Time Sensitive Networking over 5G Networks

Dancun Omondi Ogenda

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 1.08.2022

Supervisor

Professor Raimo Kantola

Advisor

Jose Costa-Requena Dr.Sc. (Tech.)

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Dancun Omondi Ogenda

Title Time Sensitive Networking over 5G Networks

Degree programme Computer, Communication and Information Sciences

Major Communications Engineering - Wireless Networks **Code of major** ELEC3029

Supervisor Professor Raimo Kantola

Advisor Jose Costa-Requena Dr.Sc. (Tech.)

Date 1.08.2022 **Number of pages** 73 **Language** English

Abstract

Time-Sensitive Networking (TSN IEEE 802.1Q), is an Ethernet technology that provides deterministic messaging on standard Ethernet. When centrally managed, the TSN technology offers the capability of guaranteed delivery of messages with reduced jitter. TSN uses time-scheduling in providing deterministic communications and works at Layer 2 (L2) of the Open System Interconnection. The advantage of TSN working at L2 is that TSN entities (switches and bridges) only need the information contained in Ethernet headers to make forwarding decisions. In addition, the information carried in Ethernet frame payloads does not have to be limited to IP only, making TSN applicable in industrial applications with different application payloads.

The goal of the thesis was to come up with a state-of-the-art design of IEEE TSN modules. This goal involved designing a topology for testing TSN, prototyping the TSN modules, and testing the modules when completed. The thesis evaluates how the developed TSN module's performance compares to IEEE WG set standards.

I carried out the experimentation based on the IEEE Working Group (WG) recommendations and publications which provided the necessary modifications to Precision Time Protocol version 2 (PTPv2) regarding packets that needed to be modified to develop generalized Precision Time Protocols (gPTP). Before entering and exiting the 5G System (5GS), the gPTP messages are changed. These encompass all the needed packet header modifications and necessary calculations to achieve the synchronization accuracies of 900 nanoseconds as stipulated by the IEEE 802.1AS standard.

The project's findings were that the functionalities stipulated by the IEEE TSN WG were possible to implement and even achieve synchronization between the different TSN modules. The thesis did not accomplish the synchronization accuracy levels specified by the IEEE TSN. This low synchronization accuracy level was understandable, considering that the 5GS and equipment needed to improve performance were missing. The thesis evaluates the exactness with which gPTP packets arriving at the TSN modules could be detected, captured, modified, and sent to end stations successfully and provides an in-depth explanation for why the synchronization accuracy levels achieved were low.

Keywords time, synchronization, rate ratio , time offset

Preface

I wish to thank Prof. Raimo Kantola and Dr. Jose Costa-Requena for their guidance and counseling while working on my thesis. The patience they offered me was extraordinary, and for that, I am grateful. The thesis topic they chose was exciting, and it has been a pleasure working with both to the end of this thesis.

Of utmost importance is my girlfriend, Greta, who has been by my side and offered words of encouragement whenever I needed extra motivation to get me going with the thesis.

Lastly, I offer my gratitude to every teacher who has ever been part of my life for the whole journey. I appreciate each of you, and I believe that without a single out of the entire lot, I would not have reached this level today.

Dancun Ogenda

Contents

Abstract **3**

Preface **4**

Contents **5**

Abbreviations and Acronyms **7**

1 Introduction **9**
1.1 Background . . . 9
1.2 Motivation . . . 10
1.3 Goals . . . 10
1.4 Implementation . . . 10
1.5 Structure of the thesis . . . 11

2 Background on Time Sensitive Networks **12**
2.1 IEEE 802.1 AS TSN . . . 12
2.2 IEEE 802.1 TSN Toolbox . . . 13
2.2.1 TSN Time Synchronization . . . 13
2.2.2 Bounded low latency . . . 14
2.2.3 Reliability . . . 15
2.2.4 Resource management . . . 15
2.3 Differences between PTP and gPTP . . . 16
2.4 TSN architecture for a packet network . . . 17
2.4.1 Important TSN entities . . . 17
2.4.2 Single gPTP domain implementing TSN . . . 18
2.4.3 Multiple gPTP domains implementing TSN . . . 19
2.4.4 TSN with redundant paths/GM PTP instances . . . 20
2.5 Time Sync . . . 22
2.5.1 Measuring delay . . . 23
2.5.2 Logical syntonization . . . 25
2.5.3 GM selection and network establishment . . . 26

3 Time Sensitive Networks in 5GS **27**
3.1 5GS-TSN architecture . . . 27
3.2 Integration concept of 5GS with the IEEE TSN . . . 29
3.2.1 TSN translators definitions and functions in 5GS and IEEE 802.1AS . . . 29
3.2.2 The elementary procedures between the AF and DS-TT . . . 30
3.2.3 The elementary procedures between the AF and NW-TT . . . 32
3.2.4 Port management procedure as initiated by the NW-TT . . . 34
3.3 TSN prototypes for LAN and 5G networks . . . 35

4 Environment requirements **38**
4.1 Hardware timestamping . . . 38
4.2 L3 TSN Switches . . . 39
4.3 TSN traffic separation between networks . . . 39
4.4 Open-Source GM Configurations . . . 39
4.5 Summary of necessary equipment . . . 40
4.6 Methods used in thesis . . . 41

5 Implementation and results **43**
5.1 Theoretical changes in 5GS to support TSN . . . 43
5.2 NW-TT functionalities implementation . . . 44
5.2.1 Structure of gPTP packet headers . . . 44
5.2.2 Hardware timestamping in Linux kernel . . . 46
5.2.3 Necessary packet modifications . . . 50
5.3 DS-TT functionalities implementation . . . 51
5.4 TSN testing topology . . . 51
5.5 GM synchronization sequence . . . 53
5.6 NW-TT synchronization sequence . . . 54
5.7 DS-TT synchronization sequence . . . 55
5.8 Thesis results . . . 56

6 Conclusions **59**
6.1 Conclusions regarding thesis work . . . 59
6.2 Future research and work related to 5G and TSN implementation . . . 60

7 Appendix **61**
7.1 NWTT Ingress Timestamping and Packet Modifications . . . 61
7.2 DSTT Egress Timestamping and Packet Modifications . . . 65

References **70**

Abbreviations and Acronyms

3GPP	3rd Generation Partnership Project
5GC	5G Core
5GS	5G System
AF	Application Function
AVB	Audio Video Bridging
BMCA	Best Master Clock Algorithm
CBR	Constant Bit Rate
CBS	Credit Based Shaper
CNC	Central Network Controller
DSTT	Device-Side TSN Translator
E2E	End-to-End
EPON	Ethernet Passive Optical Network
GM	Grandmaster
GMC	Grandmaster Clock
gPTP	generalized Precision Time Protocol
IEEE	Institute of Electrical and Electronics Engineers
IS-IS	Intermediate-to-Intermediate-System
LAN	Local Area Network
MRP	Multiple Registration Protocol
NWTT	Network-Side TSN Translator
OSI	Open Systems Interconnection
P2P	Point-to-Point
PSFP	Per Stream Filtering and Policing
PTP	Precision Time Protocol
QoS	Quality of Service
SRP	Stream Reservation Protocol
TDM	Time-Division Multiplexing
TG	Task Group
TSC	Time Sensitive Communication
TSN	Time Sensitive Networking
TSN TT	TSN Translator
UPF	User Plane Function

List of Figures

1 Tools offered by TSN and supporting standards 13
2 TSN network domain with functional links. 18
3 TSN with access link failure. 19
4 TSN with multiple gPTP domains 20
5 TSN with path redundancy synchronization with one clock providing time for two domains 21
6 TSN with GM redundancy separated in two PTP domains 22
7 gPTP message exchange for Peer delay measurement 23
8 TSN consisting of a GM plus synchronization path redundancy with a single and hot-standby GM 25
9 E2E architecture to support 5GS-TSN integration 28
10 5GS-TSN architecture consisting TSN and 5GS time domains 29
11 Ethernet port management procedure requested by TSN-AF 33
12 TSN Evaluation Kit architecture 36
13 TSN prototype for Ethernet Network 37
14 5GS-TSN Integration Prototype 37
15 Separation of traffic into two networks using switches 40
16 TSN topology for testing 52
17 GM base settings 53
18 GM console log when started 54
19 Sample gPTP message sent by NW-TT 55
20 Sample console log when DS-TT is started 56
21 Synchronization trend and eventual stabilization 57

