DIGITAL SIGNATURES FOR PTP USING TRANSPARENT CLOCKS

A Thesis

by

RACHEL ELLEN FLORES-MEATH

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jyh-Charn Liu
Co-Chair of Committee,	Daniel Ragsdale
Committee Members,	Paul Gratz
	Guofei Gu
Head of Department,	Dilma Da Silva

August 2017

Major Subject: Computer Science

ABSTRACT

Smart grids use synchronous real-time measurements from phasor measurement units (PMU) across portions of a grid to provide grid-wide integrity. Achieving synchronicity requires either accurate GPS clocks at each PMU or a high-resolution clock synchronization protocol, such as the Precision Time Protocol (PTP), specified in IEEE 1588 with the power profile in IEEE C37.238-2011. PTP does not natively include measures to provide authenticity or integrity for timestamps transmitted across an Ethernet network, though there has been recent work in providing end-to-end integrity of transmitted timestamps. However, PTP for use in the smart grid requires a version of the protocol in which network switches update the trusted timestamp in flight, meaning that an end-to-end approach is no longer sufficient. We propose two methods to provide for the integrity of the transmitted and updated timestamps as well as to ensure the authority of all network devices altering the time. In the first, we amend the PTP standard to include signatures as part of the time packet itself at the cost of increased jitter in the system. In the second, we transmit these signatures over a wireless network, reducing congestion on the original network. We test both methods on a simulated PTP switch intended for experimentation only and demonstrate that the use of a second network dedicated to verification-related information is better for current networks, as including signatures in the original packet causes more jitter than is acceptable for synchronizing PMUs in particular.

DEDICATION

To my parents, Valerie, and my beloved husband John for their love and support.

ACKNOWLEDGMENTS

I would like to thank Drs. Steve Liu and Daniel Ragsdale for their invaluable support and guidance. Special thanks to Dr. Guofei Gu, Dr. Paul Gratz, and Dr. Mladen Kezunovic for their assistance.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professors Daniel Ragsdale and Jyh-Charn (Steve) Liu, co-advisors, and Guofei Gu of the Department of Computer Science and Engineering and Professor Paul Gratz of the Department of Electrical and Computer Engineering. Additional input came from Professor Mladen Kezunovic of the Department of Electrical and Computer Engineering. All work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a fellowship from the National Physical Science Consortium and a stipend from Sandia National Laboratories.

NOMENCLATURE

PTP	Precision Time Protocol
MAC	Message Authentication Code
NTP	Network Time Protocol
GPS	Global Positioning System

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Introduction

1.1.1 Motivation

Sensor nodes in critical infrastructure, such as the phasor measurement units (PMUs) used in smart grids, produce synchronous real-time measurements at various points in the grid. In the case of PMUs, these measurements (synchrophasers) of magnitude and phase angle in the electric waves allow for the grid to maintain its integrity throughout. Minimizing the offset between clocks on the PMUs then requires accurate time synchronization.

Ideally, we use Global Positioning System signals as a time source for high resolution clock synchronization, with each PMU receiving the same time source from GPS. This means that there is no need for additional clock synchronization within each PMU, as the GPS signals are already correctly synchronized. However, GPS signals can be subject to spoofing[1], so an alternative to this clock source will be useful both if GPS spoofing cannot be sufficiently mitigated and as a fallback measure if GPS signals are currently unavailable to a system.

Without relying on GPS as a time source, we then have to synchronize with the physical clocks in PMUs. This is a well-studied problem, and in the case of a fully-trusted system, we can rely on PTP to synchronize clocks to sub-microsecond resolution.

However, it is unrealistic to assume that a network, even with isolated circuits, can be fully trusted. There are several attack scenarios, including the network switches themselves being compromised to alter timestamps and move clocks out of sync. As a result of this, it is critical to provide proof via cryptography that no node in the network has interfered with the correct transmission of synchronization messages. Digital signatures provide authentication of message sources and verify the integrity of those messages, satisfying this concern.

PTP is already in use in a few critical situations. For example, PTP synchronizes systems in substations in multiple countries [2]. Additionally, PTP can be used in Wide Area Protection, which currently relies on GPS synchronization [3]. Both of these situations require the protocol to be secure in order to avoid damage to infrastructure.

To address the potential for timestamps to be altered and cause serious errors in power systems, we assess the impact in terms of time resolution and resource use of wrapping authentication protocols around key portions of the PTP protocol. We develop measures that allow for detection of faulting nodes, with the note that correction of faults is out of the scope of this work.

1.2 Precision Time Protocol

PTP is a clock synchronization protocol with sub-microsecond resolution defined in IEEE 1588-2008[4]. In PTP, there are three types of nodes that we consider: grandmaster clocks that synchronize the rest of the network to it, slave nodes that are synchronized to the grandmaster, and transparent clocks that account for network delays in the path from grandmaster to slave. We also consider nodes to be in a hierarchy, with the grandmaster as the root of a tree, as in Figure 1.1.

The second revision of the standard, released in 2008, adds the ability to better adjust for any delays that are encountered when transmitting timestamps through the network, as we fully expect multiple hops betweeen a grandmaster and slave node. Each hop adds to a correction factor that is then added to the original timestamp from the grandmaster and accounts for the time taken to pass through the device. The net result is that we consider the transmission as though it was performed along a single hop. The problem here is that we have to trust all nodes along the path to correctly adjust the correction factor.

The protocol makes several assumptions about networks using the protocol. Messages

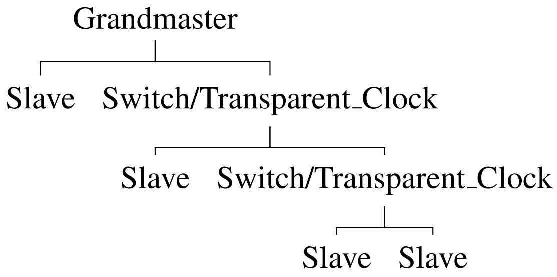


Figure 1.1: A sample PTP network hierarchy.

may occasionally be dropped, duplicated, received out of order. Both multicast and unicast models are supported.

PTP devices coexist with non-PTP devices on a network, so it is necessary to assume other traffic on the network. The devices, each of which has their own unique clock id, are divided into these types:

- Ordinary clocks
- Boundary clocks
- End-to-end transparent clocks
- Peer-to-peer transparent clocks
- Management nodes

Ordinary clocks use two logical interfaces on a single physical port to handle PTP messages. One handles general messages, while the other handles the event-related messages that require timestamping. These clocks can be grandmasters or slaves.

Boundary clocks can be either master or slave for different PTP domains.

End-to-end transparent clocks compute the residence time (egress time minus ingress time) of event messages and update the correctionField of the message to account for the delay from the message entering and leaving the switch. These clocks are commonly syntonized to the grandmaster to have their clock tick at the same rate as the grandmaster.

Peer-to-peer transparent clocks go one step further by computing the delay between itself and its peers and update the correctionField with both the peer delay from the node it received the message and the residence time.

Management nodes allow for configuration and maintenance of the PTP network and may also be combined with another device type.

While the PTP standard does not require transparent clocks, we use these as they are more accurate that the alternative method, boundary clocks.

1.2.1 Boundary versus Transparent Clocks

Boundary clocks do not adjust packets in flight. To adjust for the path delay inherent during network transmission, end-to-end delay calculation is performed[4, 11.3]. In this, the slaves receving a Sync packet track the time of transmission of the Sync message, receipt of the Sync message, transmission of a Delay_Req message and the receipt thereof. These timestamps, as well as any correction field adjustments, provide the slave with rough estimate of the round-trip time from grandmaster to slave.

Transparent clocks operate as network switches that also can adjust timestamps to reflect the residence time of a packet as well as the time a packet takes to travel between a switch and its peers. IEEE 1588-2008 introduced a new set of PTP messages to calculate the peer delay: Pdelay_Req, Pdelay_Resp, and Pdelay_Resp_Follow_Up. Peer delay calculation, detailed in [4, 11.4], requires periodic transmission of a Pdelay_Req message from a clock. The peer delay calculation strongly resembles the Sync/Delay message pairs for boundary clocks, though instead of traversing a full network, the Pdelay messages go only to immediate pairs. This means that peer delay times are known for each pair of peers in the network.

Transparent clocks should be syntonized to the grandmaster. That is, the clock ticks at the same rate as the grandmaster to ensure that updated timestamps are usable by the slave.

1.2.2 One-step and Two-step Clocks

To adjust timestamps in flight requires the ability to edit packets as the packet is leaving the physical layer. As this is not always feasible, the PTP standard provides two ways to consider how clocks transmit timestamps in the event messages. One-step clocks are able to directly provide time in a single event message, like Sync or Pdelay_Req. Two-step clocks transmit a rough timestamp in an event message and then provide the more accurate timestamp reflecting actual egress time in a general message, denoted as a Follow_Up. One-step clocks are preferable for reducing network traffic, but both clock types are acceptable.

1.2.3 Synchronization Process

In order to synchronize a network of slave nodes, a grandmaster clock with the most accurate clock is elected. This clock then sends a series of Sync messages, designed to start synchronizing with all of the slave nodes. In our transparent clock model, the switches add the network delay from the node who sent the message to them before passing the message along to the next nodes. The flow of a Sync packet is shown in Fig. 1.2. For convenience, we combine the transparent clock and an unmanaged switch into a single node for convenience, though we unicast to the transparent clock and multicast to the rest of the network.

In case the grandmaster cannot provide an accurate egress time in the original Sync message, a Follow_Up message contains that timestamp. The slave notes the ingress time of the Sync message and, in the peer delay mechanism, adds the correction field value in the Sync packet computed by the peer-to-peer transparent clocks between grandmaster

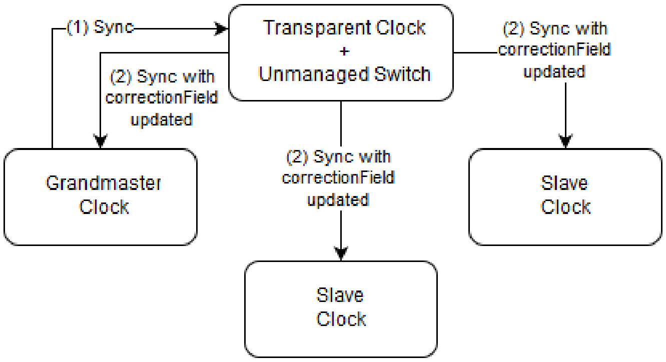

(1) Sync
Transparent Clock
+
Unmanaged Switch
(2) Sync with correctionField updated
(2) Sync with correctionField updated
(2) Sync with correctionField updated
Grandmaster Clock
Slave Clock
Slave Clock

Figure 1.2: Flow of Sync packet.

and slave.

