending on the operating conditions, and our architecture tunes frequency tolerance to compensate for errors.

Summarizing all the results, feedforward control w/feedback trim outperforms other controllers in terms of high clock accuracy with no time discontinuities. It also achieves tunable syntonization as well as clock synchronization in the presence of different kinds of delay attacks.

7 CONCLUSION AND FUTURE WORK

Security in context of time is important, as many applications are emerging that have moved away from traditional "clockless" assumption [21]. Systems are increasingly making use of synchronized clocks to enhance the accuracy of network measurements and reduce the complexity of distributed system protocols. However, adversaries are targeting timing primitives for copyright theft, illegal trade, and location theft, and so on. Cryptographic and network security mechanisms have thwarted various attacks on time transfer packets but delay attack is too strong to be mitigated completely. We pointed out the key issues in current clock synchronization architectures that make them vulnerable to delay attacks and propose a new delay attack-tolerant synchronization architecture. Built on top of a feedforward control with feedback trim clock adjustment mechanism coupled with packet filtering techniques, the architecture is capable of bounding delay attack errors.

In future, we intend to provide a formal security analysis of our delay attack-tolerant clock synchronization architecture and provide provable error bounds under all possible attacks. Showing an implementation for NTP is also an interesting direction where we can utilize the excess information from multiple servers to tighten the error bounds for strong delay attacks.

We are also considering some interesting future research directions. We know that securing time transfer packets is meaningless if the timing hardware and software stack is stealthy. Researchers show that an adversary can cause hardware faults by overclocking digital circuits and not satisfying their timing constraints [44]. A malicious OS can manipulate timer registers [24], and a host can lie about time [13]. It would be beneficial to protect all layers of the time stack—the hardware timers, system software, and the network packets.

REFERENCES

[1] 2009. Best Practices for IEEE 1588/ PTP Network Deployment. Retrieved from https://www.microsemi.com/document-portal/doc_view/133167-best-practices-for-ieee-1588-ptp-network-deployment.

[2] European Southern Observatory. 2017. Taking the first picture of a black hole (2017). https://www.eso.org/public/usa/outreach/first-picture-of-a-black-hole/blog/.

[3] Beaglebone Black. Retrieved from https://beagleboard.org/black.

[4] IEEE 1588 Compliant Moxa Switch. Retrieved from https://store.moxa.com/a/cat/industrial-ethernet/ethernet-switches/managed.

[5] IEEE Standard 1003.1, 2004. Retrieved from http://pubs.opengroup.org/onlinepubs/009695399/functions/clock.html.

[6] The Linux PTP Project. Retrieved from http://linuxptp.sourceforge.net.

[7] David W. Allan and Marc Abbott Weiss. 1980. Accurate time and frequency transfer during common-view of a GPS satellite. In *Proceedings of the 34th Annual Symposium on Frequency Control*. IEEE, 334–346.

[8] Robert Annessi, Joachim Fabini, and Tanja Zseby. 2017. It's about time: Securing broadcast time synchronization with data origin authentication. In *Proceedings of the 26th International Conference on Computer Communication and Networks (ICCCN'17)*. IEEE, 1–11.

[9] Robert Annessi, Joachim Fabini, and Tanja Zseby. 2017. SecureTime: Secure multicast time synchronization. *arXiv preprint arXiv:1705.10669* (2017).

[10] Fatima Anwar, Sandeep D'souza, Andrew Symington, Adwait Dongare, Ragunathan Rajkumar, Anthony Rowe, and Mani Srivastava. 2016. Timeline: An operating system abstraction for time-aware applications. In *Proceedings of the Real-Time Systems Symposium (RTSS'16)*. IEEE, 191–202.

[11] Fatima M. Anwar, Amr Alanwar, and Mani B. Srivastava. 2018. OpenClock: A testbed for clock synchronization research. In *2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, 1–6.

[12] Gildas Avoine, Xavier Bultel, Sébastien Gambs, David Gerault, Pascal Lafourcade, Cristina Onete, and Jean-Marc Robert. 2017. A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 800–814.

[13] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM Trans. Comput. Syst.* 33, 3 (2015), 8.