1 Introduction

1.1 Background

The recent advancements in radio technologies such as 5G promise faster speeds and low latencies. From the specification requirements of 5G, the supported data rates need to be up to 10 Gbps. Compared to previous telecommunication networking technologies, 5G is 10 to 100 times better in terms of bitrate improvements over both the 4G and 4.5G. Additionally, the latency offered in 5G is close to 1 ms. One of the notable novelties promised by the introduction of 5G is heterogeneous communication, through which support for emerging applications with the highest data rate requirements is met, which at the same time fulfills the promise of ultra-low latency communication and higher availability between large networks of interconnected devices. An example of such is in Industry 4.0, which seeks to advance the digitization of industries. [1]

While the 3GPP optimized previous radio technologies (2G, 3G, 4G, and 4.5G) for consumer markets (telecommunication providers were selling to mobile users), 5G serves the same consumer market but in addition could offer optimized solutions for intelligent industries, for example, for big data, robotics and sensors. These optimizations still depend on the evolution that the 5G is still going through. The 5G technology is developing around verticals, such as Industry 4.0 and vehicular communications. The previous wireless technologies used in these settings, such as Wi-Fi had limitations. Generally, generic cellular technologies, of which 5G is an example offer better performance compared to Wi-Fi. The generic cellular technologies manage resources better, tend to be more stable when a large number of devices are connected in simultaneous communications and offer automatic seamless handover and access control based on SIM information and licensed usage spectrums. [2]

On the other hand, Time Sensitive Networks (TSN) serve the primary goal of providing deterministic services. This goal offers the assurance of a guaranteed transport of packets with low levels of packet loss, low delay variations, and bounded latency. TSN applicability ranges from industrial automation to autonomous vehicles, which both require high levels of timing accuracy. Different existing TSN profiles, such as time synchronization (802.1AS), ultra-reliability (802.1CB), bounded low latency (802.1Qav), dedicated resources and AP (802.1Qat), all work together to achieve the TSN's aim of deterministic services. Many Task Groups (TGs) that exist today lay out the specifications relating to the requirements and usage scenarios of TSN in the 3GPP Release 16. [3], [4]

3GPP Release 16 provides provisions for how the 5G technology will push into newer areas such as industrial communications. In an attempt to achieve such requirements, the IEEE and other related organizations have come up with ways to enable the integration of 5G and TSN. The aim is to have the two forming the primary building blocks of flexible production. The focus slowly shifts from wired to wireless connections, making devices increasingly mobile, highly reconfigurable, and needed on the modular production lines. Therefore, the 5G-based wireless

communication is a viable alternative, which, when integrated with TSN, offers improved solutions to the Industry 4.0 needs when compared to the previous radio technology standards.

The possibility to extend TSN to other layers of the Open Systems Interconnection (OSI) protocol stack makes for broader use cases for devices in TSN networks. While 5G provides better capacity and low latency, TSN, on the other hand, works on top of 5G to synchronize the different entities in a TSN network among other functions. [5]

1.2 Motivation

The thesis collaborates between the Aalto University research unit and Cumucore Oy from April to November. Cumucore Oy is a startup company aiming to ensure the replacement of Ethernet cables in networks. The company provides a 4/5G Core that is easy to install and configure. The company's technologies used to achieve this are the existing standards, notably, Wi-Fi infrastructure with a license to operate within the 3.5GHz frequency band.

Cumucore Oy cooperates with Aalto University to conduct research revolving around 5G and beyond. The main aim of the thesis topic was to drive research and prototype the TSN modules. After developing the TSN modules, the thesis could test them and analyze the performance levels and accuracy.

This thesis topic was good because 5G offers endless possibilities and needs practical applications built on top of it. The higher capacities and ultra-low latencies provided by 5G could be used as a base to build applications. This research is exactly what companies such as Cumucore Oy [1] and Amazon Web Services (AWS) [2] are researching on and implementing.

1.3 Goals

This thesis aims to conduct preliminary research concerning the IEEE 802.1AS (TSN) and develop prototypes of the TSN modules. This goal involves developing the TSN modules based on the recommendations of the TSN IEEE WG. The thesis then uses the developed TSN modules to test the levels of synchronization in a given controlled network when other TSN entities such as Grandmaster Clock (GMC) are present.

The thesis provides all this information while at the same time discussing the positives and negatives concerning the initially set goals and what was attainable.

1.4 Implementation

The TSN modules, namely device-side TSN translator and network-side TSN translator (DS-TT and NW-TT) are modules that exist inside the 5GS and function to synchronize the entities inside and outside the 5GS. The work in regards to the technical specifications necessary is cut out and laid in the IEEE specification

[1] https://cumucore.com

[2] https://aws.amazon.com/about-aws/whats-new/2021/11/preview-aws-private-5g/

documents. The hard work, however, was to dig deep and understand what was needed, and find the necessary tools and equipment to accomplish the task.

The TSN modules are implemented using the C programming language. The reasons for this choice is highlighted in the thesis. There need to be switches that could switch or even have the capability to differentiate TSN traffic. For this, some research was conducted on the different possibilities in the markets, and a choice was made to go with the TSN Evaluation Kits from Innovasic. Additionally, PCs with Network Interface Cards (NIC) supporting hardware time-stamping that could be used as an application interface for executing the code and testing the TSN functionalities were necessary. These did not have to be high-performance devices. Basic devices with NIC that support hardware timestamp generation for the gPTP protocol as a whole or individual gPTP packets.

In the end, with the devices available, it was possible to come up with the TSN modules with a codebase considered stable and devoid of memory leaks. The memory handling when capturing packets in Linux was done in a way that buffer problems were avoided altogether. The lack of readily available 5G supporting equipment such as 5G modems and switches resulted in the thesis taking a little bit longer than anticipated, and a basic test environment that focused on measuring the delay was instead opted for. The performance levels and the synchronization accuracies achieved are presented and demonstrated in this thesis in later sections.

1.5 Structure of the thesis

Chapter 2 introduces TSN together with the relevant standardization supporting it. Chapter 3 introduces TSN and the interactions between the TSN components and 5G System (5GS). Chapter 4 presents a short description of the tools and equipment used in the project for the testing, troubleshooting, and verification of the TSN modules. It also touches on the different methodologies used to obtain the relevant information, from which the TSN modules prototyping and implementation were made possible. Chapter 5 presents the TSN module C code samples together with explanations for the most important functions used to implement the TSN modules. Chapter 6 concludes on the work done in this thesis project and presents the important findings in the whole project and areas in which improvements could be made in the future relating to possible research.

2 Background on Time Sensitive Networks

This chapter introduces the important background information relating to TSN and presents a cutting-edge TSN design based on publications by the IEEE-TSN WG and related industry publications. Section 2.1 provides an introduction of what TSN entails and its main purpose. Section 2.2 introduces the different components under the TSN umbrella. These are the capabilities of TSN in general. Section 2.3, talks about what differentiates gPTP from PTP. Section 2.4 introduces the architecture of different TSN networks together with the various algorithms that enable time synchronization. Section 2.5 discusses ways in which synchronization can be achieved in different settings when multiple clock sources are used in networks.

