An Analysis of Internal Attacks on PTP-based Time Synchronization Networks

Waleed Alghamdi

School of Computer Science

College of Science Engineering

National University of Ireland, Galway

Research Supervisor: Dr. Michael Schukat

Declaration

I declare that the research described in this thesis is original work, which I undertook at the National University of Ireland, Galway, during the years 2015 - 2021. This work has not previously been presented for an award at this or any other university.

Signed: Waleed Alghamdi
Date:

Acknowledgements

This thesis would have been impossible to achieve without the support I found from people who contributed to directing me to the right path and provide me advice and guidance, or people who were a source of motivation and inspiration to me, and they are as follows:

First, I would like to thank my supervisor, Dr. Michael Schukat, for agreeing to supervise this research and giving me all the facilities before my arrival to Galway until I settled in this beautiful city. The support that I found from him and his motivation for me had a great impact on what I have achieved now.

I would also like to thank all the colleagues in Room 308 for making the Ph.D. more enjoyable process. We will fondly remember all our entertaining discussions while we stalled our research.

I also do not forget Dr. Mohannad Alahmadi and Peter O'kane, who have been credited with overcoming the technical obstacles that I faced while researching. I also value the great services Peter provided me during the COVID-19 pandemic, especially during the closure of the university or while I was in Saudi Arabia.

My family was a source of inspiration and motivation, starting with my parents, who supported me throughout my educational journey from school to completing this degree, to my wife Maha, who was credited with keeping me fully dedicated to studying and helping me deal with the stress of researching. My daughters, Alaa, Elaf, and Leen, were putting a smile on my face when I come home that make me forgot all my tiredness. My youngest child and my first son, who was born during my P.h.D, was the most important and beautiful event in my life despite the academic delay caused by his born.

Finally, I would like to thank my brothers, sisters, relatives, and all my friends, as well as all those who supported me even with a word or even wished me well.

Abstract

The IEEE 1588 precision time protocol (PTP) is very important for many industrial sectors and applications that require time synchronization accuracy between computers down to microsecond and even nanosecond levels. Nevertheless, PTP and its underlying network infrastructure are vulnerable to cyber-attacks, which can stealthily reduce the time synchronization accuracy to unacceptable and even damage-causing levels for individual clocks or an entire network, leading to financial loss or even physical destruction. Of particular concern are advanced persistent threats (APT), where an actor infiltrates a network and operates stealthily and over extended periods of time before being discovered. Existing security protocol extensions only partially address this problem. This thesis provides a comprehensive analysis of strategies for advanced persistent threats to PTP infrastructure, possible attacker locations, and the impact on the clock and network synchronization in the presence of security protocol extensions, infrastructure redundancy, and protocol redundancy. It distinguishes between attack strategies and attacker types as described in RFC7384 but further distinguishes between the spoofing and time source attack, the simple internal attack, and the advanced internal attack. Our analysis shows that a sophisticated attacker has a range of methodologies to compromise a PTP network. Moreover, all PTP infrastructure components can host an attacker, making the comprehensive protection of a PTP network against malware infiltration, as for example exercised by Stuxnet, a very difficult task. Some experiments were conducted to demonstrate the impact of PTP attacks, using a fully programable and customizable man in the middle device, thereby considering the two most popular PTP slave daemons PTPd and PTP4l. In doing so, it determines suitable attack patterns and parameters to compromise the time synchronization covertly. This thesis also contributes to the detection of PTP attacks and the attacker location using a trusted supervisor node (TSN). This node collects and analyses delay and offset outputs of monitored slaves as well as timestamps sent by Sync messages, allowing it to detect abnormal patterns in the data provided. Depending on the attack scope, the TSN uses two different algorithms to detect all PTP attacks. This proposal is in line with the prong D as specified in IEEE 1588-2019.

Table of Contents

Table of Figures

List of Tables

Chapter 1

Introduction

Time synchronization has become extremely important in the networked world of today. Although it has always been necessary for industry and telecommunications, it is now also required for many other applications [10]. The techniques used for time synchronization are the basis of time-sensitive applications, particularly distributed network applications [11]. These techniques address the issue of a host adhering to a time-scale reference, which can be utilized by highly precise local clocks or a remote source. The techniques of time synchronization provide details for the sequence and order of communications between a host and reference and also the core data they exchange. A time protocol is developed from the implementation of a synchronization technique, along with a combination of the message structure and any additional operations and methods required. Time protocols are usually designed for specific types of networks [12]. The Network Time Protocol (NTP), for example, is designed for large, dynamic and variable latency Packet Switched Networks (PSNs) by effectively using sophisticated statistical techniques which minimize errors inherent to such networks [13]. Moreover, NTP meets the needs of distributed applications that require accuracies of as little as 1 millisecond over wide area networks. On the other hand, the Precision Time Protocol (PTP) [5] is used in an infrastructural network that is well-controlled PSNs and that used specialized hardware, thereby providing a high level of accuracy [14].

Applications for measurement and control, operational systems used by manufacturing, utility and telecommunication systems, and many financial markets and leading exchanges (i.e., IMC, Eurex, and NYSE) require synchronization levels beyond what NTP can provide, i.e., they need devices that are synchronized within a few microseconds of each other [15], [16]. This degree of synchronization requires the use of GPS receivers or atomic clocks. However, these approaches are often not feasible (i.e., GPS does not work indoors) or very expensive. To meet these demands, IEEE published the IEEE 1588 standard, otherwise known as PTP, in 2002 to provide alternative synchronization protocols for time synchronizing numerous interconnected computing devices with microseconds accuracy [12]. In 2008, a second more robust version of PTP was released [17], [14]. In 2020, IEEE released the latest version of the IEEE 1588 standard (PTP v2.1). It downwards compatible and includes among various enhancements a revised section on security recommendations called Annex P [18].

1.1 Research Motivation

Although PTP is necessary for many time-sensitive applications, its network infrastructure is vulnerable to cyberattacks. PTP can be infected by malware or manipulated firmware that perform PTP attacks over extended periods of time in what is known as Advanced Persistent Threats (APTs). APTs on critical infrastructure firstly became public when the Stuxnet attack on the Iranian Natanz Uranium enrichment facility was uncovered in 2010 [18], [19]. Stuxnet operated silently for many months before being detected. It used a very sophisticated strategy to manipulate PLC (programmable logical controller) and SCADA (supervisory control and data acquisition) system firmware before gradually changing control settings with the aim to compromise and eventually destroy Uranium centrifuges [20]. It is estimated that Stuxnet disabled about one-third of the centrifuges in Natanz [18].

There have been many important lessons learnt from this attack, which led eventually to a better awareness of critical infrastructure vulnerabilities and to a strengthening of their attack resilience. However, these do not provide necessary protection against the two weakest links within the cyber framework, namely the Human factor (e.g., an employee brought on a USB stick Stuxnet into the isolated Natanz facility) and zero-day exploits (e.g., Stuxnet used a total of 5 unknown exploits). As such, critical infrastructure will always be at risk [18], [20], [21].

Originally security was not considered for PTP. As a result, over the last decade, various security protocol extensions or infrastructure enhancements have been suggested. This includes the experimental security extension Annex K (introduced with the IEEE 1588 version 2) in 2008, which has been proven to be insufficient [14], [22]. PTP version 2.1 (IEEE 1588-2019), released in 2020, include a new security extension called Annex P. Annex P consist of 4 prongs as follows [7]:

- Prong A (Integrated Security Mechanism) allocates authentication type-length-value (TLV) to provide protection to PTP messages using a symmetric key.[1]

- Prong B (PTP External Transport Security Mechanisms) describes the existing external security extensions, i.e., MACsec and IPsec that can be used to protect the PTP messages.

- Prong C (Architecture Guidance) describes various redundancy approaches, i.e., redundant time system, redundant grandmaster (i.e., the most accurate clock in a PTP network), and redundant paths.

- Prong D (Monitoring and Management Guidance) describes a monitoring system to observe the PTP slave behaviour.

However, it has been shown that state-of-the-art layer-2 and layer-3 cryptographic security protocols (i.e., MACSec and IPSec) can only prevent a subset of potential attacks [23], limiting the benefits of Prong A and Prong B, while infrastructure enhancements (i.e., multiple paths [24],[25], redundant grandmaster [26], and protocol redundancy [27]) as suggested in Prong C do not provide a gold-plated way to prevent an attacker from exploiting PTP vulnerabilities either [28]. Therefore, this thesis provides a new detection model against PTP attacks using a central management node called Trusted Supervisor Node (TSN). The underlying idea is centred around analyzing data

[1] To enhance PTP security, the TLV is used to append a cryptographic integrity check value (ICV) to PTP messages.

collected from all slave clocks in a network during the synchronization process and then detect for anomalies caused by an attack.

1.2 Problem Statement and Research Questions

While the PTP infrastructure is still vulnerable to cyber-attacks, particularly internal attacks, there is an urgent need to find a way to prevent or detect these attacks and subsequently increasing protocol security. Based on the aforementioned PTP security issue, several research questions have been raised, and they need to be addressed as follows:

1. What are the attack strategies that threaten PTP? Moreover, some sub-questions are proposed as follows:
   - What are the existing security extensions?
   - To what extent can these security extensions deter PTP attacks?
2. What is the effect and impact of such attacks on slave clock synchronization, and under what circumstances do PTP slave clients recognize attacks?
3. Are there better ways to detect/prevent internal PTP attacks?

1.3 Contributions

The main contributions of this research are as follows:

Research findings that are related to the first research question are published in the journal article “Precision time protocol attack strategies and their resistance to existing security extensions”, as well as in the conference paper "Advanced methodologies to deter internal attacks in PTP time synchronization networks".

The findings of the second research question are published in the journal article “Cyber Attacks on Precision Time Protocol Networks—A Case Study”, as well as in the conference paper titled "Slave Clock Responses to Precision Time Protocol Attacks: A Case Study" and the conference paper titled "Practical Implementation of APTs on PTP Time Synchronisation Networks".

Finally, the findings of the last research question are submitted to a journal (which is currently under review in the journal of Engineering Science and Technology, Elsevier) with potential title: “A Security Enhancement of the Precision Time Protocol using a Trusted Supervisor Node” and the conference paper titled "A Detection Model Against Precision Time Protocol Attacks".

1.3.1 Peer-reviewed Journal Publications

1. W. Alghamdi and M. Schukat, "Precision time protocol attack strategies and their resistance to existing security extensions," *Cybersecurity,* vol. 4, p. 12, 2021/04/01; impact factor 1.959.

2. W. Alghamdi and M. Schukat, "Cyber Attacks on Precision Time Protocol Networks—A Case Study," *Electronics,* vol. 9, p. 1398, 2020/08/28; impact factor 2.397.

3. W. Alghamdi and M. Schukat, "A Security Enhancement of the Precision Time Protocol using a Trusted Supervisor Node," currently under review at the journal of *Engineering Science and Technology, Elsevier;* impact factor 4.36.

1.3.2 Peer-reviewed Conference Publications

1. W. Alghamdi and M. Schukat, "Advanced methodologies to deter internal attacks in PTP time synchronization networks," in *2017 28th Irish Signals and Systems Conference (ISSC)*, 2017, pp. 1-6.

2. W. Alghamdi and M. Schukat, "A Detection Model Against Precision Time Protocol Attacks," *2020 3rd International Conference on*

Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 2020, pp. 1-3.

3. W. Alghamdi and M. Schukat, "Slave Clock Responses to Precision Time Protocol Attacks: A Case Study," *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, Dublin, Ireland, 2020, pp. 1-4.

4. W. Alghamdi and M. Schukat, "Practical Implementation of APTs on PTP Time Synchronisation Networks," *2020 31st Irish Signals and Systems Conference (ISSC)*, Letterkenny, Ireland, 2020, pp. 1-5.

1.3.3 Conference Presentations

1. W. Alghamdi, “Practical Implementation of cybersecurity attacks on PTP Networks”: International Timing and Sync Conference was held on 2 – 5 November 2020 at Dusseldorf, Germany.

1.3.4 Poster Presentations

1. W. Alghamdi, “Advanced methodologies to deter internal attacks in PTP time synchronization networks”: NUIGUL research day was held on 29 April 2016 at Limerick University, Ireland.

2. W. Alghamdi, “Advanced methodologies to deter internal attacks in PTP time synchronization networks”: NUIGUL research day was held on 19 April 2017 at NUIG, Galway, Ireland.

1.4 Thesis Overview

The current chapter (Chapter 1) provides a brief introduction to the importance of time synchronization and PTP in particular. It also presents the research motivation as well as identifying research questions. At the end of the chapter, lists of contributions made are provided.

Chapter 2 describes in detail the IEEE 1588 Precision Time Protocol, its entities, and synchronization mechanisms. It also provides an analysis of the time correction algorithms implemented in two popular PTP daemons (i.e., PTP4l and PTPd) as well as slave clock adjustment.

Chapter 3 (1) analyzes and classifies all possible PTP attacks, thereby dividing internal attacks into two types, namely simple and advanced internal attacks, as well as dividing spoofing attacks into master and slave spoofing attacks, (2) outlines possible implementations in detail, (3) demonstrates the vulnerabilities of existing security measures to prevent internal attacks.

Chapter 4 presents (1) the prototype of a PTP hardware "sandbox" testbed that is able to perform various man in the middle (MitM) attacks, namely delay, time source, and packet modification attacks, (2) determines suitable attack patterns and parameters, and (3) analyzes the effect of such attacks on slave clock synchronization, operation, and behaviour using different PTP daemons and slave types.

In Chapter 5, we (1) propose a central unit or trusted supervisor node (TSN) that collects the synchronization outputs from all monitored slave clocks and rearranges or groups the collected data to common synchronization cycles, (2) that implements two attack detection mechanisms depending on the attack scope (i.e., a subset of slave clocks or all slave clocks), (3) thereby specifying the attack strategy type, (4) and determining the possible attacker location within the network using the lowest common ancestor method.

In the last chapter (Chapter 6), we summarize all our findings as well as proposing suggestions for future work.

Chapter 2

The IEEE 1588 Precision Time Protocol (PTP)

Since there are several applications that require better time synchronization accuracy than network time protocol (NTP) provides, this chapter gives a brief introduction to the precision time protocol (PTP) and how it works. This chapter is structured as follows: Section 2.1 explains the importance of PTP while addressing its origin and developments. Section 2.2 will list the PTP entities and their function, followed by a description of how the master-slave hierarchy is established in Section 2.3. Section 2.4 will explain the grandmaster election mechanism. Section 2.5 will show the process of clock synchronization. After that, the most famous PTP daemons will be discussed in Section 2.6 and how to update the slave clocks in each daemon in Section 2.7. Finally, Section 2.8 will give an overview of the importance of PTP security.

2.1 Introduction

There was a growing demand in the early 2000s for an integrated timing solution that was not only cost-effective but also more accurate and precise. In 2002, a solution was proposed to work with any type of network, i.e., industrial communication, telecom, sensor networks, high-frequency trading, and motion control, and it became a standard known as IEEE 1588, and the protocol became known as Precision Time Protocol (PTP) [1]. IEEE 1588 has been revised twice to provide more accuracy and security. The first revision was in 2008 and introduces a new PTP-aware device called transparent clock that improves the provided accuracy by better accounting for PTP packet transmission delays [29], [30]. Also, an experimental security extension was suggested to prevent PTP attacks. However, it has a number of flaws and is deprecated (as further described below). In 2020, the second revision was released. It introduces multipronged security that combines a set of security mechanisms and configuration options listed in Annex P to provide PTP security [7]. Since all the analyses and experiments used in this thesis were conducted using IEEE 1588-2008 devices, this section will focus more on the definitions and explanations contained in the document of IEEE 1588-2008 while highlighting the most prominent differences with the IEEE 1588-2019.

2.2 Entities

IEEE 1588 defines different entities that constitute a PTP network and are essential to its performance. The possible entities in the PTP network are as follows [1]:

2.2.1 PTP Domains

A domain is a group of PTP nodes that communicate with each other on a link. One PTP network can contain different PTP domains, but they are considered independent due to the scope of PTP message communication, clock data sets (i.e, clock characteristics), state, operations, and timescale may be different to each other [1].

The frame of the PTP message provides information on the domain number (*domainNumber*). Domain numbers range between 0 and 255. The default domain number is 0, while the domain numbers 1 to 3 are alternate domains. The range from 4 to 127 are user-defined, while 128 to 255 are reserved [1].

In IEEE-2019, a domain is specified by two parts: *domainNumber* and *sdoId* (standardization development organization identity). The *sdoId* of a domain consists of a 12-bit integer, its value ranging from 0 to 4095. The most significant 4 bits are called the *majorSdoId,* while the least significant 8 bits are called the *minorSdoId*. Here, the range of *domainNumber* is subject to the *sdoId* value [7].

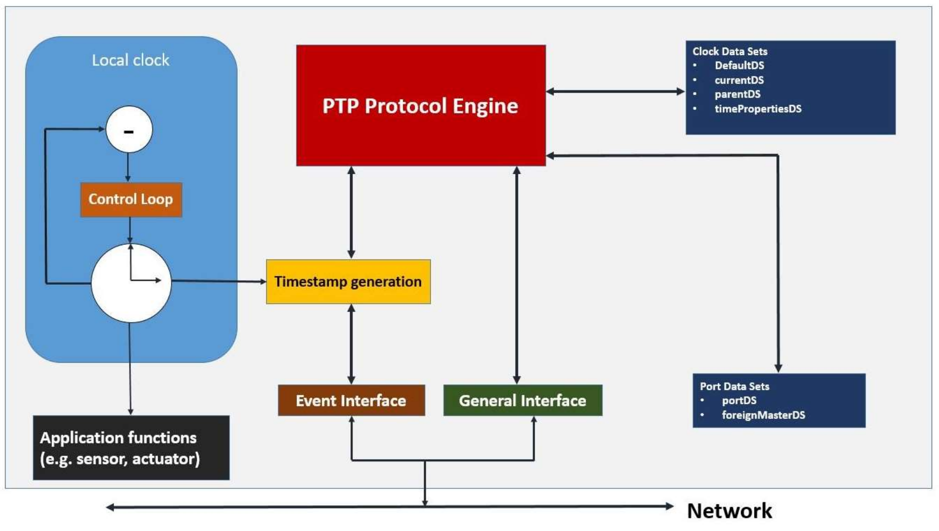

Figure 2-1. Ordinary clock model [1].

2.2.2 Clock

A. Ordinary Clock

An ordinary clock uses two logical interfaces at its port to communicate with other nodes, namely the Event Interface and the General Interface, which are responsible for distinguishing between the types of PTP messages. The Event Interface passes and timestamps only event messages using the Timestamp Generation block, whereas the General Interface allows passing other messages that are not required to be timestamped [1].

Figure 2-1 [1] shows an ordinary clock model that contains different blocks as follows:

Clock Data Set (as defined by the IEEE Standards [5]):

- “defaultDS: Attributes describing the ordinary clock.
- currentDS: Attributes related to synchronization.
- parentDS: Attributes describing the parent (the clock to which ordinary clock synchronizes) and grandmaster (the clock at the root of master-slave hierarchy).

- timePropertiesDS: Attributes of the timescale.”

The Port Data Sets block contains port attributes that include the PTP state (as defined by IEEE 1588 [5]).

The PTP Protocol Engine is responsible for all PTP communications and is responsible for synchronizing the clock to the grandmaster clock timescale. It maintains the data set as well as a two-way PTP communication with other network components. Moreover, it is responsible for the computation of the master’s time based on the PTP packets received on its port [1].

The Control Loop in the local clock block has the responsibility of adjusting the local clock to the master’s clock. This occurs at the guidance of the PTP Protocol Engine, as the PTP Protocol Engine calculates the offset that requires to be added to the local clock [5]. The PTP Protocol Engine has direct access to adjust and correct the local clock [1].

Depending on the characteristics of an ordinary clock, the clock can be a grandmaster or slave clock as follows:

1. Grandmaster Clock

A grandmaster clock is considered the most accurate clock in any PTP network and is utilized as a time reference for all other clocks [31]. It has a high-quality local oscillator that does not easily drift and provides highly stable and precise time information [1].

2. Slave Clock

A slave clock is an end device in a PTP network that is synchronized and syntonized (i.e., make a slave clock frequency follows the grandmaster clock frequency) to the grandmaster clock [1].

In IEEE 1588-2019, an ordinary clock is divided into three different sections: the external environment that contains clock sources (e.g., GNSS receivers) and clock sinks (i.e., external applications such as sensors, actuators, and computation element), PTP instance (i.e., an instance of PTP protocol that works in one domain), and the communication network. Since each ordinary clock has only one PTP physical port, this port consists of two blocks, namely a PTP port block and a Media Dependent block. Each block is separated by Media-Dependent, Media-Independent (MDMI) interface. The PTP port block is responsible for operations of PTP Port Data Sets and the operation of the best master clock algorithm (BMCA). The Media Dependent (MD) block is divided into two sub-blocks [7]:

- a PTP MD adapter that handles the reception and transmission of all PTP messages as well as handling the time transfer between the PTP Port block and the Network Interface Stack.
- the Network Interface Stack.

The communication between the PTP MD adapter block and the Network Interface Stack is conducted through three interfaces [7]:

- General Message Functions that are responsible for receiving and sending PTP general messages.
- Events Message Functions that are responsible for receiving and sending PTP event messages.
- Other Adapter Functions that handle all other functions.

The PTP Port can be an end-to-end (E2E) port that implements the E2E methodology of time transfer and determines the path delay; can be a peer-to-peer (P2P) port that implements P2P methodology of time transfer and determines the path delay; or can be a Special port to transfer time between PTP instances (not PTP timing messages) [7].

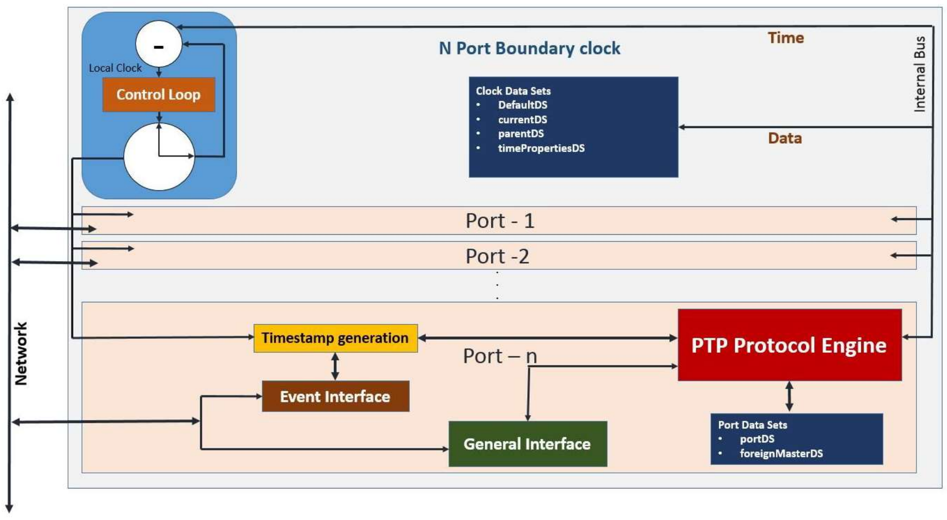

Figure 2-2. Boundary clock model [1].

B. Boundary Clock (BC)

The concept of a boundary clock is taken from IEEE 1588-2002. A boundary clock is a network component that has multiple PTP ports (see Figure 2-2); however, it does not provide direct access between slaves and their grandmaster. Instead, the boundary clock is acting as a slave to the grandmaster and as a master to slaves connected to it (i.e., the port connected to the grandmaster will be in a slave state, and the other ports will be in a master state). The boundary clock sets its internal clock according to information provided by its PTP Protocol Engine and gets synchronized to its grandmaster clock [1], [32].

In comparison to an ordinary clock, a boundary clock has the following differences [1]:

- The Clock Data Sets are common to all boundary clock ports.
- The local clock is common to all boundary clock ports.
- Each Protocol Engine determines which port provides the time signal used to synchronize its local clock.

Among all PTP messages, the management messages are only forwarded to other network components, but with limitations (i.e., forwarding management message is limited by the value of *boundaryHops*). The boundary clock is applicable only in the context of the PTP protocol; otherwise, it is an ordinary switch, repeater, router, bridge, etc. which means that the boundary clock has no additional role with other types of messages [1].

In IEEE 1588-2019, a boundary clock has the same ordinary clock structure but with multiple PTP ports [7].

C. Transparent Clock (TC)

The concept of a transparent clock was introduced in IEEE 1588-2008. In comparison to a boundary clock, a transparent clock would not block direct access to the time reference provided by the grandmaster. It measures the time taken of every PTP event message inside it and updates the packet's time correction field accordingly. In other words, a transparent clock does not differentiate between its ports (i.e., master or slave) and allows transparent access to the grandmaster in the network [1].

There are two types of transparent clocks, end-to-end transparent clock [33] and peer-to-peer transparent clock. The only difference between end-to-end transparent and peer-to-peer transparent clocks is the manner in which delay calculation takes place and the corresponding corrections in the PTP message fields. The rest are the same [1].

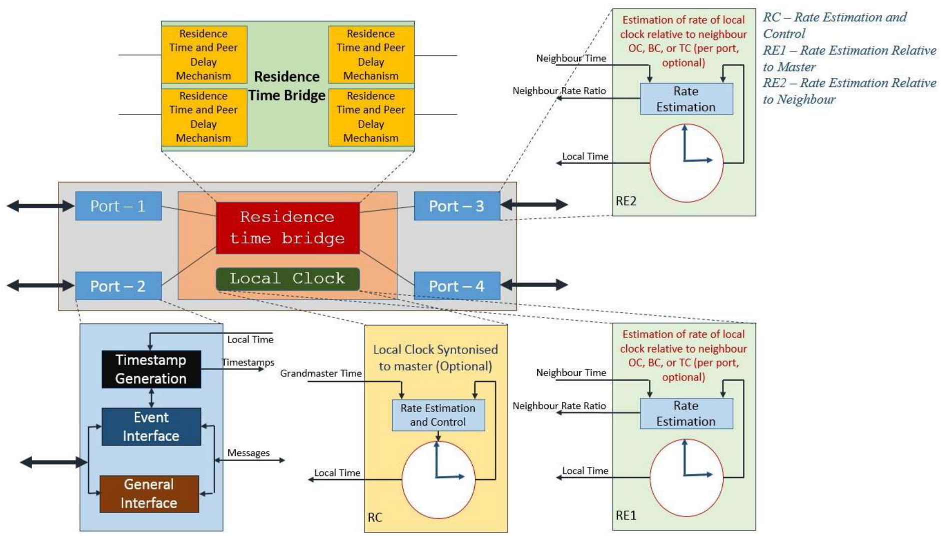
Residence Time and Peer Delay Mechanism
Residence Time and Peer Delay Mechanism
Residence Time Bridge
Residence Time and Peer Delay Mechanism
Residence Time and Peer Delay Mechanism
Estimation of rate of local clock relative to neighbour OC, BC, or TC (per port, optional)
Neighbour Time
Neighbour Rate Ratio
Rate Estimation
Local Time
RE2
RC – Rate Estimation and Control
RE1 – Rate Estimation Relative to Master
RE2 – Rate Estimation Relative to Neighbour
Port – 1
Residence time bridge
Port – 3
Port – 2
Local Clock
Port – 4
Local Time
Timestamp Generation
Timestamps
Event Interface
Messages
General Interface
Grandmaster Time
Local Clock Syntonised to master (Optional)
Rate Estimation and Control
Local Time
RC
Estimation of rate of local clock relative to neighbour OC, BC, or TC (per port, optional)
Neighbour Time
Neighbour Rate Ratio
Rate Estimation
Local Time
RE1

Figure 2-3. Peer-to-Peer Transparent clock model [1].

1. Peer-to-Peer Transparent Clock (P2P TC)

As shown in Figure 2-3 [1], a transparent clock has only one additional block (i.e., residence time bridge block) per port that is utilized to calculate and determine the delay per link. Therefore, a transparent clock can compute a peer-to-peer link delay only if its peer component in the link also has the same capability of peer-to-peer link delay calculation [1].

Pdelay_Req, *Pdelay_Resp*, and in some cases *Pdelay_Resp_Follow_Up* (two-step operating mode) messages are used to calculate the delay between peer devices per link and on all transparent clock ports. A peer-to-peer transparent clock only corrects *Sync* and *Follow_Up* messages by updating the respective field for link delay and residence time (i.e., *correctionField*) then forwards them [1].

A peer-to-peer delay measurement mechanism uses *Pdelay_Req* and *Pdelay_Resp* messages for delay request and delay response, while an end-to-end delay measurement mechanism uses *Delay_Req* and *Delay_Resp* messages for delay request-response. The two delay measurement mechanisms are independent, and their messages cannot interoperate with a different delay measurement mechanism. For example, an end-to-end transparent clock cannot work with a peer-to-peer transparent clock. In the peer-to-peer delay measurement mechanism, a master needs to generate only

Sync and *Follow_Up* messages and responds only to *Pdelay_Req* via *Pdelay_Resp* messages. As a result, the master does not receive any *Delay_Req* or issue *Delay_Resp* messages, and therefore the workload on the master is reduced [1].

To inter-connect peer-to-peer transparent clocks and end-to-end transparent clocks, a boundary clock must be used by acting as a slave on one side while acting as a master on the other [1].

2. End-to-End Transparent Clock (E2E TC)

The end-to-end transparent clock behaves as an intermediate node (i.e., ordinary router, repeater, switch, or as a bridge) for all the messages but treats PTP event messages in a different way. End-to-end transparent clock computes the residence time of every PTP event message and updates the correction field of the PTP message frame accordingly [34]. The information provided by the correction field of the PTP message frame is used to tell the PTP Protocol Engine in the slave clock about the time taken of a particular PTP message to transit the transparent clock. The slave uses this information to calculate the total offset from the grandmaster clock. It is worth noting that the residence time may differ for each message arriving on different ports of the same transparent clock [1].

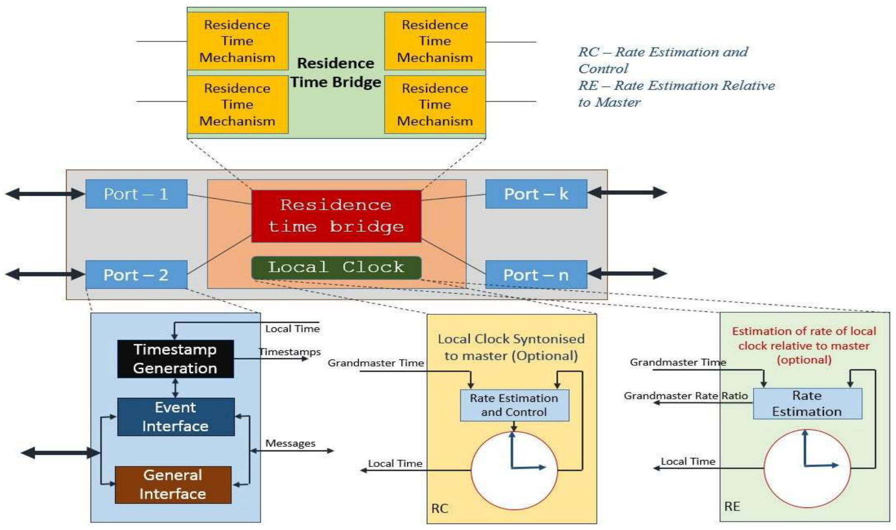

Figure 2-4. End-to-End Transparent clock model [1].

Since the residence time of a PTP event message depends on the measurements of the local clock of a transparent clock, it is important to make the local clock rate of the transparent clock run the same as the local clock rate of the grandmaster clock. This is because that when a slave calculates the offset from the grandmaster clock, even a 0.02% difference (which is likely possible) between a transparent clock and the grandmaster clock, may result in an unacceptably large error at the slave's end. For example, if a PTP event message takes 1ms to pass the transparent clock, by considering 0.02% of accumulated error that results from the mismatch rate of the transparent clock and the grandmaster clock, an overall error of 200ns can be induced at the slave's end, after calculation [1].

To avoid such an error of the mismatch rate of a transparent clock and the rate of a grandmaster clock, IEEE 1588 provides a method called a syntonization to syntonize a transparent clock to the grandmaster clock, where its clock frequency follows the grandmaster clock frequency. The syntonization process is done by observing *Sync* and *Follow_Up* (if present) messages received from the grandmaster clock. The Rate Estimation and Control block (see Figure 2-4) uses the time information provided by the *Sync* and *Follow_Up* messages alongside the time information of the local clock [1]. After collecting a set of *Sync* and *Follow_Up* messages from the grandmaster clock, the Rate Estimation and Control block calculate the difference between the local clock rate and the grandmaster clock rate. When the rate ratio between the two clocks is calculated, it can be used to adjust the transparent clock rate. The transparent clock does not necessarily change its oscillator frequency physically since the oscillator frequency can be easily drifted. Instead, a digital solution is possible by multiplying the timestamps taken by the calculated rate ratio. Rate adjustments of one node will impact the rate adjustments at all the nodes downstream [1].

In IEEE 1588-2019, a transparent clock has a Residence Time Bridge that operates in the same manner as a transparent clock introduced in IEEE 1588-2008. Unlike the ordinary clock and boundary clock, the transparent clock

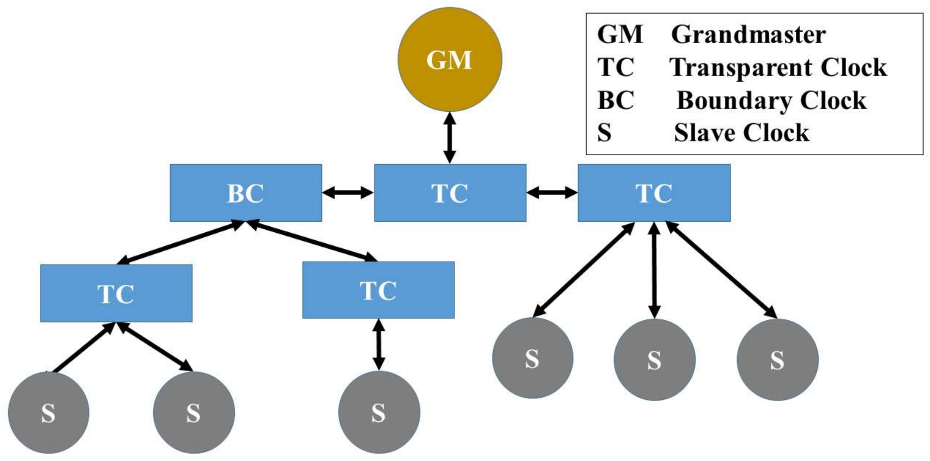

Figure 2-5: A typical PTP network.

does not have an external environment; however, it can be associated with an ordinary clock to provide real-time support for the application device (e.g., one of PTP Port of a transparent clock is internally connected to an ordinary clock) [7].