In a peer-to-peer transparent clock system, we also need to know the network delay between two adjacent nodes in a network. This peer delay is negotiated by each node, including transparent clocks.

To account for network delay as a Sync or Follow_Up message, transparent clocks edit the correction field of a Sync or Follow_Up message. This is a field separate from the grandmaster's timestamp intended for the sum of all peer delays and residence times (here, $time_egress - time_ingress$ on the switch), so that the slave nodes can compute a corrected timestamp. The residence time can be highly variable, depending on the number of packets in the queue that must be processed before the Sync of Follow_Up message can be forwarded.

1.2.4 Message Types

Thus far, we have discussed only the message types related directly to the transmission and correction of timestamps. As these messages are the only ones that must be updated by transparent clocks, they are the only ones for which existing protection schemes, used in systems with only boundary clocks, are insufficient. Therefore, we only concern ourselves with Sync, Follow_Up, Pdelay_Req, Pdelay_Resp, and Pdelay_Resp_Follow_Up. For all other message types, we assume the system is otherwise secured.

1.2.5 PTP Packets

PTP packets share a common header containing several variables used to identify the type of message and the original sender. Sync/Follow_Up and Pdelay messages append a timestamp and the Pdelay responses add a clock identity after that to indicate to whom the Pdelay response is intended.

As mentioned, the header contains critical information[4, 13.3]:

- transportSpecific and messageType: a byte indicating the type of network for a packet as well as the type of message
- domainNumber: a byte for which domain the clock is currently operating. Multiple domains can exist on a network, so the domain number is used to segregate the clocks into their correct domain
- correctionField: 8 bytes containing a timestamp correction in scaled nanoseconds, used for storing the peer path delays and residence times for the Sync/Follow_Up and Pdelay messages
- sourcePortIdentity: 10 bytes containing the packet's originating clock identity.
- sequenceId: 2 bytes of monotonically increasing values intended to prevent out-of-order handling for Sync/Follow_Up and Pdelay messages. Each clock maintains a separate pool of sequence ids for each of these two message categories. Only the most recent sequenceId is considered valid for adjusting timestamps.

The other fields are not as valuable for our work, as they do not directly contribute to the unique identity of or adjustment of timestamp-containing packets.

1.3 PTP Power Profile

The IEEE C37.238-2011[5] standard provides a profile for the particular configuration of PTP needed for power systems. There are several key takeaways, including that there are at most 16 hops from grandmaster to slave, that PTP messages are sent multicast over Layer 2 (Ethernet frames), that transparent clocks must be used, and that either one- or two-step clocks can be used. The standard specifies the use of multicast messages and IEEE 802.3/Ethernet transport mapping, further detailed in [4, Annex F].

With at most 16 hops from grandmaster to slave, there is a hard limit on the depth of the network. This is a reduction from the 255 boundary clocks that could exist in IEEE 1588[4, 6.2.f], but still provides a limit on the number of keys needed and the number of computations required to pass a timestamp from grandmaster to slave.

The requirement to use transparent clocks requires the additional trust of switches, as the use of boundary clocks had previously removed any concern of message alteration by switches.

Sync and Pdelay_Req messages must be sent every second, so the computation required to secure timestamps must be brief to avoid causing a backup of pending messages.

While the original PTP standard does not require VLAN tagging, the power profile requires IEEE 802.1Q-tagged frames[5, 5.6]. This means that we will need to handle both VLAN-tagged and untagged frames, though any transmitted frames must be tagged. The tag reduces the amount of space possible for signatures by a few bytes.

IEEE 1588-2008 provides for TLV (type, length, value) extensions at the end of a packet. The power profile requires the ORGANIZATION_EXTENSION and ALTERNATE_TIME_OFFSET_INDICATOR TLVs, but these are only appended to Announce (master clock-only) messages, which do not have timestamp updates contained within them.

Annex B requires a maximum time error of $\pm 1\mu$s when the network is at 80% wire-speed. The time inaccuracy of grandmasters should be better than 0.2μs, but the inaccuracy may exceed that so long as the 1μs error is met.

Our work cannot meet the tight timing requirements of the power profile, so we focus instead on relative performance, i.e. the additional performance hit of adding security to transparent clocks.

1.4 Prior Work

Securing PTP networks has received academic consideration, but most solutions incur overhead on the system. More critically, the effect of these measures on the jitter of the system has not been determined, so the addition of security mechanisms could degrade PTP's resolution to an unacceptable level.

Liu et al.[2] found that a variety of factors in the PTP protocol affect timing resulting in high jitter for unsecured networks. Their tests used a physical network, which means that their results will be more accurate than a software-based implementation, especially when using a virtual network adapter. Their work identified a potential issue in that a one-step transparent clock performed worse than the cascading boundary clock, contrary to theory, but they attribute the bizarre result to a hardware issue. This underlines the potential issues when working with hardware, suggesting that a well-known and well-maintained software version might be more viable for avoiding implementation issues.

They simulated a heavy traffic with the manual addition of messages that they could expect to see on a substation's network. This is not as ideal as a simulating the full system with only realistic messages involved, but it does allow for better control of the congestion on the network.

Liu et al.[2] found it necessary to perform 128 different tests to fully exercise variations in PTP implementations. By taking their recommendation that point-to-point communication over the datalink layer with a one-step clock has minimal jitter and increased resilience to background traffic, we minimize excess tests. However, holding all other variables and also testing end-to-end communication will ensure that we better test the system's resilience to rogue middle nodes. We also consider both cascaded and transparent clocks to better understand the effect of adding cryptography.

Guo et al.[6] considered lightweight schemes for synchronizing sensor networks using XMPP. Broadcast messages transfer encrypted timestamps derived through a lightweight scheme, though the precise scheme is left unspecified. In our work, we consider the pros and cons of various lightweight algorithms, with the primary consideration being that we do not need the added computation time of ensuring confidentiality.

NTP, as the more secure but less precise alternative to PTP, provides an excellent starting point when identifying concerns common to any time synchronization protocol, as it predates PTP and has thus been subject to more review. Mills[7] measured stability and accuracy of NTP servers. He also detailed precisely the properties required of a reliable and accurate time synchronization protocol, providing a checklist for the properties that must still hold when authentication is added to PTP. Mills' survey was performed across the Internet instead of in the more tightly controlled lab environment we use. The limited environment means that we will not be able to see effects across a wide variety of systems using PTP, so our measured results may be somewhat optimistic, but the theoretical analyses will still hold.

Combining PTP research with SDN-based networks allows for better protection from Byzantine actors. Mizrahi and Moses[8] approach the combination from an unusual direction by effectively reversing the protocol in ReversePTP by sending all times to a single controller node. Their approach could avoid issues resulting from Byzantine peers, as the master switch can adjust routes based on timestamps reported by the slaves. While we do not use SDN for our approach, the correction mechanisms present can be used in addition to our error detection system.

Moreira et al.[9] cover the Annex K extension to PTP that was intended to correct security vulnerabilities. The extension had several flaws, leading to a flurry of suggested new schemes, including the use of IPSec and MACSec[10]. This is further addressed in RFC 7384 [11], which outlines requirements of a system that secures PTP. IPSec and MACSec both allow for the protection of messages from the grandmaster, but there is an additional cost to computation by encrypting the message contents. Also, by encrypting the contents, implementing transparent clocks with either of these schemes requires the concatenation of the correction factor to each message (assuming it could even be identified as related to time!), causing a sharp increase in message size. While encrypting the message contents might be beneficial to prevent an adversary from eavesdropping on the messages involved, the size of packets and the overall network flow reveals the type of message, so a grandmaster could still be identified and targeted.

Cahn et al.[12] describe adding SDN to the grid in general. Their implementation used physical devices and had latency comparable with Ethernet switches, but variance in the system was not covered.

Elson et al.[13] describe an alternative to NTP that they implemented in software on commodity hardware. This high-resolution alternative to NTP also lacks built-in security, like PTP, suggesting that the work to secure PTP should be made with enough potential generality to extend to other protocols.

Siu et al.[14] profile the time error seen in cascading boundary clocks. Their bounds provide useful references for verifying our results. Further, their work emphasizes the need for transparent clocking, as cascading boundary clocks exhibit increasingly large time error.

Dalmas et al.[15] address resilience to Byzantine master clocks in PTP. Their approach elects a new grandmaster if the master's time is sufficiently different from a GPS-based time. This does not address a deliberately malicious node and requires the faulty node to cooperate with their protocol by switching itself off, a scenario that is unlikely to happen in the case of a malicious master node.

Itkin and Wool[16] propose using elliptic curve-based authentication (EdDSA). This method requires 280880 cycles to verify a signature ($14\mu s$ on a Nehalem CPU) and creates signatures in $9\mu s$ [17]. While this is a remarkably short time for elliptic curve cryptography, it is possible to reduce the cycle count while making little security sacrifice. Their key survives for 7 years at 20 Sync messages a second, so a weaker key or one with a less lengthy lifetime can still provide strong security with minimal impact on the system. Additionally, EdDSA currently lacks a hardware implementation and only exists in software, so a NIC that supports this algorithm would require a second general purpose CPU just to compute signatures, which is not feasible. A solution with both hardware and software implementations meeting the resolution of PTP is necessary to see widespread adoption.

Also, their solution notes a gap in which an in-band adversary delays the network route. Transparent clocks, in which each node updates the time as it proceeds through the network, could resolve this except that the original timestamp is what was signed. Any information in the correction field would need to be considered suspect at best. This necessitates the inclusion of a trusted controller that can manage the route of a packet through the network. It also requires a solution that can authenticate the information generated at each hop to ensure that no intermediate node can spoof a delay.