2.1 IEEE 802.1 AS TSN

The IEEE 802.1Q defines TSN: as a standard through which deterministic services can be provided on Ethernet (802) networks. Through deterministic delivery, packets are set to be delivered within a given window while eliminating delays notable for causing congestion and errors in networks. TSN is a Layer 2 (L2) technology, so it does not use the IP protocol but rather the Ethernet protocol. This means that the forwarding decisions are then meant to be handled by the TSN bridges, depending on the information contained in the Ethernet headers. [6], [7] [3]

There are various packet services supported by TSN. First, is the best effort, which finds its use in most of the routers and/or bridges. The underlying architecture dictates that it should deliver most of the packets most of the time and in a given orderly manner. It does not provide any guarantees that the packets will be delivered. Secondly is the constant bit rate (CBR), which is also offered by Time Division Multiplexing (TDM). In CBR, the latency is always fixed, whereas the jitter and delay are almost zero. Theoretically, it can minimize congestion loss to nearly zero when buffering is implemented. [8]

Last is TSN, which is based on best-effort packet delivery in the network. In TSN, a contract exists between the network and a given application. The purpose of the contract is to ensure that the TSN flow transmitter is limited to a given bandwidth. The network's part of the bargain is to reserve the needed bandwidth, buffering, and any additional scheduling resources that are needed exclusively for the given TSN traffic flow. The benefit that this offers is that it enables the possibility of bounded latency and zero congestion. Through the use of multiple paths, it is possible to schedule packet flows simultaneously. Duplicate paths are then deleted when the packets reach their destination. [8], [9]

The IEEE 802.1Qcc standard defines a data model consisting of a Centralized Network Controller (CNC) and Central User Configuration (CUC) to centrally manage TSN services and resources. The standard additionally, implements ways to minimize jitter, through packet scheduling and guarantee in delivery. The forwarding decisions, therefore, are made by TSN bridges which do not use IP addresses but

[3] https://1.ieee802.org/tsn/

rather Ethernet header contents. The advantage offered by this approach is that the payload of any given Ethernet frame, as such, can be anything and not be limited to carrying IP data. This makes TSN more appealing for use in different environments/scenarios as it can carry almost any given payload typically found in industrial applications. [8], [9]

2.2 IEEE 802.1 TSN Toolbox

A summary of the TSN components, together with the supporting standards are shown in Figure 1. The subsequent sections provide more detailed descriptions of the same.

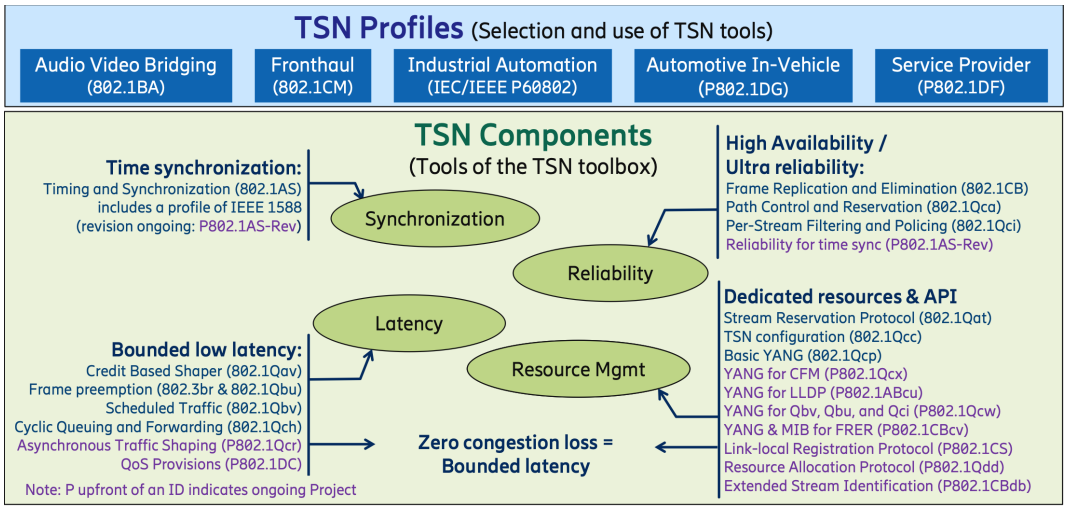

Figure 1: Tools offered by TSN and supporting standards

TSN is not a single component but rather a combination/collection of different standards and amendments, together with multiple projects published or developed by the IEEE 802.1 Task Group (TG). It defines provisions on how to achieve deterministic services which are operated on top of IEEE 802 networks. [10], [11] Depending on their needs, applications can make selections from the TSN toolbox shown below:

1. *Time Synchronization*
2. *Bounded low latency*
3. *Reliability*
4. *Resource Management*

2.2.1 TSN Time Synchronization

All network equipment in TSN must have the same concept of time. There exists a need for all the switches/bridges on the network to have their times synchronized. To

achieve this, two approaches exist. The first approach is through the IEEE 1588-2008-PTP and IEEE 8021AS 2011-Time which have an algorithm. The algorithm needs to have a reference clock which is considered as the grandmaster clock (GMC)(used in the synchronization of other clocks in the network). The GMC will then push synchronization messages to all connected devices attached to the TSN network.

The second method uses the IEEE 802.1AS, which is a unique profile adopted by the TSN TG stipulating the need for the use of IEEE 1588 specification in conjunction with those in IEEE 802.1Q. The profile assumes that some applications might not require the full functionality specified by the IEEE 1588-2008. The IEEE 802.1AS was not able to satisfy all the automation requirements necessitating the redesign to 802.1AS-Rev-CBS.

The IEEE 802.1AS-Rev defines an algorithm used for realtime data streams and a way for the data streams prioritization over best-effort traffic. The IEEE 802.1 TSN WG developed the credit-based shaping (CBS) in 2009 and it was a predecessor technology for TSN audio-video bridging (AVB). The 802.1AS-2011 provided precise time synchronization between different nodes in the network and several demonstrations proved that it was possible to achieve precision better than 1 μs, but a problem still existed. Time jumps occurred occasionally in the networks in situations where a loss of GM occurred and selecting a newer one was not instantaneous.

To solve this problem in the new redesign (IEEE 802.1AS-Rev), the focus was shifted on how to maintain a synchronized time for applications (time-sensitive) across IEEE 802 networks for duration when the operation is considered to work normally and when new network equipment are added or removed due to failure. To improve on the redundancies, configurations are made that enable multiple synchronization spanning-trees and GMs. This redesign shortens the recovery times in cases where failures occur in nodes and for a low-latency seamless handover between GMs. [10]

2.2.2 Bounded low latency

One of the noteworthy improvements to IEEE 802.1 networks concerning both AVB and TSN is ensuring that it can guarantee the delivery of messages with real-time constraints. The AVB suite performance improvements had already incorporated the IEEE 802.1-Qav (Forwarding and Enhancement) into it. TSN needed this improvement for streams that were considered time-sensitive. The goal and benefit that this standard offered was the ability for real-time transmission to be still made possible even when there was no global synchronization at the bridges and end stations.

Two published standards (IEEE 802.1Qbu and IEEE 802.1Qbv) further helped TSN improve real-time performance. In IEEE 802.1Qbu, an implementation enables ongoing noncritical transmissions to be interrupted by time-critical messages. The IEEE 802.1Qbv, on the other hand, provides a schedule-driven communication that leverages the synchronized transmission time together with the message forwarding decisions in a given network. [11], [12]

2.2.3 Reliability

One of the differentiating requirements between AVB and TSN applications is the need for highly reliable packet delivery. An example to highlight this difference is the nature of delay-sensitivity that is related to autonomous driving systems. In such scenarios, no tolerance is allowed due to delays (re-transmission of lost frames), and mechanisms have to be put in place to promptly detect when error conditions occur and further limit their propagation throughout the network.

TSN has a varied range of standards that contribute to reliability in communication such as duplicating the number of packets, eliminating some packets in an attempt to reduce the probability of loss, reservation of bandwidth for different applications, and even helping with path choice for different streams of data.

A discussion of the standards and how each helps to address the reliability of TSN applications is presented in this section. The first standard is the IEEE 802.1Qca (Path Control and Reservation), which is extended from intermediate Intermediate System to Intermediate System (IS-IS). The implementation uses the bridge with the shortest path and, through this, enables a way in which the bridge can define explicit forward control paths, which in turn makes the use of non-shortest paths possible. The standard comes bundled with tools for reserving bandwidth and streams on the forwarding path and mechanisms to control resiliency for data traffic.

The IEEE 802.1Qci Per-Stream Filtering and Policing (PSFP) is a standard that provides the necessary procedures used to make decisions about filtering and enforcing those policies per-stream. It also offers a way to guarantee quality by protecting the data stream to check whether conformance is adhered to or not. [11], [12]

2.2.4 Resource management

TSN defines many standards for the reservation and management of bandwidth resources. In instances where it is possible to predict behavior and there is a need for zero packet loss, contract negotiation between different applications is the preferred solution. [10]

For this to be made possible, the source of the TSN flows is limited to comply with a given maximum packet size and the number of packets to be transmitted over a given time interval. This reservation process provides a way in which, at most, applications can reserve all network resources. Reservations can be for specific streams moving throughout the network (bridge local area). TSN achieves stream reservation through the Stream Reservation Protocol (SRP), highlighted in the IEEE 802.1Qat standard and further enhanced in the IEEE 802.1Qcc.