Figure 2-5 shows a simple PTP network that include different PTP clock types.

2.2.3 PTP Transport Protocols

Several transport protocols can be used to transport PTP messages. These protocols are as follows [3]:

1. PTP over UDP over IPv4: the first byte of a PTP message will immediately follow the last byte of the UDP header, as shown in Figure 2-6.

2. PTP over UDP over IPv6.

3. PTP over IEEE 802.3/ Ethernet.

4. PTP over DeviceNET.

5. PTP over ControlNET.

6. PTP over IEC 61158 Type 10.

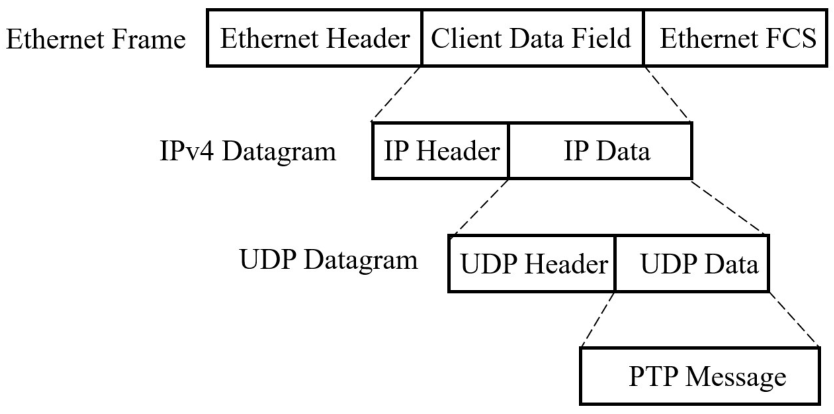

Figure 2-6. PTP message within UDP over IPv4 [3].

This thesis focuses only on the first transport protocol (i.e., PTP over UDP over IPv4); therefore, the details of the other transport protocols (as listed above) can be found in [5], [7].

2.2.4 PTP Message Types

All PTP messages contain common fields called the header, in addition to some different fields that may vary from one message to another. The structure of the header is as specified in Table 2-1 [5], [7]:

Table 2-1: Common PTP message header [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
transportSpecific/majorSdoId				messageType				1	0
reserved/minorVersionPTP				versionPTP				1	1
messageLength								2	2
domainNumber								1	4
reserved/minorSdoId								1	5
flagField								2	6
correctionField								8	8
reserved/messageTypeSpecific								4	16
sourcePortIdentity								10	20
sequenceId								2	30
controlField								1	32
logMessageInterval								1	33

1. transportSpecific/majorSdoId: transportSpecific is used in IEEE 1588-2008 to check the length of the incoming PTP event message that requires the UDP payload to be at least 124 octets in length [5]. This field is replaced by majorSdoId in IEEE 1588-2019, which contains the most significant 4 bits of the sdoId attribute.

2. messageType: it indicates the message type as further described below.

3. reserved/minorVersionPTP: in IEEE 1588-2008, this field is reserved, and all its bits are 0s. The field of minorVersionPTP is used in IEEE 1588-2019 to indicate the value of portDS.minorVersionNumber of the originating PTP Instance.

4. versionPTP: it indicates the value of portDS.versionNumber member of the originating PTP Instance.

5. messageLength: it is the total number of octets that form the PTP message.

6. domainNumber: it indicates the value of defaultDS.domainNumber member of the originating ordinary clock or boundary clock.

7. reserved/minorSdoId: in IEEE 1588-2008, this field is reserved, and all its bits are 0s. This field is replaced by minorSdoId in IEEE 1588-2019, which contains the latest significant 8 bits of the sdoId attribute.

8. flagField: it consists of two octets, and their values depend on the message type. For example, in *Sync* message, the flagField may indicate whether there is a *Follow_Up* message associated with it or not (i.e., two-step operation).

9. correctionField: it is the value of the correction measured.

10. reserved/messageTypeSpecific: in IEEE 1588-2008, this field is reserved, and all its bits are 0s. In IEEE 1588-2019, this field is replaced by messageTypeSpecific, and its value depends on the message type (i.e., general message or event message). For example, in a general message, the messageTypeSpecific is reserved while it may be used for internal implementation in the event message case.

11. sourcePortIdentity: it indicates the value of portDS.portIdentity of the PTP Port that originated this PTP message.

12. sequenceId: it indicates the sequence number of an individual message type.

13. controlField: this field is similar to the messageType field, but it conforms only with IEEE 1588-2002 and is obsolete in IEEE 1588-2019 (i.e., the field will be ignored on the recipient).

14. logMessageInterval: it is determined by the type of PTP message.

A. PTP Event Messages

A PTP event message is a message that requires to be timestamped. The following are the PTP event messages [1]:

1. *Sync*: The master clock issues a *Sync* message that includes the time of its local clock and sends it to all slaves in its domain. During this process, a delay is introduced due to the internal circuitry of the master. Therefore, there will be a difference between the *Sync* message timestamp and the actual time of leaving the master port, which will

Table 2-2: Sync message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
originTimestamp								10	34

influence the slave offset calculation. The master uses a *Follow_Up* message to carry the actual time at which the *Sync* message was transmitted. This process is known as a two-step operation. In a one-step operation, the *Sync* message is timestamped when it leaves the master's port [1]. The *Sync* message format is specified in Table 2-2 [5], [7].

a) originTimestamp: in one step operation, it contains the master time when the *Sync* message was transmitted. In a two-step operation, the value will be set to 0 [3].

2. *Delay_Req*: This type of event message will be issued by a slave in order to determine the link delay between the slave and its master. The slave records the sending time and waits for the master's [1]. The *Delay_Req* message format is similar to the *Sync* message format, as shown in Table 2-2. Here, the *originTimestamp* field contains the slave time when the *Delay_Req* message was transmitted [35].

3. *Pdelay_Req*: In a peer-to-peer mechanism, a PTP node issues *Pdelay_Req* message and records its transmitting time to determine the link delay between itself and its peer [1]. The message format is shown in Table 2-3 [5], [7].

Table 2-3: Pdelay_Req message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
originTimestamp								10	34
reserved								10	44

a) originTimestamp: it contains the requester clock time when the message was transmitted.

b) reserved: it is used to make the length of the *Pdelay_Req* message equal to the length of the *Pdelay_Resp* message.

4. *Pdelay_Resp*: In a peer-to-peer mechanism, a PTP node uses a *Pdelay_Resp* message to carry the arrival time of a *Pdelay_Req* message to its peer. The IEEE Standard provides several options in how to convey the timestamp information in the *Pdelay_Resp* message [1].

    a) The *Pdelay_Resp* message conveys the time difference between the transmission time of the *Pdelay_Resp* message and the receipt time of the corresponding *Pdelay_Req* message [5].
    b) The *Pdelay_Resp_Follow_Up* message, which follows the *Pdelay_Resp* message, conveys the time difference between the transmission time of the *Pdelay_Resp* message and the receipt time of the corresponding *Pdelay_Req* message [5].
    c) The *Pdelay_Resp* message conveys the receipt time of the corresponding *Pdelay_Req*, and the *Pdelay_Resp_Follow_Up* message, which follows the *Pdelay_Resp* message, conveys the transmission time of the *Pdelay_Resp* message [5].

The *Pdelay_Resp* message format is shown in Table 2-4 [5], [7].

a) requestReceiptTimestamp: it contains the responder clock time when the *Pdelay_Req* message was received.

b) requestingPortIdentity: it contains the sourcePortIdentity field value that is existed in the header of the associated *Pdelay_Req* message.

Table 2-4: Pdelay_Resp message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
requestReceiptTimestamp								10	34
requestingPortIdentity								10	44

Table 2-5: Announce message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
originTimestamp								10	34
currentUtcOffset								2	44
reserved								1	46
grandmasterPriority1								1	47
grandmasterClockQuality								4	48
grandmasterPriority2								1	52
grandmasterIdentity								8	53
stepsRemoved								2	61
timeSource								1	63

B. PTP General Messages

General messages are not required to be timestamped by the PTP ports. The PTP Protocol Engine uses the information provided by general messages for calculation, estimation, and state establishment[1].

1. *Announce*: this message contains characteristics of the PTP clocks in order to elect a grandmaster clock using the best master clock algorithm[1]. The Announce message format is shown in Table 2-5 [5], [7].

    a) originTimestamp: it will be set to 0, or it will contain the PTP node time that originates the Announce message.

    b) currentUtcOffset: it contains the timePropertiesDS.current-UtcOffset value.

    c) reserved: it contains the timePropertiesDS.currentUtcOffset value.

    d) grandmasterPriority1: it contains the parentDS.grandmaster-Priority1 value.

e) grandmasterClockQuality: it contains the parentDS.grandmasterClockQuality value.

f) grandmasterPriority2: it contains the parentDS.grandmasterPriority2 value.

g) grandmasterIdentity: it contains the parentDS.grandmasterIdentity value.

h) stepsRemoved: it contains the currentDS.stepsRemoved value of the data set of the node issuing the *Announce* message.

i) timeSource: it contains the timePropertiesDS.timeSource value.

2. *Follow_Up*: it will be sent after a *Sync* message in the two-step operation to carry the actual transmission time of the *Sync* message [1]. The message format is shown in Table 2-6 [5], [7].

Table 2-6: Follow_Up message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
preciseOriginTimestamp								10	34

a) preciseOriginTimestamp: it contains the master time when the *Sync* message was transmitted.

3. *Delay_Resp*: It conveys the arrival time of the *Delay_Req* message to a master clock [1]. The message format is shown in Table 2-7 [5], [7].

Table 2-7: Delay_Resp message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
receiveTimestamp								10	34
requestingPortIdentity								10	44

a) receiveTimestamp: it contains the master time when the *Delay_Req* message was received.

b) requestingPortIdentity: it contains the sourcePortIdentity field value that is existed in the header of the associated *Delay_Req* message.

4. *Pdelay_Resp_Follow_Up*: in a peer-to-peer mechanism, a PTP clock uses *Pdelay_Resp_Follow_Up* message (in case of two-step operation is applied) to carry the time at which *Pdelay_Resp* was transmitted [1]. The message format is shown in Table 2-8 [5], [7].

Table 2-8: Pdelay_Resp_Follow_Up message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
responseOriginTimestamp								10	34
requestingPortIdentity								10	44

a) responseOriginTimestamp: it contains the responder time when the *Pdelay_Resp* message was transmitted.

b) requestingPortIdentity: it contains the sourcePortIdentity field value that is existed in the header of the associated *Pdelay_Req* message.

5. *Management*: the purpose of this message is to carry information and commands to manage clocks on a PTP network [1]. The message format is shown in Table 2-9 [5], [7].

a) targetPortIdentity: it contains the target port address.

Table 2-9: Management message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
targetPortIdentity								10	34
startingBoundaryHops								1	44
boundaryHops								1	45
reserved			actionField					1	46
reserved								1	47
managementTLV								M	48

b) startingBoundaryHops: it contains the number of boundary clocks that the *Management* message is allowed to be retransmitted by.

c) boundaryHops: it contains the number of remaining boundary clocks that can retransmit a particular *Management* message.

d) reserved: it will be set to 0.

e) actionField: it contains the action type that the *Management* message must perform.

f) reserved: it will be set to 0.

g) managementTLV: it contains a management error status TLV.

6. *Signalling*: this message is used to exchange information, requests and commands between PTP clocks [1]. The message format is shown in Table 2-10 [5], [7].

Table 2-10: Signaling message format [5], [7].

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
targetPortIdentity								10	34
One or more TLVs								10	44

a) targetPortIdentity: it contains the target port address.

2.3 Synchronization Overview

Two phases are required to establish a successful synchronization.

- Establishing the Master-Slave hierarchy
- Synchronizing the clocks

These phases depend on each other since establishing the topology sets the stage for the PTP communication process. Therefore, establishing a correct master-slave topology is essential for proper synchronization. The ports' states are determined by examining *Announce* messages received on all connected network ports. *Announce* messages are analyzed, and their data are compared to determine the ports' state of each clock in the network. For this reason, the Best Master Clock Algorithm is used [1].

Any ordinary clock or boundary clock can have three possible states as follows [1]:

- MASTER: Acts as a time reference to all PTP nodes in a network.
- SLAVE: Ports that synchronize to a port in master-state.
- PASSIVE: This state is not a master nor a slave.

Each port maintains its own copy of the PTP state machine as defined above [1].

In PTP, a mesh network can form cyclic paths, which is harmful to the high precision synchronization. In such a network, some messages may be repeated in cycles creating multiple copies of the same messages and therefore creating ambiguity within the network nodes. To avoid such scenarios, outside protocols such as Spanning Tree Protocols (STP) are required to be deployed on all PTP nodes. As a result, effective tree topology is created. The grandmaster clock represents the tree root, while boundary clocks and slave clocks represent the branches and leaves of the tree, respectively [1].

2.4 Best Master Clock Algorithm (BMCA)

A local clock port may receive several *Announce* messages that describe the clocks' data and characteristics in a PTP network. For establishing and determining the state of the local clock port (Master, Slave or Passive), the BMCA analyzes the data collected and determines which clock is the best in the network. The decision of BMCA depends on the quality of the clock, stability of its local oscillator, its time base, and the closest of them (in case there is a group of clocks that have the same high quality in the network). A situation can arise when multiple *Announce* messages reflect the same foreign master clock due to the failure of removing cyclic paths. To avoid such a scenario, the distance between the foreign master clocks and the local clock is measured depending on the total number of boundary clocks in between them. This information is added in a special field called a *stepsRemoved* field in the PTP *Announce* message frame. Therefore, BMCA selects only one active grandmaster clock in the network, which has a better characteristics and its location is close to the most slaves [1].

According to IEEE 1588-2008, BMCA consists of two separate algorithms [1].

- Data set comparison algorithm
- State decision algorithm

The data set comparison algorithm specifies the quality and class of the local clock, on the basis of which the state decision algorithm decides the state of the port, i.e. master, slave, or passive [1].

The data set comparison algorithm uses a set of parameters to compare different clocks. These parameters are specified by IEEE 1588-2008 as follows:”

- priority1: A user configurable designation that a clock belongs to an ordered set of clocks from which a master is selected.
- clockClass: An attribute defining a clock’s international atomic time (TAI) traceability.
- clockAccuracy: An attribute defining the accuracy of the clock.
- offsetScaledLogVariance: An attribute defining the stability of a clock.
- priority2: A user configurable designation that provides finer grained ordering among otherwise equivalent clocks.
- clockIdentity: A tie-breaker based on unique identifiers” [5].

When the BMCA is completed, each PTP node is sure about its state machine, and therefore, a master-slave hierarchy is established, which allows smooth communication between PTP network components [1].

2.5 Synchronizing the Clocks

After a master-slave hierarchy is established, a series of PTP messages are exchanged between a master clock and its slave clocks to transfer the accurate time from the master to the slaves [36]. In these messages, the master sends its accurate time to slaves as well as responding to the slaves’ messages to determine the transmission delay in the communication path. Then, the PTP Protocol Engine of the slaves calculates the offset from the master clock, which is then added to its local clock’s time. Next, slaves get the same time as that of the master, which is known as synchronization. After that, the master and its slaves keep exchanging these messages to keep the slave clocks rate close to the master clock rate and therefore not allowing drift [1].

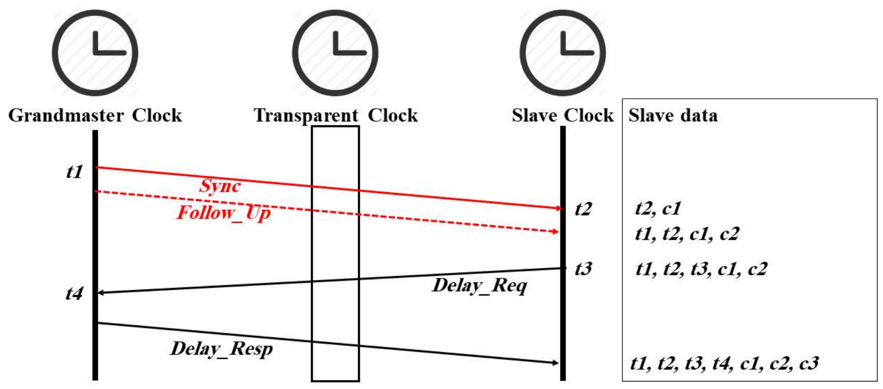

Figure 2-7. PTP timestamps and time synchronization messages in E2E delay mechanism.

2.5.1 Time Synchronization using the End-to-End request-response Mechanism (E2E)

In the E2E mechanism (see Figure 2-7), the master sends a *Sync* message to slave clocks that includes its departure time t1, while the slave clocks record its arrival time t2. If the master does not support the hardware timestamp (one-step operation mode), the master sends a *Follow_Up* message that conveys t1 to the slaves (two-step operation mode). All slaves send a *Delay_Req* message to the master and record its departure time t3. The master records their arrival times t4 and send them back to the respective slaves using the *Delay_Resp* message. If a transparent clock is located between the master and the slaves, the *correctionField* value of *Sync* (c1), *Follow_Up* (c2), and *Delay_Req* (c3) messages will be updated when the messages are crossing the transparent clock device. Synchronization messages that belong to the same cycle share the same sync sequence id *syncID*, a 16-bit counter that is incremented with each cycle [37]. Within a given cycle, the slaves have all timestamps required to calculate the offset and delay as follows [38]:

$$offset = ((t2 - t1 - c1 - c2) - (t4 - t3 - c3))/2 \qquad (2.1)$$

$$delay = ((t2 - t1 - c1 - c2) + (t4 - t3 - c3))/2 \qquad (2.2)$$

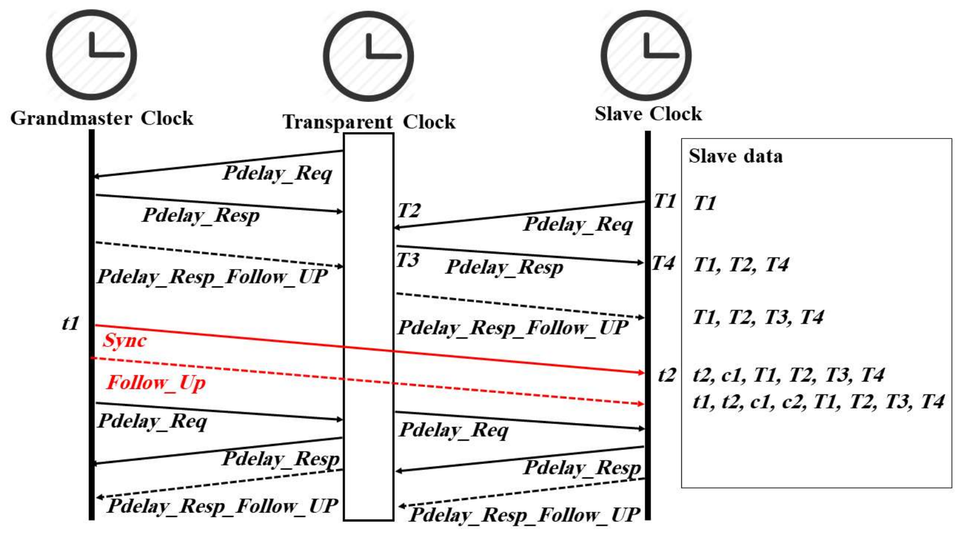

Figure 2-8. PTP timestamps and time synchronization messages in P2P delay mechanism.

2.5.2 Time Synchronization using the Peer-to-Peer Mechanism (P2P)

In the P2P mechanism (see Figure 2-8), the *Sync* and *Follow_Up* messages will work in the same manner as the E2E mechanism, and the difference is only in the delay calculation [39]. The path delay calculation is made by each P2P aware PTP port regardless of the port state (i.e., master or slave). This behaviour helps to correct the path delay immediately upon a network reconfiguration [5]. Figure 2-8 shows how PTP clocks exchange the required messages. For example, the slave firstly sends a *Pdelay_Req* message to the transparent clock and records its departure time T1. The transparent clock receives and records the arrival time T2 of this message. Next, the transparent clock sends a *Pdelay_Resp* message to the slave clock that includes T2 and records the departure time T3. After that, the slave clock generates T4 upon receiving the *Pdelay_Resp* message. Finally, the transparent clock sends a *Pdelay_Resp_Follow_Up* message to the slave clock that includes T3 [40]. The path delay between the transparent clock and the grandmaster clock will be calculated at the transparent clock side as follows [5]:

$$delay = ((T4 - T1) - (T3 - T2))/2 \qquad (2.3)$$

Depending on the applied operation mode (one-step or two-step operation mode), the calculated delay will be added to *c1* or *c2* and before the *Sync* or *Follow_Up* message leaving the transparent clock, respectively. The slave clock will do the same to measure the link delay between the transparent clock and itself. The offset then can be calculated as follows [5]:

$$offset = (t2 - t1 - c1 - c2) - slave\ link\ delay \qquad (2.4)$$

2.6 PTP Software Daemons

There are many PTP implementations that work on different platforms [41]. This section will focus on two popular open-source PTP slave software implementations, namely linuxptp (also called PTP4l) and PTPd. PTP4l is a Linux client [42], whereas PTPd is available for Linux, FreeBSD and Mac OS X [43]. While both implementations follow the message sequence, for example as shown in Figure 2-7, in practice, the GM and the slave clocks send out *Sync*/*Follow_Up* and *Delay_Req* messages independently from each other at configurable rates, i.e., the transmission of a *Delay_Req* message is not triggered by the reception of a *Follow_Up* message. The delay request measurements are only done on a best effort basis and may not be executed in a timely manner [2].

To better understand the time correction algorithms, I reverse-engineered the code of PTPd and PTP4l to determine their characteristics and their weaknesses.

2.6.1 PTP Daemon (PTPd)

PTPd [44] records the time arrival of each *Sync* message (*T2*) as well as the content of its *correctionField* (*C1*). It then extracts the *preciseOriginTimestamp* field value (*T1*) and the *correctionField* value (*C2*) of the corresponding *Follow_Up* message (that matches the *sequenceID* of the *Sync* message). After that, the PTPd calculates the delay between the master and the slave:

$$MSdelay = T2 - T1 - (C1 + C2) \tag{2.5}$$

The slave then records the departure time of a *Delay_Req* message (*T3*) and eventually receives the corresponding *Delay_Resp* message (with identical *sequenceID*). It extracts the *receiveTimestamp* field value (*T4*) and the *correctionField* value (*C3*) of this message. The slave now calculates the delay between slave and master:

$$SMdelay = T4 - T3 \tag{2.6}$$

and the mean path delay via

$$meanPathDelay = ((MSdelay + SMdelay) - C3)/2 \tag{2.7}$$

If the *meanPathDelay* value is greater than one second or a negative value, it is replaced by the last valid delay value before the offset is calculated:

$$offsetFromMaster = MSdelay - meanPathDelay \tag{2.8}$$

Finally, PTPd calculates the average of this offset and the previous offset before updating the system clock [45]. The overall offset equation (without averaging) is as follows:

$$Offset = (T2 - T1 - (C1 + C2)) - (((T2 - T1 - (C1 + C2)) + (T4 - T3)) - C3)/2 \tag{2.9}$$

More details on PTPd can be found in [46].

2.6.2 Linuxptp (PTP4l)

PTP4l [47] records the arrival time of the *Sync* message (*T2*), extracts its *correctionField* value (*C1*), waits for the arrival of the corresponding *Follow_Up* message (with identical *sequenceID*) and extracts the *preciseOriginTimestamp* (*T1*) as well as the *correctionField* value (*C2*). It records the transmission time (*T3*) of a *Delay_Req* message and extracts its arrival (*T4*) and its *correctionField* (*C3*) from the corresponding *Delay_Resp* message. PTP4l has provisions for calculating the time difference (*delayAsymmetry*) between the transmitting and receiving path (which is positive when the master-to-slave propagation time is longer and negative when the slave-to-master time is longer). However, a default value of zero is hard-coded into the software. PTP4l computes the ratio and frequency deviation of the local clock in relation to the master clock by using the last two values of *T2* and two corrected values of *T1* (including the path delay), as follows:

$$ratio = (T2n - T2n{-}1)/(T1n - T1n{-}1) \tag{2.10}$$

$$frequency = (1.0 - ratio) * 10^9 \tag{2.11}$$

The final offset and delay equations are as follows:

$$Offset = (T2 - (T1{+}C1{+}\ delayAsymmetry\ {+}C2)) - ((T2 - T3) * frequency + (T4 - C3 - (T1 + C1 + delayAsymmetry + C2)))/2 \tag{2.12}$$

$$meanPathDelay = (((T2 - T3) * ratio) + ((T4 - C3) - (T1 + C1 + C2)))\ 2 \tag{2.13}$$

It is worth noting that for both daemons, the above methodologies/algorithms did not change between recent software versions.

2.7 Slave Clock Adjustment

Both PTP daemons use the calculated offset to update the slave clock by either resetting the clock, i.e., step the clock, or gradually adjusting the clock, i.e., add a small amount of time to the clock every second and/or change the clock frequency [48], using the following configuration options:

- Disable/enable PTP clock adjustment: if the clock adjustment is disabled, the clock will be in free-running mode. If enabled, both daemons use the time offset to calculate the frequency offset between the slave and its master and gradually adjust the local clock tick duration to improve the local slave clock accuracy. Furthermore, PTPd updates the slave time by adding a small percentage of time to the slave clock depending on the calculated offset [2].

- Maximum clock frequency adjustment: this is an upper threshold for the maximal permissible (positive or negative) frequency correction as calculated above [2].

- Disable/enable clock reset: if disabled, local clock adjustments per synchronization cycle are limited to the above value. If enabled, the daemon resets the clock when synchronization starts and/or when a calculated offset is larger than a configurable threshold. Resetting the clock makes the slave time the same as the master time in one synchronization step. Further on, PTP4l will also adjust the slave clock frequency before doing a clock reset [2].

2.8 Advanced Persistent Threats and PTP

The last decade has been marked by significant security problems and the emergence of complex cyber-attacks called Advanced Persistent Threats (APTs). Most of the APTs use sophisticated procedures, methods, and tactics to manipulate their targets. As a result, most of the security countermeasures are unable to prevent or detect such attacks [49]. Ussath et al. [49] have analyzed 22 different APT reports, and it was concluded that three main phases are responsible for characterizing an APT: the initial compromise (e.g., how an attacker gain access to the target system), the lateral movement (e.g., how an attacker moves and manipulates multiple devices inside in the target system), and the command and control activity (e.g., an attacker chooses a node to act as a server for the attack).

Stuxnet is a famous example of APTs that targeted Iran's nuclear program. It is malware that infiltrated the Iranian network via a USB flash drive that has files to launch executable code and then infected programmable logic controller (PLC) programs such as WinCC and PCS 7 programs on Windows and made them deviate from their normal operation. Stuxnet is spread to other computers connected to the infected one. Stuxnet does not require an Internet connection to update itself. Instead, it uses a command-and-control server to provide the infected PCs with data and download executable files [50].

Similar to Stuxnet, an effective internal attack on a PTP network would be best conducted slowly over extended periods of time via an APT, after a successful penetration, reconnaissance, and the setup of command and control (C&C) communication. For example, an employee with sufficient privileges could unknowingly bring malware into the target organization (e.g., via phishing or an infected USB stick), which spreads within the internal PTP network to identify and reconnoitre the role of each connected PTP device (e.g., TC, BCs and the GM). After that, it would choose a node to be the C&C server for the attack while manipulating the firmware or configuration parameters of other PTP devices (e.g., TC) pivotal for the attack, thereby rendering even infrastructure redundancy useless. In the subsequent attack phase, the time synchronization of slaves would be gradually manipulated via subtle coordinated changes, therefore causing slave clocks to go slowly out of sync. The damage caused (e.g., out of sync trading transactions) could be unnoticed for a long time, similar to the way Stuxnet operated stealthily over many months in the Natanz facility [2].

Chapter 3

Precision Time Protocol Attack Strategies and their Resistance to Existing Security Extensions

The work outlined in this chapter was published in:

W. Alghamdi and M. Schukat, "Precision time protocol attack strategies and their resistance to existing security extensions," *Cybersecurity,* vol. 4, p. 12, 2021/04/01 impact factor 1.959.

This chapter provides an in-depth analysis of potential APT attack strategies on PTP networks, including attacker types (i.e., man in the middle or injector), their location within a network, their impact on clock synchronization (i.e., clock manipulation, versus clock free-running), and the impact range (i.e., affecting all slaves, a subset of slaves or a single slave) in the presence of protocol security extensions and infrastructure redundancy including Annex P, therefore extending prior research conducted by [22, 23]. This chapter is structured as follows: Section 3.1 provides an introduction to the advanced persistent threats (APTs). The attack types will be specified in Section 3.2, while Section 3.3 will define the attacker types. Section 3.4 determines the scope of the analysis. All existing security extensions are listed in Section 3.5. The potential PTP attack strategies and their implementations will be discussed in Section 3.6, where a set of experimental results will be shown in 3.7. A conclusion is given in Section 3.8. This chapter answers the first research question as outlined in Chapter 1.

3.1 Introduction

As briefly introduced in the previous chapter, Advanced Persistent Threats (APTs) use sophisticated procedures, methods, and tactics to compromise target systems through different phases. Such attacks often begin by targeting a small number of power users within the target organization with malicious software, for example, malware on secondary memory devices (i.e., USB sticks) or phishing emails. They then propagate themselves across the organization by exploiting software flaws. Several technology providers, including RSA and Google, fell victim to APTs and made it public. The emergence of APTs has demonstrated the limitations of network-centric perimeter security that has been exercised for many years, where a firewall isolates and protects infrastructure and information from unreliable networks, e.g., the Internet. With APTs, all networks are deemed unreliable, and the security perimeter has to be user-centric [19].

APTs, by their nature, are very difficult to detect [51] and typically incorporate either a static (and therefore less effective) signature-based malicious code detection or a behaviour-based detection [52] using correlation analysis, for example, of network traffic patterns. Here a recent trend to use machine learning methodologies can be observed [53].

Stuxnet and the subsequent attack on the Natanz Uranium enrichment facility in 2010 is an example of an advanced attack on critical infrastructure. It started with an infected USB stick, which was unknowingly brought into this high-security facility by an employee. Stuxnet subsequently spread across the isolated Natanz network infrastructure and took over control of PLCs and SCADA systems responsible for the centrifuges used in the Uranium enrichment process, which were subsequently damaged via subtle changes to their operating parameters over many months [20, 21]. Stuxnet would have been very hard to detect even with today's advances in machine learning techniques, as its operation caused no apparent changes in network traffic patterns or PLC behaviour.

Another important example of critical infrastructure relates to the exact clock synchronization between computer systems, as required by many sectors such as telecommunications or financial services, where local time sources (e.g., quartz-based real-time clocks) alone are not sufficient because of stability and accuracy problems that affected by temperature and age, resulting in local clock derivations in the order of milliseconds per day [54]. Over packet-switched networks, such time synchronization can be provided by two protocols, the Network Time Protocol (NTP) and the Precision Time Protocol (PTP) [55]. These protocols are the basis for time-sensitive systems, especially distributed network systems, as they manage how a host clock is adhering to a time-scale reference. Both protocols are based on a clock synchronization technique that specifies the order and sequence of message transmissions between a host and reference clock, the message structure as well as the required time synchronization processes [12].

Time synchronization protocols are typically designed for particular types of networks: The Network Time Protocol (NTP) is suitable for large and dynamic latency packet-switched networks (PSNs), using complex statistical techniques that effectively reduce the inherent synchronization errors in such networks [13], [56]. It fulfils the requirements of distributed systems that need accuracy in the order of a few milliseconds over wide area networks. On the other hand, the Precision Time Protocol (PTP) [5] is designed for infrastructure networks, i.e., well-managed PSNs, that often use specialized (PTP-aware) hardware, providing clock synchronization accuracy down to microsecond and even nanosecond level [14].

Many financial markets and leading exchanges such as IMC, Eurex, and NYSE allow PTP time synchronization from their systems with market client/participants so that they can synchronize their clocks with the exchange [27]. Here PTP failure can lead to devastating consequences. For example, Eurex uses a very sophisticated PTP time synchronization network to timestamp financial and stock transactions of their clients, including high-frequency trading. The synchronicity and accuracy of these timestamps are very important to the exchange and its customers. However, on 26th August 2013, a PTP infrastructure glitch occurred that, even though it was detected in time, forced Eurex to postpone its market opening. It later turned out that an incorrect leap second [57] calculation caused an erroneous synchronization of their critical systems [27]. Another example in smart grid infrastructure, Bonneville Power Administration reported, in June 2016, losing two 500 kV lines 40 and 80 miles long. An investigation concluded that the synchrophasors responsible for the load monitoring along these powerlines were incorrectly synchronized because of erroneous GPS timestamps, resulting in incorrect line current differential readings. The GPS malfunction itself was caused by test procedures executed on the GPS satellites used by the synchrophasors [58], [2].

While these examples demonstrate the impact of network time synchronization problems, they raise the more general question of the vulnerability of PTP- and NTP-based time synchronization packet-switched networks to APTs, which subsequently pose a high risk to many time-sensitive application areas [23]. Previous attempts at security protocol extensions, such as the (OSI layer 7) IEEE 1588 Annex K for PTP, and Autokey for NTP, are insufficient to deter cyber-attacks [14]. Moreover, state-of-the-art network layer 2 and layer 3 protocols (i.e., MACsec and IPsec) can only deter a subset of possible attack strategies, namely external attacks [23].

This section focuses on a much more devious attack type on PTP networks, the internal attacks, which are much harder to detect, as they allow an attacker to compromise PTP infrastructure components, similar to the way Stuxnet compromised industrial control infrastructure in its final stage of operation [59], [60]. As with Stuxnet, internal PTP attacks do not cause obvious behavioural changes in PTP devices or unusual network traffic patterns. Therefore this section does not analyze the effectiveness of APT mitigation and detection strategies but focuses on viable internal attack strategies.