When considering possible authentication schemes, we look to past work in sequential signature aggregation. This field authenticates each hop in a path from source to destination and thus ensures that no hop can falsify the value reported by a prior node. We still may have issues with faulty nodes forging values for themselves, but it would be constrained to only those nodes as modifying other nodes' values would cause verification to fail. We consider both public key signature aggregation and MAC aggregation, with the concession that symmetry key cryptography is faster but requires more key exchanges.

In sequential aggregation, a hop in the chain verifies the prior results. This can lead to monotonically increasing packet sizes, which is not ideal on a congested network. As such, history-free sequential MAC aggregation[18] or signature aggregation[19] can be a viable way to constrain either the packet size or network congestion if authenticated packets are sent separately. This technique does incur additional latency on the network[20], but if the computations needed on each node have minimal variance in runtime, then this can be included in final correction field calculations. The net result is that the computation on each hop needs to be of minimal jitter, but the destination node can take an arbitrarily long amount of time (constrained by drift) to finish the verification process as it has a receipt timestamp.

2. SYSTEM DESIGN

2.1 System Model

We model critical infrastructure as a hierarchy of three types of nodes: the grandmaster, slave end nodes, and intermediate hops between the grandmaster and slaves. At its simplest, this is a tree fanning out from the grandmaster with intermediate nodes bridging the gap to the slave nodes.

We could also model this as strictly peer-to-peer communication or as a client-server model. Peer-to-peer addresses the issue of trusting other hops in the network, but does not reflect the different roles of nodes. The client-server model could suffice if we only use cascading boundary clocks, but this would give us less tight synchronization.

We consider the impacts of a node in each of these categories failing. A faulty grandmaster leads to all other clocks potentially being inaccurate, so this has been addressed by a variety of works including [15]. Intermediate nodes could modify packets without authentication of prior messages or simply drop the Sync messages. A faulty slave node only affects its own clock. Each type of failure is still concerning, so detecting where a fault exists is critical to perform quickly to minimize the damage possible.

Fortunately, PTP involves bidirectional and frequent communication between master and slave nodes. The grandmaster can send the Sync message as frequently as 10Hz. In response to a Sync message, the slave node replies with a Delay_Req or Pdelay_Req, allowing the grandmaster and slave nodes to calculate a path delay.

As the rate of Sync messages increases, slave nodes are more able to tolerate dropped or faulty messages before having drifted sufficiently far to be out of sync with other clocks in the system. This means that we can reduce network congestion by only signing a fraction of messages, with each signature verifying the originator and contents of the aggregated messages. We also minimize latency by signing with less frequency, though we do increase the time to detect an error in the system to an extent by delaying the verification process.

Despite the limited tolerance of dropped packets, packet manipulation is unacceptable, so we must authenticate the original message. However, the use of transparent clocks means that we must also authenticate correction field modifications generated along the route from grandmaster to slave. Authentication performed along the route reduces the chance that the slave node is malicious by having intermediate nodes verify messages even if the slave node decides the message is inauthentic. This makes all nodes watchdogs of their neighbors and can help pinpoint the point(s) of failure in a network if a message is found to have been corrupted at some point along the path.

2.2 Constraints

In targeting the power system profile for PTP[5], we need to authenticate a system with multicast transmission of original PTP messages. Synchronization requests are sent no less than once per second, limiting the overhead afforded to our system.

As messages are transmitted across an already-congested network, authentication must either piggyback on existing messages or add minimal additional traffic to the network.

Original PTP messages must be sent unencrypted to ensure that the transparent clock alterations can be made to the original packet. This means we do not consider algorithms providing authenticated encryption, such as TriviA[21], which could otherwise avoid the issue of switches maliciously editing packets.

Unless we can be wholly assured that we compute signatures in a constant number of cycles, we must tighten the constraints of the power profile to only use two-step clocks. That is, the grandmaster's initial Sync message causes all switches in the path from grandmaster to slave to record the residence time and parent peer delay time for that Sync to be transmitted in the Follow_Up packet issue by the grandmaster.

By using a two-step clock, we avoid having to estimate our signing time, which degrades the resolution of the clock. The estimate is necessary for a one-step clock as we must sign the residence time, which is the egress time less the ingress time, in the packet that has not yet left the switch.

The use of two-step clocks adds another requirement that the path taken by Sync and Follow_Up packets is identical[4, 7.4.2]. While we have control of the switches used to calculate path delays, other networking hardware may exist and could try to reroute traffic. Therefore, transparent clocks can only accept incoming traffic from other clocks, a feature we would require to maintain accurate path delay calculations anyhow.

2.2.1 Out of Scope Faults

In a networked system, any number of faults can occur. We consider the situations that we can protect against, while taking care to note that our defenses do not and cannot cover all faults.

In particular, jamming, of either the trusted clock source or of the network itself, would prevent the PTP traffic from reaching its destination. If the PTP packets cannot go through the network, we hardly expect our verification data to reach its destination as well.

Without a trusted clock source, our grandmaster and transparent clocks are forced to rely upon a free-running local clock. This is not ideal, but it does provide a fallback. The loss of trusted clock source is detectable in a log and therefore is outside the scope of this work.

We require timestamps to be accurately taken. Without trust in our time source, we cannot begin to consider the issues of protecting that time.

Grandmaster election is not considered here. The choice of best master clock is not dependent on the path delays, and transparent clocks do not provide any adjustment to

BMCA packets. Therefore, an existing end-to-end scheme, such as in [16] suffices to ensure that grandmasters are appropriately elected.

2.3 Overview

Synchronization messages travel between grandmaster and slave nodes, via switches that may alter the contents of packets. These switches act as transparent clocks and are supposed to modify the correction field, but a malicious switch could alter other portions of the packet or introduce an incorrect value to the correction field.

Switches edit the correction field to reflect the residence time ($t_{egress} - t_{ingress}$ of the packet in the switch, as expected. However, the switch also stores a set of at most m residence times in an array, ordered by the sequence id of the packet they modify. This means that any dropped packets, for which the switch has not seen the sequence id, will have their corresponding residence time set to all bits high, matching PTP's overflow on residence time.

Slave and grandmaster nodes verify the preceding m packets from the grandmaster/slave, respectively. If no packets are missing, the node verifies that the sent signature fits the sent packets with the correction field set to 0 for each packet. If packets were missing, then the resulting signature cannot be verified. Further verification of the Pdelay value can be done between adjacent nodes, ensuring that no switch can spoof the actual Pdelay value between it and another node.

2.3.1 Symmetric versus Asymmetric Authentication

In deciding whether to use symmetric or asymmetric cryptography for authenticating messages, we considered the tradeoffs. Namely, symmetric authentication, using MACs, is faster and requires less hardware than an asymmetric signature. However, while the PTP messages are multicast, a symmetric approach requires each node, including switches, to store separate keys to ensure that no one node can spoof another. A signature needs to only be performed once per round of m messages.

The fundamental issue boils down to the resources available to all nodes. Switches by far have the tightest constraint, as they are expected to handle all traffic quickly. We can work around the switch issue by having the ingress/egress times forwarded to another machine that signs packets, but this simply reintroduces the problem of sending trusted information through an untrusted network or requires each switch to have a tagalong computer. Neither case is desirable.

In lightweight cryptography, there is a limited number of algorithms available for both symmetric and asymmetric cryptography. Of these algorithms, Poly1305-AES[22] and EdDSA[17], respectively, provide authentication with as little overhead as possible.

Since we want to minimize the storage requirements on our slave clocks, asymmetric authentication algorithms are used here, though both will work as long as system space allows. Additionally, this resolves issues related to redirection of packets, since packets can be multicast without concern as to the recipients. If verification information is not sent to all slave clocks, then the verification step on the slave clocks missing the data will note that the verification information never arrived.

We associate clock identities with the key used for signing and verification. This association must be provided as part of the key distribution algorithm, as we do not want to send the public key along with signed messages for fear of tampering.

2.3.2 Handling Faults

While we trust that timestamps are taken accurately, we do not trust that any particular network device will not willfully modify a timestamp from a prior source. That is, a transparent clock might alter the correction field of a Follow_Up packet to include the switch's update and only that time. The slave at the end would not receive the true path delay. Therefore slaves must verify received timestamps. Transparent clocks should verify received timestamps, though the added overhead may preclude this.

A master clock may try to masquerade as the grandmaster by sending Sync packets. However, by requiring the grandmaster to sign its Sync and Follow_Up packets, no master clock may spoof the grandmaster.

A slave clock may try to claim that a verification failed. However, by requiring all slaves to verify and by urging transparent clocks to verify as able, the lack of an error message on the majority of the system limits the search for the faulting node.

To prevent a replay attack involving a proper signature being retransmitted, we must include a nonce of sorts. As it happens, the PTP standard requires the transmission of an monotonically increasing sequence id with each Sync/Follow_Up packet and a separate counter for the Pdelay messages. This means that a signature could not be replayed as long as the sequence id is used in the computation of the signature, especially as PTP does not permit more than one packet per message type with a given sequence id for the Sync and Pdelay messages.

There is one crucial caveat to this sequence id rule. If a Follow_Up packet of given sequence id is received more than once before the next Sync message, both Follow_Up packets might be used to adjust the clock. This means that implementations must log the spurious arrival.

2.4 Inline Packet Modification

Ideally, we would not place more frames on the Ethernet network in the course of providing integrity of timestamps and authority of the switches. This suggests that inline packet modification for Sync/Follow_Up packets is a viable process.

In order to verify timestamps, we must first identify the smallest portion of data that proves a timestamp update belongs to a given clock for a given sequence id (to avoid replay attacks).

We consider Sync and Follow_Up packets in tandem, as both packets are of equal length and append an original, 10 byte, timestamp (with higher accuracy in the Follow_Up) to the PTP header. The header itself contains the sequence id, domain number, correction field, and message type, among other fields.

We could sign the whole packet with the updated correction time, but the cost of the signature increases with the message length. Therefore, we focus on signing this limited message to reduce computation. The message type is necessary to prevent replay attacks from other signatures, such as those that might be needed for Pdelay messages (discussed in the next section). Otherwise, we must use both Sync and Follow_Up messages to first compute the path delay and then to sign the delay, respectively.