The IEEE 802.1-2014 contains the IEEE 802.1Qat-SRP, which is one of the standards of AVB. Through it, it is possible to register and reserve resources (buffers and queues) within bridges (between the talker(s) and listener(s)). This resource reservation makes it possible to enable end-to-end (E2E) management of quality of service (QoS) for streams that have guarantees for both bandwidth and latency.

The IEEE 802.1Qcc-SRP (Enhancements and Performance Improvements) is an amendment of IEEE 802.1Qat standard, and it defines the different protocols that

are needed to provide support for configurable stream reservation classes and streams. Through the modifications implemented in the IEEE 802.1Qcc-SRP, TSN can fulfill the requirements of industrial, automotive markets, professional audio/video, and consumer needs.

An additional standard that is considered relevant to the management of resources is the 802.1CS(IEEE) (Link-local Registration Protocol). The IEEE 802.1CS protocol defines the database replication process when applications use point-to-point (P2P) links from one given end to the other. This way facilitates the creation process of application protocols for distributing information through parts or the whole network. It hugely arises from the need for different applications such as industrial automation and audio/video systems in large stadiums, which demand far more configuration/registration/reservation data. The protocol tends to handle the data better than in IEEE 802.1 Multiple Registration Protocol (MRP). MRP has been optimized for databases of up to 1500 bytes but might begin to slow down significantly when used for larger databases. The IEEE 802.1CS is better than the IEEE 802.1 MRP when large data replication is required. [4]. [11], [13]

2.3 Differences between PTP and gPTP

The IEEE 802.1AS (gPTP) differs from the original IEEE 1588-2019 (PTP) in many ways. The most significant difference is that, in gPTP, the underlying assumption is for all communication between the different PTP Instances to be done using MAC PDUs and addressing. The communication method differs from PTP, which supports legacy technologies such as L2 and even L3-4 methods of communication.

In scenarios demanding multiple technologies, a specification exists in gPTP for media-independent sub-layer that dramatically simplifies single timing domain integration. These might be technologies that are radical with different media access protocols. For each of the protocols, gPTP specifies a given media-dependent sub-layer. The sub-layering necessitated a provision to generalize the information exchanged between the PTP Instances to offer support for different management schemes and packet formats suitable for given networking technology. PTP, on the other hand, provisions a new architectural design based on media-independent and media-dependent sub-layers which offers support for IPv4, IPv6, Ethernet LANs, and a multitude of industrial automation control protocols.

In gPTP, there is only two PTP instance types present (PTP End Instances and PTP Relay Instances), while in PTP, there are Boundary Clocks, Ordinary Clocks, E2E Transparent Clock, and PTP Transparent Clocks. An Ordinary Clock in PTP corresponds to a PTP End Instance while the Boundary clock corresponds to a Relay Instance.

In gPTP, the information is communicated directly from PTP Instances to only those directly connected, making the gPTP domain consist of only PTP Instances. As a result, non-PTP Instances can therefore not be used to relay gPTP information. The difference with PTP is that in PTP, it is possible to use non-PTP-aware relays

[4]https://www.ieee802.org/1/files/public/docs2021/dp-farkas-TSN-profiles-0221-v01.pdf

in the PTP domain. This specification slows the conference time (time it takes for PTP entities to synchronize) and introduces jitter and wander, which needs to be filtered by any PTP clock in the network.

In scenarios demanding full-duplex links PTP uses E2E delay measurements while gPTP uses the P2P delay mechanism. Additionally, for such links, gPTP uses two-step processing which consists of FollowUp and PdelayRespFollowUp messages to communicate timestamps. An optional one-step is made possible too and can embed into the Sync messages timestamps on the fly when they are in the transmission process. PTP allows for either one-step or two-step processing for both the Sync and Delay messages depending on the specific profile in question.[14]–[16]

2.4 TSN architecture for a packet network

This section presents and offers an explanation for the different entities that make up TSN together with the mechanisms through which networks can be made time-aware. Additionally, a model that highlights the gPTP and its application in a packet network is discussed.

2.4.1 Important TSN entities

TSN networks are considered time-aware and mainly based on gPTP, which is the primary protocol used in packet networks to achieve synchronization of clocks. The gPTP protocol is a successor of the earlier IEEE 1588-2019 PTP.

The time-aware networks consist of time-aware devices such as bridges, end-stations, or routers. When working and interconnected together through a gPTP capable network element, all the devices form a gPTP network. Every gPTP instance supported by a time-aware system creates a gPTP domain. The PTP Instances of that gPTP then become part of that given gPTP domain. A time-sensitive device can support or be part of more than one gPTP domain. The mechanism that makes it possible to execute a gPTP in a given PTP domain is the PTP instance. A time-aware system can be associated with multiple PTP Instances in other gPTP domains. [5]

In a time-sensitive network, there are two types of PTP instances. Firstly is the PTP End Instance, which is usually the GM but in some cases can be a recipient of time information. Secondly is the PTP Relay Instance, which might again be a GM, but if not, then any recipient of time information that the GM pushes.

The Standard for Local and Metropolitan Area Networks covered in IEEE 802.1AS-2020 defines the different mechanisms for achieving delay measurements. These standard-based procedures range from IEEE 802.3 Ethernet, which uses full-duplex point-to-point links, IEEE802.3 EPON links, IEEE 802.11 wireless, and generic coordinated shared networks (GCSN). [14]

[5]https://www.ieee802.org/1/files/public/docs2021/60802-farkas-tsn-network-configuration-entity-0721-v01.pdf

2.4.2 Single gPTP domain implementing TSN

This section presents a single domain gPTP of a time-aware network. In a gPTP domain, there can be multiple PTP Instances, all of which have clock sourcing capabilities. When selecting the GM PTP Instance, the BMCA (Best Master Clock Algorithm) is used to select any PTP instances. The BMCA achieves the GM PTP Instance selection by comparing different parameters in all instances and choosing the best one to act as the source of accurate clock information. To a larger extent, the algorithmic implementation of BMCA is peculiarly similar to that defined in IEEE 1588-2019, with some simplifications. [16]

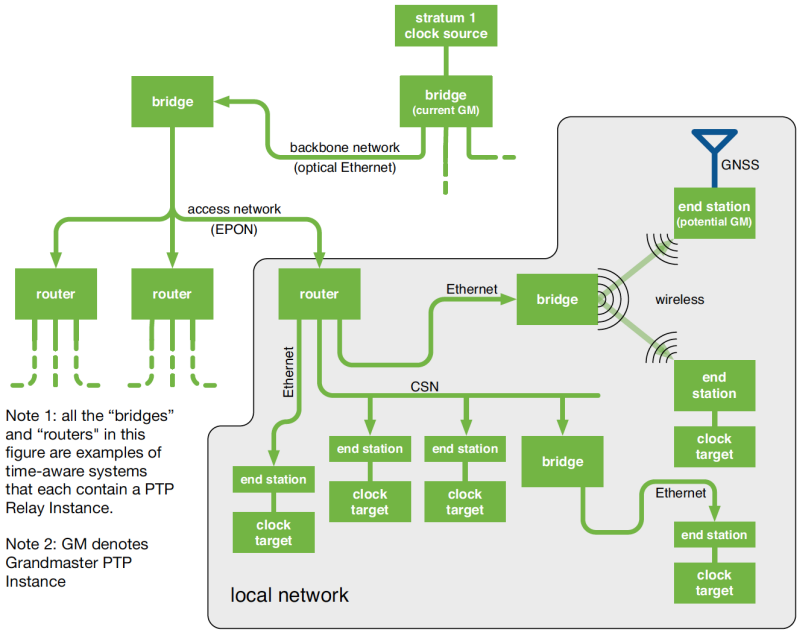

Figure 2: TSN network domain with functional links.

Figure 2 shows an example topological architecture of a time-aware network with the underlying assumption that all the routers and bridges support and have a PTP Relay Instance and that the GM denotes the Grandmaster PTP Instance.