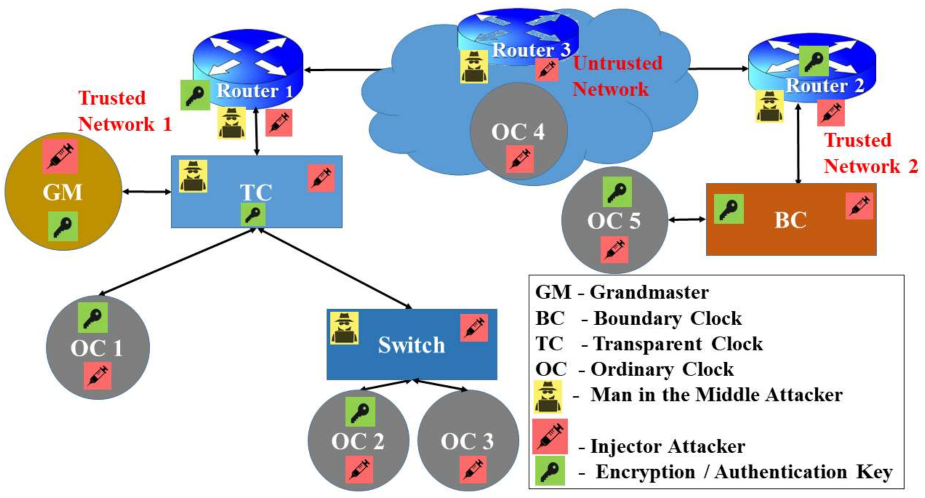

Figure 3-1. PTP Network with Attack and Attacker Positions.

3.2 Attack Types

RFC7384 [61] entails cyber-attack threats to time synchronization protocols. It distinguishes between internal and external attacks by either a man-in-the-middle (MitM) or an injector attacker [14]. Figure 3-1 illustrates such attacks, the attackers, and their locations within a PTP network model that incorporates the various PTP infrastructure components previously mentioned. The diagram distinguishes between two trusted networks (1 and 2), which are interconnected via an untrusted network. Both trusted networks are fully managed and have (potentially) implemented the same L2, L3, or L7 (i.e., Data Link, Network, or Application Layer) security mechanisms. Similarly, [22] distinguishes between insider and outsider adversaries. The outside adversary can only see multicast messages, while the inside adversary can see all protocol messages.

3.2.1 Internal Attack

Here the attacker has access to a trusted component of the network and may have access to the security (i.e., authentication/encryption) keys used. An internal attacker can maliciously manipulate legitimate network traffic or create new packets that appear legal to the manipulated nodes [61]. The internal attack will be further classified into two sub-categories, as follows:

A. Simple Internal Attack

Here an attacker resides within a trusted network, either on a secretly-added untrusted device or on a legitimate trusted device, but without having access to cryptographic keys. This kind of attacker has limited capabilities that may include packet removal, packet delays, or traffic generation to perform a denial of service (DoS) attack. In Figure 3-1, the switch and OC3 are points from which to launch a simple internal attack.

B. Advanced Internal Attack

Here the attacker gains full access to a device, including access to the encryption/authentication keys used by means of a malware infection or a manipulated firmware upgrade. Subsequently, the attacker takes control of the device behaviour or configuration, for example, by changing its clock properties to fool the BMC algorithm. This kind of attack can also change packet content in transit or generate new legitimate-looking packets. In Figure 3-1, Router1, Router2 GM, BC, TC, OC1, OC2, and OC5 are points from which to launch such an advanced internal attack.

3.2.2 External Attack

Here the attacker does not have possession of secret network encryption or authentication keys and resides outside the trusted network. In Figure 3-1, Router3 and OC4 are possible external attack points.

3.3 Attacker Types

3.3.1 Man-in-the-Middle Attacker (MitM)

A man-in-the-middle (MitM) attacker is located in a position where it can intercept and modify protocol packets in flight. It has physical access to a node of the PTP network or has gained full control of one device in the network [61]. For example, in Figure 3-1, Router1, TC and Switch are possible internal MitM attackers that reside in a trusted network (i.e., Trusted Network 1), while

Router2 is another example of an internal MitM attacker who has access to an intermediate node with the cryptographic keys in another trusted network segment (Trusted Network 2). Please note that while BC is an intermediate node, it acts as an endpoint between uplink and downlink and does not forward any event messages between the grandmaster and the other slaves. In contrast, Router3 is an example of an external MitM attacker who can prevent some or all protocol messages from arriving at their destinations.

3.3.2 Packet Injector Attacker

A traffic injector attacker is located in a position that allows it to generate network traffic. In Figure 3-1, an internal injector attacker can reside and inject traffic within the main network (Router1, GM, TC, Switch, OC1, OC2, and OC3) or has access to a node in another trusted network (Router2, BC and OC5). Router3 and OC4 are external injectors with limited attack capabilities [61].

3.4 Scope of Analysis

This thesis will mainly focus on internal attacks via packet injectors or man-in-the-middle since a PTP network is typically a tightly managed and therefore trusted (and potentially even isolated) infrastructure that is confined within an organization. External attacks are only considered in the context of Figure 3-1, with Router3 and OC4 being potential entry points.

The assumption is that an attacker gains access to one or more PTP infrastructure components via a malware infection [62] (e.g., by means of phishing or USB exploits as documented with Stuxnet) and, once established, launches an APT with the aim of compromising synchronization of PTP clocks stealthily, potentially over an extended period of time, in order to cause infrastructure failure or degraded service.

3.5 PTP Safeguards

3.5.1 Cryptographic Protocol Security

A. IEEE 1588 Annex K

IEEE 1588-2008 defines an experimental L7 security extension to PTP called Annex K [63]. It provides group source authentication, message integrity, and replay protection security using symmetric keys. It creates a trust relationship [64] utilizing a challenge-response three-way handshake mechanism based on pre-defined keys that are reached by subsets or the entire PTP domain [22]. Since its release in 2008, various flaws have been discovered [22], [65], [66], which resulted in various suggestions for protocol improvements, including the use of public-key encryption [22] and an improved handshake and replay counter [14]. The three-way handshake only increases traffic but provides no additional security [65]. Also, an attacker can stage an Annex K-specific type of DoS attack or can get the symmetric keys from any of the existing clocks [66], allowing any PTP slave to masquerade as the grandmaster [22]. These and other flaws caused Annex K to dropped in favour of other cryptographic protocol security extensions, as outlined further below.

B. E2E Protocol Security (IPsec)

IPsec provides L3 security protocols for IP networks by authenticating and encrypting IP packet payloads or by authenticating the non-modifiable sections of the IP header, hereby supporting both a transport mode between endpoints and a tunnel mode between security gateways [67]. Also, IPsec provides E2E message integrity that is retained from the protocol packet sender to the receiver [38], [68]. IPsec is designed to deter some external attacks, such as eavesdropping, replay attacks, and packet modification [69]. However, it is not designed to work in tandem with PTP [70]. For example, as an L3 protocol, it does not allow for PHY layer hardware timestamping or the easy integration of intermediate TCs to deliver the best possible slave clock synchronization [14], [23]. Also, tunnel mode IPsec does not support the integration of on-path intermediate BCs, while its cryptographic engine causes extra latency/jitter that negatively impacts the synchronization performance [71].

C. P2P Protocol Security (MACsec)

MACsec is an L2 security protocol that relies on IEEE 802.1X (for key management and session initiation) and IEEE 802.1AE (which specifies the authentication and encryption protocol) [72], [73]. As an L2 protocol, MACsec provides a hop-by-hop authentication and encryption mechanism, therefore supporting hardware timestamping and the full integration of BCs and TCs [14]. MACsec protects the connection between trusted segments of the network infrastructure but cannot prevent attacks that are launched from these trusted segments. It is complementary to end-to-end security protocols, as it can protect application data independently of network operations but cannot necessarily protect the operation of network segments [74]. This gives an opportunity to the advanced internal attacker, who resides in a trusted node, to launch a PTP attack (e.g., packet content modification attack) and therefore degrading the synchronization accuracy [38].

D. Type Length Value (TLV)

IEEE 1588-2019 uses an AUTHENTICATION TLV to share security-related information required to calculate the integrity check value (ICV) that is used for integrity and authenticity verification purposes [75]. This TLV is attached to all PTP messages to be secured. A secret key is used to create a unique ICV that is appended to a PTP message [75]. IEEE 1588-2019 provides two types of AUTHENTICATION TLV, namely immediate and delayed security processing. The former uses a key management protocol, e.g., Group Domain of Interpretation (GDOI), to process the AUTHENTICATION TLV by sharing all security parameters required to calculate the ICV before the PTP message content is further processed. This approach enables a transparent clock to make the required change to the *correctionField* value in a PTP message before it leaves the transparent clock port. For example, master and slave clocks can use immediate security processing to protect the outgoing messages by creating and appending AUTHENTICATION TLV and ICV to these messages and verify them at each PTP clock upon the reception [75]. In contrast, the delayed security processing depends on an associated key management protocol, e.g., Time Efficient Stream Loss-Tolerant Authentication (TESLA), which supports delaying the distribution of required security parameters, including the secret key. Here the master uses a specific secret key to protect the outgoing messages, and the slave receives and buffers these messages for later verification. Eventually, the master uses a new secret key and discloses the old one, allowing the slaves to verify the integrity of the buffered messages [75]. As a result of this approach, the *correctionField* value cannot be updated when a PTP message passes a transparent clock. Both security processing (i.e., immediate and delayed security processing) can be included in one PTP message by attaching two AUTHENTICATION TLVs. The first AUTHENTICATION TLV can be used to authenticate the PTP messages sent by the originator, while the second AUTHENTICATION TLV is used to protect the mutable fields, e.g., *correctionField.* In this case, the mutable parts of PTP messages are authenticated by the immediate security processing (i.e., a TC can update the *correctionField value*) while the other parts are authenticated by the delayed security processing [7]. However, the AUTHENTICATION TLV has the same limitation as IPsec and MACsec, i.e., it is vulnerable to delay attacks and the advanced internal attacker [23], [38].

3.5.2 Infrastructure Enhancements

A. Multiple Paths

The variability of network latencies presents a challenge, as the accuracy of clock synchronization relies on the symmetry and steadiness of propagation delays in the uplink and downlink direction between the master clock and the slave clock. A computer network is prone to path asymmetry and variable network latency, depending on the nature of the underlying network [76], [77]. Multiple network paths can improve fault-tolerance and PTP performance by providing multiple PTP message paths between a master and its slaves [78], [79]. Such means also improve security, as it complicates MitM attacks [24].

Multiple paths can be achieved via VLAN [24] or via High-availability Seamless Redundancy (HSR) in combination with the Parallel Redundancy Protocol (PRP) [25], [80]. Han [81] and Neyer [82] show that multiple packet propagation paths can detect delay attacks by comparing the offset values computed by each slave port. However, an attacker can manipulate all paths and therefore desynchronize the slave clocks.

B. Redundant Grandmaster

Multiple redundant grandmasters can be utilized to compensate for byzantine failures, where the master clock provides an incorrect time reference [26]. Here the redundant grandmasters compare the active master's time with their own time [83]. If the computed difference exceeds a particular value, one of the passive grandmasters becomes the main grandmaster. Again, an attacker can target all redundant grandmasters, and therefore the attack remains unnoticed.

C. Protocol Redundancy

Multi-time protocol synchronization of PTP slaves provides another mechanism to prevent byzantine failures [27]. Here a slave uses NTP in parallel with PTP and determines offsets from multiple stratum time sources. Their median value is compared against the measured PTP offset, and the former is used to correct the local clock if the difference between the two values is larger than a threshold. However, the NTP and PTP protocols are vulnerable to the same attack strategies such as packet content modification attack, replay attack, and DoS attack.

D. De-militarized Zone

The De-Militarized Zone (DMZ) is a method of creating a semi-secure network that works as the first line of defence to secure the internal infrastructure of an organization from external attackers [84]. DMZ is useful for networks that need to share devices or endpoints (e.g., web servers) publicly. As such, it does not protect against an internal attacker who is already inside a trusted network.

3.5.3 IEEE 1588 Annex P

IEEE 1588-2019 introduced a new security extension called Annex P, which retains backward compatibility to previous PTP versions. It addresses security in four prongs as follows [82], [85]:

- Prong A (PTP Integrated Security Mechanism) describes a type-length-value (TLV) extension for message authentication using symmetric encryption. There are two different operating modes supported; (1) immediate security processing that relies on a shared group key (2) delayed security processing that is supported by the Timed Efficient Stream Loss-Tolerant Authentication (TESLA) protocol. This prong can be classified as cryptographic protocol security as described above.

- Prong B (PTP External Transport Security Mechanisms) describes existing external security mechanisms, including IPsec and MACsec.

- Prong C (Architecture Guidance) describes an overview of architectural security measurements, namely redundancy. With redundancy, an attacker must compromise multiple points to manipulate the time synchronization. IEEE 1588 defined three types of redundancy: redundant time system, redundant grandmaster, and redundant paths [86]. This prong is similar to infrastructure enhancements, as already described in this section.

- Prong D (Monitoring and Management Guidance) describes a monitoring mechanism to observe the PTP behaviour to detect (rather than deflect) a potential attack such as a DoS attack via monitoring slave clock parameters, including offset and delay measurements.

Please note that this research focuses on prevention means rather than detection.

3.6 PTP Attack Strategies and Implementations

Reference [87] derives a set of conditions from securing the PTP network under the assumption that the attacker does not have access to cryptographic keys. However, previous research has shown that MACsec, IPsec, and Annex K only protect against certain external attacks, but not against internal attacks, as any trust token that either identifies the origin or guarantees the security/integrity of PTP messages may be compromised [23], [88]. Consequently, Prong A and B of PTP 2.1 (IEEE 1588-2019) Annex P do not provide protection against internal attacks.

This section will extend these results by providing a similar vulnerability analysis of infrastructure enhancements, including Annex P Prong C in the presence of an internal attacker. This is complemented by an assessment of possible internal attack implementations and their impact/severity in the presence of cryptographic protocols or infrastructure enhancements.

Table 3-1: PTP Attack Strategies

NO	Attack Strategy
1	Packet Content Manipulation Attack
2	Packet Removal Attack
3	Packet Delay Manipulation Attack
4	Time Source Degradation Attack
5	Master Spoofing Attack
6	Slave Spoofing Attack
7	Replay Attack
8	BMCA Attack
9	Denial of Service Attack

Table 3-1 summarizes different PTP attack strategies as outlined in [23] and further distinguishes between master spoof attacks and slave spoof attacks, as further described in this section.

Figure 3-2 shows a PTP network model that incorporates all relevant network elements (i.e., routers, secured and unsecured network segments) and PTP hardware elements (GM, TC, BC, and OC). For each element, it shows what attacker type, as described in Section 3.3, can use what attack strategy as listed in Table 3-1. Here, yellow and red stars denote if the strategy can or cannot be averted by at least one of the PTP security safeguards as listed in Section 3.5. In other words, Figure 3-2 illustrates the attack types that can b launched in each PTP node and the possibility of preventing such attacks using the existing security extensions. Note that infrastructure enhancements are not explicitly integrated into this diagram; instead, multi-paths redundancy is referred to [24] and [25], while protocol redundancy is referred to [27].

All considered attacks must be persistent (i.e., continuously manipulate PTP traffic for the duration of the attack) in order to have the desired effect. Once an attack is terminated, normal PTP operation will resume, and affected slave clocks will slowly resynchronize again.

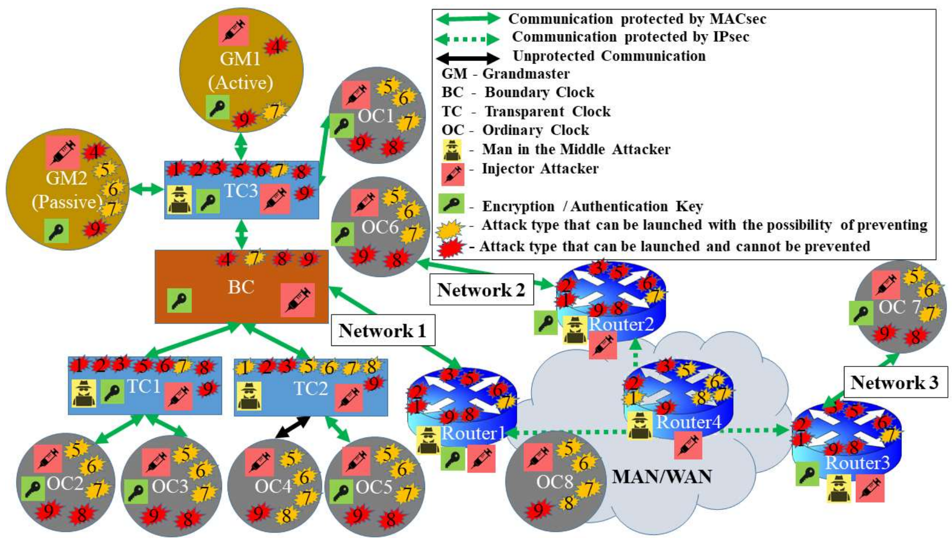
Communication protected by MACsec
Communication protected by IPsec
Unprotected Communication
GM - Grandmaster
BC - Boundary Clock
TC - Transparent Clock
OC - Ordinary Clock
- Man in the Middle Attacker
- Injector Attacker
- Encryption / Authentication Key
- Attack type that can be launched with the possibility of preventing
- Attack type that can be launched and cannot be prevented
GM1 (Active)
GM2 (Passive)
TC3
TC1
TC2
BC
OC1
OC6
OC2
OC3
OC4
OC5
OC8
OC 7
Network 1
Network 2
Network 3
Router1
Router2
Router3
Router4
MAN/WAN

Figure 3-2. Single-path PTP Network Model with Attacker Type as listed in Table 3-1.

3.6.1 Packet Content Manipulation Attack

A. Attack Overview:

In a packet content manipulation attack, a *MitM attacker* manipulates suitable fields of time protocol packets in transit, thereby manipulating the clock synchronization of all clocks downstream or making them go into free-running mode [61].

B. Attack Implementation:

In Figure 3-2, Router1, Router2, Router3, Router4, TC1, TC2, and TC3 are all suitable points from which to launch a packet content manipulation attack (red and yellow stars number 1); for example:

1. TC1 has access to the network security key(s). So, an attacker who resides on TC1 can launch an advanced internal MitM attack on OC2 and OC3 by intercepting and changing *all* Sync/Follow_Up messages as follows:

    a) For every Sync/Follow_Up message, add a fixed or an incremental error value to the *originTimestamp* / *preciseOriginTimestamp* or *correctionField* fields. Since PTP clients disregard clock offset calculations beyond a certain threshold and go into free-running mode instead, such values must be carefully selected. Likewise, the sudden termination of such an attack would cause the slave to detect the cumulated error.

    b) Change the *versionPTP* value from (version) 2 to (version) 1; OC2 and OC3 won't support the obsolete older version of PTP and will eventually go into free-running mode.

    c) PTP clocks can only communicate with each other if they share the same *domainNumber* value. Changing this parameter will cause them to discard all synchronization messages they receive and eventually go into free-running mode.

2. TC2 is a suitable (simple internal MitM) attack point from which to perform a packet content manipulation attack in the absence of a cryptographic security protocol. As a result, OC4 and OC5 are compromised.

3. Router 4 is a suitable (external MitM) attack point from which to perform a packet content manipulation attack in the absence of a cryptographic security protocol. As a result, OC6 and OC7 are compromised.

PTP safeguards have the following impact:

1. Cryptographic security protocols can prevent the simple internal attacker (i.e., yellow stars number 1), but the advanced internal MitM attacker (i.e., red stars number 1) has legitimate network access and can perform such an attack.

2. Multiple paths cannot prevent such attacks, especially if the manipulated packet arrives faster than the others in the case of the HSR approach or if all the intermediate nodes were attacked by a MitM attacker.

3. A redundant GM cannot mitigate packet content manipulation, as a MitM attacker can manipulate all packets regardless of the sender. In other words, if the passive GM2 (see Figure 3-2) recognizes that it has better accuracy than the active GM1, it will become the active GM, but the attacker can manipulate its messages in the same manner.

4. With protocol redundancy, NTP messages are also vulnerable to a packet content manipulation attack by a MitM attacker.

3.6.2 Packet Removal Attack

A. Attack Overview:

In a packet removal attack, protocol packets are intercepted and removed by a *MitM attacker*, which again either leads to clock synchronization errors of all clocks downstream or makes them go into free-running mode. An internal (and an external) MitM attacker can perform such an attack, as it only requires them to reside in an intermediate node, regardless of whether the attacker has access to the authentication/encryption keys [61].

B. Attack Implementation:

Most of the intermediate nodes (Router1, Router2, Router3, Router4, TC1, TC2, and TC3) are points from which to launch a packet removal attack (red stars number 2, as shown in Figure 3-2); for example:

1. TC1 (advanced internal MitM attack point) in Figure 3-2 can selectively intercept and remove PTP messages (i.e., delay request messages only), causing degradation of OC2 and OC3 synchronization. TC1 also can remove *all* PTP messages, forcing OC2 and OC3 to go into free-running mode.

2. TC2 (simple internal MitM attack point) in Figure 3-2 can launch a similar attack to OC4 as in example 1, but since it cannot distinguish between encrypted PTP packets and other network traffic, it would randomly remove messages to/from OC5. TC2 would certainly not block all OC5 traffic, as this could be easily spotted by the slave. Instead, packets have to be removed more subtly so that the TCP retransmission mechanism compensates for packet loss of other affected network services.

3. Router4 (external MitM attack point) in Figure 3-2 can randomly or systematically drop PTP messages to/from OC6/OC7, causing either a

slave clock synchronization degradation or a switch into free-running mode.

PTP safeguards have the following impact:

1. Cryptographic security protocols cannot protect against packet loss.

2. Multiple paths can mitigate such an attack unless all intermediate nodes are simultaneously manipulated by a MitM attacker.

3. A redundant GM cannot mitigate such an attack, as it would be targeted as well once it is active.

4. In protocol redundancy, NTP messages are also vulnerable to a packet removal attack by a MitM attacker.

3.6.3 Packet Delay Manipulation Attack

A. Attack Overview:

IEEE 1588 requires symmetric network delays between master and slave in order to achieve optimal clock synchronization [5]. If the time propagation delays of a sync message and its corresponding delay request message are not equal, the slave clock will calculate an inaccurate offset [89]. A packet delay manipulation occurs when the transmission of protocol packets is purposely delayed by a *MitM attacker* [23]. As a result, all clocks downstream from the attacker location will be manipulated. An internal (and even external) MitM attacker can perform such an attack, as it only requires them to reside in an intermediate node without having access to the authentication/encryption keys used [61].

B. Attack Implementation:

A packet delay manipulation attacker can use an intermediate node to selectively hold PTP packets for a certain time before forwarding them to their destination. Such an attack must happen in one direction only (uplink or downlink) to produce an asymmetric delay between the master and slave.

Large instantaneous delays cause large slave clock offset errors and are likely to be picked up by PTP slave daemons, so incremental delay over time must be used.

Most of the intermediate nodes (Router1, Router2, Router3, Router4, TC1, TC2, and TC3) are points from which to launch a packet delay manipulation attack (red stars number 3, as shown in Figure 3-2), for example:

1. TC1 (advanced internal MitM attack point) can repeatedly delay *all* Sync or Delay_Req messages, resulting in an asymmetric path delay between the master and its slaves. As a result, there is a degradation of the synchronization of both OC2 and OC3.

2. TC2 (a simple internal MitM attack point) can similarly attack OC4 and OC5 by delaying *all* packets that go towards or come from these endpoints.

3. Router4 (external MitM attack point) can launch a packet delay manipulation attack on OC6 and OC7.

PTP safeguards have the following impact:

1. Cryptographic protocols do not guarantee that messages will be delivered to their destinations in a fixed or deterministic time.

2. Multiple paths can mitigate such an attack unless the intermediate nodes along all network paths delay PTP packets synchronously.

3. The same applies to protocol redundancy, where NTP packets (coming from multiple time servers) are synchronously delayed on their way to the host.

4. GM redundancy cannot address this problem.

3.6.4 Time Source Degradation Attacks

A. Attack Overview:

Time source attacks occur when an *internal injector attacker* compromises the precise time source of the master clock, i.e., GM or BC, as shown in Figure 3-2. Subsequently, all clocks downstream are manipulated.

B. Attack Implementation:

GM1, GM2, and BC are targets of such an advanced internal injector attack, for example:

1. Since GPS is usually used as a network time reference, an attacker can jam or spoof the satellite signals, causing the grandmaster clock to become an incorrect reference time [61].

2. An attacker can target GM1 (the active GM in Figure 3-2) by manipulating its firmware. Subsequently, GM1 provides inaccurate timestamps to all PTP nodes causing degradation of synchronization.

3. An attacker can manipulate the BC in the same manner as in example 2. As a result, all PTP slave clocks downstream will be manipulated.

PTP safeguards have the following impact:

1. Cryptographic security protocols cannot prevent the degradation of the time source.

2. Multiple paths do not provide a solution either since the attack occurs at the endpoint of a network (the BC will act as an endpoint for all PTP messages).

3. All redundant active/passive GMs can be simultaneously compromised.

4. Protocol redundancy can mitigate such attacks unless NTP synchronization is interrupted or manipulated as well.

3.6.5 Master Spoofing Attack

A. Attack Overview:

In a master spoofing attack, an *injector attacker* is depicted as a legitimate master by generating and transmitting PTP packets [61]. The attacker impersonates the master clock and distributes false synchronization messages, causing all clocks downstream to be compromised.

B. Attack Implementation:

All non-master PTP nodes are suitably located to launch a master spoofing attack (yellow and red stars number 5, as shown in Figure 3-2); for example:

1. TC1 (advanced internal injector attack point) in Figure 3-2 can masquerade as the master BC by using its IP address, continuously generate manipulated Sync/Follow_Up packets, and send them to OC2/OC3.

2. OC1 (advanced internal injector attack point) can similarly masquerade as an active GM (GM1) and send manipulated Sync/Follow_Up packets to BC. As a result, BC as well as all nodes downstream (OC2 to OC7), will be affected. Note that this attack can only occur if no cryptographic security (i.e., MACsec) is applied.

3. TC2 (a simple internal injector attack point) can continuously send spoofed Announce/Sync messages to OC4 and OC5 if no cryptographic security protocol (i.e., MACsec) is used. As a result, OC4 and OC5 will be compromised [90].

4. Router4 or OC8 (external injector attack points) can similarly attack networks 2 and 3 if no cryptographic security protocol (i.e., IPsec) is used. As a result, OC6 and OC7 will be manipulated.

Note that an attacker can send malicious messages from an active GM or BC as a time source degradation attack rather than a master spoofing attack.

PTP safeguards have the following impact:

1. Cryptographic security protocols cannot prevent such an attack if the spoofed messages use the same security keys and come from a trusted intermediate node (red stars number 5, as shown in Figure 3-2).

2. In the multiple paths approach, all intermediate nodes can be simultaneously attacked and send orchestrated spoofed master messages.

3. A redundant GM cannot mitigate such attacks, as the attacker can spoof any active GM.

4. With protocol redundancy, NTP can mitigate such attacks if it is not otherwise manipulated.

3.6.6 Slave Spoofing Attack

A. Attack Overview:

In a slave spoofing attack, an *injector attacker* masquerades as the target (a legitimate intermediate or a slave clock) and transmits delay request messages to the master sooner than the attacked node. The master responds to the spoofed node, which in turn calculates its delay using incorrect timestamps [61]. Note that if the slave receives a spoofed delay response message with a sequence number that does not match its last delay request message, the response message will be discarded, and this attack attempt fails.

B. Attack Implementation:

All PTP nodes (except the active GM and the BCs) are suitably located to launch slave spoofing attacks (yellow and red stars number 6, as shown in Figure 3-2); for example:

1. Router1 (an advanced internal injector attack point) can continuously create spoofed delay request packets using OC6's or OC7's IP address and their expected sequence numbers and send them to BC. As a result, OC6 and OC7 will be manipulated because of the asymmetric uplink/downlink path between the master and the slave.

2. TC2 or OC4 (simple internal injector attack points) can similarly attack OC5, but only if no cryptographic security protocol (i.e., MACsec) is used.

3. Likewise, Router4 or OC8 (external injector attack points) can attack OC6 and OC7, as long as no cryptographic security protocol (i.e., IPsec) is used.

PTP safeguards have the following impact:

1. Cryptographic security protocols cannot prevent such an attack if the spoofed messages use the same security keys and come from a trusted intermediate node (red stars number 6, as shown in Figure 3-2).

2. In the multiple paths approach, multiple intermediate nodes along all paths between the master and a slave can be simultaneously manipulated and send spoofed delay request messages to the master in order to produce an asymmetric delay.

3. A redundant GM cannot mitigate such an attack as there is no reason for the passive GM to take action.

4. In protocol redundancy, NTP can mitigate such an attack as long as it is not separately manipulated.

3.6.7 Replay Attack

A. Attack Overview:

In a replay attack, an *internal* (or even *external*) *injector/MitM attacker* continuously records protocol packets and transmits them later without modification.

B. Attack Implementation:

All network nodes are suitably located to launch a replay attack (yellow stars number 7, as shown in Figure 3-2); for example:

1. GM2 (advanced internal injector attack point) in Figure 3-2 can replay multicast Sync/Follow_Up messages from GM1. As a result, all nodes downstream will be compromised.

2. OC4 (simple internal injector attack point) can replay multicast Sync/Follow_Up messages from BC and replay them later to OC5. As a result, OC5 will be manipulated.

3. Router4 or OC8 (external injector attack points) can similarly compromise OC6 and OC7.

PTP safeguards have the following impact:

1. All cryptographic security protocols have a replay protection mechanism (based on a sequence number field), protecting against such an attack [61, 91].

2. With the multiple paths approach, intermediate nodes along all paths between the master and a slave can be simultaneously manipulated and record and resend later Sync/ Follow_Up messages to the slaves in

order to manipulate the time synchronization. Moreover, the replay attack can also be performed by an injector attacker rather than a MitM (i.e., a different slave), which cannot be avoided by the multiple paths approach.

3. A redundant GM cannot mitigate such attacks as the attacker can record and replay packets from any active GM.

4. In protocol redundancy, NTP is also vulnerable to replay attacks.

3.6.8 BMCA Attack

A. Attack Overview:

In a BMCA attack, an *advanced internal attacker* guides other network clocks to elect it as the best master by tampering with the BMC algorithm. Here the BMCA attacker does not fake its identity but tampers with the master election process by advertising exaggerated and incorrect clock characteristics [61] and – once elected – manipulates the synchronization of all slave clocks.

B. Attack Implementation:

All PTP nodes are suitably located to host a BMCA attack (yellow and red stars number 8, as shown in Figure 3-2), for example:

1. OC1 (advanced internal injector attack point) becomes a rogue master. It subsequently sends continuously crafted announce messages that carry the best clock attributes (i.e*., priority1, clockClass, clockAccuracy, offsetScaledLog-Variance, priority2,* and *clockIdentity*) of the entire network to tamper with the BMC algorithm, as explained in [5]. As a result, all nodes downstream (OC2 to OC7 and BC) will rely on this compromised time reference.

2. OC4 (a simple internal injector attack point) can launch this attack if no cryptographic security protocol is present. As a result, all nodes

downstream (BC and OC1 to OC7 excluding OC4) will rely on an inaccurate time source.

3. Router4 or OC8 (external injector attack points) can launch this attack if no cryptographic security protocol is present. As a result, OC1 to OC7 and BC will be manipulated.

PTP safeguards have the following impact:

1. Cryptographic security protocols can only stop an external or simple internal injector attacker. An advanced internal injector attacker can stealthily perform such an attack.

2. The multiple paths approach cannot prevent this manipulation as the attacker can infiltrate an endpoint to become the rogue grandmaster.

3. A redundant GM cannot mitigate such an attack, assuming that a rogue master *always* has better clock attributes than the other grandmasters.

4. In protocol redundancy, NTP can mitigate such an attack unless it is separately compromised or disabled.

3.6.9 Denial of Service Attack

A. Attack Overview:

A denial-of-service attack can be initiated by an *injector attacker*. There are many potential Layer 2 and Layer 3 DoS or DDoS attacks, such as MAC flooding, ARP spoofing, and IP spoofing, which compromise the target's availability and timely execution of the PTP protocol [61]. In addition, an attacker can utilize cryptographic execution attacks by sending bogus IPsec or MACsec packets, which cause a high CPU load when the receiver’s cryptographic engine tries to verify the validity of these packets. This attack can be launched by any internal (and even external) attacker [61] and forces all affected clocks to go into free-running mode.

B. Attack Implementation:

All PTP nodes are suitably located to launch a DoS attack (red stars number 9, as shown in Figure 3-2; for example:

1. OC2 (advanced internal injector attack point) in Figure 3-2 performs an ARP spoofing attack to bind its MAC address to OC3's IP address. As a result, OC3 cannot receive PTP messages and eventually goes into free-running clock mode.

2. OC4 (a simple internal injector attack point) can launch a DoS attack by continuously transmitting protocol packets using a fake security key to OC5, which causes a high utilization of OC5's cryptographic engine. As a result, OC5 cannot process other PTP messages in time and goes into free-running clock mode.

3. Router4 or OC8 (external injector attack points) can launch a DoS attack as described in examples 1 and 2 in order to manipulate all slaves in networks 2 and 3. As a result, OC6 and OC7 will go into free-running mode.

PTP safeguards have the following impact:

1. Cryptographic security protocols cannot prevent but may even support (D)DoS-style attacks, as shown in example 2.

2. The multiple paths approach fails if all interfaces of an endpoint are targeted.

3. A redundant GM cannot address this issue, as the attacker aims to compromise the slave availability rather than the existing GMs.