It is not sufficient to simply transmit a Sync message as in the IEEE 1588 standard, as the path delay values sent in a Follow_Up of sequence id i must correspond to the matching Sync to be able to trust the path delay. Since we assume that other nodes can try to spoof the grandmaster, it is entirely possible that a node could delay a grandmaster's Sync message or even create a new Sync message to masquerade as the grandmaster. We must verify that a Sync message has been sent by the grandmaster, and we need to prevent replay attacks by including the sequence id as part of the signature. That is, no node can cause a switch to calculate the path delay for a given Sync message unless that Sync message originated at the grandmaster.

Since dropped packets are possible, a switch could drop a valid Sync message and inject a spoofed one. Then, subsequent switches would calculate peer delays and sign these to a legitimate Follow_Up message with the same sequence id as the dropped Sync. However, these peer delays may correspond to a prior network configuration and may no longer accurately describe the time from grandmaster to slave. For instance, if a node drops out of the network suddenly, the dropped Sync message may have path delays that reflect the new routing, not the old one that was intended.

It is still possible for a node to delay the Sync packet for arbitrary amounts of time. However, the peer delay and residence times for the delayed Sync message still correspond to the delayed message. Additionally, the grandmaster then signs the precise egress time of the Sync message in the Follow_Up message, so it is not possible for this value to be falsified.

Each switch receiving this Sync message must first verify that the signature is valid, passes along the packet, and records the residence time and peer delay for the given sequence id. The verification can be performed in the background, as long as the time is logged.

The grandmaster must also transmit a Follow_Up message with the close estimate of the egress time. This value and the clock id of the grandmaster are signed and concatenated to the back of the packet as the first of the chain of signatures.

Each switch receiving the Follow_Up message then appends its clock id, residence time, and peer delay, updating the original signature with these values.

When received by a slave node, the Sync message is verified and the slave clock is set into a state for receiving Follow_Up messages.

When received by a slave node, the Follow_Up message is verified as follows. First, each portion of the signature is verified based on the known (clock id, public key) pairs. As we use an aggregate signature, we can verify in bulk, often for less cost than verifying each signature independently. With the signatures checked, the node then ensures that the sum of the path delay values equals the value in the correction field, ensuring that no tampering occurred along the path. The last check is not optional, as it provides a sanity check on the timestamp.

It is necessary for the grandmaster to sign the Sync packet to prevent a rogue master clock from spoofing a Sync. For this, since the origin timestamp is not guaranteed to be accurate, we need only sign the tuple consisting of the vlan, domain, clock id of the grandmaster, sequence id, and message type. This signature is concatentated to the end of the Sync packet and should be verified by all network nodes. However, since the values in the Sync packet are not used in the computation of the Follow_Up, we can perform the verification out of the critical path of the PTP program for each node.

For Follow_Up messages, we now have an accurate timestamp, so the grandmaster again signs the accurate time of egress of the Sync, vlan, domain, its clock id, the sequence id, and message type. This signature is concatenated to the end of the Follow_Up as before.

As the Follow_Up packet flows through the network, transparent clocks update a correction field to reflect the peer delay and residence time of the Sync packet. Transparent clocks must provide a signature for their vlan, domain, clock id, sequence id, message type, and the correction field value to ensure that their correction field update is accurate. Signatures and intermediate values (clock id) are concatentated to the end of the Follow_Up for verification by the slave clocks.

This system of concatenating signatures provides a key problem in ensuring that no packet is dropped. If we consider that each signature is separate, we may encounter a situation in which a rogue switch simply overwrites a prior signature with its own, effectively cutting a signature out of the loop and in such a way that no other node could detect the error. We consider how to correct for this now.

2.4.1 Aggregate Signatures

To ensure that no signature can be dropped from a Follow_Up packet, we must employ aggregate signatures. In these signatures, the new signature and a prior signature are combined to create a new signature. This new signature is then combined with later signatures as the Follow_Up packet proceeds through the network such that only a single signature is in the packet at any given time. If a node tries to remove the contribution of a prior node, the verification at the end will fail.

For more detail on the consequences of not using aggregate signatures, we consider the following ideas.

1. Given a network path from a source to destination node with exactly one switch node, if the switch node cannot correctly update its own portion of the correction field, then the timestamp sent from the source to destination via the switch cannot be trusted.

   Consider the switch updates at least the correction field, a field initialized to 0 by the source node. If this single field cannot be trusted to be correct, the destination node cannot trust the resulting timestamp.

2. Given a network path from a source to destination node with an unknown number of switch nodes, and each switch node correctly updates its own portion of the correction field. Even if each switch node signs its update and most prior updates, concatenating the signature onto the packet, the resulting time cannot be trusted.

   Consider the switch overwrites the signature and update of its immediate predecessor to effectively cut the predecessor out of the path. . As the destination node cannot know the number of hops between it and the source node, it cannot discern that a switch's update was omitted.

   Fig. 2.1 illustrates the modifications that a switch can make and how it is indistinguishable from a correctly transmitted packet. In order for the modified packet to be accepted, the malicious switch must provide verification information for itself with a falsified correction field value. The slave receives verification information for the malicious switch and the grandmaster. The correction field portions sum to the correction field value in the packet, so the slave would accept this update as legitimate, even though a portion of the route was omitted.

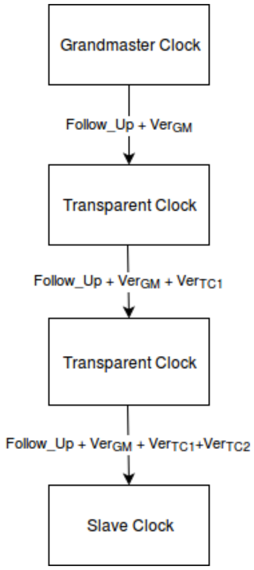
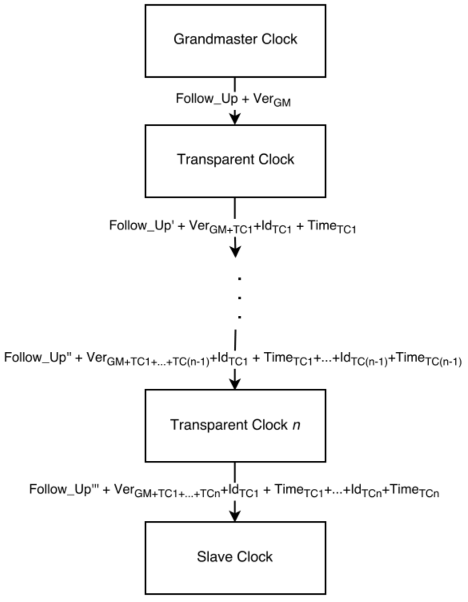

Grandmaster Clock
Follow_Up + Ver$_{GM}$
Transparent Clock
Follow_Up + Ver$_{GM}$ + Ver$_{TC1}$
Transparent Clock
Follow_Up + Ver$_{GM}$ + Ver$_{TC1}$+Ver$_{TC2}$
Slave Clock
Grandmaster Clock
Follow_Up + Ver$_{GM}$
Transparent Clock
Follow_Up + Ver$_{GM}$ + Ver$_{TC1}$
Malicious Transparent Clock
Follow_Up + Ver$_{GM}$ +Ver$_{TC2'}$
Slave Clock

Figure 2.1: Concatenation alone of signatures means that a rogue node can omit prior verification information and make it seem as though there are fewer hops between a grandmaster and slave clock.

With this in mind, it is clear that aggregate signatures of some form must be used. Aggregate signatures are conceptually closely related to multisignatures, where all signers sign the same message. Some aggregate signature algorithms require a signer to verify all prior signatures before signing, while history-free ones simply require the signer to sign its portion and include it in the signature. Additionally, it is possible to sequentially aggregate signatures and thus enforce an order to signing. This is advantageous in S-BGP, where BGP routing information needs to be properly ordered. While it is also helpful to ensure that messages are transmitted in the correct order, PTP has no inherent route verification and can only provide information on a route for a given Announce message, not for all Sync/Follow_Up messages. For that reason, we focus on simply having all hops and the grandmaster sign their portion of the Sync/Follow_Up packet and accept that the hop portions may be out of order without negatively affecting the system.

When using an aggregate signature algorithm, the packet flows of both Sync and Follow_Up packets is as shown in Fig. 2.2. In the figure, Ver is the signature, ID is the clock id of the transparent clock, and T is the partial correction of the transparent clock. These values are concatenated on to the end of the original Follow_Up packet as it flows through the network.



Grandmaster Clock
Follow_Up + Ver_{GM}
Transparent Clock
Follow_Up' + $\text{Ver}_{\text{GM+TC1}}$+$\text{Id}_{\text{TC1}}$ + Time_{TC1}
Follow_Up'' + $\text{Ver}_{\text{GM+TC1+...+TC(n-1)}}$+$\text{Id}_{\text{TC1}}$ + Time_{TC1}+...+$\text{Id}_{\text{TC(n-1)}}$+$\text{Time}_{\text{TC(n-1)}}$
Transparent Clock *n*
Follow_Up''' + $\text{Ver}_{\text{GM+TC1+...+TCn}}$+$\text{Id}_{\text{TC1}}$ + Time_{TC1}+...+Id_{TCn}+Time_{TCn}
Slave Clock
Grandmaster Clock
Sync+Ver_{GM}
Transparent Clock
Sync+ Ver_{GM}
Sync + Ver_{GM}
Transparent Clock *n*
Sync + Ver_{GM}
Slave Clock

Figure 2.2: Flow of Follow_Up and Sync packets using aggregate signatures.

Boneh-Lynn-Shacham is one of the oldest aggregate signature algorithms and originated as a means to produce short signatures. It produces n bits of security for roughly $2n$ bits of signature. The algorithm is based on bilinear elliptic curve pairing and relies upon the difficulty of computational Diffie Hellman and the existence of random oracles.

The algorithm works as follows. A signer i has a public key, v, and a private key s. Using their private key, a signature ω_i is produced for a given message m with hash h. Another signer, j can piggyback off of the first signer by simply multiplying their ω_j with ω_i. In this way, signatures can be chained together into a single 20-byte field, in the case of using SHA-1 as the hash. Verification is more complex.

Other aggregate signature algorithms exist[23] and some do not rely upon the existence of random oracles[24], but BLS has the advantage of being a short, history-free signature of fixed length. That is, we do not require a lengthy computation and take a constant amount of time to sign no matter how deep in the PTP hierarchy we are. Signatures minimally strain the network bandwidth, with each hop adding more raw data than the size of the whole signature.