The different routers and bridges with clock sourcing capabilities have an equal potential of being selected as a GM based on the BMCA. The result of the BMCA ensures that in a given domain, all the PTP Instances as such will have and use only one GM. It is common in networks for failures to occur. This context describes the BMCA operations in situations where there is a loss in the access network connection (EPON). Since the access network currently has the lone (single) GM, the local network has selected the local GM due to connection loss. This GM has a reference to the (Global Navigation Satellite System) GNSS source and, as such, becomes the active GM.

Consequently, now there are two domains (access and local network). Both have their own active GMs, which is different from the previous case before link loss, in

which only a single GM existed for both the domains. The resulting topology as a result of the link loss is as depicted in Figure 3.

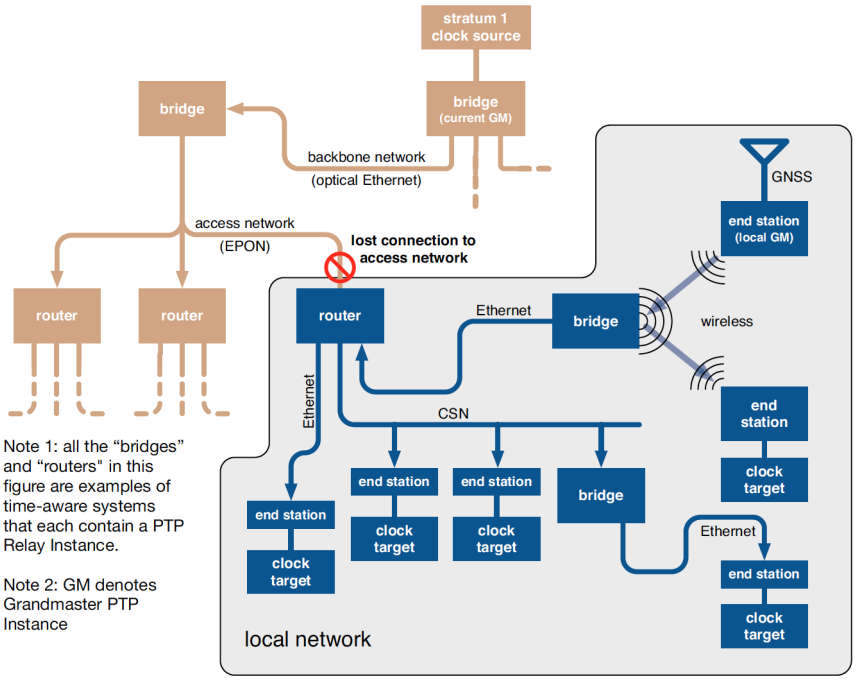

Figure 3: TSN with access link failure.

It is a requirement that for a given TSN system that supports multiple domains, one of the domains has to be domain zero (0) to offer backward compatibility with the IEEE 802.1 2011 standard. The domain 0, however, needs not necessarily be active in TSN networks. It is worth noting that at given short periods during network reconfiguration, more than one GM might be active while the BMCA process is ongoing. [16], [17]

2.4.3 Multiple gPTP domains implementing TSN

In an industrial application where the number of network elements might be strewn in different areas of the same building, working with a single domain might not be possible, and there is a need for multiple domains. The networks might have different timescales depending on which domain the time is originating.

From the Figure 4, domain 0 uses the PTP timescale while domain 1 uses the arbitrary (ARB) timescale. It is noteworthy that every PTP Instance located in domain 1 has domain 0 active in this implementation. Those PTP instances belonging to the same domain need to have direct connections between them.

It is not possible for time to be transported from a given PTP Instance in domain 0 to the other PTP Instance in domain 0 through a time-aware system that does not have the domain 0 active. Just as in the single-domain, the BMCA selects the

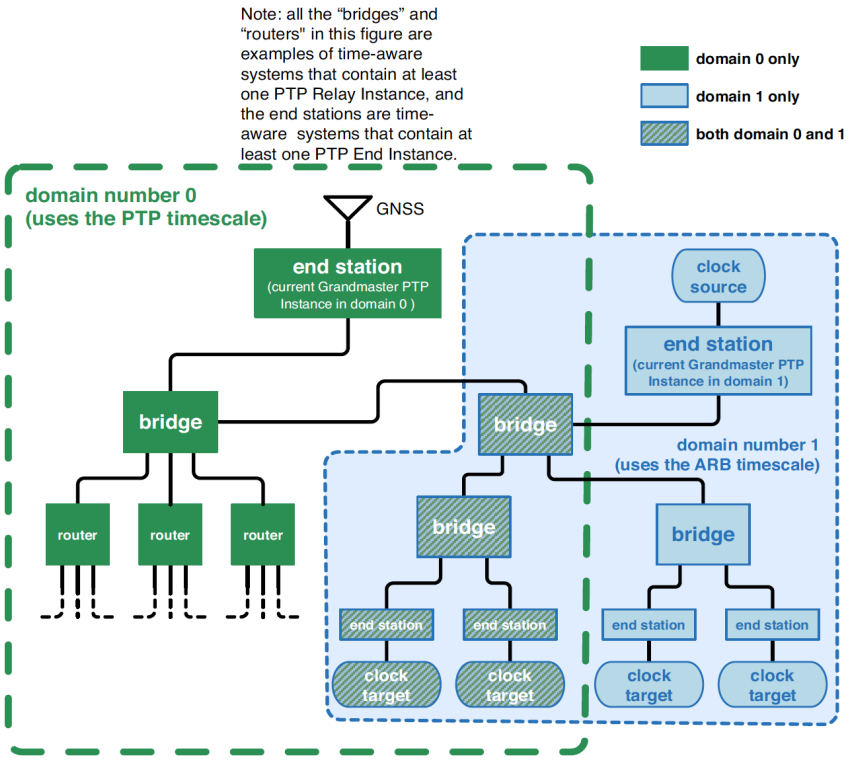

Figure 4: TSN with multiple gPTP domains

GM. The significant difference in multiple gPTP domains is that in each, a different BMCA instance will be invoked. [16], [18]

2.4.4 TSN with redundant paths/GM PTP instances

Ethernet normally runs a single spanning tree through which all the traffic is carried, meaning that there is no redundancy. To implement redundancy in Ethernet, multiple spanning trees depending on the number of VLANs is implemented. This ensures that redundant connectivity can be maintained and provisions the multiple paths needed for clock synchronization packets that originate from multiple clocks.

Through redundancy, TSN can achieve different sophistication levels, improve performance and reduce costs. For these reasons, some level of redundancy is consistently implemented in time-aware networks. These implementations of redundancy differ from one application to the other. No matter the redundancy implementation, the solution needs components to detect anomalies. Furthermore, correction components for such anomalies need to be present, and finally, an action component that acts to correct the detected anomalies. The correction components function to determine the appropriate corrective action. On the other hand, the action component performs the required action(s) necessary to fix the detected problems.

The BMCA standard provides a basic level of redundancy which consists of a detection component. The trigger in the detection component is activated any time

after the GM stops working, which might be caused by a loss of sync or announce messages. Additional reasons for such triggers are when links that connect to a GM fail. When the network detects any of the above failures, the correction component works by triggering the BMCA, which sends the announce messages. This process allows for the new GM to be selected. Upon success, the elected GM begins to send, Announce, and Sync messages (through the action component), making all the participating PTP Instances listen to the new GM.

However, this standard does not fully specify redundancy but instead provides insights into how redundancy can support more sophisticated network configurations. Such networks, as a result, then support additional levels of clock path redundancy and GM. Figure 5 shows how such networks can provide the different redundancy levels as one way of dealing with failures.