4. In protocol redundancy, NTP is also vulnerable to a DoS attack.

Table 3-2: PTP Attack Strategies and their Impacts

NO	Attack Strategy	Attack Impact	Impact Scope	Internal Attack Type (the RAG rating is used to highlight the severity)			
				Cryptographic Security	Multiple Paths	Redundant GM	Protocol Redundancy
1	Packet Content Manipulation	Clock Manipulation or Clock free-running	Based on attacker location All/subset/single slave(s)	Advanced MitM	Simple MitM	Simple MitM	Simple MitM
2	Packet Removal	Clock Manipulation or Clock free-running	Based on attacker location All/subset/single slave(s)	Simple MitM	Simple MitM	Simple MitM	Simple MitM
3	Packet Delay Manipulation	Clock Manipulation	Based on attacker location All/subset/single slave(s)	Simple MitM	Simple MitM	Simple MitM	Simple MitM
4	Time Source Degradation	Clock Manipulation	All slaves	Advanced Injector	Simple Injector	Simple Injector	Simple Injector
5	Master Spoofing	Clock Manipulation	Based on attacker location All/subset/single slave(s)	Advanced MitM	Simple (Injector/ MitM)	Simple (Injector/ MitM)	Simple (Injector/ MitM)
6	Slave Spoofing	Clock Manipulation	Single slave	Advanced MitM	Simple (Injector/ MitM)	Simple (Injector/ MitM)	Simple (Injector/ MitM)
7	Replay	Clock Manipulation	Based on attacker location All/subset/single slave(s)	N/A	Simple (Injector/ MitM)	Simple (Injector / MitM)	Simple (Injector/ MitM)
8	BMCA	Clock Manipulation	All slaves	Advanced (Injector/ MitM)	Simple (Injector/ MitM)	Simple (Injector/ MitM)	Simple (Injector/ MitM)
9	Denial of Service	Clock free-running	Based on attacker location All/subset/single slave(s)	Simple (Injector/ MitM)	Simple (Injector/ MitM)	Simple (Injector/ MitM)	Simple (Injector/ MitM)

3.7 Experimental Validation of PTP Attacks

Table 3-2 shows the potential impact of the various attack strategies outlined in Section 3.6. From a slave clock perspective, the most effective attack that directly manipulates all clock synchronization downstream from the physical location of the attacker in the network (i.e., slaves that must go through the manipulated node to communicate with the master) is represented by the label "Clock Manipulation". In contrast, "Clock free-running" indicates that all downstream clocks go into free-running (non-PTP synchronized) mode, which is usually not picked up by a host operating system and causes a slow desynchronization over time, as outlined in Section 3.1. This table also shows the various attack strategies (i.e., simple/advanced attack, and MitM /injector)

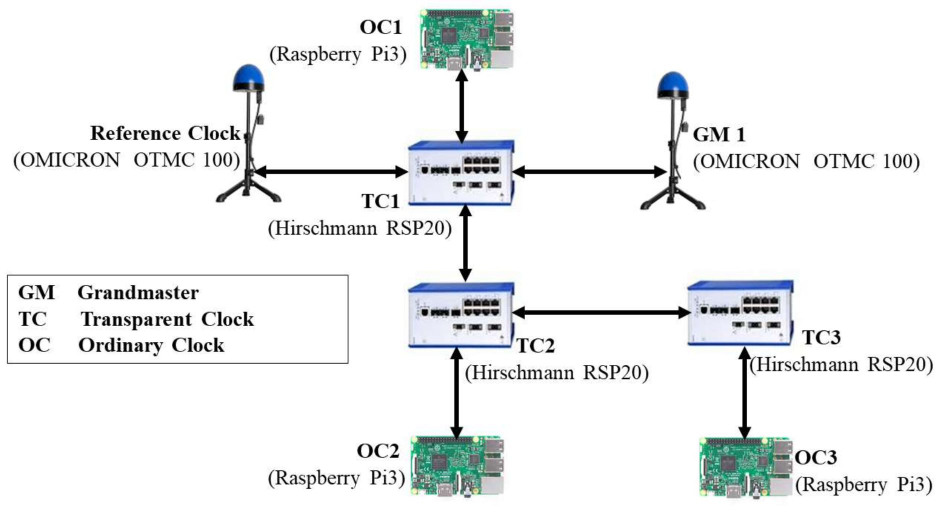

Figure 3-3. Testbed.

that can be applied for each strategy, and their severity using the RAG rating: The red colour indicates that the PTP safeguards, as listed in Section 3.5 do not provide protection, while the yellow colour indicates that a given attack strategy can be detected. The green colour indicates that the attack can be averted by the PTP safeguard.

Continuing on from this work, a testbed was set up to simulate and experimentally validate some attack strategies (i.e., time source degradation, packet content manipulation, packet delay manipulation, replay, and DoS attack) that have a different impact on PTP slave(s). The testbed (see Figure 3-3) consists of three slaves (OC - Raspberry Pi 3 model B), three transparent clocks (TC - Hirschmann RSP20), one grandmaster clock (GM - OMICRON OTMC 100), and one reference clock (OMICRON OTMC 100). The experiments were done using the PTP slave daemon PTPd. The reference clock provided an accurate time reference (similar to the grandmaster clock in normal operation - no attack), but it does not participate in the time synchronization process, and it is assumed to be secure and outside the attack scope. It also collected timestamps from all other clocks in the network and subsequently computed the time drift of these clocks by calculating the difference between its timestamps and the timestamps received from the other slaves minus the time taken to transfer these timestamps from the slaves to the reference clock. All devices in the network are connected via CAT5e Ethernet cables with a data rate of 1000 Mbps.

In detail, an attacker has the following options:

1. Desynchronize all PTP clocks downstream via a BMCA- or time source degradation attack: These approaches exploit the grandmaster's role as a time source and propagate an inaccurate time reference to all other clocks in a network. Like all the other attacks presented, it has to be persistent to continuously manipulate PTP clocks. This attack was performed by attaching a new OC device to the network that advertises itself as the best clock. Figure 3-4 shows the possible impact of such an attack on PTP slaves. In this experiment, master timestamps are given an increasing negative offset of 100 µs per second before being circulated to the slaves via Sync messages. Similarly, the BMCA attack would have the same impact on the slave clock when the difference of the clocks frequencies (the new master and the reference clock) introduces a 100 µs time offset per second.

2. Manipulate a subset of PTP clocks by using packet content manipulation or packet delay manipulation strategies: Figure 3-5 shows the impact of a packet content manipulation attack when an attacker

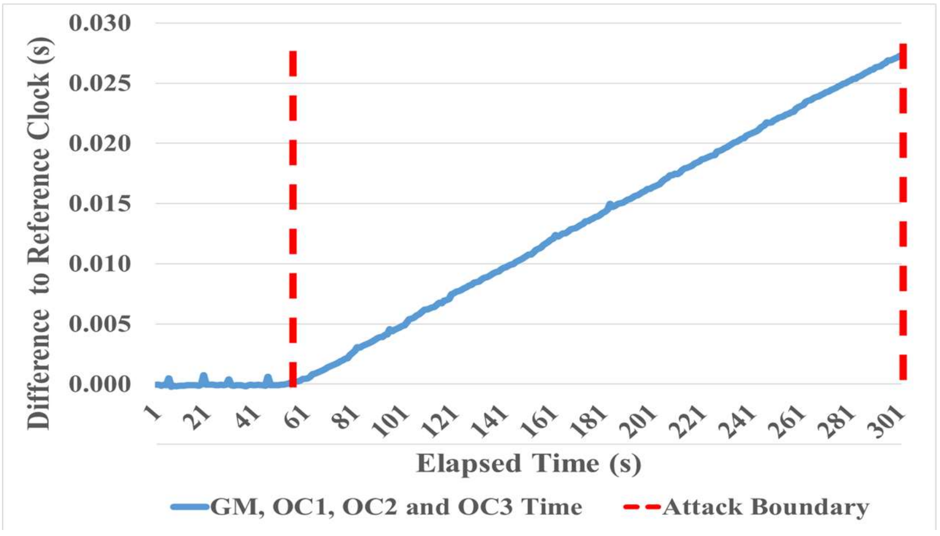

Figure 3-4. Impact of BMCA or time source degradation attack on slaves.

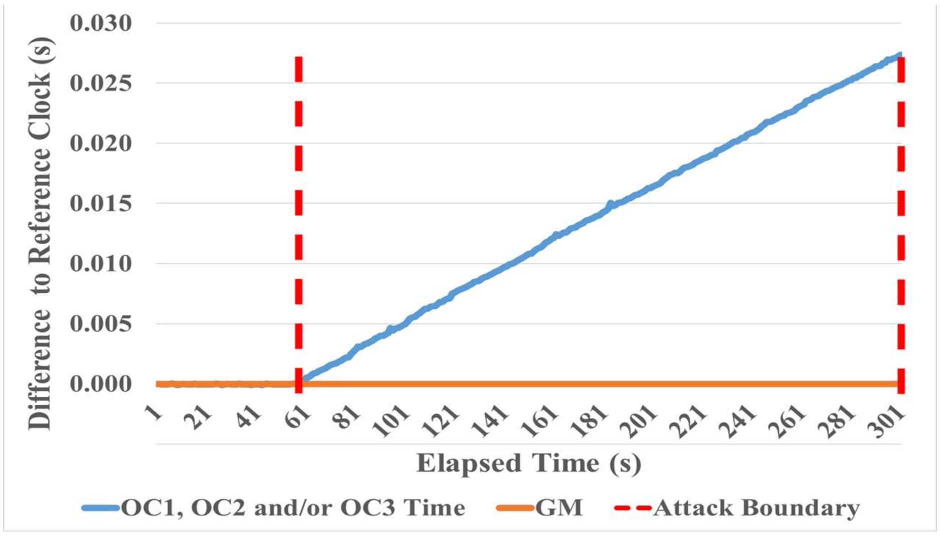

Figure 3-5: Impact of packet content manipulation attack.

intercepts either Sync or Follow_Up messages and decreases their timestamps by a value that is incremented by 100 µs per second. Similarly, Figure 3-6 shows the impact of an asymmetric delay attack when an attacker intercepts each Sync message and holds it for 20 ms before forwarding it to its destination. Since many applications require a smooth and monotonically increasing time base, PTP daemons were designed to take this feature into account, especially when the time error introduced is within the preconfigured threshold. Figure 3-6 showed that the PTP daemon increased the slave clock frequency gradually, starting from 60 s into the experiment, due to the introduced asymmetric delay, to meet the master clock frequency. At the 100th

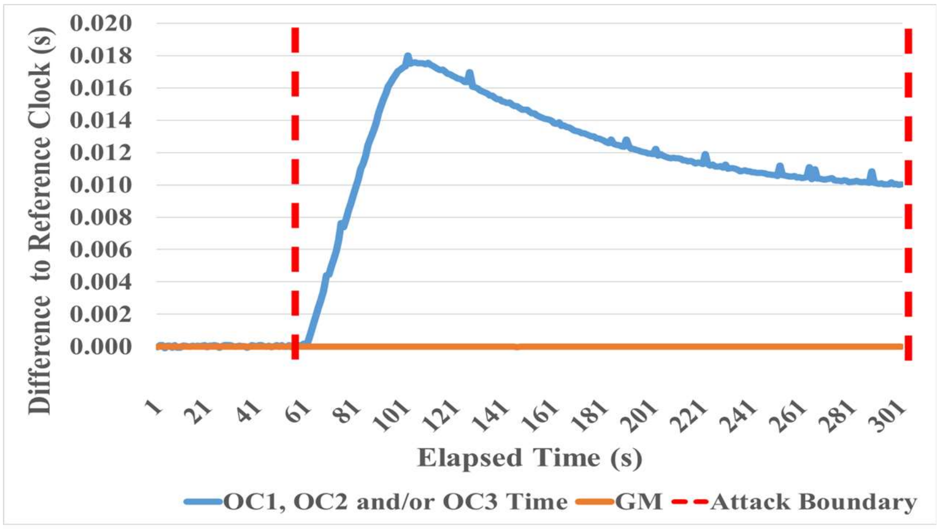

Figure 3-6: Impact of asymmetric packet delay manipulation attack.

second, the PTP daemon realized that the slave clock frequency became faster than the master clock frequency and subsequently decreased the slave clock frequency gradually until both clock frequencies were close to each other. This experiment was conducted by adding a network impairment emulator device between the GM and TC1 to affect all slaves or between TCs to affect some slaves (see Figure 3-3). The emulator device is able to intercept and delay/manipulate the content of specific packets (i.e., Sync/Follow_Up messages) and then forward them to their destination.

3. Interfere with the clock synchronization process via master spoofing or replay attacks: With each of these, a slave may receive valid and fresh sync messages as well as spoofed or replayed sync messages over time, making it swing between a synced and an unsynced state to the master. Figure 3-7 shows the impact of a replay attack when an attacker records the last Sync/Follow_Up messages sent by the master every 5 seconds and replays them to their destination. Such an attack can be performed by any of the existing slaves. It is worth noting that the PTP daemon in this experiment applied a clock reset instead of gradually adjusting the clock frequency because the introduced time error exceeded the preconfigured threshold (i.e., if the time error is greater than one second).

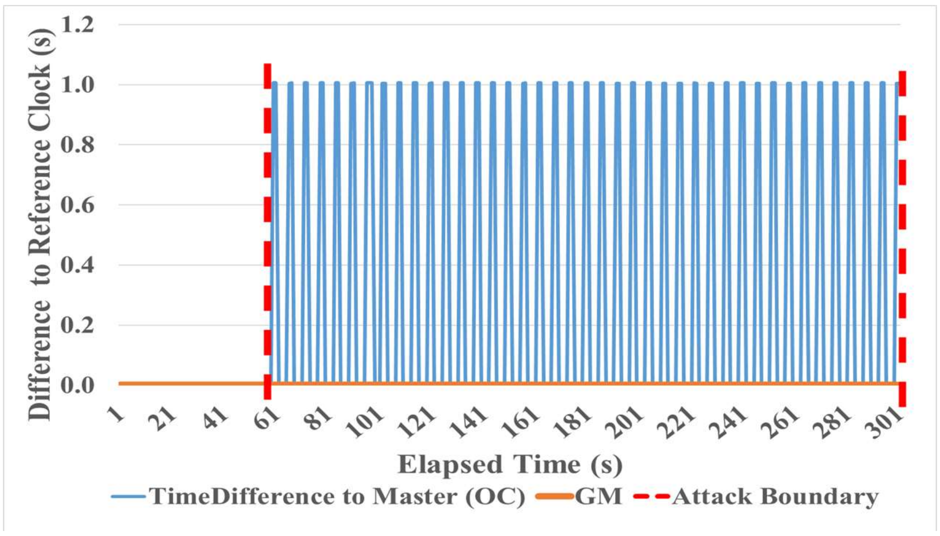

Figure 3-7: Impact of replay attack.

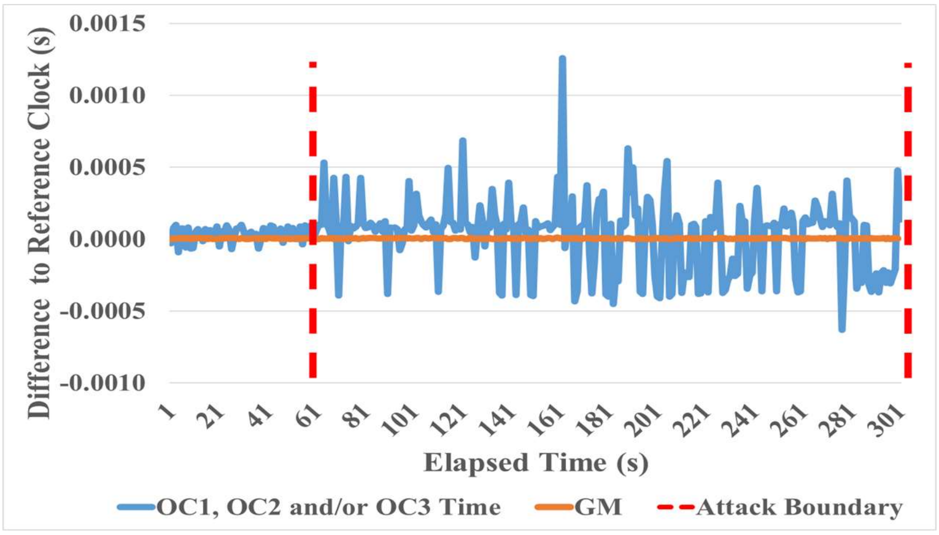

Figure 3-8. Impact of DoS attack/packet removal attack.

4. Target a single PTP clock and manipulate it by using the slave spoofing attack: This has the lowest impact on a PTP network.

5. Launch a DoS attack or packet removal attack that makes affected slave clocks go into free-running mode: A denial of service can be relatively easily detected by a network or a slave, as it affects all network services. Figure 3-8 shows the impact of a DoS attack when an attacker prevents slave(s) from receiving the PTP messages. Here, the PTP slave clock will be in free-running mode, and subsequently, its frequency will be unstable, making its time ahead of and sometimes behind the reference clock. The emulator device is used here again to intercept and remove the Sync message, preventing them from being received by the slaves, which subsequently go into free-running mode.

3.8 Conclusion

This chapter investigates the problem of advanced persistent threats to PTP networks. It distinguishes between attack strategies and attacker types as described in RFC7384 but further distinguishes between the spoofing and time source attack, the simple internal attack, and the advanced internal attack. This research takes into account the new security features of the emerging Annex P.

Our analysis shows that an internal attacker has a range of methodologies to compromise the time synchronization of PTP slaves, ranging from slave spoofing that targets individual slaves to BCMA attacks that compromise all endpoints in a network. While prior research has validated that cryptographic security via MACsec or IPsec is a blunt instrument against most internal attacks, the previous sections have shown that infrastructure or protocol redundancy does not provide viable protection either. Moreover, all PTP infrastructure components (GM, BC, and TC) and even slave clocks (e.g., ordinary personal computers) can host an attacker, as shown in Figure 3-2. This makes the comprehensive protection of a PTP network against malware infiltration, as for example exercised by Stuxnet, a very difficult task .

While this chapter also presents some experimental findings with regard to attack implementation and their impact, the next chapter will explore these threats in more depth. We are particularly interested in the exact behaviour of different PTP client daemons in the presence of the aforementioned attacks using different parameters. These results will help us to reach our long-term goal, a PTP intrusion detection system based on the aforementioned trusted supervisor node to protect time synchronization networks against APTs.

Chapter 4

Cyber Attacks on Precision Time Protocol Networks—A Case Study

The work outlined in this chapter was published in:

W. Alghamdi and M. Schukat, "Cyber Attacks on Precision Time Protocol Networks—A Case Study," *Electronics,* vol. 9, p. 1398, 2020/08/28 impact factor 2.397.

This chapter will build on the Chapter 3 findings and present the results of the experimental proof-of-concept implementations of the most far-reaching APT attacks identified, therefore making the following contributions: (1) an analysis of the time correction algorithms (and their characteristics/weaknesses) implemented in two popular PTP daemons (i.e., PTP4l and PTPd) using two different slave types (i.e., Galileo and Raspberry Pi), (2) prototyping of a PTP hardware "sandbox" testbed that is able to perform various of man in the middle (MitM) attacks, namely delay, time source, and packet modification attacks, (3) determining suitable attack patterns and parameters, and (4) analyzing the effect of such attacks on slave clock synchronization, operation and behaviour. The MitM implementation goes beyond existing research in [81], [82], and [8] as this thesis simulates the APT behaviour using a hardware-based programmable MitM that can manipulate PTP packets and change the attack parameters dynamically over time. In doing so, it determines suitable attack patterns and parameters to compromise the time synchronization covertly. This chapter is structured as follows: Section 4.1 provides a summary of PTP attack strategies. Section 4.2 describes an experimental testbed based on different PTP devices (i.e., slaves, switches, grandmaster—GM and transparent clock—TC), presents the MitM device, and characterizes different slave clocks. The experimental results of packet propagation attacks (i.e., delay and transparent clock attacks) and time reference attacks will be shown in Section 4.3, followed by a conclusion in Section 4.4. This chapter answers the second research question as outlined in Chapter 1.

4.1 Summary of PTP Attack Strategies

PTP clock synchronization assumes that the network delay between slaves and their master is symmetric while relying on a single accurate time reference and correctly circulated timestamps. This makes PTP susceptible to a range of attack strategies as described in the previous chapter, [92], [93]:

- Systematically modify PTP message content, i.e., timestamps *T1*, *T4*, *C1*, *C2* and/or *C3*, in transit via a man-in-the-middle (MitM) attacker located in an intermediate node, such as a router, switch, or transparent clock. For example, Figure 4-1 shows the manipulation of the *correctionField* segment (that contains the transparent clock residence time) of PTP messages;

- Selectively delay PTP message propagation by a MitM to induce an asymmetric uplink/downlink delay;

- Directly manipulate the time reference by compromising the grandmaster clock or by introducing a Byzantine master clock. The latter requires an ordinary clock to become a rogue master (a clock that

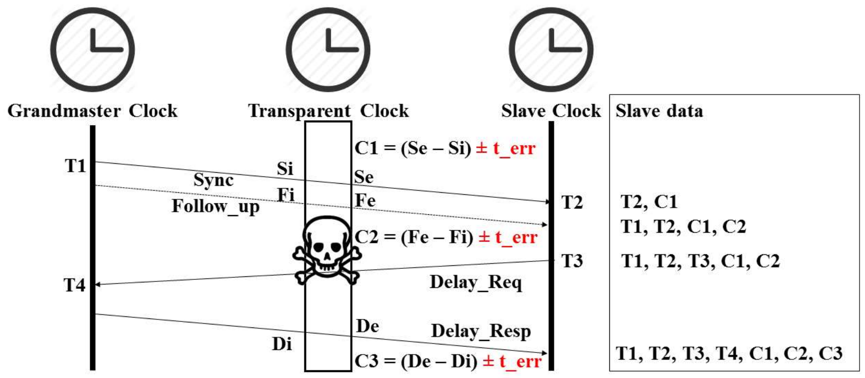

Figure 4-1. Manipulated residence time in transparent clock.

pretends to be the best in the network) by circulating announce messages with overrated clock attributes such as the *PriorityOne* field with the aim to manipulate the BMCA [94]. Once the device becomes the grandmaster, it propagates inaccurate timestamps and desynchronize attached slaves, and ultimately desynchronize the entire network [95].

This chapter excluded some PTP attacks (i.e., packet removal attack, master spoofing attack, slave spoofing attack, replay attack, and denial of service attack), as they may influence only individual PTP slaves or are easy to detect by PTP daemons. In contrast, the PTP attack strategies as listed above affect all slave clocks downstream from the location of the attacker while not causing apparent changes in PTP network traffic patterns or packet content, [94].

Previous research has evaluated or simulated some of these attacks. Ullmann [9] has analyzed the delay attack mathematically with a proposal to detect this attack by computing the uplink path (i.e., the delay between the master and its slave) at the beginning of synchronization and store it at the master side, while the slave computes and stores the downlink path delay. Here the attack can be detected when the new uplink/downlink path computed has a significant difference in the reference values. However, the *Delay_Req* message is not triggered by the arrival of a *Sync* message in the conventional PTP protocol and is processed on a best-effort basis, which may provide an unstable path delay over time, and therefore false alarms could be triggered. Other researchers argued that redundant paths in a PTP network could detect the delay attack by comparing the offset values that are calculated by each port [81],[82]. Nevertheless, APT can compromise all redundant paths, which may affect the offset values in both slave ports. Moussa [6] proposed a detection and mitigation method for the delay attack by using a redundant reference clock. However, their detection method works only for delayed *Sync* messages. Moussa [8] has simulated GM, TC, and asymmetric delay attacks, and they propose another network time reference to monitor the *Sync* messages as well as collect timestamps from slaves in order to detect an attack. However,

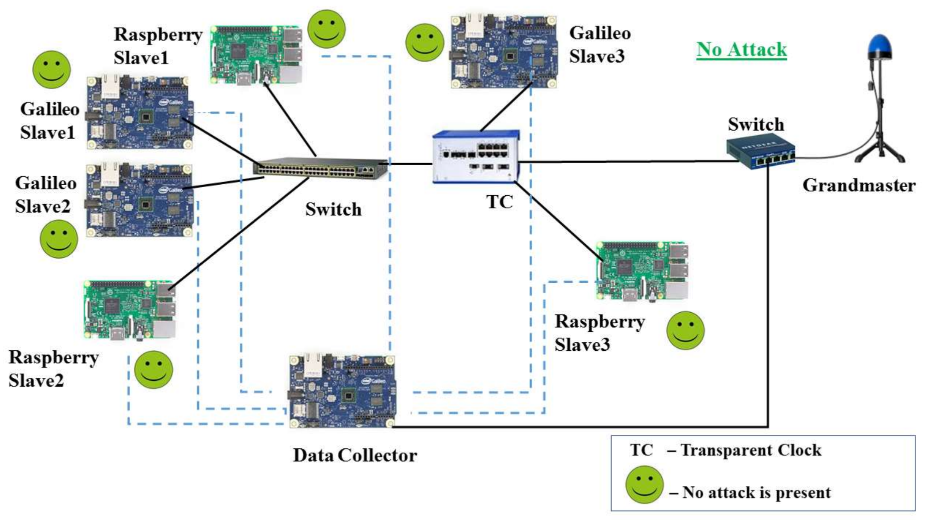

Figure 4-2. The PTP testbed.

the proposal fails in detecting some attacks if the attacker can manipulate the timestamp sent by slaves. The delay attack, packet modification attack, spoofing attack and denial of service attack have been simulated by [81], but without proposing a detection or mitigation method against these attacks. Itkin [22] has simulated some spoofing attacks as well as the BMCA attack.

4.2 Testbed and MitM Device

The testbed, as shown in Figure 4-2, contains one grandmaster clock (OMICRON OTMC 100-antenna-integrated PTP Grandmaster Clock) and two different types of slave devices (Intel Galileo Gen 1 and Raspberry Pi 3 model B) that are interconnected by ordinary switches and one transparent clock (Hirschmann RSP20). The PTP4l daemon is used by the Galileo devices, whereas PTPd is installed on the Raspberry Pi. Since the slave devices do not support hardware timestamping, all experiments were conducted using a two-step operation mode. In addition, the ordinary switches used do not support the peer delay mechanism, and hence the end-to-end mechanism was used. The data collector is an ordinary slave, which is always properly synced to the GM. Its primary role is to collect PTP data from nodes under attack while providing a correct time reference. All devices in the network are connected via CAT5e Ethernet cables with a data rate of 1000 Mbps. Also, the network only carries time synchronization traffic to minimize the network load, any

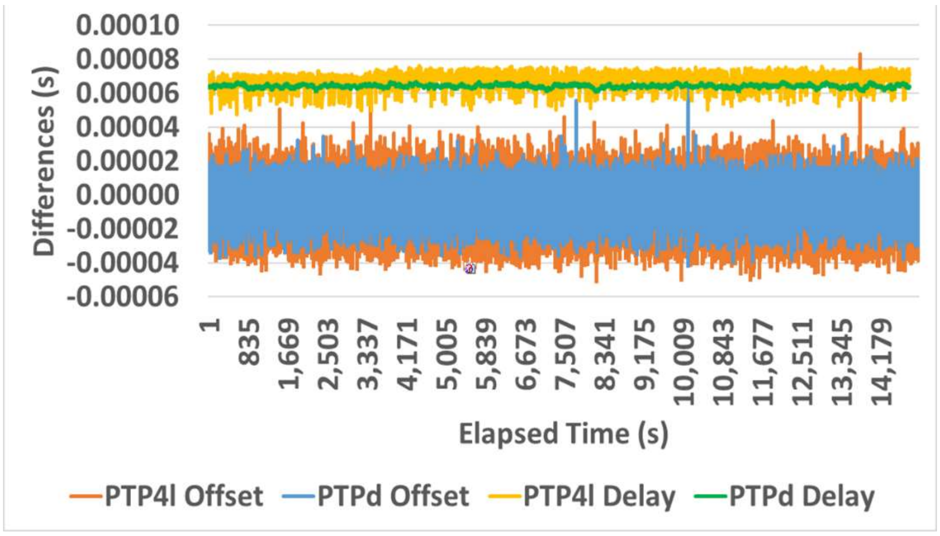

Figure 4-3. Offset and delay range of two PTP daemons running on the same machine.

non-deterministic packet delay, and jitter, while the CPU load of all devices is kept at a minimum.

The MitM device is a Linux computer with two network ports. It is located in the path between a master and its slaves to intercept and manipulate PTP packets in transit. The two network ports are connected via a bridge, and the ebtables tool is used to intercept PTP packets. A user-space program written in C language manipulates the intercepted packets and forwards them to their destination. Therefore, the programmable MitM simulates the effect of APTs on time synchronization, as it allows to slowly manipulate PTP timestamps over time, for example, with small increments in the order of microseconds.

Section 2.6 has analyzed PTP4l and PTPd. The different offset/delay calculation/averaging mechanisms in both daemons result in variations of the offset/delay values range, as shown in Figure 4-3. Here a slave computer ran both daemons concurrently over a four hours period using the same grandmaster. It can be seen that PTP4l has a higher offset/delay range in comparison to PTPd, which is a result of applying different filter mechanisms as described in the previous section 2.6. For example, PTPd uses the average of the last two offsets calculated as a final offset result of the new

Table 4-1: Offset/Delay statistics.

Configuration	Clock characteristics after 20 minutes			
	Offset		Mean Path Delay	
	Average (µs)	Standard Deviation (µs)	Average (µs)	Standard Deviation (µs)
Galileo without MitM	-0.066	15	424	3
Galileo with MitM	-0.031	30	537	6
Raspberry without MitM	0.125	8	169	1
Raspberry with MitM	0.065	14	280	2

synchronization cycle, while PTP4l uses the ratio and frequency deviation of the local clock in its offset calculation. This difference makes the offset range for PTPd smaller than for PTP4l.

Table 4-1 shows the time synchronization baseline (i.e., offset and mean path delay) with orderly working PTP daemons after 20 min of operation with and without a programmable MitM device in the packet path. Here the MitM is added between the GM and TC as shown in Figure 4-2. Please note that all offset and delay values collected during this period were computed at the same slave and were used to calculate the statistics as shown in Table 4-1. It can be seen that the MitM device introduces a symmetric uplink/downlink delay in the PTP packet path, as well as some jitter, as it is not a fully deterministic soft real-time system, i.e., packets are internally processed on a best-effort basis. Nonetheless, it allows for mimicking an attacker, which would otherwise be located in a switch, a GM or a TC.

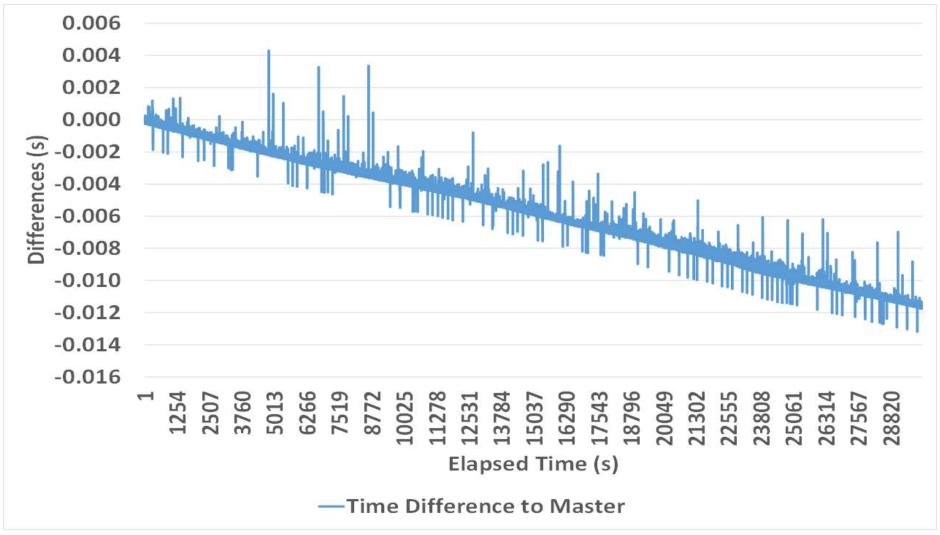

Figure 4-4. Galileo clock in free-running mode.

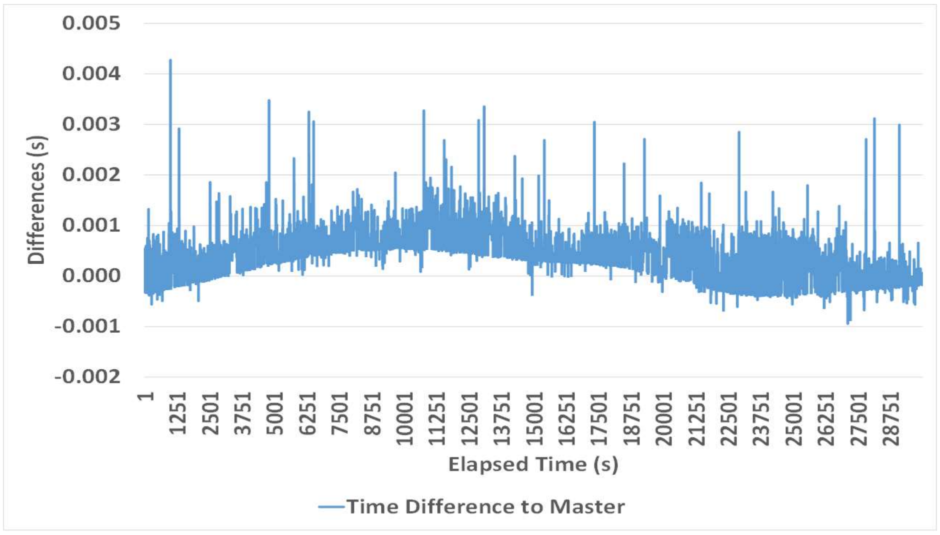

Figure 4-5. Raspberry Pi clock in free-running mode.

While slave (quartz-oscillator) hardware clocks are inherently unstable and tend to drift [54], there are subtle differences in their quality. This is shown in Figure 4-4 and Figure 4-5, where both slave device types did run over an 8 hours period (from 2 pm to 10 pm) in free-running mode and periodically send timestamps to the GM-synchronized data collector device (as shown in Figure 4-2). The Galileo hardware clock shows have better *stability* compared to the Raspberry when exposed to different temperature conditions (24 degrees Celsius during the day dropping to 17 degrees Celsius at night), while the latter shows better *accuracy*.

Table 4-2: Summary of investigated attacks.

Attack Type	Typical Attacker Location
Delayed packet transmission	Switch
correctionField manipulation	TC
GM timestamp manipulation	TC
Byzantine attack	GM

4.3 Experimental Results

Four different attack types, as listed in Table 4-2, were mimicked by the appropriately positioned MitM device. All attacks need to be persistent and continuously executed in order to effectively desynchronize a slave. GM timestamp manipulation and Byzantine attack were combined in one experiment (time source attack), as both result in incorrect grandmaster timestamps. Similarly, delayed packet transmission and *correctionField* manipulation were combined under one attack category (packet propagation attacks), as both result in a false path delay measurement. All experiments

Table 4-3: Experiments parameters and settings.