A key requirement of BLS is that all messages to be signed are unique. This is inherently enforced by our inclusion of clock ids, which are unique, in the message to be signed.

Despite our usage of BLS as the aggregate signature algorithm for this system, any aggregate signature algorithm can be used, though we note several performance-related caveats to using an arbitrary algorithm.

- History-free signatures do not require verification of prior signatures. At the cost of verification time, though, errors in the switches can be detected sooner.
- Signatures that grow in length may exceed the size of a single frame.
- At most 17 signings will occur from grandmaster to slave, so short verification times, a common side effect of longer signing times, may enable the use of histories in the algorithm. However, it is worth considering that these switches need to perform computations other than just PTP timestamping.

2.5 Using a Secondary Channel

We could simply have switches flood a single wireless network with the verification data each time they transmit a FollowUp frame. However, PTP has no notion of how the

Ethernet network is structured, so the slave nodes cannot determine which switches are relevant in verifying the partial sums of timestamp updates.

Without the hierarchy, we then consider creating our own record of a hierarchy. We know that Pdelay messages must be used for accurate path delays. In the prior scheme, we did not deem it necessary to protect the Pdelay messages from alteration, but to produce a true and verifiable hierarchy, we must now sign all Pdelay_Resp and Pdelay_Resp_Follow-_Up packets with our clock id, sequence id for Pdelay, and message type. The clock that sent the original Pdelay_Req will now have a way to verify its immediate peers.

Sync messages are signed by the grandmaster, but the signature and the signed values are forwarded over WiFi to all clients on the network. Additionally, we use Ed25519[17] to sign the packet, since it is much faster.

For the sake of this implementation, forwarding is achieved here by tracking which clients are connected to the PTP network and unicasting the packet to all such clients to avoid performance issues related to multicasting over WiFi[25].

Transparent clocks forward their vlan, domain, clock id, sequence id, correction field update, message type, and the clock id of the peer from which they received a Follow_Up to all clients on the network.

We might be able to trust that our wireless network is secure, but slave clocks have no way to verify if verification data they have received is legitimate. Therefore, we need a signature again. A simple Ed25519 signature suffices since we no longer need to combine signatures. A packet flow can be seen in Fig. 2.3. The signed packets are broadcast to all listening clocks.

However, since we are sending verification information in separate packets, we no longer have the risk of a rogue switch dropped our information. Instead, we have a problem from jamming. Since each 802.11 frame is sent separately, the slave clocks receive them at separate times and must reassemble the Ethernet hierarchy through which they received

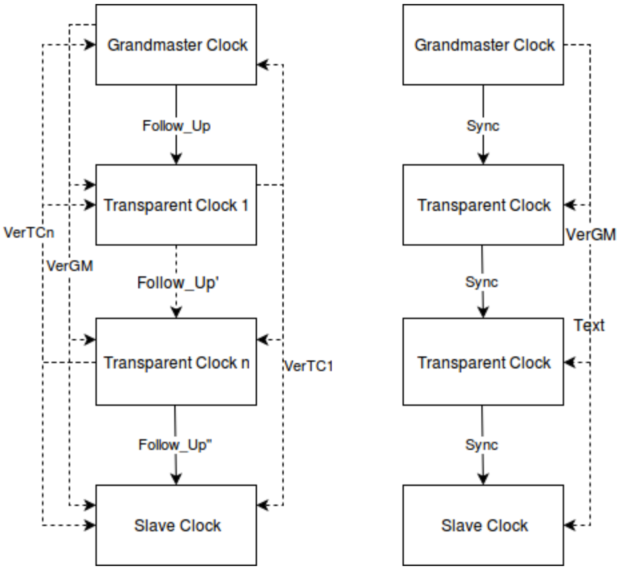


Figure 2.3: Flow of Follow_Up and Sync packets using a mix of WiFi and Ethernet.

a Follow_Up packet. Not only does this now require a slave clock to track its parent transparent clock, but we have no guarantee that a transmitted 802.11 frame may ever reach the slave. We set a timeout by which time a slave assumes that if it has not received enough packet to build its hierarchy, that an error has occurred.

It is not sufficient for the partial sums of correction fields to add up correctly, since it is possible that a rogue switch could have simply altered its correction field to account for another switch that is being jammed. A full hierarchy from grandmaster to slave must be constructed in order for the chain of trust to exist.

2.6 Physical System Design

Our system simulates a single hop from grandmaster to slave by using three Raspberry Pi 3 Model B computers. These ARM-based machines run the lightweight release of Raspbian Jessie and use a rebuild kernel to allow for particular hardware features.

One Pi acts as the grandmaster of the domain, running PTPd[26]. Its system clock is synchronized via NTP, based on LinuxPPS documentation [27] but achieves nanosecond accuracy by the use of a PPS signal, provided by a uBlox EVK-M8T connected via the GPIO pins on the Pi.

Our transparent clock is a second Pi running our kernel module. It also receives clock updates via NTP with the PPS signal.

The third Pi is the slave for the domain and runs PTPd.

To connect the machines, we use an unmanaged switch and unicast traffic from the grandmaster or slave to the transparent clock. The transparent clock, acting as a switch, then multicasts PTP packets as per IEEE 1588, unicasting peer delay responses as necessary. This setup is illustrated in Fig. 2.4. The use of an unmanaged switch introduces some variability into route times, but reduces some of the strain on the transparent clock by ensuring that only PTP traffic is managed by it.

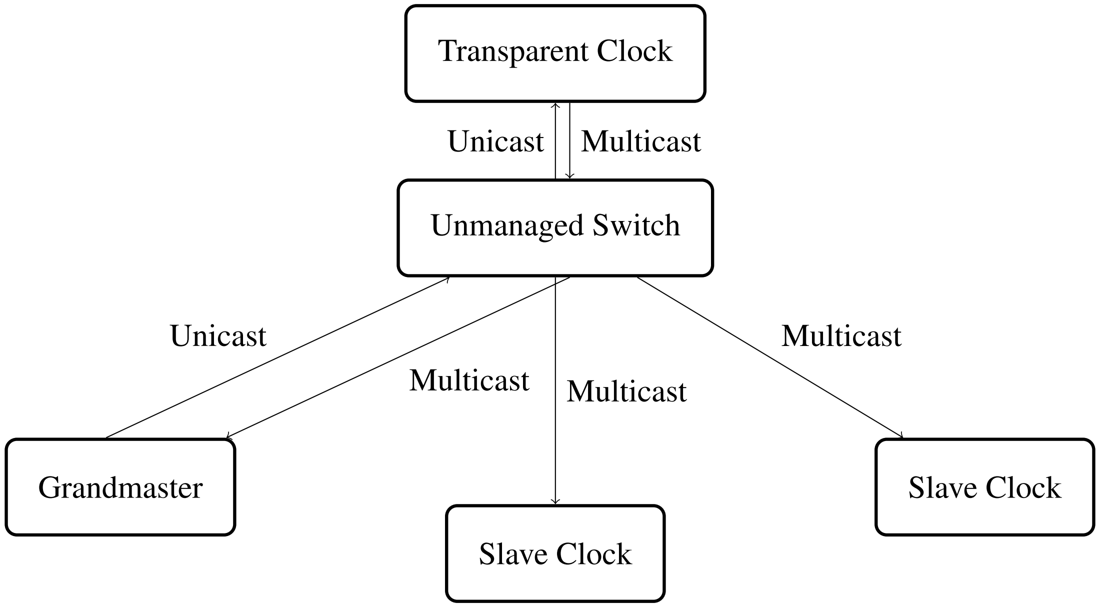


Figure 2.4: System layout.

The unmanaged switch we used was a Netgear JFS524. This device experienced an average round-trip time of 0.332ms with a standard deviation of 0.039ms. This is already an excessive amount of jitter for the network, so we focus on resource impact concerns on the extra bandwidth and computation required to sign.

2.6.1 Alternate Designs

We considered several alternative designs for turning a switch into a transparent clock, particularly since our approach requires a full router to chain multiple switches without disseminating packets beyond the appropriate level of the hierarchy. Among these were the use of OpenVSwitch on the Raspberry Pi switch or simply multiple USB-to-Ethernet adapters connected to the Pi.

OpenVSwitch (OVS) provides a software-based OpenFlow switch with kernel-space routing for known flows[28]. This means that we could treat PTP packets as a known flow for faster routing than in userspace. OVS has been successfully tested on Raspberry Pis with a throughput of 30 Mbps[29].

With OVS, we would have access to the raw skbuff structures, which contain transmit and receipt times[30]. Our approach uses the timestamp returned by sock_recv() and the time immediately after we transmit a packet. Suárez Marín also used OVS for PTP in his implementation of a synchronous SDN network, though his work did not implement the full PTP standard but simply amended the timestamp field on a Sync message [31]. While there are certain benefits to using OVS, like easy access to skbuff without altering an Ethernet driver, we do not alter the flow of packets enough to merit using the full suite.

Without the need for the full capabilities of OVS, we considered a simpler approach, wherein we bridge several Ethernet ports on the Pi, created by attaching USB-to-Ethernet adapters. Then, the external switch or router is not needed. However, Ethernet on the Pi actually uses the onboard USB hub[32], so we cannot leverage the full speeds possible with Ethernet using a traditional NIC. Any non-PTP traffic would be unnecessarily delayed.

This approach would be appropriate for a full PTP transparent clock implementation and is easily achieved with the current software, since outbound Ethernet packets are multicast, but the limited number of USB ports on a Pi made this less than ideal for stress-testing the system.

2.7 Transparent Clock Design

There is no existing software-based transparent clock implementation that is open for modification. Therefore, we must implement our own.

In designing the transparent clock, the kernel-level routing provided by OVS inspired the use of a kernel module, since this allows us to get closer to the hardware and get more accurate timestamps. By using a kernel module, we also have a generic implementation of IEEE 1588 that can be ported to other hardware with less effort.

Since we are using cryptosystems not provided by the kernel, signing and forwarding are performed via netlink sockets to userspace. The MIRACL library provides a BLS implementation, which we modified slightly to support multiple signatures[33]. As such, our userspace program uses MIRACL to perform both BLS and EdDSA signing and forwarding, either over Ethernet or WiFi, respectively. The performance hit required to transmit to userspace makes up for the inaccuracy of using timestamping in userspace. The signing and forwarding operations are less time-sensitive than the forwarding of Sign packets.