[14] Timothy Broomhead, Julien Ridoux, and Darryl Veitch. 2009. Counter availability and characteristics for feed-forward based synchronization. In *International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS'09)*. IEEE, 1–6.
[15] Nirupama Bulusu, John Heidemann, and Deborah Estrin. 2000. GPS-less low-cost outdoor localization for very small devices. *IEEE Pers. Commun.* 7, 5 (2000), 28–34.
[16] Chen Chen, Himanshu Raj, Stefan Saroiu, and Alec Wolman. 2014. cTPM: A cloud TPM for cross-device trusted applications. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*. 187–201.
[17] Guoqiang Jerry Chen, Janet L. Wiener, Shridhar Iyer, Anshul Jaiswal, Ran Lei, Nikhil Simha, Wei Wang, Kevin Wilfong, Tim Williamson, and Serhat Yilmaz. 2016. Realtime data processing at facebook. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1087–1098.
[18] Sanchuan Chen, Xiaokuan Zhang, Michael K. Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 7–18.
[19] Omer Deutsch, Neta Rozen Schiff, Danny Dolev, and Michael Schapira. 2018. Preventing (network) time travel with chronos. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'18)*.
[20] John C. Eidson and John Tengdin. 2003. IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems and applications to the power industry. In *Proceedings of the Distributech Conference*. 4–6.
[21] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2018. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association.
[22] Gerhard P. Hancke and Markus G. Kuhn. 2008. Attacks on time-of-flight distance bounding channels. In *Proceedings of the 1st ACM Conference on Wireless Network Security*. ACM, 194–202.
[23] Mohammad Kamrul Hasan, Rashid Abdelhaleem Saeed, Aisha-Hassan Abdalla, Shayla Islam, Omer Mahmoud, Othman Khalifah, Shihab A. Hameed, and Ahmad Fadzil Ismail. 2011. An investigation of femtocell network synchronization. In *Proceedings of the IEEE Conference on Open Systems (ICOS'11)*. IEEE, 196–201.
[24] intel sgx. 2016. Intel(R) Software Guard Extensions Software Developer Manual. Retieved from https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3d-part-4-manual.pdf.
[25] Eyal Itkin and Avishai Wool. 2017. A security analysis and revised security extension for the precision time protocol. *IEEE Trans. Depend. Secure Comput.* 17, 1 (2017), 22–34.
[26] Anantha K. Karthik and Rick S. Blum. 2018. Estimation theory-based robust phase offset determination in presence of possible path asymmetries. *IEEE Trans. Commun.* 66, 4 (2018), 1624–1635.
[27] Andrew J. Kerns, Daniel P. Shepard, Jahshan A. Bhatti, and Todd E. Humphreys. 2014. Unmanned aircraft capture and control via GPS spoofing. *J. Field Robot.* 31, 4 (2014), 617–636.
[28] J. Eidson and K. Lee. 2002. IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. In *2nd ISA/IEEE Sensors for Industry Conference*. IEEE, 98–105.
[29] Yang Li, Rui Tan, and David K. Y. Yau. 2017. Natural timestamping using powerline electromagnetic radiation. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM, 55–66.
[30] Martin Lukac, Paul Davis, Robert Clayton, and Deborah Estrin. 2009. Recovering temporal integrity with data driven time synchronization. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*. IEEE Computer Society, 61–72.
[31] Aanchal Malhotra, Matthew Van Gundy, Mayank Varia, Haydn Kennedy, Jonathan Gardner, and Sharon Goldberg. 2017. The security of NTP's datagram protocol. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 405–423.
[32] David L. Mills. 1991. Internet time synchronization: The network time protocol. *IEEE Trans. Commun.* 39, 10 (1991).
[33] Tal Mizrahi. 2012. Slave diversity: Using multiple paths to improve the accuracy of clock synchronization protocols. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS'12)*. IEEE, 1–6.
[34] Tal Mizrahi. 2014. *Security Requirements of Time Protocols in Packet Switched Networks*. Technical Report.
[35] Bassam Moussa, Mourad Debbabi, and Chadi Assi. 2016. A detection and mitigation model for PTP delay attack in an IEC 61850 substation. *IEEE Transactions on Smart Grid* 9, 5 (2016), 3954–3965.
[36] Lakshay Narula and Todd Humphreys. 2018. Requirements for secure clock synchronization. *IEEE J. Select. Top. Signal Process.* 12, 4 (2018), 749–762.
[37] Mark L. Psiaki and Todd E. Humphreys. 2016. GNSS spoofing and detection. *Proc. IEEE* 104, 6 (2016), 1258–1270.

[38] Dima Rabadi, Rui Tan, David K. Y. Yau, and Sreejaya Viswanathan. 2017. Taming asymmetric network delays for clock synchronization using power grid voltage. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 874–886.

[39] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, et al. 2016. fTPM: A software-only implementation of a TPM chip. In *Proceedings of the USENIX Security Symposium*. 841–856.

[40] Julien Ridoux, Darryl Veitch, and Timothy Broomhead. 2012. The case for feed-forward clock synchronization. *IEEE/ACM Trans. Netw.* 20, 1 (2012), 231–242.

[41] Daniel P. Shepard, Jahshan A. Bhatti, Todd E. Humphreys, and Aaron A. Fansler. 2012. Evaluation of smart grid and civilian UAV vulnerability to GPS spoofing attacks. In *Proceedings of the ION GNSS Meeting*, Vol. 3. 3591–3605.

[42] Dave Singelee and Bart Preneel. 2005. Location verification using secure distance bounding protocols. In *Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005*. IEEE, 7–pp.

[43] Kun Sun, Peng Ning, and Cliff Wang. 2006. TinySeRSync: Secure and resilient time synchronization in wireless sensor networks. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*. ACM, 264–277.

[44] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. 2017. CLKSCREW: Exposing the perils of security-oblivious energy management. In *Proceedings of the 26th USENIX Security Symposium*.

[45] Markus Ullmann and Matthias Vögeler. 2009. Delay attacks—Implication on NTP and PTP time synchronization. In *Proceedings of the International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS'09)*. IEEE, 1–6.

[46] Peter Volgyesi, Abhishek Dubey, Timothy Krentz, Istvan Madari, Mary Metelko, and Gabor Karsai. 2017. Time synchronization services for low-cost fog computing applications. In *Proceedings of the 28th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*. ACM, 57–63.

[47] Qingyu Yang, Dou An, and Wei Yu. 2013. On time desynchronization attack against IEEE 1588 protocol in power grid systems. In *Proceedings of the IEEE Energytech 2013*. IEEE, 1–5.

Received January 2019; revised October 2019; accepted February 2020