Parameter	Setting	Comment
Delay mechanism	End-to-End	The ordinary switches used do not support peer delay mechanism
Operation Mode	Two-Step	Slave NIC does not support HW timestamping
IP protocol	Version 4	-
Log Sync Interval	0	The master sends a Sync message every 2^0 second
Log Announce Interval	1	The master sends an Announce message each 2^1 seconds
Announce receipt timeout	3	After 3 unsuccessful announce intervals a slave clock will change into free-running mode
Log delay request interval	0	The slave sends a new Delay_Req message every 2^0 second
Clock Frequency	512 ppm	Maximum absolute frequency change that can be applied to the clock servo
Clock Adjustment	enabled	Slave clock update enabled
Clock Reset & Step threshold (PTP4l)	On start-up	Reset the clock only at the beginning of the synchronization
Clock Reset & Step threshold (PTPd)	offset >1 s	Reset the clock only if the offset from the master is greater than one second

were conducted by using the parameters and settings as shown in Table 4-3. These parameters and settings correspond to many PTP profiles such as the IEEE1588 Default profile [5], ITU-T Telecom G.8275.2 profile [96], and SMTPE ST- 2059-2 profile [97]. Hence, all the attacks' effects as described in this section will be applicable to any sector that use the aforementioned profiles.

All experiments in this section were conducted over relatively short time periods to cover different attack scenarios within a reasonable timeframe. However, in a real APT such attack could be stretched over much longer time periods (e.g., via using smaller delay increments, as further described in this section), with the overall result being the same.

4.3.1 Attack 1: Packet Propagation Attack

A. Delayed Packet Transmission (Compromised Switch)

Figure 4-6 shows the experimental setup to mimic this attack. The MitM systematically changes the residence time of synchronization packets either one-way (*Sync* or *Delay_Req* packets) or two-ways (both *Sync* and *Delay_Req* packets), introducing therefore either an asymmetric or a symmetric delay that affects timestamps *T2* (arrival time of a *Sync* message), *T4* (arrival time of a *Delay_Req* message), or both. Delay values are either fixed or are incremented between synchronization cycles. The skull icon (see Figure 4-6) indicates the attacker location while the colourful faces indicate whether the node is in the

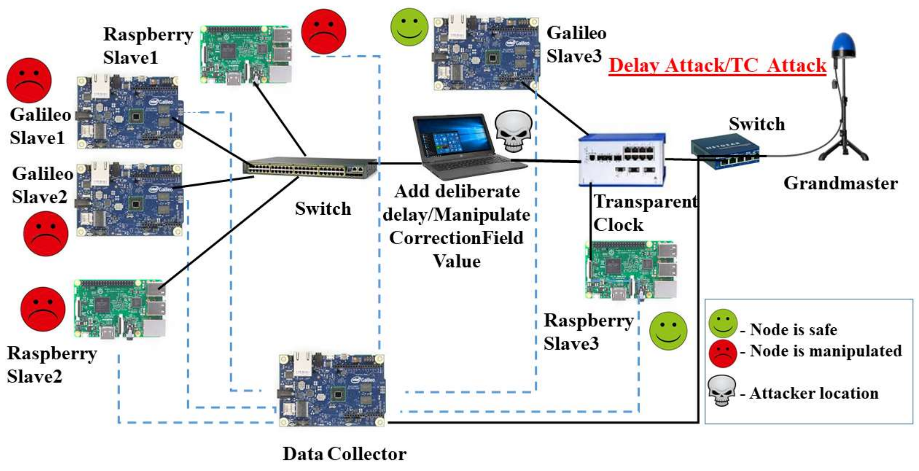

Figure 4-6. The packet propagation attack.

Table 4-4: Experiments and delay attack parameters.

Experiment No	Attack Parameters							
	Delay Increment between Cycles	Interval	Duration of Attack	Delay Type	PTP Daemon	Accumulated Slave Clock Drift	Offset Average	Delay Average
1	5 ms	1 s	200 s	Asymmetric	PTP4l	93 ms	226 ms	234 ms
					PTPd	101 ms	164 ms	74 ms
				Symmetric	PTP4l	84 ms	24 ms	432 ms
					PTPd	61 ms	51 ms	187 ms
2	10 µs	1 s	910 s	Asymmetric	PTP4l	5 ms	8.6 µs	2.8 ms
					PTPd	5 ms	13 µs	1 ms
				Symmetric	PTP4l	0	13 µs	5 ms
					PTPd	0	- 3.3 µs	2.4 ms
3	20 ms (once-off increment)	-	910 s	Asymmetric	PTP4l	10 ms	135 µs	10 ms
					PTPd	10 ms	75 µs	9 ms
				Symmetric	PTP4l	0	114 µs	20 ms
					PTPd	0	268 µs	17 ms

attack domain or not (i.e., red face indicates that the node is in the attack domain but the green face is not). Table 4-4 provides a summary of the conducted experiments. The accumulated slave clock drift is calculated as the difference between the correctly set data collector device time and the timestamps received by the manipulated slaves plus the time taken to traverse these timestamps from the slaves to the data collector. The latter also submits calculated offset and delay values, as shown in their PTP daemon's log files. The offset and delay average calculations consider all reported values during an attack.

An asymmetric delay (downlink or uplink) results in the following:

$$Offset\ (PTPd) = ((T2 + delay) - T1 - (C1 + C2)) - ((((T2 + delay) - T1 - (C1 + C2)) + (T4 - T3)) - C3)/2 \tag{4.1}$$

$$Offset\ (PTP4l) = ((T2 + delay) - (T1 + C1 + delayAsymmetry + C2)) - (((T2 + delay) - T3) * frequency + (T4 - C3 - (T1 + C1 + delayAsymmetry + C2)))/2 \tag{4.2}$$

$$\mathit{meanPathDelay\ (PTPd)} = ((((T2 + \mathit{delay}) - T1 - (C1 + C2)) + (T4 - T3)) - C3)/2 \tag{4.3}$$

$$\mathit{meanPathDelay\ (PTP4l)} = ((((T2 + \mathit{delay}) - T3) * \mathit{ratio}) + ((T4 - C3) - (T1 + C1 + C2)))/2 \tag{4.4}$$

$$\mathit{Offset\ (PTPd)} = (T2 - T1 - (C1 + C2)) - (((T2 - T1 - (C1 + C2)) + ((T4 + \mathit{delay}) - T3)) - C3)/2 \tag{4.5}$$

$$\mathit{Offset\ (PTP4l)} = (T2 - (T1 + C1 + \mathit{delayAsymmetry} + C2)) - ((T2 - T3) * \mathit{frequency} + ((T4 + \mathit{delay}) - C3 - T1 + C1 + \mathit{delayAsymmetry} + C2)))/2 \tag{4.6}$$

$$\mathit{meanPathDelay\ (PTPd)} = (((T2 - T1 - (C1 + C2)) + ((T4 + \mathit{delay}) - T3)) - C3)/2 \tag{4.7}$$

$$\mathit{meanPathDelay\ (PTP4l)} = (((T2 - T3) * \mathit{ratio}) + (((T4 + \mathit{delay}) - C3) - (T1 + C1 + C2)))/2 \tag{4.8}$$

where *delay* represents the added delay introduced by an attacker.

Equations (4.1)–(4.4) show how the offset and delay are affected (in both daemons) when an attacker increments the delay from the master to its slaves, while Equations (4.5)–(4.8) show the opposite effect, i.e., increment the delay from the slaves to their master.

Adding a notable fixed delay (i.e., in the order of milliseconds) to either the uplink or downlink path results in an instantaneous spike of the offset error, as shown in Figure 4-7 and Figure 4-8, which can be easily picked up by a PTP slave daemon, while a single smaller fixed delay that blends in with normally occurring delay measurement fluctuations (i.e., in the order of microseconds) do not have a significant impact on slave clock desynchronization. This leads to the conclusion that a high-impact APT requires carefully selected incremental path delays over time.

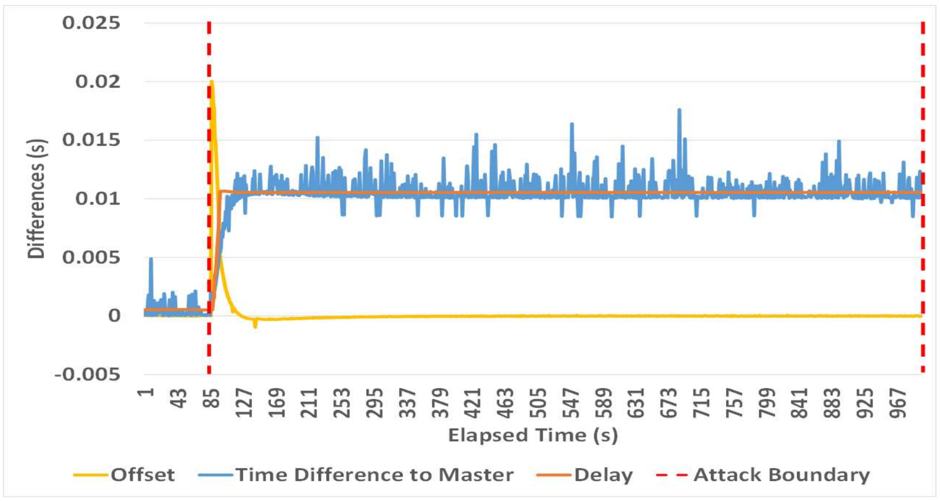

Figure 4-7. A single asymmetric delay increment of 20 ms (PTP4l).

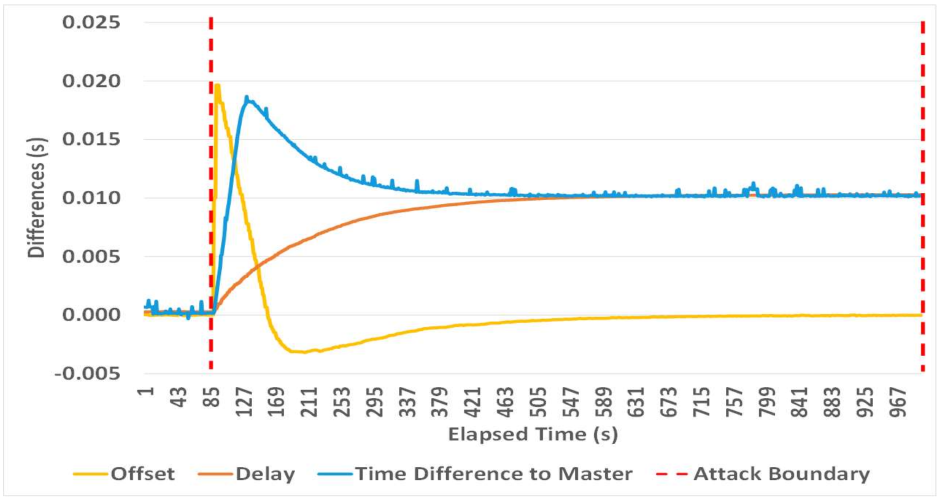

Figure 4-8. A single asymmetric delay increment of 20 ms (PTPd).

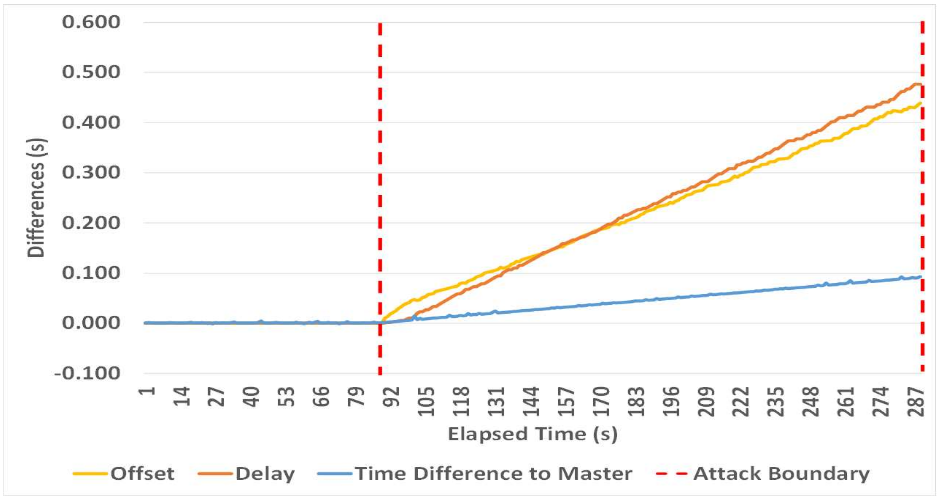

Figure 4-9. Asymmetric delay attack with a 5 ms increment every second (PTP4l).

Figure 4-9 and Figure 4-10 show the observed offset, delay and clock errors when a (Galileo-PTP4l) and (Raspberry–PTPd) slaves were exposed to an asymmetric delay (where *Sync* message transmissions were delayed) with 5 ms increments per second from about 90 s into the experiment, resulting in a slave clock error of 93 ms (Galileo) and 101 ms (Raspberry) after 200 s of an attack. It is notable that the introduced delay increment is too large to be immediately compensated by slave clock frequency adjustments (which are limited to 512 ppm per synchronization cycle throughout all experiments), resulting in a noticeable cumulated offset error over time. This can be avoided if the increment is limited to a compensable error, as shown in Figure 4-11 and Figure 4-12. Here the delay was reduced to 10 μs every second from about 90 s into the experiment, resulting in a slave clock error of around 5 ms after 910 s of the attack and an observable delay drift, while the offset error remained unchanged.

Manipulating a switch's firmware to introduce delays only in one direction may not be feasible; therefore, the impact of a symmetric delay attack was investigated, where a switch would apply a delay to all packets that cross it. Any fixed symmetric delay is naturally compensated by PTP (see (2.1)), as shown in Figure 4-13 and Figure 4-14, while small symmetric delay increments have no effect either as shown in Figure 4-15, Figure 4-16, and Table 4-4 (i.e., Table 4-4 shows that the accumulated slave clock drift was not affected in the second and third symmetric experiments due to the PTP ability to compensate such behaviour). However, introducing a symmetric delay increment of 5 ms per second shows some unexpected results, as the time synchronization error increased to around 84 ms (Galileo) and 61 ms (Raspberry) after 200 s into the attack, also showing a delayed increase of the measured offset errors (Figure 4-17 and Figure 4-18); here, the offset eventually starts drifting after the introduced delay amount exceeds the delay request interval that was set to one second (Table 4-4). This behaviour is a result of the PTP client software implementation mentioned before, where GM-initiated sync packets and slave-initiated delay measurements are not synchronized and are affected by different delay increments.

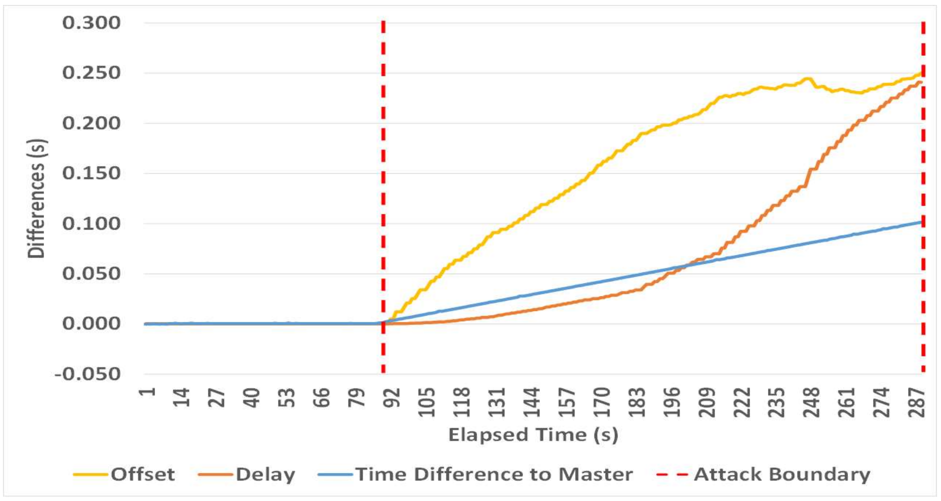

Figure 4-10. Asymmetric delay attack with a 5 ms increment every second (PTPd).

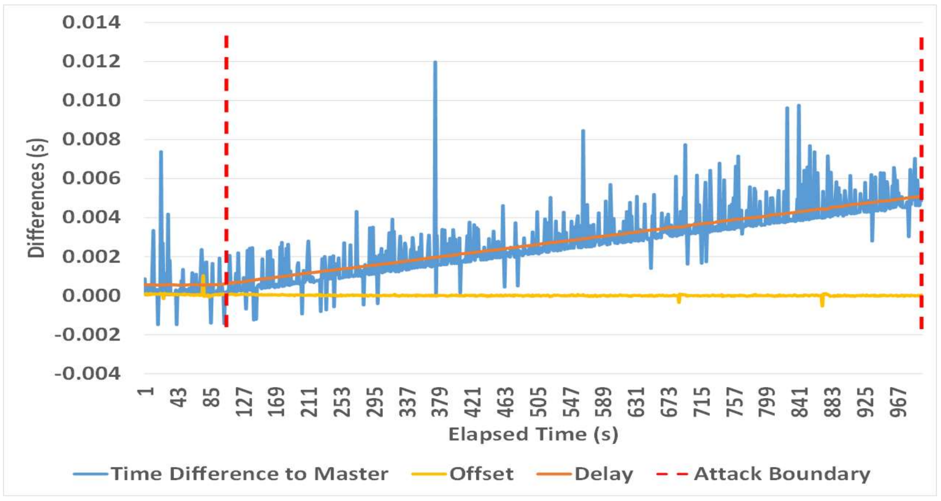

Figure 4-11. Asymmetric delay attack with a 10 µs increment every second (PTP4l).

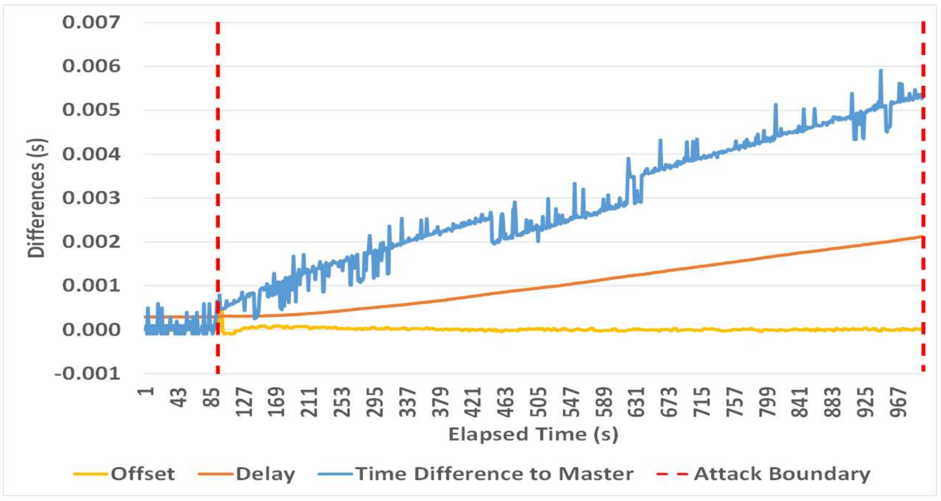

Figure 4-12. Asymmetric delay attack with a 10 µs increment every second (PTPd).

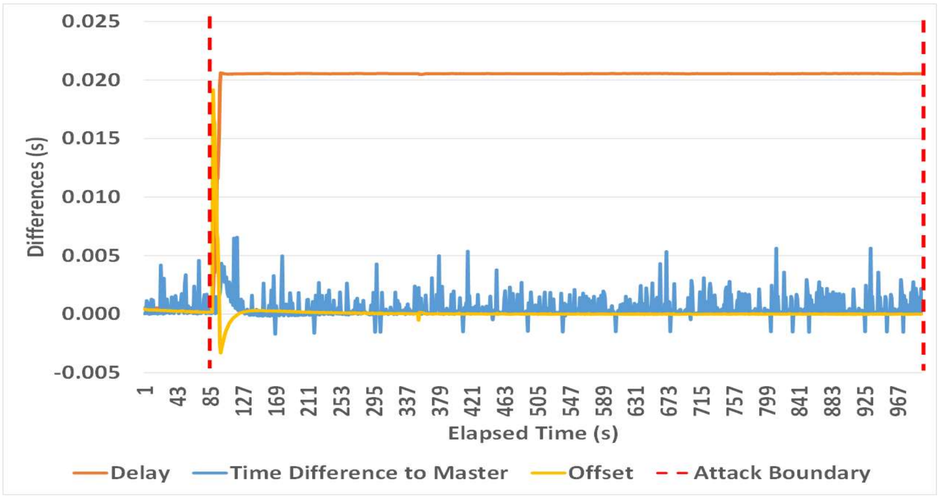

Figure 4-13. A single symmetric delay increment of 20 ms (PTP4l).

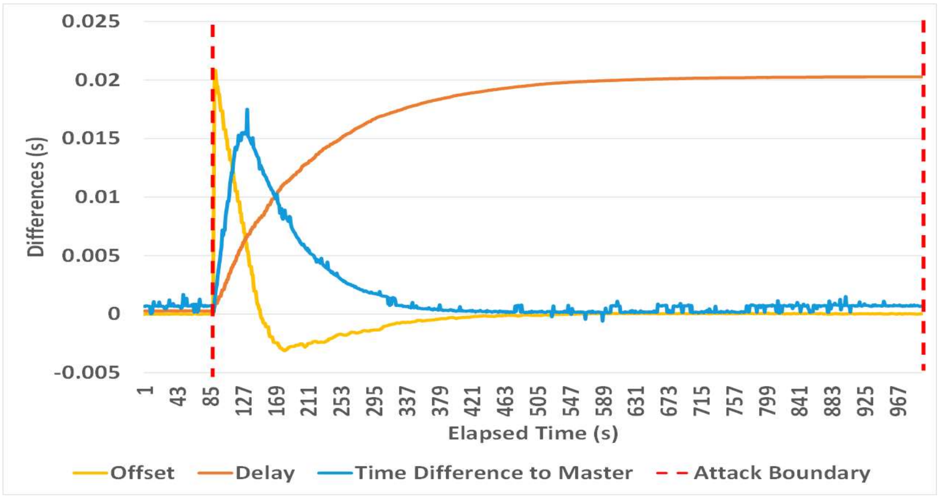

Figure 4-14. A single symmetric delay increment of 20 ms (PTPd).

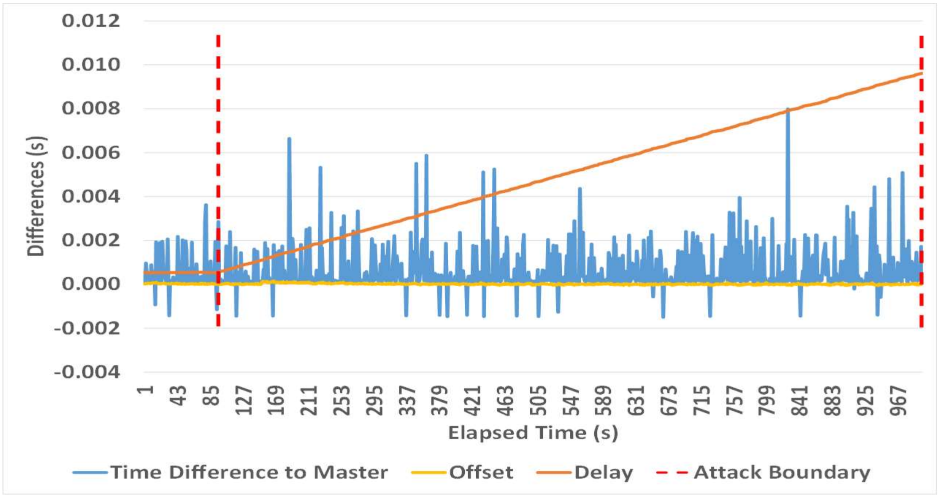

Figure 4-15. Symmetric delay attack with a 10 μs increment every second (PTP4l).

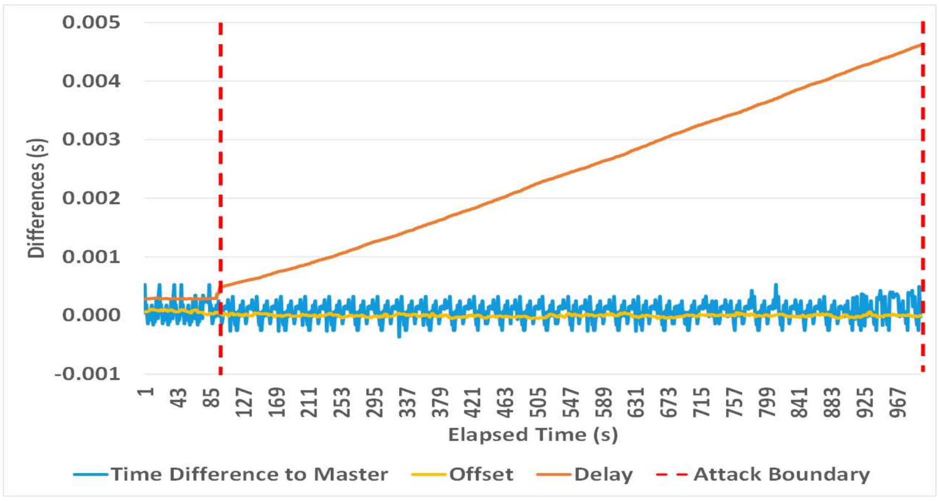

Figure 4-16. Symmetric delay attack with a 10 µs increment every second (PTPd).

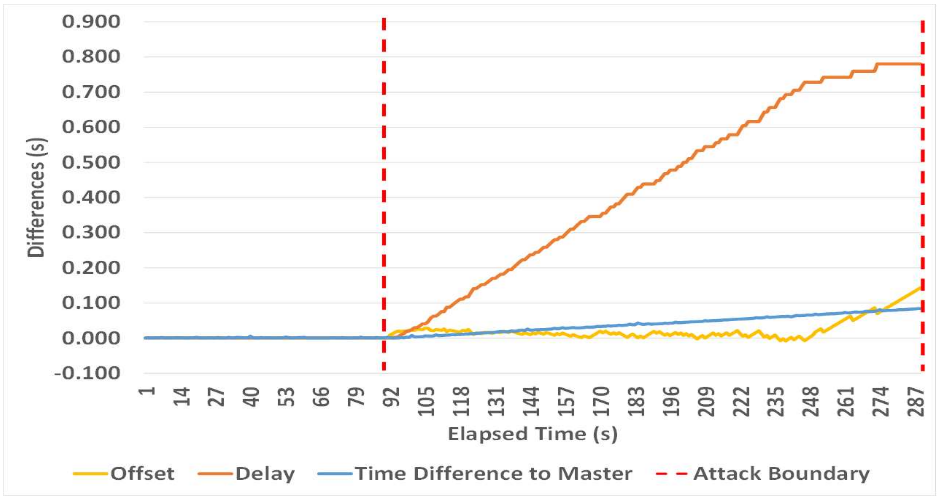

Figure 4-17. Symmetric delay attack with a 5 ms increment every second (PTP4l).

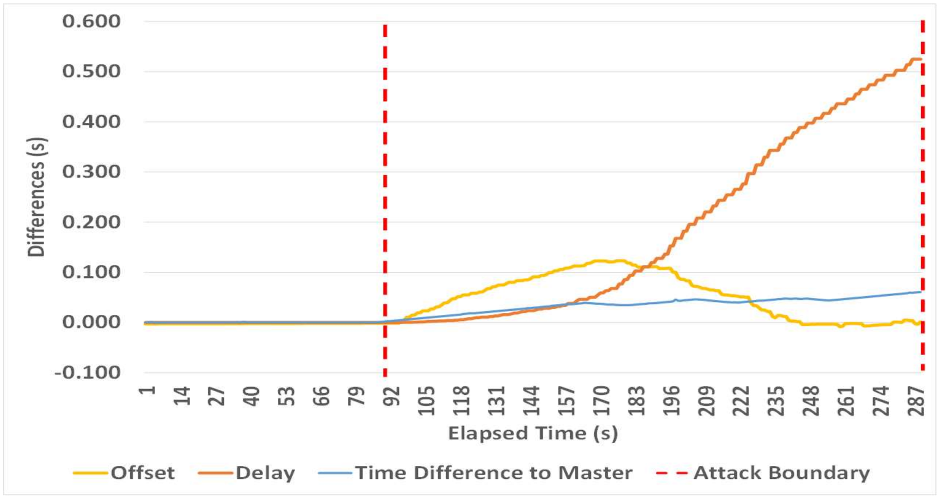

Figure 4-18. Symmetric delay attack with a 5 ms increment every second (PTPd).

It can be concluded that delay measurement variations could be a potential indicator to detect a delay attack, as Figure 4-9, Figure 4-10, Figure 4-11, Figure 4-12, and Figure 4-17 show that the delay distribution follows the time error distribution. Also, Table 4-1 and Table 4-4 show that the calculated delay averages are different and do change due to the attack.

In summary, an attacker can use the delay attack to desynchronize slave clocks efficiently as follows:

1. Introduce an asymmetric delay in the path between the slaves and their master, resulting in a slave clock drift of approximately half the maximum asymmetric delay (e.g., Figure 4-7 and Figure 4-8 show how the slaves drift by 10 ms as a result of a 20 ms asymmetric delay). Here smaller increments result in a small adjustment in the clock frequency and less apparent fluctuations of a slave clock offset.

2. Introduce a consecutive large delay in the uplink and downlink path between the master and its slaves that makes a slave clock always slew with the maximum frequency. Here, the value of the slave clock drift is variable and subject to the PTP daemon type and the filtering mechanism that is applied. Such an attack causes an increased slave clock offset over time, which makes the attack easy to be detectable.

B. CorrectionField Manipulation (Compromised TC)

Here the MitM attacker intercepts the *Follow_Up* and/or *Delay_Resp* messages and gradually changes their correction field value (as normally done by a TC) to provide a false path delay (see Figure 4-6). Providing a false *C2* (*correctionField* of *Follow_Up* message) or/and *C3* (*correctionField* of *Delay_Resp* message) introduces inaccurate offset and mean path delay values accordingly as follows:

$$Offset\ (PTPd) = (T2 - T1 - (C1 + C2 + Err)) - (((T2 - T1 - (C1 + C2 + Err)) + (T4 - T3)) - C3)/2 \tag{4.9}$$

$$Offset\ (PTP4l) = (T2 - (T1 + C1 + delayAsymmetry + C2 + Err)) - ((T2-T3) * frequency + (T4 - C3 - (T1 + C1 + delayAsymmetry + C2 + Err)))/2 \tag{4.10}$$

$$meanPathDelay\ (PTPd) = (((T2 - T1 - (C1 + C2 + Err)) + (T4 - T3)) - C3)/2 \tag{4.11}$$

$$meanPathDelay\ (PTP4l) = (((T2 - T3) * ratio) + ((T4 - C3) - (T1 + C1 + C2 + Err)))/2 \tag{4.12}$$

$$Offset\ (PTPd) = (T2 - T1 - (C1 + C2)) - (((T2 - T1 - (C1 + C2)) + (T4 - T3)) - (C3 + Err))/2 \tag{4.13}$$

$$Offset\ (PTP4l) = (T2 - (T1 + C1 + delayAsymmetry + C2)) - ((T2 - T3) * frequency + (T4 - (C3 + Err) - (T1 + C1 + delayAsymmetry + C2)))/2 \tag{4.14}$$

$$meanPathDelay\ (PTPd) = (((T2 - T1 - (C1 + C2)) + (T4 - T3)) - (C3 + Err))/2 \tag{4.15}$$

$$meanPathDelay\ (PTP4l) = (((T2 - T3) * ratio) + ((T4 - (C3 + Err)) - (T1 + C1 + C2)))/2 \tag{4.16}$$

$$Offset\ (PTPd) = (T2 - T1 - (C1 + C2 + Err)) - (((T2 - T1 - (C1 + C2 + Err)) + (T4 - T3)) - (C3 + Err))/2 \tag{4.17}$$

$$Offset\ (PTP4l) = (T2 - (T1 + C1 + delayAsymmetry + C2 + Err)) - ((T2-T3) * frequency + (T4 - (C3 + Err) - (T1 + C1 + delayAsymmetry + C2 + Err)))/2 \tag{4.18}$$

$$meanPathDelay\ (PTPd) = (((T2 - T1 - (C1 + C2 + Err)) + (T4 - T3)) - (C3 + Err))/2 \quad (4.19)$$

$$meanPathDelay\ (PTP4l) = (((T2 - T3) * ratio) + ((T4 - (C3 + Err)) - (T1 + C1 + C2 + Err)))/2 \quad (4.20)$$

where *Err* represents the added time value introduced by an attacker.

Equations (4.9)–(4.12) show how the offset and delay are affected (in both daemons) when an attacker manipulates the *correctionField* value of *Follow_Up* messages, while Equations (4.13)–(4.16) show the opposite effect, i.e., manipulates the *correctionField* of *Delay_Resp* messages. In contrast, the symmetric manipulation (i.e., the attacker manipulates the *correctionField* of *Follow_Up* and *Delay_Resp* messages with the same increment) are shown in Equations (4.17)–(4.20).

Table 4-5 provides a summary of the conducted experiments. They are based on the experiments of the previous section with the difference that the MitM attacker manipulates only the *correctionField* of either the *Follow_Up* message or *Delay_Resp* message to provide an asymmetric manipulation or manipulate both (*Follow_Up* and *Delay_Resp*) to provide a symmetric

Table 4-5: Experiments and correctionField manipulation parameters.

Experiment No	Attack Parameters							
	CorrectionField Timestamp Increment between Cycles	Interval	Duration of Attack	Increment Type	PTP Daemon	Accumulated Slave Clock Drift	Offset Average	Delay Average
1	5 ms	10 s	200 s	Asymmetric	PTP4l	-50 ms	-1 ms	-22.8 ms
					PTPd	-98 ms	-2.2 ms	116 µs
				Symmetric	PTP4l	-3 ms	-0.807 µs	-48 ms
					PTPd	-75 ms	-2.3 ms	28 µs
2	10 µs	10 s	910 s	Asymmetric	PTP4l	-230 µs	6 µs	388 µs
					PTPd	-326 µs	- 0.802 µs	163 µs
				Symmetric	PTP4l	≈ 0	-0.077 µs	108.5 µs
					PTPd	-500 µs	-0.822	114 µs

manipulation when they pass the MitM node as shown in Figure 4-6. In all experiments, *Err* was increased every 10 s, while the other attack parameters remained untouched.

Figure 4-19 to Figure 4-22 show that an incremental *correctionField* offset in *Follow_Up/Delay_Resp* messages causes a proportional desynchronization of the affected slave. The resulting delays are compensated for by the aforementioned clock adjustment mechanisms; therefore, the offset oscillations are shown in the diagrams.