To allow for the use of the power profile, we include VLAN tagging as a supported socket option. We would append the appropriate TLV extensions[5, 5.12], except that we do not alter the Announce messages for which TLV extensions apply; therefore, we simply forward all PTP packets as appropriate.

The kernelspace socket library provides access to raw packets without layers of overhead found in userspace. While not as quick as a native Ethernet driver, the code is general enough to be relocated to an Ethernet driver should the need arise.

2.8 Ensuring Clock Accuracy

Given that we are constructing a switch that provides hop-by-hop updates to a timestamp, we must ensure that the switch's time is as accurate as possible. Raspberry Pi lacks an RTC that we could use as a clock source.. Though NTP does not provide a sufficiently precise time resolution just based on an Internet clock source, Mills added kernel support for a nanosecond resolution clock that can use either PPS from an external source like a GPS or a local PLL [34]. PLL resolution is limited by the STC. This leaves PPS as our clock source, with a stratum 1 clock used to disambiguate seconds[27]. This combination provides a resolution of 1μs on the Pi.

With a stable clock, we then ensure that the timestamping functions are as close to the correct instant as possible. The SO_TIMESTAMPING functionality allows for timestamping to be performed when packets are received and transmitted[35]. The information is provided as part of a control message associated with particular packets. Adding this functionality requires rebuilding the kernel, though this was alongside the inclusion of PPS support.

The kernel configuration options set were:

- CONFIG_PPS to provide PPS support
- CONFIG_NETWORK_PHY_TIMESTAMPING to provide timestamps on packet egress
- PTP_1588_CLOCK to test against LinuxPTP[36]

Following modification of the Pi Ethernet driver to explicitly provide network timestamps on packet egress, we confirmed in ethtool that software timestamping, where the time provided is the time the packet enters or leaves the kernel, was enabled.

In our experience, the receive timestamp is reliably received, though the Pi's Ethernet driver did not support TX timestamping originally and still did not provide the requested timestamp after modification. As such, transmit timestamping is performed using the kernel timestamping function getnstimeofday() and its userspace equivalent clock_gettime() using the realtime clock. This clock is adjusted by the NTP offset and provides a low-overhead nanosecond-resolution timestamp. Ideally, based on IEEE 1588, we would have the transmit timestamp recorded as soon as the packet leaves the NIC. Since we do not record the time until after sock_send() returns, we cannot guarantee that the egress time will be perfect. However, since the NIC does not support timestamping upon a packet leaving the physical layer as is, the highest fidelity we could achieve on a Pi is a timestamp immediately after the packet leaves the software, which is prior to our time measurement. As such, we overestimate the transmit time and thus overestimate the residence time for a given packet.

If the processor used supports it, RDTSC (or a similar counter) could also be used as a cycle-based count of the residence time[37]. However, this requires conversion from cycles to nanoseconds or requires all PTP clocks to be syntonized. While the standard generally requires transparent clocks to be syntonized to the grandmaster of a given domain, this may not be feasible in practice.

2.9 Implementation Details

All software runs on Raspbian Jessie, using Linux 4.4.46. The particular build of Raspbian does not provide a GUI and is intended to be lighter.

2.9.1 Transparent Clock

The transparent clock is implemented in a combination of kernel module and userspace program to better leverage the speed of kernel modules with the flexibility of userspace. The bulk of the switch behavior is in the kernel module, with only signing and verification operations being sent via netlink to a userspace program.

The kernel module binds to the Ethernet interface on the Pi and sets socket options for receive timeouts of 1s (to avoid hanging the system), software timestamping, as well as only accepting packets with an Ethernet type of PTP (0x88F7). This allows for other network transmissions to be processed as before the kernel module is inserted. We also enable VLAN tagging, though the kernel strips tags before passing the contents to the kernel-level socket. We also create a netlink socket to which we can multicast information that we need to sign or verify and over which we can receive signatures to forward over Ethernet.

The Raspberry Pi 3 has four cores, so we use multiple threads for the kernel module portion of the transparent clock. One thread only sends Pdelay_Req packets and stores the egress time in a hashtable. Using kernel-level rwlock locks, this egress time is stored in association with the Pdelay sequence id while avoid race conditions.

The second thread is the main thread for the transparent clock. This thread operates in an event loop, processing all received PTP packets based on the version and message type. When a packet is received, there is an associated control message for when the packet first entered the kernel level. This ingress time is stored.

- Since we implement a transparent clock, all PTP version 1 messages are forwarded as though the transparent clock was just a normal switch. Future PTP versions are dropped.

- If we receive a Pdelay_Req, then we issue a Pdelay_Resp and Pdelay_Resp_Follow-_Up with the latter containing the residence time of the Pdelay_Resp[4, 11.4.3.c].

- Pdelay_Resp packets require the use of the stored Pdelay_Req egress time. We use the lock on the hashtable to obtain the appropriate egress time and calculate the mean path delay if no follow up is expected or store the response's ingress time if we expect a follow up. Since we are simulating a transparent clock and have multiple ports, we also store the clock id of the packet to allow us to associate peer delays with the correct peer. We handle multiple responses with the same clock id and sequence id by logging the error, as this indicates an error in the PTP network.

- Pdelay_Resp_Follow_Up packets complete the path delay handshake and are used in combination with the other Pdelay values for this sequence id and clock id to compute the mean path delay.

- Sync packets are forwarded as soon as possible. The residence of the packet is stored for that sequence id in an array that wraps around to bound the number of pending Sync/Follow_Up packets. Every n packets, we start filling from the start of the array again. By storing the residence time, we can quickly access the value when the Follow_Up arrives.

  However, IEEE 1588 allows for Sync and Follow_Up packets to be transmitted out of order. To accommodate this, a Follow_Up arriving before a Sync sets a flag that requires us to store the signature of the grandmaster and create a new Follow_Up packet with that signature when the Sync finally arrives.

  If the Sync packet had the flag for a Follow_Up unset, then we create the Follow_Up and send it. In an unprotected network, this is acceptable as there is no way for the slave clocks to know that the grandmaster only sent the Sync. However, since we now have to work with signed packets, we enforce that the grandmaster uses a two-step clock. The lack of a Follow_Up indicates an error in network configuration and is logged in the transparent clock.

- Follow_Up packets may arrive before their corresponding Sync packet. If this happens, we set a flag to send the Follow_Up once we have a Sync and store the grandmaster signature if there is one.

The creation of Follow_Up packets requires communication with the userspace program to either sign the time information for this clock or to forward the same over WiFi. Since we may already have several signatures, we use a struct to store all of the past signatures as well as the information for our own signature. This is sent to userspace, with the clock's portion signed and then sent back to a handler for the netlink socket. The handler uses the struct to construct the Follow_Up and finally transmit it.

Avoiding the back-and-forth transmission of signatures and identifiers requires that the signing occur in kernel space or that the userspace program transmits the Follow_Up packet itself.

- Delay messages associated with the non-use of transparent clocks are discarded.
- All other PTP packets are forwarded to the rest of the network.

The behavior of the kernel module can be seen in Fig. 2.5.

While the kernel module operates largely in response to received packets, the userspace program operates at the behest of the kernel module. Following initial setup of the cryptography, MIRACL[33] for BLS and libsodium[38] for Ed25519, we create a file descriptor for the netlink socket and listen for requests from kernelspace. Multiple programs can be used to verify and create new signatures, since we multicast from the kernel module. This allows for more modular use of the transmitted data.

2.9.2 Grandmaster Clock

To support signing on the grandmaster clock, we modified PTPd to sign Sync and Follow_Up messages. This required changing several files:

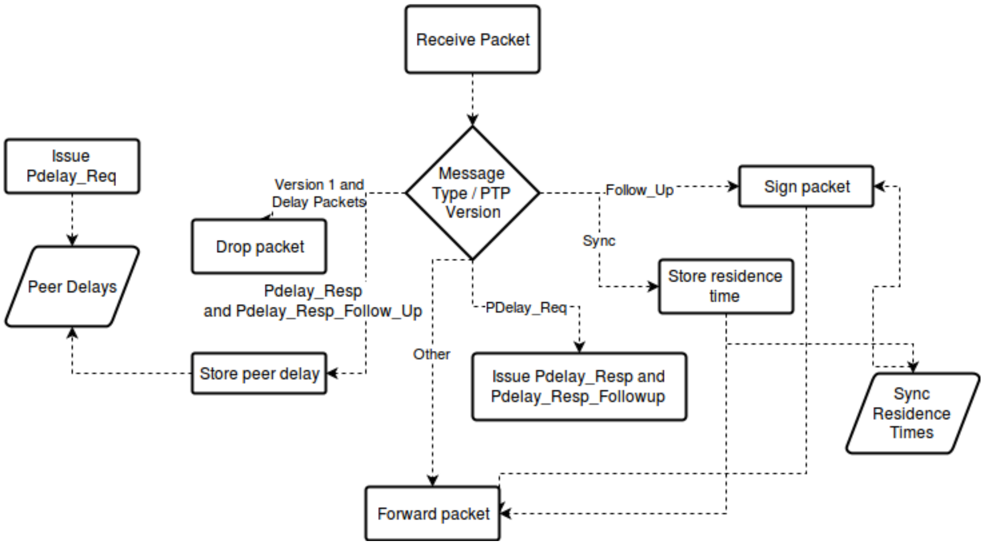


Figure 2.5: Event loops for transparent clock kernel module.

- protocol.c - This contains the main state machine and required changing issueSync() and issueFollowUp() to transmit verification information over WiFi for our second system.

- dep/msg.c - msgPackSync() and msgPackFollowUp() copy data specific to the message type into the PTP pack. This is where we sign messages using either BLS or Ed25519.

- dep/constants_dep.h - PTP packet size is defined here and is changed to 1500 to allow us to use the full frame if needed. PTP4L[36] uses the full frame by default.

- dep/startup.c - We create or load the public and private keys in ptpdStartup(). This requires additional startup time but means we do not need to load keys later when signing.

- datatypes.h - A PtpClock structure is passed around PTPd and effectively stores system global variables. We add the space for our public and private keys here.

libsodium stores its local variables in unsigned char arrays. This means that we can use the signature function in the C PTPd library just by linking the library during compilation.