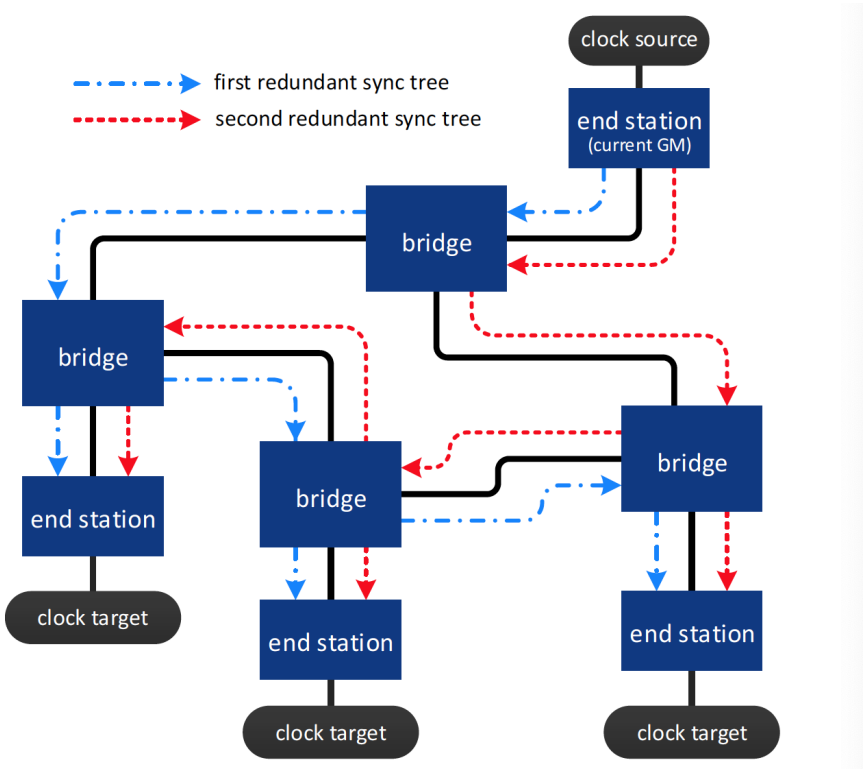

Figure 5: TSN with path redundancy synchronization with one clock providing time for two domains

Figure 6 shows two redundant GMs: one acting as the primary and the other secondary GM. Each of the GMs has at least one out of the two redundant synchronization trees originating from it, and the setting supports hot-standby operation mode. The hot-standby mode is a requirement for the secondary GM to be synchronized to the other primary GM. The synchronization tree of the primary GM includes the secondary GM, which is why both of the GMs need to be synchronized.

The first assumption made in this setup is that the different methods used in merging redundant Sync messages received at different end-stations are left out.

Secondly, all the various bridges in the figure are assumed to be time-aware and contain PTP Relay Instances while the end-stations are also time-aware and contain PTP End Instances. Lastly, the GM denotes that a Grandmaster is a Grandmaster PTP Instance. [16], [19]

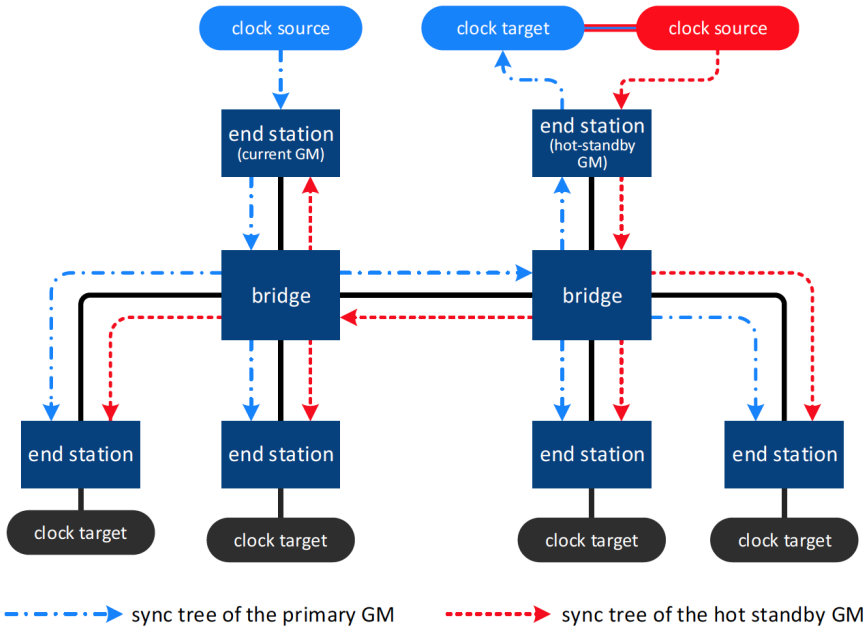

Figure 6: TSN with GM redundancy separated in two PTP domains

2.5 Time Sync

The IEEE 1588-2019 and time synchronization (gPTP), to a certain extent, have the exact implementation regarding how to achieve synchronization. It involves a GM PTP Instance, which sends time information that includes the current synchronized time to each PTP Instance directly attached. It is a requirement for each PTP Instance to correct the received synchronized time.

Each PTP Instance performs time synchronization by adding the propagation time: which is the time that it takes information to be transmitted from the GM PTP Instance to the receiving PTP Instance using the gPTP communication path. PTP Instances considered PTP Relay Instances must forward the corrected time information (which might include additional correction delays encountered in the forwarding process) to every directly attached PTP Instance.

To ensure that the synchronization process works and that timing information is distributed accurately, there needs to be precise information about two-time intervals: the forwarding delay/residence time (the time difference from when messages enter a device to that time they exit) and the propagation delay, that is the time it takes for the synchronized time information to be transmitted between PTP Instances.

It is easy to calculate the residence time as it is local to a given PTP Relay Instance. For the gPTP communication path delay, on the other hand, the calculation of the time is a little complicated as the time is dependent on a variety of things such as the media properties, length of the path, e.t.c.

2.5.1 Measuring delay

There are different ways to measure propagation delay for every communication path (PTP/LAN). There is the absolute and well-known timing information measurement for when parts of a given message(s) are transmitted from a given device (talker) until the receiver(listener) gets them. The PTP Instances can measure the propagation delay in the reverse direction by sending the same messages from the receiver to the sender.

Figure 7 details the delay measurement procedure between the master clock (MC) and slave clock (SC) time PTP Instances with the assumption of the link being symmetric in uplink and downlink directions.

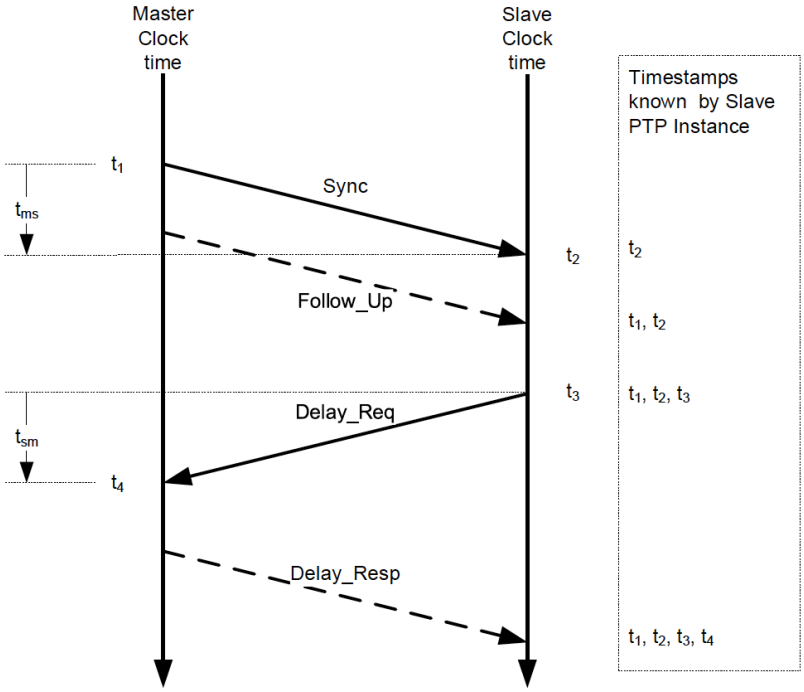

Figure 7: gPTP message exchange for Peer delay measurement

From the Figure 7, the first step in the timing message exchange involves the Master PTP Instance sending a Sync message and noting the time t_1 when it sent the message to the Slave PTP Instance. The PTP Slave Instance notes the time that it receives the message: the time of reception t_2. The Master PTP Instance needs to convey the timestamp t_1 to the PTP Slave Instance. There are two ways to convey this timing information: firstly, the Master PTP Instance can embed

the timestamp t_1 in the Sync message. This process requires hardware processing levels that offer high precision levels. Secondly, the t_1 can also be embedded in the Follow_Up message.