Increasing the *correctionField* value of *Follow_Up/ Delay_Resp* messages may cause the calculated mean path delay to become smaller than zero. PTP4l accepts these values as shown in the figures and can therefore be used as an indicator for this attack. In contrast, PTPd rejects any negative path delay (from master to slave or slave to master), and it uses the last valid non-negative mean path delay calculated instead, as described in Section 2. This makes the mean path delay value keeping the last valid value calculated before the attack; therefore, the delay appears as a straight line (see Figure 4-20 and Figure 4-22). As a result, PTPd's delay calculations cannot be used to indicate this attack, which is a major weakness that can be specifically exploited in a PTP attack.

Figure 4-23 to Figure 4-26 show the impact of a symmetric TC attack on both clients with a similar ripple effect on offset values. PTP4l shows robustness against this attack with very small slave clock time deviations and easily detectable delay measurements. PTPd, in contrast, shows significant slave time errors and normal delay values. Again, this is a major weakness that can be specifically exploited in a PTP attack. This attack causes a time synchronization error ranging from hundreds of microseconds to 98 ms depending on the attack parameters and PTP daemon type, as shown in Table 4-5.

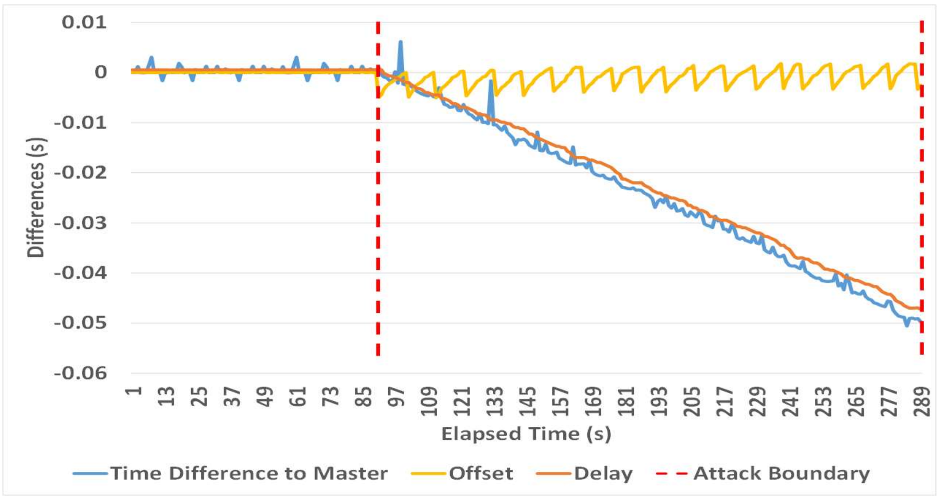

Figure 4-19. Asymmetric TC manipulation with a 5 ms increment every 10 s (PTP4l).

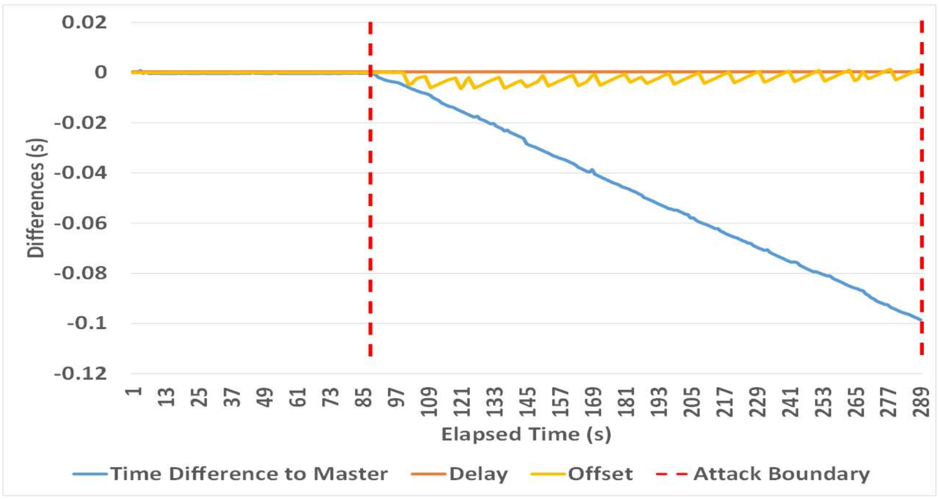

Figure 4-20. Asymmetric TC manipulation with a 5 ms increment every 10 s (PTPd).

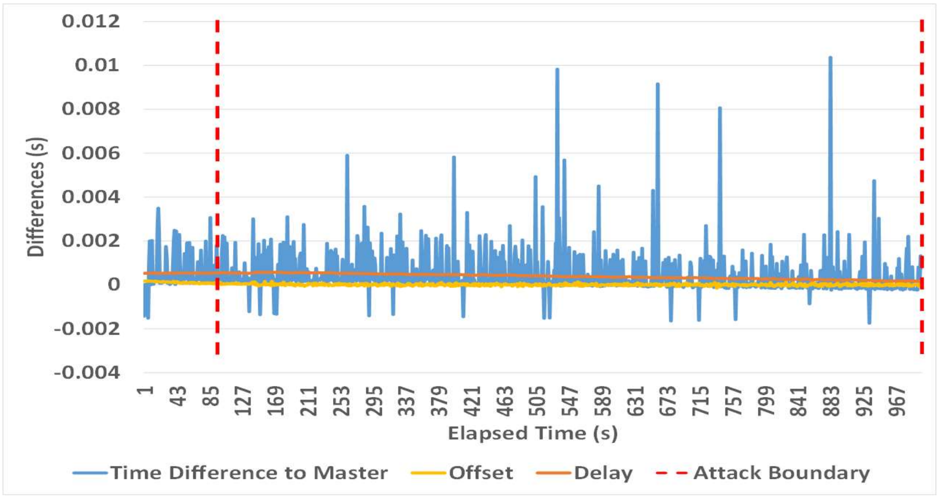

Figure 4-21. Asymmetric TC manipulation with a 10 µs increment every 10 s (PTP4l).

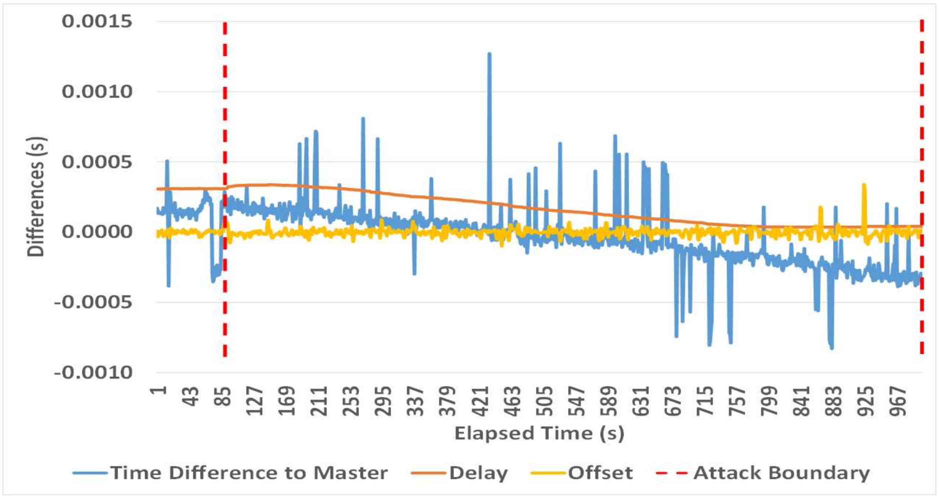

Figure 4-22. Asymmetric TC manipulation with a 10 µs increment every 10 s (PTPd).

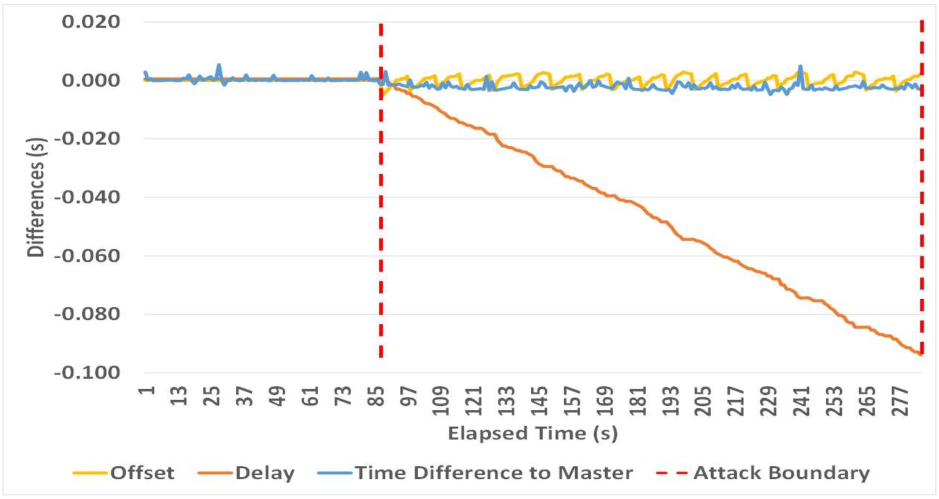

Figure 4-23. Symmetric TC manipulation with a 5 ms increment every 10 s (PTP4l).

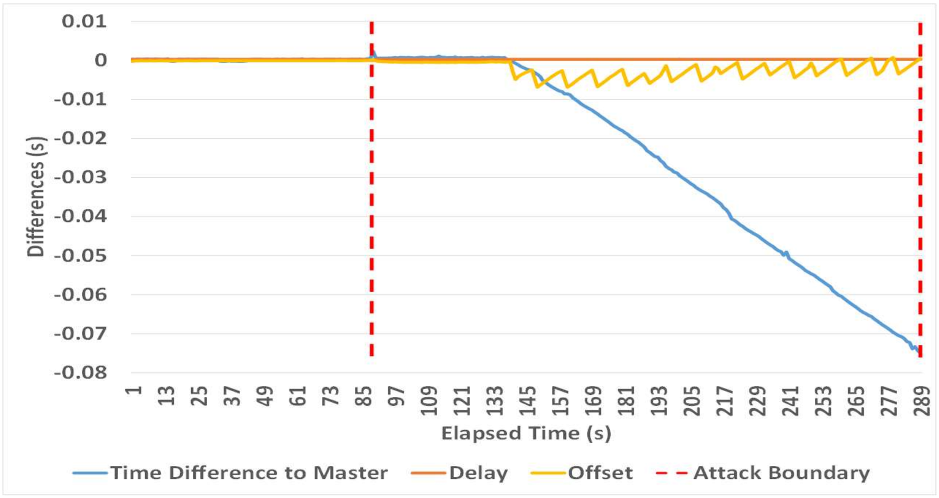

Figure 4-24. Symmetric TC manipulation with a 5 ms increment every 10 s (PTPd).

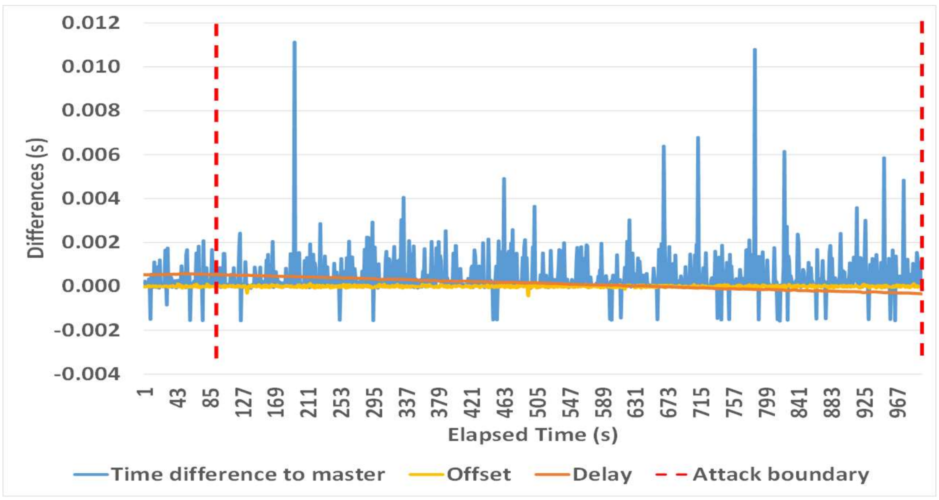

Figure 4-25. Symmetric TC manipulation with a 10 µs increment every 10 s (PTP4l).

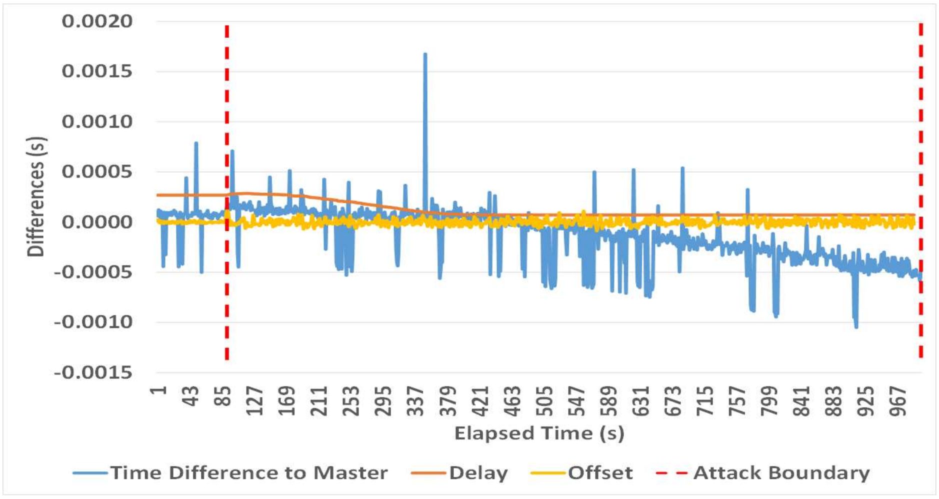

Figure 4-26. Symmetric TC manipulation with a 10 µs increment every 10 s (PTPd).

In summary, an attacker can use the *correctionField* attack to desynchronize slave clocks efficiently as follows:

1. In the case of PTPd, introduce an asymmetric/symmetric incremental *correctionField* offset in the path between the slaves and their master. Here smaller increments result in a small adjustment in the clock frequency, and less apparent fluctuations of a slave clock offset and delay value. PTPd is not only vulnerable to both symmetric and asymmetric TC attacks, but its delay values are not correctly reported, making the daemon blind to these attacks.

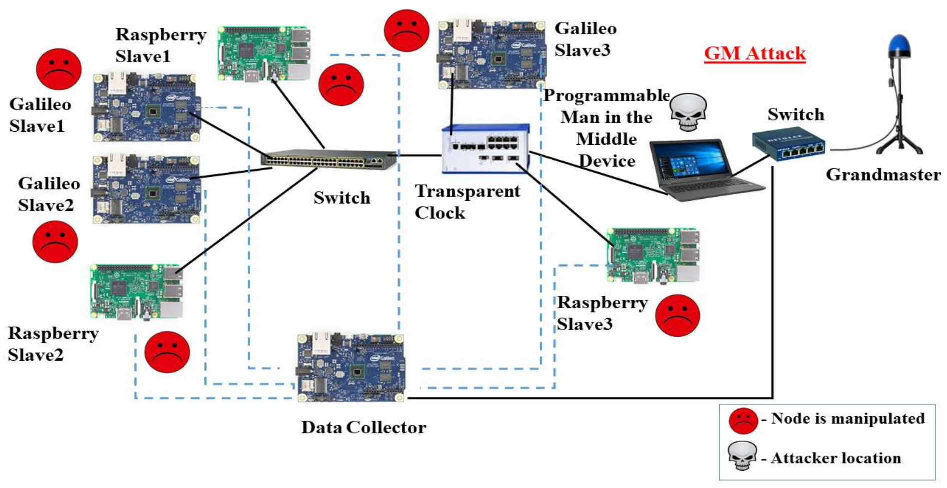

Figure 4-27. GM attack setup.

2. In the case of PTP4l, introduce an incremental asymmetric *correctionField* offset in the path between the slaves and their master. Here the small increment will not affect the offset values, but the delay values will decrease over time, therefore providing an indicator for this attack.

4.3.2 Attack 2: Time Reference Attack (Compromised TC or GM)

In this attack, GM timestamps *T1* and *T4* are falsified by the MitM device (see Figure 4-27), emulating a manipulation either at source by the GM itself (as a result of a Byzantine attack) or in transit by a manipulated TC. This is done by manipulating the *preciseOriginTimestamp* of *Follow_Up* messages (*T1*) and *receiveTimestamp* of *Delay_Resp* messages (*T4*), whereby *T1* and *T4* are synchronously and gradually incremented/decremented. This leads to inaccurate offset and mean path delay values as follows:

$$Offset\ (PTPd) = (T2 - (T1 \pm Err) - (C1 + C2)) - (((T2 - (T1 \pm Err) - (C1 + C2)) + ((T4 \pm Err) - T3)) - C3)/2 \qquad (4.21)$$

$$Offset\ (PTP4l) = (T2 - ((T1 \pm Err) + C1 + delayAsymmetry + C2)) - ((T2 - T3) * frequency + ((T4 \pm Err) - C3 - ((T1 \pm Err) + C1 + delayAsymmetry + C2)))/2 \quad (4.22)$$

$$meanPathDelay\ (PTPd) = (((T2 - (T1 \pm Err) - (C1 + C2) + ((T4 \pm Err) - T3)) - C3)/2 \quad (4.23)$$

$$meanPathDelay\ (PTP4l) = (((T2 - T3) * ratio) + (((T4 \pm Err) - C3) - ((T1 \pm Err) + C1 + C2)))/2 \quad (4.24)$$

where *Err* represents the manipulated timestamp introduced by an attacker.

Table 4-6, Figure 4-28, Figure 4-29, Figure 4-30, and Figure 4-31 show how both PTP daemons behave in the presence of this attack. As already seen in previous experiments, large introduced time errors leave a slave clock slew with the maximum frequency, causing a graduate compensation of offset values as shown in the oscillations in Figure 4-28 and Figure 4-30, while small errors have less apparent effects (Figure 4-29 and Figure 4-31).

However, this attack does not directly manipulate the delay path and causes only unassuming variations in delay calculations, as shown in Table 4-6. This makes the time reference attack very devious and hard to detect. It is worth noting that this attack cannot be mitigated via redundant GMs (as the attacker can be positioned in a TC), while infrastructure redundancy is not effective if all TCs, existed in redundant paths, are manipulated simultaneously.

Table 4-6: Experiments and timestamp manipulation parameters.

Experiment No	Attack Parameters							
	Grandmaster Timestamp Increment between Cycles	Interval	Duration of Attack	Increment Type	PTP Daemon	Accumulated Slave Clock Drift	Offset Average	Delay Average
1	5 ms	10 s	200 s	Symmetric	PTP4l	-92 ms	-3.2 ms	452 μ
					PTPd	-101 ms	-2.3 ms	310 μ
2	10 μs	10 s	910 s		PTP4l	-910 μ	-0.913 μ	583 μ
					PTPd	-910 μ	-16 μ	315 μ

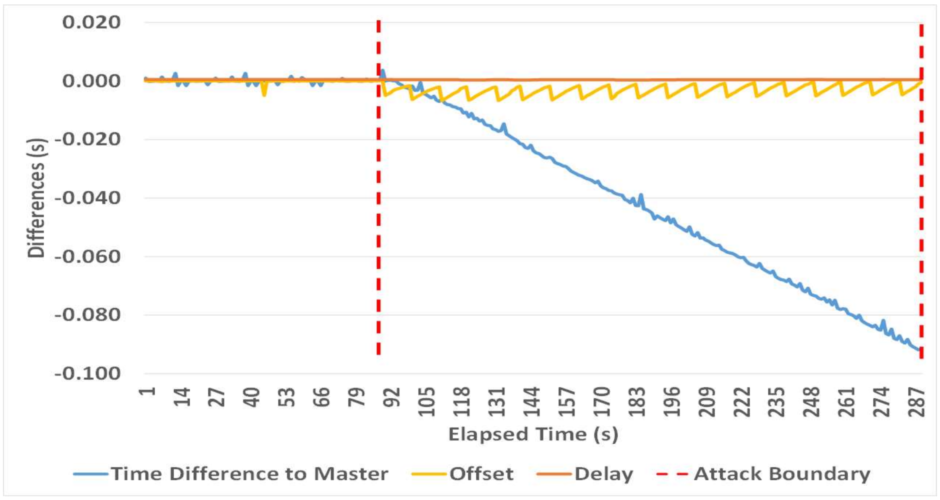

Figure 4-28. GM timestamp increment of 5 ms every 10 s (PTP4l).

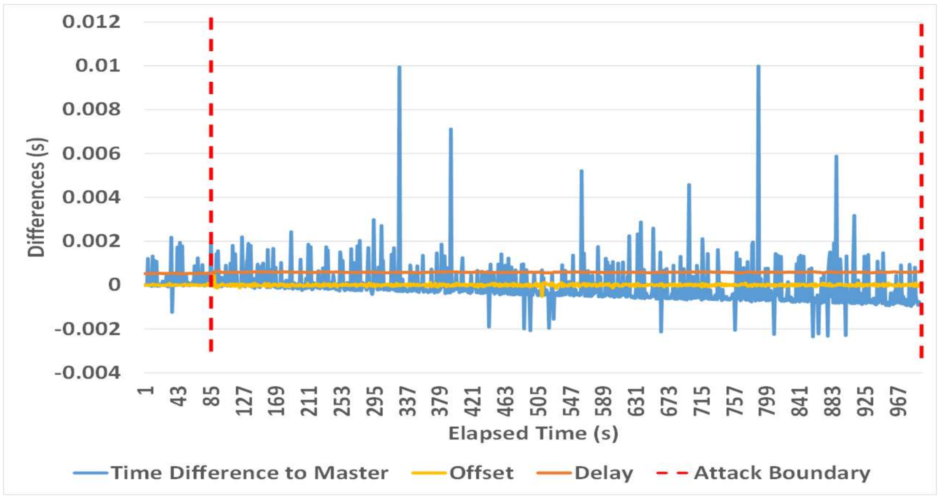

Figure 4-29. GM timestamp increment of 10 μs every 10 s (PTP4l).

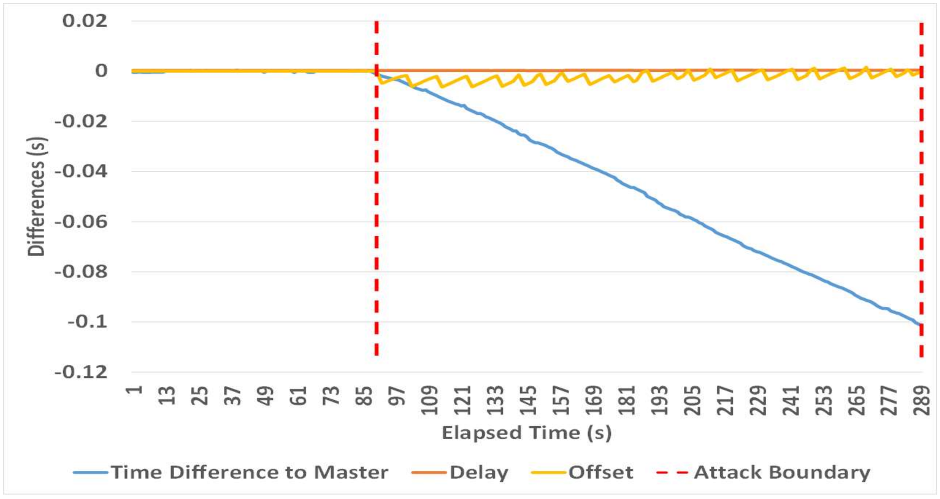

Figure 4-30. GM timestamp increment of 5 ms every 10 s (PTPd).

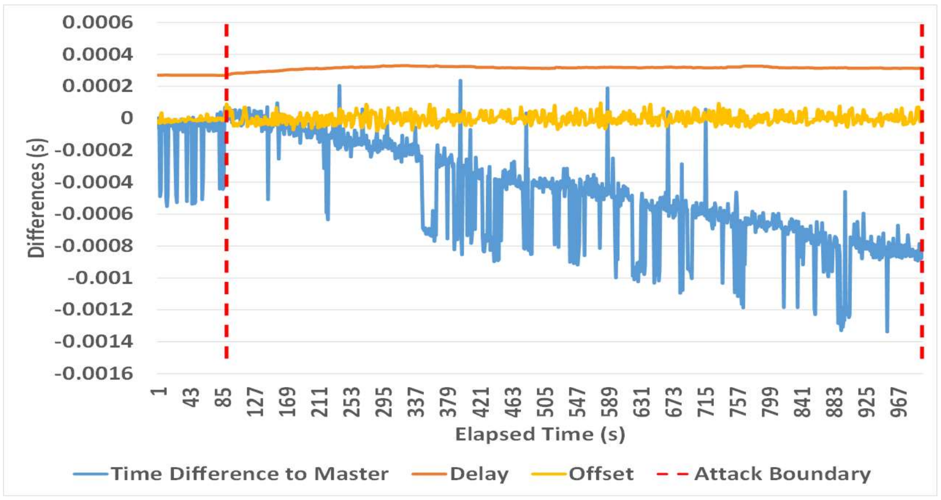

Figure 4-31. GM timestamp increment of 10 µs every 10 s (PTPd).

4.4 Conclusion

This chapter provides an experimental validation and characterization of the four most important APTs on PTP networks, comparing the behaviour of the two widely used PTP clients PTPd and PTP4l. It presents a testbed and a programmable hardware MitM device that allows conducting a variety of attacks on time synchronization networks.

The experiments show that both clients are vulnerable to the attacks presented and that the resulting time synchronization errors as well as fluctuations of delay and offset values correlate with the chosen attack parameters, i.e., delay or error increments. Their behaviour concurs with their specific implementations, which are also outlined and contrasted.

A delayed packet transmission attack can be picked up by both PTP clients by simply observing monotonic increases of delay measurements. A TC (i.e., *correctionField*) attack, in contrast, can only be reliably detected via delay measurements by PTP4l, while PTPd presents incorrect and unassuming delay values during such an attack. This daemon can also be manipulated via a symmetric *correctionField* attack, making it, therefore, even more vulnerable.

In contrast, a time source attack (either implemented in a GM or a TC) does not result in incremental delay calculations but in oscillating offset values for either daemon and is, therefore, more difficult to detect. This makes it the most suitable PTP exploit for a high impact APT.

While this chapter outlines an attack detection mechanism based on a single monitoring node, the next chapter will determine suitable time-series analysis methods for attack detection and their limitations with regard to sensitivity and specificity, which depend on the attack type, the duration of the attack, the attack parameters and the number of slaves affected.

Chapter 5

A Security Enhancement of the Precision Time Protocol using a Trusted Supervisor Node

The work outlined in this chapter is currently under review as:

W. Alghamdi and M. Schukat, "A Security Enhancement of the Precision Time Protocol using a Trusted Supervisor Node," currently under review at *Engineering Science and Technology, an International Journal* impact factor 4.36.

Chapter 3 has shown that Annex P prong A and B cannot provide protection against some PTP attacks, such as the delay attack [23], [4], [61]. Moreover, prong C does not provide comprehensive protection if an attacker simultaneously compromises all redundant systems [27], redundant paths [24], [25], or redundant grandmasters [26]. Chapter 4 has investigated the impact of all PTP attack strategies as outlined in Chapter 3 [4] on the two most popular PTP daemons, PTP4l and PTPd, thereby highlighting the need for advanced attack detection methods. This chapter proposes a detection system that is aligned with prong D as described in IEEE 1588-2019 Annex P. The underlying idea is centred around analyzing data collected from all slave clocks in a network during the synchronization process and then detect anomalies caused by an attack. The proposed detection system can work efficiently on a device without an exact time reference (e.g., an unsynchronized stand-alone device) or on a dual clock device that has a PTP slave clock and an accurate unsynchronized local clock (e.g., a non PTP-synchronized device with an oven-controlled crystal oscillator), thereby providing different levels of capabilities in detecting attacks. Section 5.1 discusses related work. Section 5.2 highlights the TSN concept, including the data collection mechanism. An experimental testbed, which combines hardware and software required to perform the TSN role and to launch PTP attacks, as well as the conducted experiments, will be presented in Section 5.3. A summary of results will be discussed in Section 5.4, followed by a conclusion and future work in Section 5.5. This chapter answers the third research question as outlined in Chapter 1.

5.1 Related Work

Since the attack impact can be noticeable at slave clocks regardless of preventive methods used to protect a PTP network, collecting and analyzing data from these slaves can be the cornerstone to detect such attacks [4]. Annex P prong D of IEEE 1588-2019 outlines such monitoring and management mechanisms. The underlying idea is that monitoring the performance of the PTP network can provide a good indication of potential PTP attacks, particularly the delay attack. For example, collecting the link delay between nodes (P2P mechanism) or between the master and its slaves (E2E mechanism) using a central management system can help to detect a potential attack. Furthermore, the central management system can monitor unexpected offset jumps or large offset corrections that reflect a large drift of the local oscillator. IEEE 1588-2019 (Annex J) provides a list of potential parameters and statistics counters that can be used to monitor the performance of the PTP network and the PTP clock operation. These parameters include the master-slave delay, the slave-master delay, the mean path delay, and the offset from the master. Annex J calculates the average, maximum, minimum, and standard deviation of each aforementioned parameter over 15 minutes and over 24 hours. Moreover, Annex J counts the announce messages sent/received by/from the current GM, the announce messages received from foreign masters, *Sync* messages sent/received by/from the master, *Follow_Up* messages sent/received by/from the master, *Delay_Req* that have been sent/received, and *Delay_Resp* that have been sent/received in each 15 min and 24 h interval [7]. However, Annex J does not outline how to compare these statistics, nor does it determine a security threshold that can be used to trigger an alarm.

In line with prong D, some researchers have proposed a detection method against PTP attacks using different ways such as comparing the slave offset in a reference clock, round trip delay (RTD) measurement, and sudden changes in the offset or the delay values obtained by the network clocks [101]. Moussa et al. [6], [8] propose a detection method against some PTP attacks. Their proposal requires another time reference that is capable of collecting timestamps from slaves and comparing them with a reference timestamp in order to detect PTP attacks. Nevertheless, their proposal does not provide protection to all potential PTP attacks. Also, their solution is vulnerable to advanced persistent threats (APTs) [19], where an attacker simultaneously manipulates all time references in a network or manipulates all timestamps sent by slaves. Using the same approach (i.e., comparing the slave offsets in a reference clock), Moradi et al. [101] suggest dividing all PTP clocks in a network into different blocks. Each block consists of at least one master clock (i.e., GM or BC) and other clocks. Each block calculates its offset from its master and sends the offset value to an analysis unit. If the offset is a non-zero value, the analysis unit will announce an attack on that block. However, achieving a zero offset in a PTP network can be very difficult to achieve even in the most optimal PTP network. In addition, the proposal focuses only on the delay attack only. On the other hand, Ullmann [9] suggests another detection method based on calculating the delay in each path (i.e., uplink and downlink paths) separately and store them at the master side as a reference. Here a delay attack can be detected when a measured uplink/downlink path delay has a significant difference from the reference value. However, normal network traffic can encounter some delays, which may result in a false alarm raised. In addition, the proposal does not provide protection against the other attack types. Similar work has been proposed by [102] (i.e., using RTD measurement). Lisova et al. [102] suggest monitoring messages sent by the grandmaster at the slave side. The authors assumed that these messages should be received at equal intervals unless an attack is present. However, any computer network is prone to path asymmetry and variable network latency, depending on the nature of the underlying network [76], [77] and therefore a false alarm can be raised. Also, there is no guarantee that the grandmaster will always send messages in regular intervals, instead differences in the order of milliseconds can be experienced. With regards to the last detection appraoch, i.e., sudden changes in the calculated offset/delay values, the underlying idea is that an attack will be announced when a sudden change of the delay/offset amount is observed at least in one slave clock [101]. However, it has been shown that a time source attack does not result in incremental delay calculations and therefore make such kind of attack detection insufficient [2].

Please note that this section focuses only on prong D and ignore the other protection ways that rely on prong A, B, and C as they already discussed in Chapter 3. Also, it is worth noting that the idea of prong D is still new and has not resulted in a significant amount of research yet.

This chapter proposes a trusted supervisor node (TSN) that acts as a central management system. It provides two different algorithms consistent with prong D that are able to detect all PTP attacks as listed in [4]. In contrast to the other approaches, this proposal does not necessarily require the existence of another time reference to detect all PTP attacks. Also, it takes advantage of more PTP clock parameters to identify and classify attacks. More details and comparisons between TSN and the other approaches, including Prong D, will be further discussed below.

5.2 The Trusted Supervisor Node Concept

In line with the ideas of Annex P Prong D, a detection system is proposed to detect PTP attacks using a monitor unit. The underlying idea of the detection system is that although individual slave clocks are intrinsically inaccurate and likely to drift [54], a subset of slaves may show a statistically significant deviation in their synchronization outputs if they are simultaneously exposed to manipulation [14]. The trusted supervisor node (TSN) is proposed to act as a monitoring unit for all slaves in a PTP network.

The TSN is an isolated and protected device that can be either:

- Type-A: a stand-alone device with a normal unsynchronized clock oscillator,
- Type-B: a dual clock device. Here, the TSN contains an unsynchronized accurate clock with very predictable and stable drifts, e.g., an oven-controlled crystal oscillator and a normal PTP synchronized slave clock.

Each TSN type has different capabilities in detecting PTP attacks: Since a Type-A TSN only has access to the collected data from the monitored slave clocks, it can only use statistical variations in this data to detect a PTP attack. A Type-B TSN exposes itself as a potential attack target. Having two independent clocks in a TSN device (i.e., a synchronized slave clock and an unsynchronized local clock) provides more baseline data to detect the most silent attack, i.e., the time source degradation attack.

The TSN, as a protected endpoint, uses TPM (Trusted Platform Module) to store encryption keys/hash values that are utilized for hardware validation and to authenticate its firmware when booted, thereby hardening the system against malicious malware injections or hardware manipulations that would render its operation useless. Data communication from slaves is authenticated via digital certificates using public key infrastructure, thereby avoiding the problem of MitM and injector attacks aimed to falsify the slaves' clock information by applying best industry practices. These certificates are exchanged using a handshake protocol and mutually authenticate the other side's cert to build a trust relationship. In this process, the TSN and slaves agree on pairwise shared private keys, which are subsequently used to encrypt/authenticate all data coming from a slave. While this process requires additional computational resources, e.g., the integration of a TLS / HTTPS protocol stack, it does not limit the detection capabilities of the TSN. The received and authenticated slave clock data, transported in slave clock status messages (SCSMs), is processed by the TSN and stored locally. Using this approach, the injection, manipulation, out of order delivery, and replay of (unsolicited or stale) SCSMs is easily detected. Packet transmission latencies or jitter (deliberately caused or a result of network traffic) do not matter either. Furthermore, each slave can use a combined digital/attribute certificate [98] to report its clock characteristics, including clock stability and typical clock drifts to the TSN, which can be subsequently used to complement a slave clock's baseline parameters outlined further below.