MIRACL, however, is a C++ library, so we created sign() and verify() methods that could be used externally by C. Additionally, we wrap the PFC (pairing-friendly curve) class in MIRACL so that we can store the related structs in the PtpClock structure.

Since we have the ability to use signing and verification algorithms directly in PTPd, the grandmaster signing functions do not use an external program, like the userspace program for the transparent clock kernel module.

2.9.3 Slave Clock

PTPd can be used for both grandmaster and slave clocks, so we implement the slave clocks' verification measures in the same source as the grandmaster clock code.

- ptp_datatypes.h - MsgSync and MsgFollowUp store the timestamp that extends beyond the PTP header. Since we need to accommodate signatures, clock ids, and timestamp portions, we extend these structs to fit the signature (for Sync and Follow_Up) and the clock ids and timestamps (for Follow_Up).

- dep/msg.c - msgUnpackSync() and msgUnpackFollowUp() unpack the raw buffer into the MsgSync and MsgFollowUp structures. We fill these structures in accordance with how much information is in the buffer.

- protocol.c - To verify messages transmitted over WiFi, we use message passing to send the Follow_Up packet received in handleFollowUp() to a second verification program. Verification takes place in handleSync() once the message is unpacked.

- datatypes.h - Since we pass received Follow_Up packets to our external verification program, we store the file descriptor for message passing in PtpClock.

The external verification program binds to the wireless adapter and stores received verification packets in a vector. When a Follow_Up is sent from PTPd, the program starts a timer for receipt of 17 (16 hops plus the grandmaster signature) packets. The timer is of variable length.

If the timer elapses without enough received packets with timestamps to sum to the correction field value sent from PTPd, then we log that too much time elapsed. If we instead receive enough packets' timestamps to sum to or exceed the correction field value, we verify each signature in turn.

3. AUTHENTICATION IMPACT

The addition of authentication to a system short on time, computational resources, and bandwidth necessitates measuring the additional costs added by the proposed system.

3.1 Bandwidth

Regardless of which scheme is used to protect data, there is always an extra amount of data transmitted. Most of the data needed to uniquely identify a packet is contained as part of the PTP header (802.1Q identifier excluded), though the clock identity provided there is only for the grandmaster.

3.1.1 Inline Packet Modification

To avoid replays on Sync messages, the grandmaster signs using Ed25519. This adds 64 bytes to the Sync message.

Follow_Up messages are signed by the grandmaster and each of the hops on the way to the slave clocks. This signature is done through BLS. In the chosen implementation, using MNT curves, this adds 20 bytes for the signature with another byte used to store least significant bit of the current points's y-coordinate.

While the signature scheme we use requires 20 bytes and an extra bit for the signature, other aggregate signature schemes use longer signatures and may be more efficient for a given system. With the 1454 bytes for the full PTP packet in an Ethernet frame less the 44 bytes for a Sync/Follow_Up message, we can support signature lengths of 994 bytes (if not combining peer delay and residence time) or 1122 bytes (if combining those values) for the 16 hop maximum of our profile. Depending on if we choose to sum the peer delay and residence time or not, we use at most an additional:

$$21 + 16 * (8 + 10) = 309 bytes$$

or

$$21 + 16 * (8 + 8 + 10) = 437 bytes$$

Neither of these is enough to require another frame even with the maximum number of hops. Therefore, the packet priority needs to be set high enough to minimize asymmetry from the larger packet sizes.

3.1.2 Secondary Channel

Verification information over WiFi includes two separate portions for the Sync and the Follow_Up, as well as a modification to the Pdelay messages.

For Pdelay, the additional signature is included as part of the Pdelay_Resp_Follow_Up. The Ed25519 signature requires an extra 64 bytes.

Securing Sync requires the grandmaster to send the verification information over WiFi. The values required are shown in Table 3.1. The total of 16 bytes of data in addition to the Ed2519 signature leads to a transmission of 80 bytes, which fits into the data portion of a single frame.

Value	Size (octets)
VLAN Identifier	2
domainNumber	1
sourcePortIdentity	10
sequenceId	2
messageType	1
Total	16

Table 3.1: Forwarded grandmaster verification information.

Securing Follow_Up requires the grandmaster to provide the values in Table 3.1 in additional to 10 octets for the accurate egress time of the Sync packet, for a total of 90 octets in the final signed frame.

The transparent clocks do not provide the 10 octet time, which is a concatenation of a second and nanosecond field, but instead provide the scaled nanosecond time in 64 bits. The full set of values is as shown in Table 3.2. Even with the Ed25519 signature included, we still have under a full frame's worth of data.

Value	Size (octets)
VLAN Identifier	2
domainNumber	1
clockIdentity	10
sequenceId	2
messageType	1
peer clockIdentity	10
residence time + peer delay time	8 (or 16 if sent separately)
Total	34 (or 42 if times sent separately)

Table 3.2: Forwarded transparent clock packet verification information.

3.2 Computation

The number of bytes signed will not match the number of extra bytes transmitted, as we rely on some information already required to be in the PTP header. As such, the actual length of computation matters.

Unless modifications to the standard are made to assure that there is only a single domain or VLAN for a given PTP network, we must include these variables as part of the bytes to be signed.

To test the additional computation, we first benchmarked the time to reach the actual output of Sync and Follow_Up packets from the time of receipt (for the transparent clock) or the start of a new event loop (for the grandmaster). Since our PTP network is intended only for experimentation and already exhibits jitter than would be unacceptable in real-world applications, we want to measure the relative increases in computation time, rather than just how many cycles are needed to sign or verify.

For benchmark the kernel module, we generated corresponding Sync and Follow_Up packets and unicast them to the kernel module. To avoid jitter from overloading the system, we waited 1s between the Sync and Follow_Up pair, emulating the behavior of a real PTP network.

For the grandmaster, we did not have the clear start point that the kernel module has by virtue of receving a packet. Starting our measurement at the top of a new iteration of the main event loop adds more computation but provides a more realistic idea of how much our modifications impact the system.

For each benchmark, 500 samples were obtained.

3.2.1 BLS Verification

Any BLS verification operations were performed in userspace as we used MIRACL. Therefore, to test the relative impact of signing and verifying preformatted messages, we signed and verified up to 17 signatures with random private keys. The time to sign can be seen in Fig. 3.1 and to verify in Fig. 3.2.

Signing of just the grandmaster's data alone takes 4.14ms. Signing transparent clock data involves taking the signature of the grandmaster combined with the previous hops, so the additional computation means that signatures average 4.85ms.

Verification scales linearly with the number of signatures.

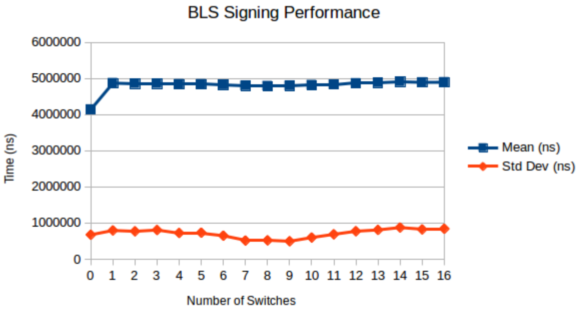


Figure 3.1: BLS signing performed on verification data with a grandmaster and up to 16 transparent clocks.

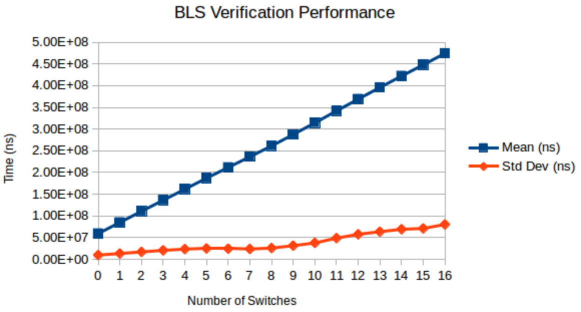


Figure 3.2: BLS verification performed on verification data with a grandmaster and up to 16 transparent clocks.

3.2.2 Inline Packet Modification

Using the timestamping points from our benchmarks, we found that, for a transparent clock, the time to receive a Follow_Up, forward the packet to userspace, and then sign the relevant portions averaged 17 times longer than simply processing the packet without any signing, shown in Table 3.3. This time does not include the time necessary to verify signatures, as we do not need verification to occur prior to sending a Follow_Up in a system geared solely to detection of faults, not prevention of faults.

The grandmaster signs Sync messages with Ed25519, with times shown in Table 3.4.

	Mean time (ms)	Std dev (ms)
Default mode	0.3671	0.1184
Signing mode	6.3897	2.0970

Table 3.3: Signing time for transparent clocks.

The sign is performed before the packet is transmitted, leading to nearly double the time before transmission, though the standard deviation is small enough, given that this is an experimental system, to suggest that on a properly implemented real-world system, inline signatures are viable.

	Mean time (ms)	Std dev (ms)
Default mode	1.2785	0.5302
Signing Sync	2.5379	0.7793

Table 3.4: Signing time for grandmaster clocks.

The grandmaster's Follow_Up signature is performed in BLS and took an average of 4.16ms more than simply transmitting the Follow_Up as-is. There is a significant amount of computation performed by PTPd prior to sending the Follow_Up, so the increase in time is in line with our benchmark BLS signing time.

As previously discussed, BLS verification scales linearly with the number of hops. Since the Follow_Up packet contains the entirety of the data needed to verify, with the exception of the stored public keys, verification times follow Fig. 3.2 closely.

3.2.3 Secondary Channel

By using WiFi and a separate process to verify signed data, we offload the bulk of the computation to outside of the critical timing path. All of the signatures use Ed25519, so th signing and verification are quick, though we did not use batch verification for this.

4. SUMMARY AND CONCLUSIONS

4.1 Challenges

The lack of a real-time clock and the use of a USB hub to provide Ethernet on the Raspberry Pi both incur increased inaccuracy in our measurements. While we use PPS from a GPS to improve the time resolution, moving the whole of switch behavior to the Pi from the combination of Pi and unmanaged switch increases the load on the Pi. This increased load may limit the use of a Pi in a full PTP system. This is part of why we consider the system to be a simulation.