The Slave PTP Instance sends to the Master PTP Instance a Delay_Req message and notes the time t_3. Upon sending the Delay_Req message, the Slave PTP Instance notes the time t_3 at which the Slave PTP Instance sent it. The Master PTP Instance receives the Delay_Req and records the time of reception t_4. The Master PTP Instance sends the Delay_Req message with the timestamp t_4 embedded to the Slave PTP Instance.

When the message exchanges between the Master and Slave PTP Instances end, the Slave becomes time-aware and knowns all the needed time instances: (t_1, t_2, t_3, t_4), which it uses to calculate the mean propagation delay (meanLinkDelay). Additionally, when time shifts occur in the network, the rate ratio can be used to adjust the propagation delay. The rate ratio and the resulting mean propagation delay are shown in the following equations.

$$r = \frac{t3_{\text{i}} - t3_{\text{i-1}}}{t4_{\text{i}} - t4_{\text{i-1}}} \tag{1}$$

$$t_{\text{ms}} = t_2 - t_1 \tag{2}$$

$$t_{\text{sm}} = t_4 - t_3 \tag{3}$$

$$D = t_{\text{ms}} + t_{\text{sm}} = t_2 - t_1 + t_4 - t_3 = (t_3 - t_1) - (t_4 - t_2) \tag{4}$$

These timestamps are helpful when computing the offset of the Slave PTP Instance concerning that of the Master PTP Instance and the mean propagation time. The mean propagation delay (D), when considering two clocks, is the path delay when PTP messages are sent as part of PTP message exchange between them as denoted by t_{ms} and t_{sm}. The formula above removes the skew that exists in the delay measurements when the clocks are not in sync.

Again the rateRatio value shown in the formula is not a single value, but rather averages of many measurements taken with each message sent or received when measuring the delay. Depending on the value of the rateRatio, it is possible to get an indication of how two different clocks behave. The rateRatio is then accumulated and used to adjust the delay between the clocks.

The underlying assumption when computing the offset and propagation time is that slave-to-master and master-to-slave propagation times are the same. Any asymmetry that the system might introduce concerning the propagation time makes the computed value of the clock offset erroneous, making the computed mean propagation time different from the actual propagation time. Provisions to allow for asymmetry in radio transmissions can be made possible. [15]

2.5.2 Logical syntonization

The time synchronization discussed in 2.5.1 is dependent on the measured accuracy of the residence time and delay. In some cases, clocks can be frequency-locked (syntonized) to the GM, necessitating a sign time-base for all the time interval measurements. Time interval corrections are done by the PTP Relay Instances using the GM frequency ratio, which is a better method than adjustments done to the frequency of an oscillator that is slow and prone to gain peaking effects.

Figure 8 displays the topology of a time-aware network that consists of a GM plus additional path redundancy achieved by having a primary and an additional hot-standby GM. Each GM establishes two sync trees, for each source, resulting in four overall sync trees separated into four gPTP domains. [20]

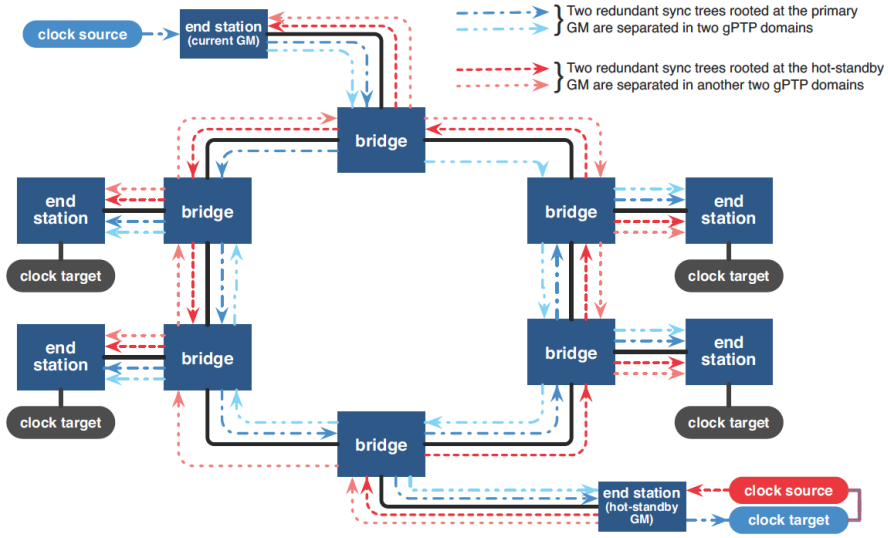

Figure 8: TSN consisting of a GM plus synchronization path redundancy with a single and hot-standby GM

The PTP Instance at each port measures its ratio of the frequency in comparison to those of the end PTP Instance clock frequency. The measure of the frequency offset between the master clock and its own local clock is known as the rate ratio. The GM accumulates its rate ratio to that of the local clock frequency to the Follow_Up message. The PTP Instances trying to synchronize to the GM use the rate ratio retrieved from the Follow_Up messages to calculate what adjustments are necessary. The calculation of propagation time is done by taking the neighbor's frequency ratio relative to that of the local clock.

The accumulation of frequency ratios when measuring the GM frequency ratio is beneficial when network failures occur, and reconfiguration is needed. The existing

GM connection is lost, and the new GM is selected based on readily available Pdelay measurements received from the first Follow_Up message. The Pdelay is a special protocol used in measuring the time it takes for packets/messages to move between two ports i.e. from ingress to egress ports. The process reduces the transient error duration during reconfiguration in the synchronized time, which is helpful in a variety of E2E audio applications.

Secondly, adjustments of the oscillator frequency, for example through the use of Phase-Locked Loop (PLL), is both slow and causes gain peaking effects. This gain peaking does not exist since an error occurring in the frequency offset at a given PTP Relay Instance, and its resulting residence time error do not have a direct influence on the frequency offset experienced at the downstream PTP Relay Instance. [21], [22]

2.5.3 GM selection and network establishment

According to the IEEE 802.1AS protocol, all PTP instances participate in the process for the best master selection. The process can be done or determined using a synchronization spanning tree (STP), different from the forwarding spanning-tree defined by the IEEE 802.1Q Rapid Spanning Tree Protocol (RSTP). The spanning-tree determination through the use of RSTP is considered sub-optimal and inadequate for synchronization purposes or for cases where a different node topology exists other than that of a synchronization spanning tree.

Since gPTP is the protocol used, it is a requirement that all the participating systems that exist in the gPTP domain are all time-aware. This means that the protocol only transfers timing information over systems that are time-aware. These time-aware systems, when trying to determine whether an end device (at the end of the link or between itself and a Pdelay responder) are time-aware, use point-to-point (P2P) delay mechanism at each port. The PTP P2P is a delay mechanism used in propagation of delay measurements between different points.

Suppose upon sending a Pdelay_Req, either a response is not received, too many responses are received, or the measured propagation delay exceeds the specified threshold. In that case, the protocol decides that the E2E Transparent Clock (TC) or a non-time-aware system is present, deeming the link attached to the port incapable of running gPTP. As such, the BMCA ignores that device while the port continually attempts propagation delay measurements through P2P delay mechanism or multipoint control protocol (MPCP) messages, or IEEE 802.11 messages. This process is done repeatedly in an attempt to check whether such links are or are not capable of running IEEE 802.1AS. [21]

3 Time Sensitive Networks in 5GS

IEEE standardization work in Release 16 has introduced 5GS features that enable the inter-working between the 5GS and wired TSN networks. The main aim of the specifications is a seamless E2E transmission of TSN streams over the 5GS. Release 16 specifies a centralized TSN model that enables a centralized SDN-type architecture. There is a separation of the control plane from the user plane. The 5GS, modeled as a local or virtual bridge, provides TSN ports and control plane connectivity at the user plane. TSN translators are included in the model. [16], [23], [24] The 3GPP offers the support for 5GS-TSN with the features listed below:

- 5GS integration to TSN is implemented as a logical TSN bridge when integrated with an external network
- There are different TSN models, but for 5GS-TSN integration adopts the fully centralized TSN model
- The different features of the 5GS function to support TSN, thus allowing for a transparent 5GS integration
- From the perspective of time synchronization, the entire 5GS (E2E) is considered time-aware, thus following the IEEE 802.1AS
- The TSN translators (DS-TT and NW-TT) located at the edges need to support the IEEE 802.1AS operations
- All the different 5GS components such as UE, gNB, NW-TT, DS-TT, and UPF must synchronize with the 5G GM

3.1 5GS-TSN architecture

The 5GS, to be integrated into any given external network, has to be implemented as a TSN bridge. The bridge includes the TSN translator functionalities needed for the 5GS-TSN integration both at the user and control planes. Figure 9, shows the different components involved in an E2E TSN with the 5GS implemented as a logical bridge.