The TSN collects data from authenticated slave clocks and analyses them in real-time using time series analysis methods, allowing it to detect an attack in real-time. The baseline parameters of every slave are locally stored and collected using dedicated software developed by me, namely the *slave monitor client* program and the *data concentrator server* program. The slave monitor program is installed on each slave clock endpoint, and its role is, after successful mutual authentication, to send per synchronization cycle an SCSM to the TSN, that includes:

- a unique slave identifier *slaveID* (i.e., the authenticated slave's IP address),

- the latest *offset* and *delay* values calculated by the slave daemon (taken from the PTP log file),

- the difference between the arrival time of the last received *Sync* message (t2) and the momentary delay value as calculated by the PTP slave, i.e., the estimated time of *Sync* packet departure from a slave's perspective (estimated master timestamp - *EMT*),

- key data from the latest *Sync/Follow_Up* message sent by the GM, i.e., the *SyncID*, GM timestamp (t1), *correctionField* content (c1 or c2), the grandmaster clock ID (*grandID*), and the sync interval value (*syncInterval*) in the second scale.

In contrast, the data concentrator program is installed on the TSN device, where it collects, authenticates and analyses SCSM from all slaves in real-time after the aforementioned initial mutual authentication with each monitored slave.

The TSN requires some baseline data before going into the monitoring stage, as every device/network link may have unique characteristics. In the conducted experiments, this data is collected during an initial calibration period prior to an attack. All slave clocks have the same *syncInterval*, and *syncID* wrap-arounds are not dealt with. It is assumed that there may be one or more attackers performing possibly multiple attacks simultaneously.

Since PTP attacks may target synchronization packets in transit, or the grandmaster clock itself, the TSN provides two different attack detection algorithms that work with both TSN types with minor restrictions, as further outlined below. The baseline determination stage and the attack detection algorithms are as follows:

5.2.1 Baseline Determination Stage

During this initialization stage, for each monitored slave that presents itself to the TSN (i.e., initiates an authentication handshake), a trust relationship is established. Subsequently, the TSN determines the slave's *slaveID*, *grandID,* and the *syncInterval* value and processes over a configurable period (C_P) incoming SCSM, which are stored in a circular buffer of size N_B.

Once the circular buffer contains N_B elements, the TSN calculates/updates with each new SCSM arriving:

- the largest observed *correctionField* value c1 or c2 (c_{max})

- the moving average of *delay*, as calculated by the slave's PTP daemon, thereby updating an upper and lower delay average value ($delay_{max}$ and $delay_{min}$)

- the maximum and minimum observed *offset* values calculated by the PTP daemon ($offset_{max}$ and $offset_{min}$). Here, the TSN determines and eliminates offset outliers using a significance test based on a configurable z-score [99] first before updating both values. This pre-selection is necessary, as PTP daemons do occasionally report intermittent offset spikes, for example, because of momentary network congestions or instability of a slave clock frequency.

Finally, two threshold values are to be set that are required for both detection algorithms:

- The acceptable timestamp difference (T_{Δ}). For a type-A TSN, this is the maximum tolerated difference between any two *EMTs* calculated by slaves for the same synchronization cycle (*SyncID* is used as an identifier); additionally, for a Type-B TSN, T_{Δ} is the maximum tolerated difference between its slave clock and the TSN clock.

- The minimum number of consecutive suspicious SCSM (N_{SCSM}) before an attack is considered.

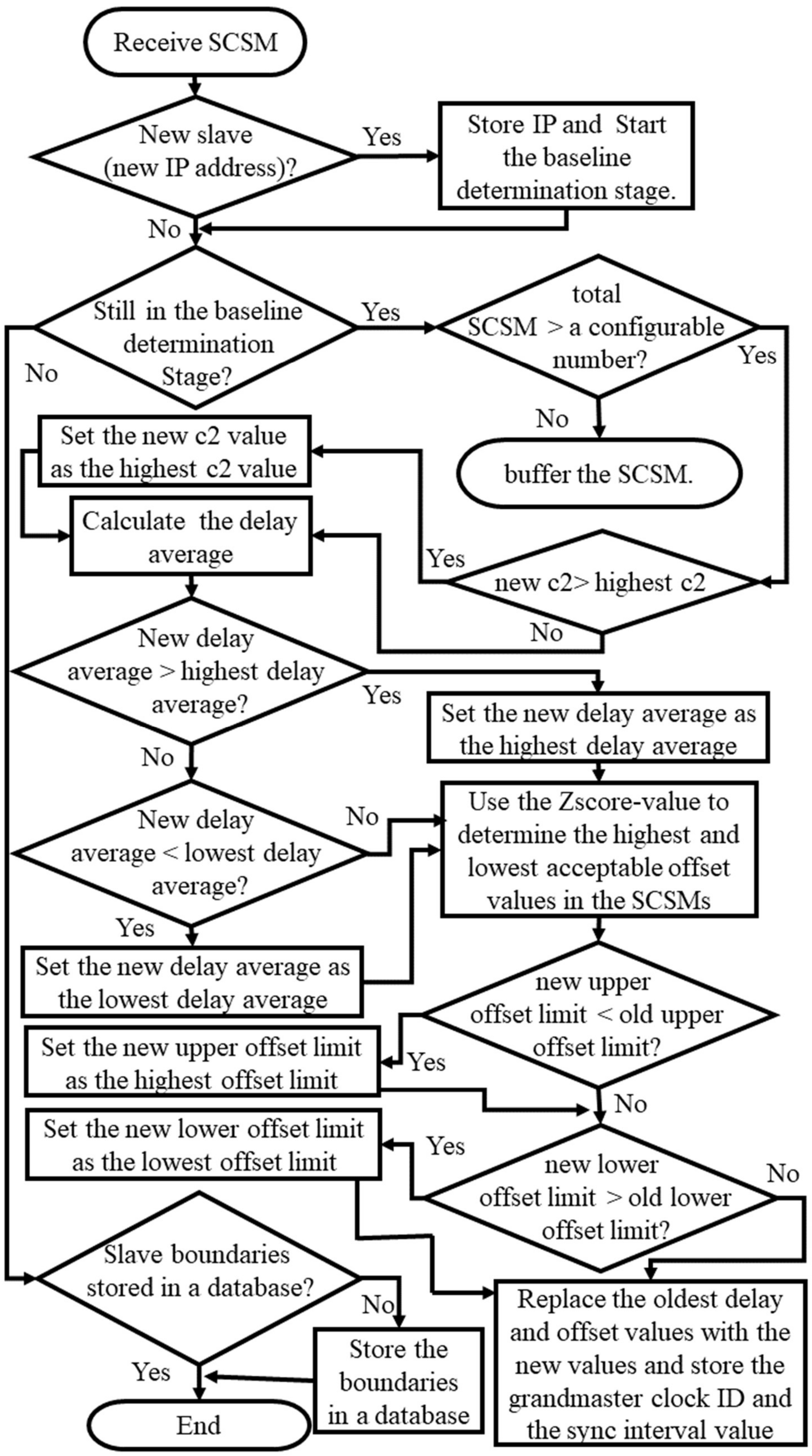
Receive SCSM
New slave (new IP address)?
Yes
Store IP and Start the baseline determination stage.
No
Still in the baseline determination Stage?
Yes
No
total SCSM > a configurable number?
Yes
No
buffer the SCSM.
Set the new c2 value as the highest c2 value
Calculate the delay average
Yes
new c2> highest c2
No
New delay average > highest delay average?
Yes
Set the new delay average as the highest delay average
No
New delay average < lowest delay average?
No
Use the Zscore-value to determine the highest and lowest acceptable offset values in the SCSMs
Yes
Set the new delay average as the lowest delay average
new upper offset limit < old upper offset limit?
Set the new upper offset limit as the highest offset limit
Yes
No
Set the new lower offset limit as the lowest offset limit
Yes
new lower offset limit > old lower offset limit?
No
Slave boundaries stored in a database?
No
Replace the oldest delay and offset values with the new values and store the grandmaster clock ID and the sync interval value
Store the boundaries in a database
Yes
End

Figure 5-1. The baseline determination stage flowchart.

Figure 5-1 shows a summary of the baseline determination stage process and how both TSN types determine the slaves' baselines parameters.

After completing the baseline determination stage for a given slave, all parameters are stored in a hash table with *slaveID* as key, and the TSN enters the slave monitoring stage.

5.2.2 Slave Monitoring Stage

Since all slaves use the same *syncInterval*, SCSMs are expected to arrive at the same average rate, but inter-arrival times may differ because of high slave CPU loads or network congestion. Also, SCSM may be duplicated (i.e., SCSMs that have the same *syncID* and *slaveID*) because of a PTP replay/spoofing attack. Therefore, the TSN needs to handle both as further described below.

Incoming SCSMs are buffered and subsequently processed based on their *syncID*, with X denoting the current synchronization cycle to be considered. Packets that cannot be authenticated or are stale indicate an SCSM modification/fabrication, a replay, or a DoS attack on the TSN itself, and an alarm will be raised.

At the beginning of the monitoring stage, the TSN collects SCSM for three sync intervals starting from the arrival of the first SCSM. At this point (and subsequently in one-synchronization cycle intervals), the TSN moves all authenticated SCSMs from the buffer into different buckets based on their *syncID*. The buckets are as follows:

- staleBucket: contains SCSMs that have stale *syncIDs* (*syncID* < X). A non-empty bucket indicates a very crude replay/spoofing attack on one or more slaves. After determining the potential attacker location using the LCA method (see below), an alarm is raised.

- activeBucket: contains all SCSMs to be processed next (SyncID == X).

- nextBucket: contains SCSMs with *syncID* > X. After the TSN processes the content of activeBucket, its content will be replaced with SCSMs of the next cycle (*syncID* = X+1). If there is no such SCSM in the buffer (the result of a DoS attack or a network-wide synchronization fault), an alarm will be raised. The remaining SCSMs with a larger *syncID* than expected (*syncID* > X + 3) indicate a very crude replay/spoofing attack, resulting in the same action as with staleBucket.

Subsequently, the TSN processes the content of activeBucket using two separate algorithms in parallel as follows:

A. Algorithm 1: Synchronization Attack Detection

The TSN processes the activeBucket content by updating each slaves' delay moving average and extracting *EMT*, c1 (or c2), and t1. While temporary congestion or instability of the network may affect a slave clock synchronization momentarily, a deliberate attack must be executed continuously and affect specific synchronization messages [4], [2]. Therefore, a slave is deemed to be attacked if N_{SCSM} consecutive SCSM show an increased c1 (or c2) value (> c_{max}) and/or if the calculated delay average is outside the established $delay_{max}$ and $delay_{min}$ boundaries. Compared to [9], the TSN decreases the probability of false-positive alarms, as once-off glitches are ignored. Also, the TSN uses *EMT* and t1 to further detect PTP attacks as further described below, which distinguishes between different attacks as follows:

Algorithm 1: Synchronization attack detection

```
1   Input:   activeBucket
2   Output: messages indicating attack types, their root locations, and
            manipulated slaves if any
3   delayMax:         Hash table containing the upper delay averages of all
                      slaves ($delay_{max}$)
4   delayMin:         Hash table containing the lower delay average of
                      each slave ($delay_{min}$)
5   delaySample:      Hash table containing lists of the last $N_B$ delay values
                      of each slave
6   delay:            Hash table containing each slave's delay value
                      extracted from its SCSM
7   cMax:             Hash table containing the largest correction field
                      value of each slave as previously determined ($c_{max}$)
8   c:                Hash table containing the c1/c2 values extracted from
                      its SCSM
9   t1:               Hash table containing current t1 values extracted
                      from SCSMs
10  EMT:              Hash table containing current EMT values extracted
                      from SCSMs
11  IP:               A list containing the slaveID of all registered slaves
12  IP2:              A list containing the slaveID extracted from SCSM
13  NSCSM:            Threshold value ($N_{SCSM}$) as set before
14  delayAverage:     Hash table containing the new delay average of each
                      slave
15  $T\Delta$:               The maximum acceptable difference between EMTs
                      as determined before
16  DoS:              Hash table containing different counters based on
                      different slaveIDs attacked by DoS attacker
                      (initialized with 0s)
17  content1:         Hash table containing different counters based on
                      different slaveID attacked by packet content attacker
                      (t1)
18  content2:         Hash table containing different counters based on
                      different slaveID attacked by packet content attacker
                      (c) (initialized with 0s)
19  replaySpoofing:   Hash table containing different counters based on
                      different slaveID attacked by replay attacker or
                      spoofing attacker
20  Delay:            Hash table containing different counters based on
                      different slaveID attacked by delay attacker
                      (initialized with 0s)
21  unknown:          Hash table containing different counters based on
                      different slaveID attacked by unclassified attack
22  LCA:              A method to determine and print the lowest common
                      ancestor for each attack type
23  baseline1         t1 value of the first slave that is not in replaySpoofing
24  baseline2         EMT value of the first slave that is not in
                      replaySpoofing
```

```
25  Set all values in content1, replaySpoofing, and unknown to 0s
26  for i = 0 until IP.size - 1 do
27      for j = i +1 until IP.size - 1 do
28        if IP[i] = = IP[j] then
29             replaySpoofing[IP[i]] = NSCSM + 1
30        end if
31      end for
32        if replaySpoofing[IP[i]] == 0 then
33           Replace the oldest delay in delaySample[IP[i]] with the
             delay[IP[i]]
34        end if
35  end for
36  for i = 0 until IP.size - 1 do
37      for j = 0 until IP2.size - 1 do
38        if IP[i] = = IP2[j] then
39           DoS[IP[i]] = 0
41               delayAverage[IP[i]]=(sum of delaySample [IP[i]] /N_B)
42           break
43         else if j = = IP2.size - 1 then
44               DoS[IP[i]]++
45         end if
46      end for
47  end for
48  for i = 0 until IP2.size - 1 do
49      if replaySpoofing[IP2[i]] = 0 then
50         baseline1 = t1[IP2[i]]
51         baseline2 = EMT[IP2[i]]
52         break
53      end if
54  end for
55  for i = 0 until IP2.size - 1 do
56    if replaySpoofing[IP2[i]] == 0   then
57     if baseline1 < > t1[IP2[i]]    then
58           content1[IP[i]] = NSCSM + 1
59     end if
60     if c[IP2[i]] > cMax[IP2[i]] then
61           content2[IP2[i]]++
62     else
63           content2[IP2[i]] = 0
64     end if
65      if delayAverage[IP2[i]] > delayMax[IP2[i]] or
           delayAverage[IP2[i]] < delayMin[IP2[i]] then
66         Delay[IP2[i]]++
67      else
68            Delay[IP2[i]] = 0
69      end if
70    if abs (baseline2 – EMT[IP2[i]]) > TΔ then
71           unknown[IP2[i]] = NSCSM + 1
72    end if
```

```
73    end if
74  end for
75  for j = 0 until IP.size - 1 do
76      if replaySpoofing[IP[j]] > NSCSM then
77          print "replay/spoofing attack"
78          print replaySpoofing
79          LCA (replaySpoofing)
80      else if DoS [IP[j]] > NSCSM then
81          print "DoS attack"
82          print DoS
83          LCA (DoS)
84      else if content1[IP[j]] > NSCSM then
85          print "content attack (t1)"
86          print content1
87          LCA (content1)
88      else if content2[IP[j]] > NSCSM then
90          print "content attack (c)"
91          print content2
92          LCA (content2)
93      else if Dealy[IP[j]] > NSCSM then
94          print "delay attack"
95          print delay
96          LCA (delay)
97      else if unknown[IP[j]] > NSCSM then
98          print "unknown attack"
99          print unknown
100         LCA (unknown)
101     end if
102 end for
```

1. Replay and Spoofing Attack

Replay and spoofing attacks have the same impact on slave clocks; therefore, they are merged into one section. Besides the aforementioned *syncID* validation, the TSN raises an alarm if the activeBucket has more than one SCSM with the same *slaveID* (lines 28 – 30 and 76 – 79 in Algorithm 1).

2. Denial of Service (DoS) Attack

A DoS attacker, located between the GM and the targeted slaves, can interfere with clock synchronization in different ways. Firstly changing the *domainNumber*, and *versionPTP* fields in *Sync* messages causes a slave's clock to go into free-running mode [4]. Also, an attacker can intercept all *Sync/Follow_Up* messages and preventing them from reaching their destination. Since SCSM messages are linked to completed synchronization cycles, the TSN keeps track of missing SCSM per slave and raises an alarm if the threshold value N_{SCSM} is exceeded (lines 43 – 45 and 80 – 83 in Algorithm 1). This threshold is required to limit false positive alarms, as PTP uses the UDP transport layer protocol, which does not compensate for occasional packet losses.

3. Packet Content Manipulation Attack

Here the TSN checks and compares synchronization information of all slaves for inconsistencies that point to a content manipulation that took place along the path between the GM and the slaves. This includes differences in t1 values (which must be identical) and excessive c1 or c2 values that are larger than c_{max} (see lines 57 – 64 and 84 – 92 in Algorithm 1). Since intermittent network congestion can make PTP messages reside in a TC longer than usual, the N_{SCSM} threshold reduces the probability of false-positive alarms. Changes of the *versionPTP* or *domainNumber*, which cause a slave clock to go into free-running mode, are already dealt with in the DoS attack section.

4. Packet delay manipulation attack

Local clock offsets correlate to asymmetric uplink/downlink delays, which result in increased delay calculations of an affected slave. Subsequently, the TSN checks for each slave if the measured delay exceeds the previously determined $delay_{max}$ / $delay_{min}$ boundaries for more than N_{SCSM} SCSMs (see line 65 – 69, and 93 – 96 in Algorithm 1), thereby accommodating intermittent network uplink/downlink delay asymmetries again because of traffic volumes.

Both TSN types also provide additional verification to detect any unknown attack which targets a subset of slave clocks, by calculating the difference between the slaves' *EMTs*. If their difference exceeds T_{Δ}, an alarm will be raised immediately (see lines 70 – 72 and 97 – 101 in Algorithm 1). In addition, a Type-B TSN compares its synchronized PTP clock with its local clock and immediately raises an alarm if the difference between its clocks exceeds T_Δ. Detecting an attack using T_Δ means that an attacker has manipulated some slaves using a new attack strategy or has kept an attack always below the N_{SCSM} threshold. This threshold, while making the algorithm more robust to compensate for glitches, also allows an attacker to interfere with the synchronization for just less than N_{SCSM} cycles, for example, by manipulating the c1/c2 value. Once the attack is temporarily paused, and normal c1/c2 values below c_{max} are reported again, the TSN resets its counters. Such an attack will eventually desynchronize a slave clock and affect its *EMT*. Hence, the EMT comparison will detect all attacks that cannot be recognized by other comparisons.

The proposed algorithm is able to detect and track simultaneous attacks on multiple groups of slaves. As a set of attacks may simultaneously target one or more slaves, both TSN types are able to distinguish between them by dedicating a counter for each attack, including the unknown attack and announce the attack when its counter exceeds the N_{SCSM} threshold.

Since slave clocks can be interpreted as leaves in a tree with the GM as the root, and network routers, switches, BC and TC as inner nodes, the lowest common ancestor algorithm (LCA) [100] can be used to identify the closest common ancestor node where the attack most likely took place. Figure 5-2 shows a PTP network that contains one grandmaster, 5 transparent clocks, and

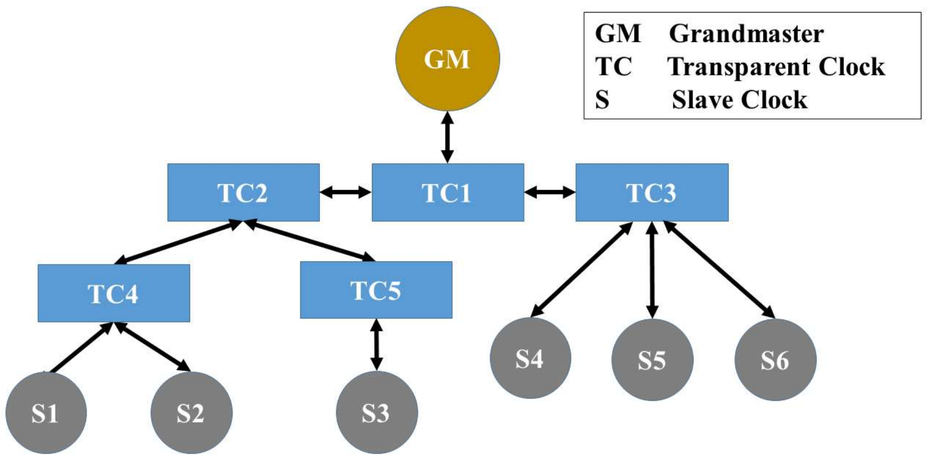

Figure 5-2. a simple PTP network.

6 slave clocks. If a MitM attacker executes a delay attack only on S1 and S2, which is subsequently detected by the TSN, the LCA algorithm will return TC4 as the lowest common ancestor as the probable location of the attacker. However, calculating the LCA during a packet content manipulation attack or unknown attack will return the lowest common ancestor either of the attacked nodes or the unaffected nodes. This is because the TSN picks one (manipulated or not manipulated) SCSM as the baseline for its analysis. Also, if two identical attacks are launched from inner node siblings, the algorithm will incorrectly return their parent as the location of the attacker.

B. Algorithm 2: Time Source Attack Detection

The BMCA attack or the time source degradation attack targets the time synchronization source, affecting all slave clocks. It has been already established that both attacks cause an instantaneous spike of the offset error of manipulated slaves [4], [2]. Hence, the emergence of such behaviour in all slave clocks is a significant indicator for such an attack, as follows:

1. Time Source Degradation Attack

A time source degradation attack may cause an instantaneous and simultaneous spike of the offset error calculated by each slave clock, which correlates to the size, the increment, and the increment interval of the introduced time error [2]. If the cumulated offset error of all slaves as reported by their SCSM exceeds the previously determined threshold in at least one synchronization cycle, an alarm is raised regardless of the TSN type used.

Algorithm 2: Time source attack detection

```
Input:   activeBucket
Output: Attack detected (Y/N)

offset:       Hash table containing slaves' offset values extracted from SCSM
offsetMax:    Hash table containing the maximum offset value of each slave
              ($offset_{max}$)
offsetMin:    Hash table containing the minimum offset value of each slave
              ($offset_{min}$)
counter:      counter counts how many slaves exceed its $offset_{max}$ and $offset_{min}$
GrandID:      Hash table containing slaves' grandID extracted from SCSM
GrandID2:     the grandID as stored in the database
IP:           A list containing the slaveID extracted from SCSM

counter = 0
for each i in IP do
    if offset[i] > offsetMax[i] or
       offset[i] < offsetMin[i]    then
        counter = counter + 1
    end if
    if GrandID[i] < > GrandID2   then
        counter = IP.size + 1
    end if
end for
if counter = IP.size then
   print "Time Source attack or BMCA Attack"
else if counter = IP.size +1 then
   print "BMCA Attack"
end if
```

2. BMCA Attack

In addition to the above method, a BMCA attack may also coincide with a changing *grandID* (the ID of the new GM), which is checked for by the TSN. However, if an attacker is able to spoof the original GM's *grandID*, an attack will be incorrectly labelled as a time source degradation attack.

However, an attacker can exploit the vulnerability of TSN Type A by manipulating the GM timestamp as well as keeping offset errors within the previously established offset boundaries. Such an attack can be implemented by introducing a very small increment to t1 in a long attack interval that makes the calculated offset remain in the normal range. Also, the BMCA attack detection will fail if the rogue grandmaster uses the same identity and network location as the old grandmaster, and both of them have the same accuracy and frequency only at the beginning of the attack (i.e., the rogue master was synchronized to the old grandmaster). Nevertheless, the Type-B TSN has the ability to detect such attacks by comparing its PTP clock, which will be affected by the attack as well, with its local clock and raises an alarm if the difference between them exceeded T_{Δ}.

5.2.3 Comparison with the State-of-the-Art

Table 5-1 shows a comparison of existing attack detection methods in terms of detectable attacks and their limitations. Despite Annex P prong D being able to detect packet delay manipulation attacks, DoS attacks, BMCA attacks, and packet content manipulation attacks, it has limitations in detecting the rest of the PTP attacks such as replay attacks, spoofing attacks, and time source degradation attacks. Also, it does not provide a mechanism to identify an attack-type nor state how an alarm can be triggered. Moussa [6] provides only a mechanism to detect the delay attack while the other attacks are ignored. In addition, the provided mechanism does not cover all delay attack scenarios.

Table 5-1: Efficiencies and limitations of existing attacks detection methods

Proposed Detection Method	Detectable Attacks	Limitations
Annex P (Prong D)	• Delay attack • DoS attack • BMCA attack • Packet content manipulation attack	• Does not provide a mechanism to identify an attack type nor how an alarm can be triggered. • Replay, spoofing, time source degradation attacks are ignored.
Moussa [6]	• Delay attack	• Other attacks are not addressed. • Does not provide a detection method against delaying a PTP packet from slave to master path.
Moussa [8]	• Attacks on GM • Attacks on TC • Attacks on slave • Delay attack	• Relies on a redundant time source that is vulnerable itself to an attack. • Requires sending timestamps from slaves to the redundant time source which may be prone to delay attack or packet content manipulation attack.
Ullmann [9]	Delay attack	• Does not cover other attacks. • Low specificity.

For example, it works efficiently when an attacker delays *Sync* messages but fails when the attacker delays *Delay_Req* messages. Moussa [8] provides a mechanism to detect attacks depending on attacker locations, i.e., GM, TC, slave, and a communication path (e.g., delay attack). However, the provided mechanism relies on a redundant time source that is vulnerable itself to an attack (e.g., an attacker simultaneously manipulates all time references in a network). Moreover, an attacker can manipulate timestamps sent by slaves to the redundant GM in order to keep the attack is hidden. Finally, Ullmann [9] provides only a mechanism to detect the delay attack while the other attacks are ignored. Furthermore, the suggested solution may raise incorrect alarms.

In comparison to these methods, the TSN is able to detect all potential PTP attacks as outlined in Chapter 3, as well as the likely location of the attacker within the network. Also, the TSN itself, as well as all communication with slaves, has been hardened against attacks that compromise the operation of the TSN. Furthermore, the TSN does not necessarily require another time reference to detect all PTP attacks, making it more attack resilient. Finally, the TSN reduces the false positive alarms by using the N_{SCSM} threshold or eliminating them using its own time reference as a benchmark for attacks (the Type-B TSN). In other words, the TSN has overcome all the limitations of the other approaches.

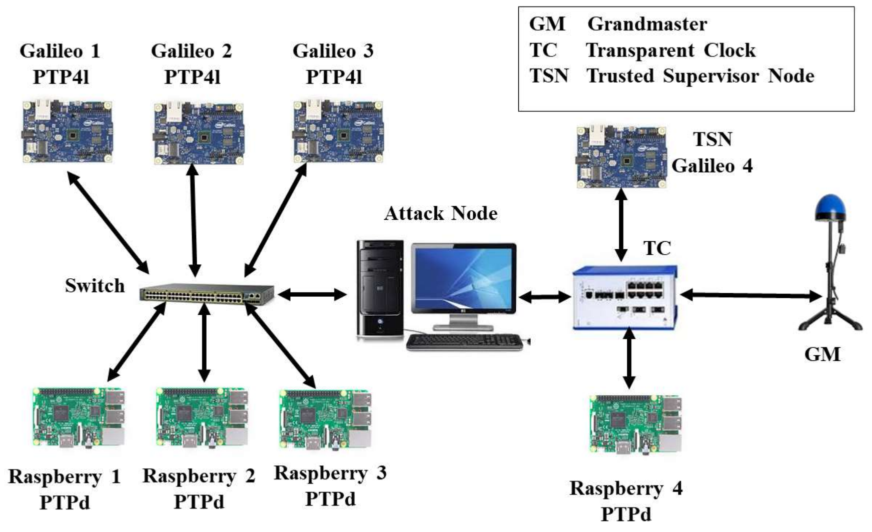

Figure 5-3. The experimental testbed.

5.3 The Testbed and Experiments

5.3.1 Testbed

The experimental testbed to validate the TSN concept is shown in Figure 5-3. It consists of the following hardware and software:

1. Grandmaster Clock: The testbed has only one time source that acts as a grandmaster clock, the Omicron OTMC 100-antenna-integrated PTP Grandmaster Clock.

2. Slave Clocks: There are seven slave clocks in the testbed comprising of three Intel Galileo Gen 1 and four Raspberry Pi 3 model B. The PTP4l daemon is used by Galileo devices, whereas the PTPd is used by Raspberry devices. A separate Galileo 4 implemented a Type-A TSN. Both slaves run a Linux OS.

3. Intermediate Nodes: The testbed consists of both PTP-aware and PTP-unaware intermediate nodes. The latter comprises a Cisco 16-port switch, whereas the former is made of a Hirschmann RSP20 transparent clock.

4. Attack Node: Since manipulating the switches' or TC's firmware to implement an attack is a tedious and complex task, the more flexible option of a programmable MitM attacker has been used. This device is a Linux computer with two network interfaces, which are connected by a bridge using the ebtables tool. A user-space program selectively manipulates incoming PTP messages from the input port before forwarding them to the output port. While the device is placed between the GM and the targeted slaves, it effectively behaves like an attacker located on the network infrastructure hardware, with the exact location being picked by the LCA algorithm [2].

5. Software: Both the slaves' monitor program and the data concentrator program on the TSN are written in C. The slave monitor program works with both aforementioned PTP slave daemons and generates SCSM packets utilizing the payload from grandmaster synchronization messages (i.e., *syncID*, t1, c2, *grandID*, and *syncInterval*), a slaves' IP address, offset and delay values provided by the daemon, and calculates the estimated timestamp *EMT*.

All hardware in the testbed is connected by CAT5e Ethernet cables with a data rate of 1000 Mbps. Since the slaves do not support the P2P delay mechanism and hardware timestamping, the E2E delay mechanism with two-step mode operation are used. In addition, the log sync interval is set to 0 in all conducted experiments, which makes the grandmaster send *Sync/Follow_Up* messages every second. Also, all conducted experiments were conducted using IEEE 1588-2008. It is worth noting that these configurations are used by many profiles, such as the IEEE 1588 default profile [5], SMTPE ST-2059-2 profile [97], and ITU-T Telecom G.8275.2 profile [96].

5.3.2 Experiments

Several experiments were conducted to examine the TSN's ability to detect various PTP attack types using the setup shown in Figure 5-3.

Table 5-2: TSN settings

Fields	Values
Database Building Time C_P	24 Hours
circular buffer size N_B	100 entries
Z-score-boundary to detect suspicious delay values Z_{max}	3.6 standard deviations
The maximum acceptable timestamp difference T_Δ	1 millisecond
SCSM Attack Threshold N_{SCSM}	10 packets

A. Baseline Determination Stage

The TSN requires predefined settings to build and determine the slaves' baselines parameters. Table 5-2 shows the TSN settings that were used for the experiments. A Z-score boundary of 3.6 standard deviations in conjunction with a buffer size of 100 values provided a 1% significance level for acceptable offset values [99]. N_{SCSM} is set to 10.

The testbed was powered up for 15 minutes before the baseline determination stage. As shown in Table 5-3, calculated $delay_{max}$ and $delay_{min}$ values vary depending on the PTP daemon type used, as each daemon type applies a

Table 5-3: Baseline parameters under normal conditions

Devices	$delay_{max}$	$delay_{min}$	$offset_{max}$	$offset_{min}$	c_{max}
Galileo 1	538.2 µs	495.1 µs	122.6 µs	-141.8 µs	11.6 ms
Galileo 2	540.7 µs	496.4 µs	144.2 µs	-102.2 µs	11.6 ms
Galileo 3	542.9 µs	495.4 µs	128.9 µs	-140.5 µs	11.6 ms
Raspberry 1	276.7 µs	256.4 µs	111.6 µs	-109.5 µs	11.6 ms
Raspberry 2	343.3 µs	239.4 µs	83.2 µs	-68.6 µs	11.6 ms
Raspberry 3	295.1 µs	241.8 µs	62.4 µs	-82.4 µs	11.6 ms
Raspberry 4	89.4 µs	82.6 µs	27.2 µs	-23.3 µs	11.6 ms

different delay filter [2], and the slave location within the network (i.e., the presence of a normal switch in the sync packet path causes a larger offset range). c_{max} is determined by the peak traffic volume that passes the TC.

B. Synchronization Attack Detection

In the conducted experiments, the TSN is exposed to all potential PTP attacks as listed in [4], i.e., packet content manipulation, packet removal, packet delay manipulation, spoofing, replay, and Denial of Service (DoS) attacks. All experiments begin with a 5-minute phase of normal time synchronization operation.

1. Packet Content Manipulation Attack

a) The attack node (see Figure 5-3) intercepts all Sync messages and increments t1 by 10 µs. The TSN immediately detects the attack and raises an alarm, as the unaffected Raspberry 4 reports a different t1. Since the TSN picked Galileo 1 as the baseline of t1 comparison, the LCA algorithm incorrectly returns the TC as a probable attack location, as already explained above.

b) The experiment is repeated by adding an incremental 20 µs to the *correctionField* value (c2) with every synchronization cycle (i.e., every second). The value of c2 increases from 3.4 ms to 12.7 ms 465 seconds later, at which point the TSN detected the attack. Here, the c2 value of Galileo 1, Galileo 2, Galileo 3, Raspberry 1, Raspberry 2, and Raspberry 3 (see Table 5-3) are manipulated and subsequently exceed their c_{max} (see Table 5-4), while c2 of Raspberry 4 remains within normal range. The LCA algorithm returns the switch as the source of the attack.