The use of digital signatures increases the time to sign. Message authentication codes (MACs) provide quicker computation but require a more complicated key negotiation strategy with each switch in the path from grandmaster to slave to have a unique key for each slave under it. It remains an open question if the increased storage costs could make up for the computation time.

Our system is not sufficient for real-world use as it is. Reducing the impact of the implementation on the results of the protocol is an ongoing work.

4.2 Further Study

The verification methods here are tested only with a single hop. While we test the time to verify a full set of 16 hops, a full system test would reveal further difficulties in the extension of our system to a full PTP network.

Additionally, we test our system on the ”happy path” of having all clocks in the system behaving correctly. While we provide a theoretical analysis of potential weaknesses and how we address them, this does not preclude the existence of a severe bug that could only be found by testing the implementation.

We also do not address a resilience to jamming. While jamming on the inline network would simply bring down the whole PTP system, jamming on wireless prevents any verification. Though this motivates our inclusion of a user-defined timeout on verifying a Sync/FollowUp pair, this is essentially a fallback to a fully unprotected PTP system, which is precisely what we try to prevent.

A given network hierarchy is unknown to PTP. By design, peer delays are associated with the port on the switch calculating the delay to its peers, rather than associating with the connected peer. With peer delay calculations being performed every second, it seems reasonable to model the network's evolution and the response to network errors to determine if there is a high probability of using outdated peer delay values. For instance, a rapidly evolving topology would run the risk of using incorrect peer delay values more frequently than a stable topology.

4.3 Conclusion

With PTP seeing increased use in power systems to provide high-resolution clock synchronization, the ability to ensure the integrity of transmitted timestamps and to verify their source is critical. Past efforts to protect PTP focused on end-to-end security, which does not suffice for systems using transparent clocks. We considered two ways to protect against modified timestamps and showed the relative advantages and disadvantages of each. We intend our efforts to be a first step towards securing networks with transparent clocks and therefore focused on a simulated transparent clock. Though neither of our systems can protect against jamming, we have shown that it is possible to provide integrity and authentication in PTP systems intended for power systems.

REFERENCES

[1] D. P. Shepard, T. E. Humphreys, and A. A. Fansler, “Evaluation of the vulnerability of phasor measurement units to GPS spoofing attacks,” *International Journal of Critical Infrastructure Protection*, vol. 5, no. 3-4, pp. 146–153, Dec. 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1874548212000480

[2] H. Liu, J. Liu, T. Bi, J. Li, W. Yang, and D. Zhang, “Performance analysis of time synchronization precision of PTP in smart substations,” in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Oct. 2015, pp. 37–42. [Online]. Available: http://ieeexplore.ieee.org/document/7324677/

[3] Bonian Shi, Daonong Zhang, and Jiong Hu, “Preliminary investigation in Wide Area Protection implementation using IEEE 1588 precision time protocol,” in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Oct. 2015, pp. 43–47. [Online]. Available: http://ieeexplore.ieee.org/document/7324678/

[4] Institute of Electrical and Electronics Engineers and IEEE-SA Standards Board, *1588-2008 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.* Institute of Electrical and Electronics Engineers, 2008, oCLC: 958709955. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=4579757

[5] ——, *C37.238-2011 IEEE standard profile for use of IEEE 1588 precision time protocol in power system applications.* New York: Institute of Electrical and Electronics Engineers, 2011, oCLC: 757338195. [Online]. Available:

http://ieeexplore.ieee.org/servlet/opac?punumber=5963697

[6] Longhua Guo, Jun Wu, Jingwei Li, Jianhua Li, and W. J. Miller, “A lightweight secure time synchronization mechanism for ISO/IEC/IEEE 21451 sensor networks,” in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Oct. 2015, pp. 13–18. [Online]. Available: http://ieeexplore.ieee.org/document/7324673/

[7] D. L. Mills, “Internet time synchronization: the network time protocol,” *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, Oct. 1991.

[8] T. Mizrahi and Y. Moses, “Using ReversePTP to distribute time in Software Defined Networks,” in *2014 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Sep. 2014, pp. 112–117. [Online]. Available: http://ieeexplore.ieee.org/document/6948702/

[9] N. Moreira, J. Lazaro, J. Jimenez, M. Idirin, and A. Astarloa, “Security mechanisms to protect IEEE 1588 synchronization: State of the art and trends,” in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Oct. 2015, pp. 115–120. [Online]. Available: http://ieeexplore.ieee.org/document/7324694/

[10] T. Mizrahi, “Time synchronization security using IPsec and MACsec,” in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Sep. 2011, pp. 38–43. [Online]. Available: http://ieeexplore.ieee.org/document/6070153/

[11] ——, “RFC 7384 security Requirements of Time Protocols in Packet Switched Networks.” [Online]. Available: https://tools.ietf.org/html/rfc7384

[12] A. Cahn, J. Hoyos, M. Hulse, and E. Keller, “Software-defined energy communication networks: From substation automation to future smart grids,” in *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, Oct. 2013, pp. 558–563. [Online]. Available: http://ieeexplore.ieee.org/document/6688017/

[13] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, p. 147, Dec. 2002. [Online]. Available: http://portal.acm.org/citation.cfm?doid=844128.844143

[14] S. Siu, W.-H. Tseng, C.-S. Liao, H.-f. Hu, S.-Y. Lin, and Y.-L. Lai, “Analysis of the dynamic time error in a chain of cascading telecom boundary clocks,” in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Oct. 2015, pp. 105–110. [Online]. Available: http://ieeexplore.ieee.org/document/7324692/

[15] M. Dalmas, H. Rachadel, G. Silvano, and C. Dutra, “Improving PTP robustness to the byzantine failure,” in *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Oct. 2015, pp. 111–114. [Online]. Available: http://ieeexplore.ieee.org/document/7324693/

[16] E. Itkin and A. Wool, “A security analysis and revised security extension for the precision time protocol,” in *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, Sep. 2016, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7579501/

[17] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, Sep. 2012. [Online]. Available: http://link.springer.com/10.1007/s13389-012-0027-1

[18] O. Eikemeier, M. Fischlin, J.-F. Gtzmann, A. Lehmann, D. Schrder, P. Schrder, and D. Wagner, "History-Free Aggregate Message Authentication Codes," in *Security and Cryptography for Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 6280, pp. 309–328, dOI: 10.1007/978-3-642-15317-4_20. [Online]. Available: http://link.springer.com/10.1007/978-3-642-15317-4_20

[19] M. Fischlin, A. Lehmann, and D. Schrder, "History-Free Sequential Aggregate Signatures," in *Security and Cryptography for Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7485, pp. 113–130, dOI: 10.1007/978-3-642-32928-9_7. [Online]. Available: http://link.springer.com/10.1007/978-3-642-32928-9_7

[20] L. Malina, J. Hajny, and V. Zeman, "Trade-off between signature aggregation and batch verification," in *2013 36th International Conference on Telecommunications and Signal Processing (TSP)*, Jul. 2013, pp. 57–61.

[21] A. Chakraborti, A. Chattopadhyay, M. Hassan, and M. Nandi, "TriviA: A Fast and Secure Authenticated Encryption Scheme," in *Cryptographic Hardware and Embedded Systems – CHES 2015*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, vol. 9293, pp. 330–353, dOI: 10.1007/978-3-662-48324-4_17. [Online]. Available: http://link.springer.com/10.1007/978-3-662-48324-4_17

[22] D. J. Bernstein, "The Poly1305-AES Message-Authentication Code," in *Fast Software Encryption*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3557, pp. 32–49, dOI: 10.1007/11502760_3. [Online]. Available: http://link.springer.com/10.1007/11502760_3

[23] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, “Sequential Aggregate Signatures, Multisignatures, and Verifiably Encrypted Signatures Without Random Oracles,” *Journal of Cryptology*, vol. 26, no. 2, pp. 340–373, Apr. 2013. [Online]. Available: http://link.springer.com/10.1007/s00145-012-9126-5

[24] R. Canetti, O. Goldreich, and S. Halevi, “The random oracle methodology, revisited,” *Journal of the ACM*, vol. 51, no. 4, pp. 557–594, Jul. 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1008731.1008734

[25] A. Phonphoem and S. Li-On, “Performance Analysis and Comparison Between Multicast and Unicast over Infrastructure Wireless LAN,” in *Technologies for Advanced Heterogeneous Networks II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 4311, pp. 75–89, dOI: 10.1007/11930181_6. [Online]. Available: http://link.springer.com/10.1007/11930181_6

[26] “ptpd: PTPd official source,” Jul. 2017. [Online]. Available: https://github.com/ptpd/ptpd

[27] “LinuxPPS.” [Online]. Available: http://linuxpps.org/mediawiki/index.php/Main_Page

[28] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, “The Design and Implementation of Open vSwitch,” in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[29] K. Ohira, “Performance evaluation of an OpenFlow-based mirroring switch on a laptop/raspberry Pi,” in *Proceedings of The Ninth International Conference on*

Future Internet Technologies (CFI'14). ACM Press, 2014, pp. 1–2. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2619287.2619307

[30] "linux/include/linux/skbuff.h - Elixir - Free Electrons." [Online]. Available: http://elixir.free-electrons.com/linux/latest/source/include/linux/skbuff.h

[31] R. Suárez Marín, "Design of a control and management system for frequency-and time-synchronous SDN networks," Master's thesis, Universitat Politécnica de Catalunya, 2016. [Online]. Available: https://upcommons.upc.edu/handle/2117/88748

[32] "USB - Raspberry Pi Documentation." [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/usb/README.md

[33] "MIRACL Cryptographic SDK," Jul. 2017. [Online]. Available: https://github.com/miracl/MIRACL

[34] D. Mills, "Generic Nanosecond Kernel Timekeeping Support." [Online]. Available: http://www.slac.stanford.edu/comp/unix/package/rtems/src/ssrlApps/ntpNanoclock/index.htm

[35] "Linux timestamping documentation." [Online]. Available: https://www.kernel.org/doc/Documentation/networking/timestamping.txt

[36] R. Cochran, "The Linux PTP Project." [Online]. Available: http://linuxptp.sourceforge.net/

[37] G. Paoloni, "How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures," Sep. 2010. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

[38] F. Denis, “libsodium: A modern and easy-to-use crypto library,” Jul. 2017. [Online]. Available: https://github.com/jedisct1/libsodium