2. Replay and Spoofing Attack

a) The attack node copies each *Sync/ Follow_Up* message and replays them after 1 ms. The TSN immediately identifies the attack as a

Table 5-4: Experiments Results Summary

Devices	Packet Content Manipulation Attack Experiment 1	Packet Content Manipulation Attack Experiment 2	Replay and Spoofing Attack 1	Replay and Spoofing Attack 2	Packet Delay Manipulation Attack	DoS Attack	Unknown Attack	Time Source Attack 1	Time Source Attack2	Time Source Attack 3
	Parameter used to detect the attack									
	t1	c2	slaveID	slaveID	Delay average	SCSM	EMT	offset	offset	offset
Galileo 1	1608980367.866912616	12.7 ms	Duplicated	Duplicated	538.7 µ	No SCSMs Received	1609081350.872350145	215 µs	-43 µs	GM
Galileo 2	1608980367.866912616	12.7 ms	Duplicated	Duplicated	541.1 µ	No SCSMs Received	1609081350.872024267	190 µs	-34 µs	231 µs
Galileo 3	1608980367.866912616	12.7 ms	Duplicated	Duplicated	543.1 µ	No SCSMs Received	1609081350.872141431	257 µs	-13 µs	221 µs
Raspberry 1	1608980367.866912616	12.7 ms	Duplicated	Duplicated	276.8 µ	No SCSMs Received	1609081350.872816678	206 µs	5 µs	258 µs
Raspberry 2	1608980367.866912616	12.7 ms	Duplicated	Duplicated	343.4 µ	No SCSMs Received	1609081350.872546247	214 µs	10 µs	250 µs
Raspberry 3	1608980367.866912616	12.7 ms	Duplicated	Duplicated	295.2 µ	No SCSMs Received	1609081350.872394132	215 µs	7 µs	251 µs
Raspberry 4	1608980367.867112616	9.2 ms	-	-	84.7 µ	SCSMs Received	1609081350.873466305	183 µs	22 µs	290 µs

replay/spoofing attack and determines the attacker's probable location (i.e., the switch), as the TSN received duplicated SCSMs from all slave clocks except for Raspberry 4.

b) In a second experiment, each *Sync*/*Follow_Up* message is copied, the *preciseOriginTimestamp* content of *Follow_Up* messages is increased by 5 ms before the copied messages are sent to their destination, therefore simulating the spoofing attack. Again, the TSN recognizes and classifies the attack correctly, while the LCA algorithm returns the correct attacker location.

3. Packet Delay Manipulation Attack

The attack node intercepts each *Sync* message and holds it for 50 µs. The TSN detects the attack after all buffered delay values (i.e., a circular buffer of size N_B) are affected by the increment (i.e., 100 seconds later) and after N_{SCSM} is exceeded. The new delay averages of Galileo 1, Galileo 2, Galileo 3, Raspberry 1, Raspberry 2, and Raspberry 3 are 538.7 µ, 541.1 µ, 543.1 µ, 276.8 µ, 343.4 µ, and 295.2 µ respectively at the attack detection point (Table 5-4). The LCA correctly identifies the switch as the potential attacker location.

4. Denial of Service (DoS) Attack

The attack node intercepts and removes all *Follow_Up* messages. As a result, the manipulated slaves (excluding Raspberry 4) stop sending their data to the TSN device, which is picked up by the TSN after N_{SCSM} missed synchronization cycles. The LCA algorithm returns correctly the switch as the potential attacker location.

5. Unknown Attack

It is possible that an attacker may develop a PTP attack strategy that fools some parameters used in the detection process (e.g., N_{SCSM}) that leads to hide the attack type. However, all previous attacks are also detected by the *EMT* comparison when Galileo 1, Galileo 2, Galileo 3, Raspberry 1, Raspberry 2, and Raspberry 3 clocks are behind/ahead of the GM/Raspberry 4 clock by T_{Δ}. To highlight the importance of the *EMT* comparison, the *correctionField* manipulation experiment (i.e., packet content manipulation attack – experiment no. 2) is repeated with pausing the attack before the affected SCSMs exceed $N_{SCSM.}$ In this case, the attacker deceives the *correctionField* comparison, but the EMT comparison triggers an alarm when the difference between the Raspberry 4 *EMT* and the other slaves' *EMTs* exceeded T_{Δ} (i.e., 1 ms). Since the TSN picked Galileo 1 as the baseline of the EMT comparison, the LCA algorithms incorrectly return the TC as a probable attack location.

6. Simultaneous Attacks

The experiments of manipulating *correctionField* value (c2) and manipulating the delay average are simultaneously repeated. As a result, the TSN detects that Galileo 1, Galileo 2, Galileo 3, Raspberry 1, Raspberry 2, and Raspberry 3 were manipulated by the packet content manipulation attack and the packet delay manipulation attack via the switch.

C. Time Source Attack Detection

1. Time Source Degradation Attack

a) The attacking node is moved between GM and TC in order to intercept and manipulate the t1 value in all *Follow_Up* messages, which will affect all slave clocks, therefore simulating the time source degradation attack. The attacking node increments the *preciseOriginTimestamp* value by 200 µs every 5 minutes. The TSN immediately detects the attack, as all slaves' offsets exceeded their baselines values (see Table 5-4).

b) The same experiment is repeated with an increment of 10 µs every 15 minutes. In contrast, the TSN is unable to detect this attack as all slave offsets remain with their normal range.

c) Galileo 1 is configured to be the new grandmaster by changing the *priority1* value to be smaller than the original GM *priority1* value. As a result, all slave clocks show abnormal offset values (see Table 5-4), and subsequently, the TSN can detect the attack.

5.4 Summary Results and Discussion

Table 5-4 shows a summary of all conducted attacks. The data shown represent parameter values sent by slaves during a specific synchronization cycle when the TSN detects the attack. All slaves that are located between the switch and the attacking node are manipulated by the packet content manipulation attack, replay/spoofing attack, packet delay manipulation attack, DoS attack, or unknown attack; therefore, they exceed at least one of their baseline values, while the parameters of Raspberry 4, which is out of synchronization packet attacks scope, remain within the normal range. In contrast, all slaves' offsets (including Raspberry 4) are affected by the time source attack 1 and 3. Although the time source attack 2 does also manipulate all slaves, their offsets remain between offset$_{max}$ and offset$_{min}$, since the attacker introduced a small increment to the t1 value (i.e., 10 µs) over a long attack interval (i.e., 15 minutes); therefore it did not create instantaneous spike of offset errors in all slaves. To tackle this problem, the Type-B TSN is introduced. The Type-B TSN calculates the difference between its TSN clocks, i.e., the PTP clock and the local clock, and raises an alarm if the difference exceeded TΔ. As a result, the Type-B TSN is able to detect any attack. Please note that keeping the PTP clock of a Type-B TSN synced to the GM does not limit the TSN role, as the TSN PTP clock will give an opportunity to detect the most silent attack, i.e., time source attack, as described above.

The TSN may fail to identify the type of intermittently paused attacks that never exceed the N_{SCSM} threshold, as done in the unknown attack experiment. Also, the LCA may incorrectly return the probable location of the attacker if a manipulated parameter is chosen as a baseline value, as happened in the packet content manipulation attack (experiment no.1) or the unknown attack experiment.

Also, a high fluctuation of delay values during the baseline determination stage leads to a high delay range, giving an attacker more flexibility to manipulate PTP packets without being detected (i.e., the better managed a network is, the tighter the margins for the delay and offset values are, the smaller the N_{SCSM} threshold can be chosen, making it harder for an attacker to manipulate PTP packets).

However, all of these limitations do not reduce/cancel the TSN's ability (the Type-B in particular) in detecting the PTP attacks, as the Type-B TSN will use its own time reference as a benchmark for attacks.

5.5 Conclusion and Future Work

This chapter introduces two algorithms to detect and determine the type and location of potential PTP attacks using two different types of trusted supervisor nodes (TSN). It follows the ideas of prong D as specified in IEEE 1588-2019 Annex P. Both algorithms collect baseline data over a training period, thereby determining slave-specific boundaries of clock parameters. As such, the proposed methods provide another layer of defence against PTP attacks.

The first algorithm detects attacks that manipulate PTP synchronization packets in transit, thereby manipulating downstream slave clocks that subsequently show different characteristics compared to unaffected nodes. The second algorithm tests for attacks that originate at the time synchronization source. Both algorithms use baseline data from each slave clock that is collected during an initial training phase.

The experiments show the ability of the TSN to detect the packet content manipulation attack, replay and spoofing attack, packet delay manipulation, and DoS attack, including the packet removal attack as an example of a DoS attack.

While the Type-A TSN is able to detect all PTP attacks that affect the parameters sent through SCSM, the Type-B TSN can detect abnormalities even if the sent parameters remain within their normal range, allowing to detect the most silent attack, i.e., the time source attack.

The TSN uses the lowest common ancestor method (LCA) to determine the probable location of the attacker. However, the execution of LCA may return an incorrect result with a packet content manipulation attack (t1 attack) and an unknown attack.

Instable delay paths between the GM and its slaves results in high fluctuations in delay values and therefore increasing the delay and offset ranges for a slave. Such behaviour delays the detection of a delay attack or makes all time source attacks undetectable by the Type-A TSN. So, the smaller the delay/offset range, the earlier the attack is detected.

While the TSN provides only detection methods, further lines of defence must be considered to prevent such attacks, for example, by hardening network infrastructure components via TPM.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The primary aim of this research is to analyze the PTP internal attacks and provide a detection system against these attacks. In this chapter, we summarize the main contributions of this research.

First of all, this thesis provides a comprehensive analysis of strategies for advanced persistent threats to PTP infrastructure, possible attacker locations, and the impact on a clock and network synchronization in the presence of security protocol extensions, infrastructure redundancy, and protocol redundancy. The analysis shows that a sophisticated attacker has a range of methodologies to compromise a PTP network. Secondly, a set of experiments were conducted to demonstrate the impact of PTP attacks, using a fully programable and customizable man in the middle device, thereby considering the two most popular PTP slave daemons PTPd and PTP4l. As a result, it determined suitable attack patterns and parameters to compromise the time synchronization covertly. Finally, this thesis also contributes to the detection of PTP attacks and the attacker location using a trusted supervisor node (TSN). This node collects and analyses delay and offset outputs of monitored slaves as well as timestamps sent by Sync messages, allowing it to detect abnormal patterns in the data provided. Depending on the attack scope, the TSN uses two different algorithms to detect all PTP attacks. This proposal is in line with the prong D as specified in IEEE 1588-2019.

We have made a valuable contribution to answering the research questions as they are listed in Chapter 1. For the first research question, the potential PTP attack strategies, as well as the existing PTP security extensions, are covered in Chapter 3 and summarized in Section 6.1.1. In addition, Chapter 4 answers the second research question and Section 6.1.2 summarized the findings.

Finally, the third question is answered and summarized in Chapter 5 and Section 6.1.3, respectively.

6.1.1 Precision Time Protocol Attack Strategies and their Resistance to Existing Security Extensions

This thesis focused on the internal PTP attack, which can have destructive consequences for time-sensitive networks. Traditional security protocol extensions (Annex K, MACsec, and IPsec) are insufficient to prevent such attacks, while infrastructure enhancements (multiple paths, redundant grandmaster, and protocol redundancy) can be compromised and do not provide a sound and gold-plated way to protect PTP networks from manipulation either; therefore Chapter 3 described potential internal attack strategies and their impact on time synchronization. Also, it gave different examples of how an internal attacker can compromise a PTP network.

PTP attacks have different impacts on PTP network endpoints:

Firstly, an attacker can desynchronize all PTP clocks (this is having the highest impact on a PTP network) by using a BMCA or time source degradation attack. These attacks exploit the grandmaster role as a time source and propagate an inaccurate time reference to all other clocks in a network. This attack needs to be persistent to continuously manipulate PTP clocks. For example, in the BMCA attack, an attacker must keep the rogue master as the best clock in the network over time. In case of a time source degradation attack, an attacker must continuously generate and transmit inaccurate timestamps to all slaves.

Secondly, an attacker can manipulate a subset of PTP clocks by using packet content manipulation, packet removal, or packet delay manipulation attacks. Such attacks need to be persistent as well to continuously manipulate PTP clocks.

Thirdly, an attacker can make the clock synchronization unstable by using master spoofing or replay attacks. When one of these attacks occurs, a slave may receive valid and fresh sync messages as well as fake sync (spoofed or replayed) messages over time, making it swing between a synced and an unsynced state to the master.

Fourthly, an attacker can target a single PTP clock and manipulate it by using the slave spoofing attack. This has the lowest impact on a PTP network. Here the attack must also be persistent to continuously compromise the slave clock.

Finally, persistent DoS attacks force affected slave clocks to go into free-running mode. DoS attacks can be relatively easy detected by a network or a slave, as they affect all network services.

6.1.2 Cyber Attacks on Precision Time Protocol Networks—A Case Study

In addition, this thesis focused on how PTP daemons (i.e., PTP4l and PTPd) respond to the PTP attacks by simulating the internal man in the middle attacker. Although both PTP daemons implement the synchronization process as specified in the standard of PTP, they showed a different response to some PTP attacks. An asymmetric delay, a consecutive large symmetric delay, or changing the protocol packet in transit (timestamp or correction field) can degrade the protocol accuracy. In the first step, a set of experiments are conducted that characterize typical offsets, delays, and drifts of local clocks on embedded systems deployed in a testbed. Secondly, two types of internal attacks were conducted (packet propagation attack and time reference attack). The data showed drift rates, offset, and delay distributions, which can be used to identify malicious clock de-synchronization caused, for example, by the aforementioned attacks. The offset distribution can be easily drifted if an attacker succeeds in making a slave slew with the maximum frequency. This could happen if the attacker introduced a high amount of delay between the master and its slaves or making a significant change in the master timestamp or the correction field of *Follow_Up* and *Delay_Resp* messages. However, adding small asymmetry delay or small change to the timestamp, correction field of *Follow_Up*, and/or *Delay_Resp* messages can gradually de-synchronize clocks over an extended period of time, result in clock offsets that might still fall within the expected range of offset measurements. In contrast, the delay measurements showed great efficacy in detecting the aforementioned attacks.

6.1.3 A Security Enhancement of the Precision Time Protocol using a Trusted Supervisor Node

Finally, we introduce a new PTP security model that allows the detection of all PTP attacks such as replay attack, spoofing attack, DoS attack, removal attack, packet content manipulation attack, delay attack, time source attack, and BMCA attack. It is loosely based on the recommendations of the aforementioned Prong D.

The proposed model relies on a monitor unit called the trusted supervisor node (TSN), which is able to collect synchronization data provided by a large number of slave devices and use statistical variations in this data to detect a PTP attack. The TSN introduces two algorithms (i.e., synchronization attack detection and time source attack detection) to detect and determine the type and location of potential PTP attacks using two different types of trusted supervisor nodes (TSN) (i.e., Type-A TSN and Type-B TSN). The synchronization attack detection algorithm is dedicated to detecting the attack that manipulates PTP messages after they are sent by the grandmaster. Also, the algorithm is able to detect attacks based on false PTP messages (e.g., spoofing attacks). In contrast, the time source attack detection algorithm is responsible for detecting the attack that results from an inaccurate or compromised time source. The conducted experiments showed the efficiency of the TSN in detecting all PTP attacks.

6.2 Future Work

In this thesis, we focused on highlighting the drawbacks of existing PTP security extensions by providing a new model for detecting PTP attacks. Also, this thesis presented the possible effects of the PTP attacks on a group of slaves with the existence of security extensions in theory. More experiments involving prong A, prong B, prong C, and TSN in a single testbed and monitoring the efficiency of this approach in preventing and detecting the PTP attacks are required.

Further research is also required to enhance PTP security by introducing new advanced methodologies to make PTP devices or PTP networks more resistant to PTP attacks.

6.2.1 Trusted Platform Module

A Trusted Platform Module (TPM) is a special chip deployed on an endpoint device. It stores encryption keys that are used for hardware validation in a host system. The module is used to authenticate a node and its firmware when it is booted, so it can detect malware infections. By doing so, it prevents, for example, a virus-infected clock from becoming a rogue master clock [14], [62].

The TSN will be required to have TSN implemented to protect its integrity. However, in order to avoid the hostile infiltration and takeover of other PTP endpoints, TSN should become a standard feature of all devices, including slave clocks, switches, GM, TC and BC. While this is a costly endeavour, it provides another layer of security.

6.2.2 Digital Certificates and Public Key Infrastructures

A rogue master attack can also be thwarted through the use of public key infrastructures (PKI) and digital certificates (DC). In this case, each clock has a digital certificate containing its characteristics. The certificate is underwritten by a trusted third party. Certificates are exchanged and mutually validated with announcing messages, so each clock can determine if a received certificate is authentic and from the sender. In addition, applying such validation prevents the rogue master from generating counterfeit clock certificates and therefore manipulates the BMC algorithm [14], [62]. In other words, all PTP devices are required to secure their communication as TSN do, using the PKI and DC.

6.2.3 Network Intrusion Detection System

A network intrusion detection system (NIDS) has the ability to monitor data communication between nodes in a time synchronization network. NIDS are placed at strategic points within a network that can detect DoS attack, packet removal attack, packet delay manipulation attack, and time source attack [69]. Therefore, NIDS can identify rogue device activity and raise an alarm [14], [62].

6.2.4 Commercial Exploitation of Research

The cybersecurity industry is a fast-growing sector with an annual global turnover of USD 153.16 billion in 2020. New defense software and hardware technologies emerge constatntly, addressing specific needs of the ICT industry (e.g. cloud security). While this research covers the niche application area of time synchronization security, it provides a promising solution to tackle a threat landscape with potentially devastating consequences for industries and consumers. Therefore, it is planned to seek a collaboration with either a PTP infrastructure provider or an industry partner (e.g. Oil and Gas) to refine and extend the proposed technology into a viable product. If an interested partner can be identified, the most suitable funding model will be determined. In an Irish context, this could be an industry co-funded Innovation Partnership, or a government funded Commercialisation Project.

References

[1] A. Pandey, "An Analysis of Precision and Security of the IEEE 1588v2 Precision Time Protocol," Discipline of Information Technology, NUIG, 2015.

[2] W. Alghamdi and M. Schukat, "Cyber Attacks on Precision Time Protocol Networks—A Case Study," *Electronics,* vol. 9, p. 1398, 2020.

[3] C. S. Ltd. (2009). *Implementing IEEE 1588v2 for use in the mobile backhaul*. Available: http://opencores.org/websvn,filedetails?repname=ha1588&path=%2Fha1588%2Ftrunk%2Fdoc%2Ftool%2Fptpv2_timing_analyzer%2FCalnex_-_IEEE_1588v2_PTP.pdf

[4] W. Alghamdi and M. Schukat, "Precision time protocol attack strategies and their resistance to existing security extensions," *Cybersecurity,* vol. 4, p. 12, 2021/04/01 2021.

[5] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002),* pp. 1-269, 2008.

[6] B. Moussa, M. Debbabi, and C. Assi, "A Detection and Mitigation Model for PTP Delay Attack in an IEC 61850 Substation," *IEEE Transactions on Smart Grid,* vol. 9, pp. 3954-3965, 2018.

[7] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008),* pp. 1-499, 2020.

[8] B. Moussa, M. Kassouf, R. Hadjidj, M. Debbabi, and C. Assi, "An Extension to the Precision Time Protocol (PTP) to Enable the Detection of Cyber Attacks," *IEEE Transactions on Industrial Informatics,* vol. 16, pp. 18-27, 2020.

[9] M. Ullmann and M. Vögeler, "Delay attacks—Implication on NTP and PTP time synchronization," in *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2009, pp. 1-6.

[10] K. Harris, "An application of IEEE 1588 to industrial automation," *Precision Clock Synchronization for Measurement, Control and Communication,* pp. 71-76, 2008.

[11] A. Dreher and D. Mohl. *Precision Clock Synchronization*. Available: http://www.industrialnetworking.com/pdf/Hirschmann_IEEE_1588.pdf

[12] J. Shannon, "Improved techniques for time synchronisation over WiFi and wireless sensor networks," 2013.

[13] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications,* vol. 39, pp. 1482-1493, 1991.

[14] W. Alghamdi and M. Schukat, "Advanced methodologies to deter internal attacks in PTP time synchronization networks," in *2017 28th Irish Signals and Systems Conference (ISSC)*, 2017, pp. 1-6.

[15] E. Technology. (2017, 10/2/2017). *Precision Time Protocol*. Available: https://www.endruntechnologies.com/pdf/PTP-1588.pdf

[16] P. V. Estrela and L. Bonebakker, "Challenges deploying PTPv2 in a global financial company," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2012 International IEEE Symposium on*, 2012, pp. 1-6.

[17] Wikipedia. (2017, 10/2/2017). *Precision Time Protocol*. Available: https://en.wikipedia.org/wiki/Precision_Time_Protocol

[18] W. Alghamdi and M. Schukat, "Practical Implementation of APTs on PTP Time Synchronisation Networks," in *2020 31st Irish Signals and Systems Conference (ISSC)*, 2020, pp. 1-5.

[19] E. Baize, "Developing Secure Products in the Age of Advanced Persistent Threats," *IEEE Security & Privacy,* vol. 10, pp. 88-92, 2012.

[20] T. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer,* vol. 44, pp. 91-93, 2011.

[21] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy,* vol. 9, pp. 49-51, 2011.

[22] E. Itkin and A. Wool, "A Security Analysis and Revised Security Extension for the Precision Time Protocol," *IEEE Transactions on Dependable and Secure Computing,* vol. 17, pp. 22-34, 2020.

[23] T. Mizrahi, "Time synchronization security using IPsec and MACsec," in *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication*, 2011, pp. 38-43.

[24] A. Shpiner, Y. Revah, and T. Mizrahi, "Multi-path Time Protocols," in *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*, 2013, pp. 1-6.

[25] T. Koskiahde and J. Kujala, "PTP monitoring in redundant network," in *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2016, pp. 1-5.

[26] 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)*, 2015, pp. 111-114.

[27] P. V. Estrela, S. Neusüß, and W. Owczarek, "Using a multi-source NTP watchdog to increase the robustness of PTPv2 in financial industry networks," in *Precision Clock Synchronization for*

Measurement, Control, and Communication (ISPCS), 2014 IEEE International Symposium on, 2014, pp. 87-92.

[28] W. Alghamdi and M. Schukat, "Slave Clock Responses to Precision Time Protocol Attacks: A Case Study," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2020, pp. 1-4.

[29] R. Cohen "Precision Time Protocol."

[30] Y. Mingwu and H. Zhenlin, "An enhanced end-to-end transparent clock mechanism for the kernel-based virtual machines," in *Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), 2017 IEEE International Symposium on*, 2017, pp. 1-5.

[31] I. Freire, I. Sousa, A. Klautau, I. Almeida, C. Lu, and M. Berg, "Analysis and Evaluation of End-to-End PTP Synchronization for Ethernet-Based Fronthaul," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1-6.

[32] A. Corporation. (2016). *Altera 1588 System Solution*. Available: https://www.altera.com/documentation/kly1423707073680.html

[33] G. M. Garner, "IEEE 1588 Version 2," *ISPCS Ann Arbor,* vol. 8, pp. 1-89, 2008.

[34] Z. Chaloupka, N. Alsindi, and J. Aweya, "Clock Synchronization Over Communication Paths With Queue-Induced Delay Asymmetries," *IEEE Communications Letters,* vol. 18, pp. 1551-1554, 2014.

[35] H. Weibel, "IEEE 1588 tutorial," in *Conference on IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Gaithersburg, USA, 2-4 October 2006*, 2006.

[36] G. M. Garner, "Description of Use of IEEE 1588 Peer-to-Peer Transparent Clocks in A/V Bridging Networks," ed: Revision, 2006.

[37] J. Wu and R. Poloquin, "Synchronizing device clocks using IEEE 1588 and Blackfin embedded processors," *19 Automobile Tail-Lamp and Brake-Lamp Controller,* p. 6, 2012.

[38] W. Alghamdi and M. Schukat, "A Security Enhancement of Precision Time Protocol using Trusted Supervision Node," *Engineering Science and Technology, an International Journal, under review,* 2021.

[39] R. Klecka. (2015). *Fundamentals of Precision Time Protocol*. Available: https://www.odva.org/Portals/0/Library/Conference/2015_ODVA_Conference_Klecka_Fundamentals-of-PTP.pdf

[40] A. Diarra, T. Hogenmueller, A. Zimmermann, A. Grzemba, and U. A. Khan, "Improved clock synchronization start-up time for Ethernet AVB-based in-vehicle networks," in *Emerging Technologies &*

Factory Automation (ETFA), 2015 IEEE 20th Conference on, 2015, pp. 1-8.

[41] (2020, 17/04/2020). *List of PTP implementations*. Available: https://en.wikipedia.org/wiki/List_of_PTP_implementations

[42] (18/04/2020). *The Linux PTP Project*. Available: http://linuxptp.sourceforge.net/

[43] A. Kempen, S. Kreuzer, G. Neville, and W. Owczarek. (18/04/2020). *Precision Time Protocol daemon*. Available: https://manpages.debian.org/testing/ptpd/ptpd.8.en.html

[44] Github. (2018). *PTPd*. Available: https://github.com/ptpd

[45] P. Ohly, D. N. Lombard, and K. B. Stanton, "Hardware assisted precision time protocol. Design and case study," in *LCI Intl. Conf. on High-Perf. Clustered Comp*, 2008.

[46] K. Correll, N. Barendt, and M. Branicky, "Design considerations for software only implementations of the IEEE 1588 precision time protocol," in *Conference on IEEE*, 2005, pp. 11-15.

[47] Github. (2019). *linuxptp*. Available: https://github.com/openil/linuxptp

[48] Y. Xinhua, Z. Shike, and L. Hongwei, "Research and implementation of precise time synchronization system of microgrid based on IEEE 1588," in *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2016, pp. 258-261.

[49] M. Ussath, D. Jaeger, F. Cheng, and C. Meinel, "Advanced persistent threats: Behind the scenes," in *2016 Annual Conference on Information Science and Systems (CISS)*, 2016, pp. 181-186.

[50] S. Al-Rabiaah, "The “Stuxnet” virus of 2010 as an example of a “APT” and its “Recent” variances," in *2018 21st Saudi Computer Society National Computer Conference (NCC)*, 2018, pp. 1-5.

[51] P. Hu, H. Li, H. Fu, D. Cansever, and P. Mohapatra, "Dynamic defense strategy against advanced persistent threat with insiders," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 747-755.

[52] D. X. Cho and H. H. Nam, "A Method of Monitoring and Detecting APT Attacks Based on Unknown Domains," *Procedia Computer Science,* vol. 150, pp. 316-323, 2019.

[53] S. Quintero-Bonilla and A. Martín del Rey, "A New Proposal on the Advanced Persistent Threat: A Survey," *Applied Sciences,* vol. 10, p. 3874, 2020.

[54] J. Shannon, H. Melvin, and A. G. Ruzzelli, "Dynamic flooding time synchronisation protocol for WSNs," in *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012, pp. 365-371.

[55] J. Eidson and K. Lee, "IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control

systems," in *Sensors for Industry Conference, 2002. 2nd ISA/IEEE*, 2002, pp. 98-105.

[56] M. Schukat, "Enterprise Time Distribution: NTP and PTP," securities technology analysis center2016.

[57] Wikipedia. (2018, 15-1-2018). *Leap second*. Available: https://en.wikipedia.org/wiki/Leap_second

[58] N. T. S. T. Force, "Time Synchronization in the Electric Power System," Technical report, North American Synchrophasor Initiative2017.

[59] E. D. Knapp and J. T. Langill, "Industrial network security: securing critical infrastructure networks for smart grid, SCADA, and other industrial control systems," 2015.

[60] J. R. Vacca, *Computer and Information Security Handbook*: Morgan Kaufmann, 2017.

[61] T. Mizrahi, "Security requirements of time protocols in packet switched networks," 2014.

[62] M. Schukat and J. Desbonnet, "Securing Time Protocols in Packet Switched Networks," 2015.

[63] B. Hirschler and A. Treytl, "Validation and verification of IEEE 1588 Annex K," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on*, 2011, pp. 44-49.

[64] R. Cohen, "PTP Security Tutorial," 2007.

[65] C. Önal and H. Kirrmann, "Security improvements for IEEE 1588 Annex K: Implementation and comparison of authentication codes," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2012 International IEEE Symposium on*, 2012, pp. 1-6.

[66] Y. Pathan, A. Dalvi, A. Pillai, D. Patil, and D. Reed, "Analysis of selective packet delay attack on IEEE 1588 Precision Time Protocol," 2014.

[67] S. Kent and K. Seo, "Security architecture for the internet protocol," 2070-1721, 2005.

[68] A. Treytl and B. Hirschler, "Securing IEEE 1588 by IPsec tunnels - An analysis," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2010, pp. 83-90.

[69] M. Schukat, "Network Timing," white paper, NUIG2015.

[70] D. T. Bui, M. Le Pallec, and N. Le Sauze, "Protocol agnostic On-Path Supports," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2010 International IEEE Symposium on*, 2010, pp. 129-134.

[71] D. Chen, "Secure 1588 in HeNB / Femtocell Application," presented at the Time & Sync in Telecoms, Lisbon, 2013.

[72] L. Ellegaard, "PTP security using MACsec," 2014.

[73] T. Joergensen, "Secure PTP - Protecting PTP with MACsec without losing accuracy," 2014.

[74] "IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security," *IEEE Std 802.1AE-2006,* pp. 1-150, 2006.

[75] D. Maftei, R. Bartos, B. Noseworthy, and T. Carlin, "Implementing proposed IEEE 1588 integrated security mechanism," in *2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2018, pp. 1-6.

[76] "Cyber-Security of Time-aware Cyber-Physical Systems," in *Re-Industrialisation of the EU*, 2016.

[77] E. T. Limited. (2016). *IEEE 1588 PTP clock synchronization over a WAN backbone*. Available: https://www.endace.com/ptp-timing-whitepaper.pdf

[78] T. Mizrahi, "A game theoretic analysis of delay attacks against time synchronization protocols," in *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*, 2012, pp. 1-6.

[79] N. Moreira, J. Lázaro, J. Jimenez, M. Idirin, and A. Astarloa, "Security mechanisms to protect IEEE 1588 synchronization: State of the art and trends," in *Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), 2015 IEEE International Symposium on*, 2015, pp. 115-120.

[80] H. Kirrmann, M. Hansson, and P. Muri, "IEC 62439 PRP: Bumpless recovery for highly available, hard real-time industrial networks," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, 2007, pp. 1396-1399.

[81] M. Han and P. Crossley, "Vulnerability of IEEE 1588 under Time Synchronization Attacks," in *2019 IEEE Power & Energy Society General Meeting (PESGM)*, 2019, pp. 1-5.

[82] J. Neyer, L. Gassner, and C. Marinescu, "Redundant Schemes or How to Counter the Delay Attack on Time Synchronization Protocols," in *2019 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2019, pp. 1-6.

[83] T. Murakami and Y. Horiuchi, "A master redundancy technique in IEEE 1588 synchronization with a link congestion estimation," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2010 International IEEE Symposium on*, 2010, pp. 30-35.

[84] K. Dadheech, A. Choudhary, and G. Bhatia, "De-Militarized Zone: A Next Level to Network Security," in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, pp. 595-600.

[85] E. Shereen, F. Bitard, G. Dán, T. Sel, and S. Fries, "Next Steps in Security for Time Synchronization: Experiences from implementing IEEE 1588 v2.1," in *2019 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2019, pp. 1-6.

[86] K. O. Donoghue, D. Sibold, and S. Fries, "New security mechanisms for network time synchronization protocols," in *2017 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2017, pp. 1-6.

[87] L. Narula and T. E. Humphreys, "Requirements for Secure Clock Synchronization," *IEEE Journal of Selected Topics in Signal Processing,* vol. 12, pp. 749-762, 2018.

[88] T. Mizrahi, "Time Synchronization Security using IPsec and MACsec," in *Israel Networking Seminar*, 2012.

[89] I. Song, "IEEE 1588 & PTP USING EMBEDDED LINUX SYSTEMS," linux foundation2015.

[90] C. DeCusatis, R. M. Lynch, W. Kluge, J. Houston, P. Wojciak, and S. Guendert, "Impact of Cyberattacks on Precision Time Protocol," *IEEE Transactions on Instrumentation and Measurement,* pp. 1-1, 2019.

[91] W. Stallings, *Cryptography and network security: principles and practices*: Pearson Education India, 2006.

[92] A. Treytl, G. Gaderer, B. Hirschler, and R. Cohen, "Traps and pitfalls in secure clock synchronization," in *Precision Clock Synchronization for Measurement, Control and Communication, 2007. ISPCS 2007. IEEE International Symposium on*, 2007, pp. 18-24.

[93] J.-C. Tournier and O. Goerlitz, "Strategies to secure the IEEE 1588 protocol in digital substation automation," in *Critical Infrastructures, 2009. CRIS 2009. Fourth International Conference on*, 2009, pp. 1-8.

[94] W. Alghamdi and M. Schukat, "Advanced Persistent Threats to Precision Time Protocol Networks A Vulnerability Analysis," *under review,* 2020.

[95] M. Schukat, "PTP and Security: Understanding Risks and best Practices," 2016.

[96] ITU. (2020, 6/2020). *G.8275.2: Precision time protocol telecom profile for time/phase synchronization with partial timing support from the network.* Available: https://www.itu.int/rec/T-REC-G.8275.2-202003-I/en

[97] "ST 2059-2:2015 - SMPTE Standard - SMPTE Profile for Use of IEEE-1588 Precision Time Protocol in Professional Broadcast Applications," *ST 2059-2:2015,* pp. 1-19, 2015.

[98] M. Schukat, P. Castilla, H. Melvin, and F. Hu, "Trust and Trust Models for the IoT," in *Security and Privacy in Internet of Things (IoTs): Models, Algorithms, and Implementations*, ed: CRC Press, 2016.

[99] F. E. Grubbs and G. Beck, "Extension of sample sizes and percentage points for significance tests of outlying observations," *Technometrics,* vol. 14, pp. 847-854, 1972.

[100] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin, "Lowest common ancestors in trees and directed acyclic graphs," *Journal of Algorithms,* vol. 57, pp. 75-94, 2005.

[101] M. Moradi and A. H. Jahangir, "A new delay attack detection algorithm for PTP network in power substation," Int. J. Electr. Power Energy Syst., vol. 133, p. 107226, 2021.

[102] Elena Lisova, Marina Gutiérrez, Wilfried Steiner, Elisabeth Uhlemann, Johan Åkerberg, Radu Dobrin, Mats Björkman, "Protecting Clock Synchronization: Adversary Detection through Network Monitoring", Journal of Electrical and Computer Engineering, vol. 2016, Article ID 6297476, 13 pages, 2016. https://doi.org/10.1155/2016/6297476