Contained in the 5GS translator functionalities are the Network-Side TSN translator, which supports link layer connectivity and reporting relevant for the discovery of attached Ethernet devices. The Device-Side TSN translator (DS-TT) also perform link layer connectivity and discovery of attached Ethernet devices. More functions performed by these TSN translators will be presented.

Located in the 5GS implemented as a TSN bridge are the TSN components together with the 5G network functions. The device-side consists of the DS-TT and UE with the NW-TT located within the UPF. Connected to the 5GS are 5G base stations (gNB), which communicate with the UE through the radio access network (RAN). The UE connected to the gNB terminates one end of the 5G TSN virtual bridge.

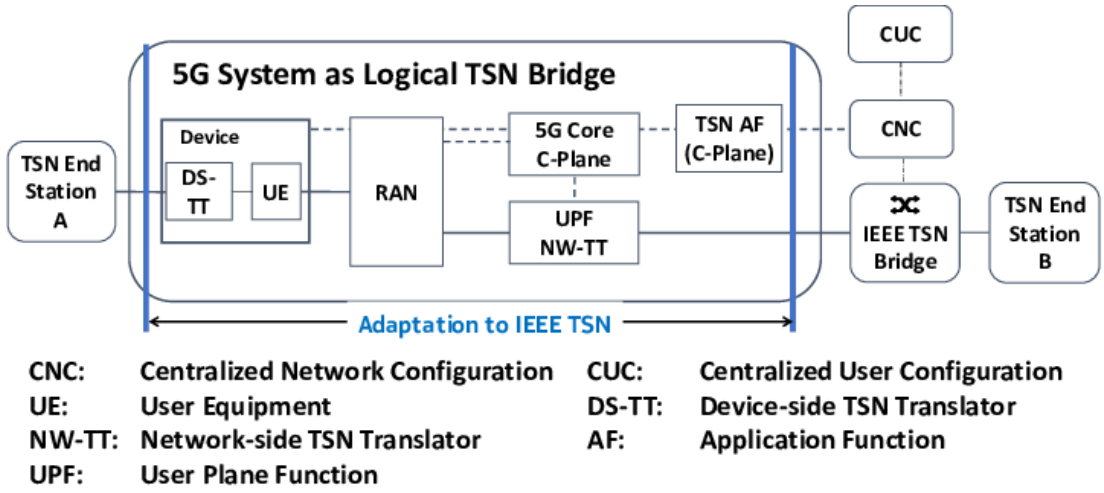

Figure 9: E2E architecture to support 5GS-TSN integration

The TSN application function (TSN AF) interacts with the TSN system (control system). This interaction happens in the control plane. The control system could, for example, be a Centralized Network Configuration Function (CNC). The purpose of the interaction between the 3GPP Core Network and AF entity is to enable services such as QoS mapping needed for stream transmission over the 5GS. The AF traditionally could influence the routing of traffic while at the same time allowing interaction necessary for policy control which is different in the 5G-TSN case, where the AF serves as a TSN Translator. The function of the AF is to help the CNC to exchange with the 5GS the QoS and port management information. For port management information that is considered standardized and deployment-specific, these are transferred transparently between the translators and the AF.

As for the user plane, TSN translators located at the UPF and UE act as gateways and connect the TSN systems to the 5GS, which enables the support for gPTP time-based synchronization. Consideration is necessary for the 5GS to either be or consist of one or multiple TSN bridges. This consideration depends on whether the UE can establish multiple packet data unit (PDU) sessions to different UPFs and whether the TSN networks do not allow Ethernet shared media. To avoid the use of shared media, modern Ethernet uses only P2P duplex links, ensuring that a given Ethernet link connects two nodes (which can either be an end-station or bridge). The reason for the avoidance of shared media is that it can significantly slow down Ethernet protocols convergence and is why the 5GS TSN model presented uses only P2P links.

The TSN translators are required when the 5GS has to be integrated with an external network. In this architectural scenario, the 5GS acts as a TSN bridge, thus abstracting the internal workings or aspects such as the radio connections. These logical TSN bridges need to be present on both ends of the 5GS to ensure that any devices with a 5G connection will appear as standard TSN devices. The bridges additionally have TSN translator functionality implemented that helps with the

interoperability between the 5GS and TSN systems and is presented for both the control and user plane. [16], [25]

3.2 Integration concept of 5GS with the IEEE TSN

This section describes the specific integration needed between the 5GS and the TSN functionalities. They are relevant as they provide a better understanding on how the 5G functions interact with the TSN components.

3.2.1 TSN translators definitions and functions in 5GS and IEEE 802.1AS

The 5GS implementing the TSN translators (DS-TT and NW-TT) becomes a virtual TSN bridge (VTB) capable of interfacing with TSN devices to ensure an E2E transmission/communication between the different end-stations (talkers or listeners).

The DS-TT integrates with the UE and NW-TT with the UPF. The gNB then serves the wireless connections to the UEs. The UPF terminates the gPTP messages signaling in the 5GS. With the enhancements to incorporate the TSN translators, the VTB appears like any other standard TSN bridge in the network when connected to other TSN bridges. To hide the complexity of the 5GS, the DS-TT and NW-TT act as logical ports of the VTB. As a result, DS-TT and NW-TT towards the network adopt TSN bridge ingress and egress port operations. Figure 10 shows a high-level abstraction of the 5GS-TSN consisting of two time domains. [21]

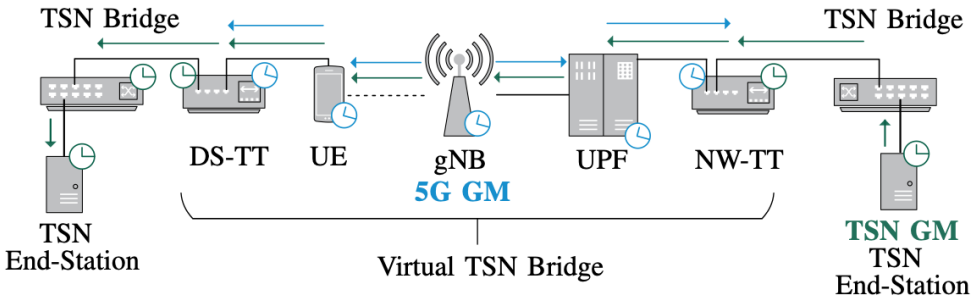

Figure 10: 5GS-TSN architecture consisting TSN and 5GS time domains

Release 16 of the IEEE 802.1AS covers the different mechanisms involved in the distribution of TSN clock and time-stamping. When a downlink (DL) gPTP message is received by the NW-TT, it generates an ingress time-stamp (TSi) for each message. In calculating the upstream TSN node (e.g., in GM time) delay, the NW-TT utilizes the cumulative rate ratio obtained from the gPTP message payload.

The NW-TT calculates the new cumulative rate ratio, which modifies the gPTP message payload by replacing the previous rate ratio (from upstream) with the new one. The NW-TT adds a suffix field to the gPTP packet and the delay from the upstream TSN node to the GM time and puts it into the correction field. The resulting gPTP messages are then forwarded to the UE through the UPF and transmitted on

a QoS flow per the upper bound residence time, which is a requirement specified in the IEEE 802.1AS.

Upon receiving a new gPTP message, the UE forwards it to the DS-TT, creating egress time-stamping (TSe) for the gPTP event messages for external domains. The 5GS calculates the residence time relating to the gPTP messages by taking the difference between the TSi and TSe. The residence time spent in the 5GS is then converted by the DS-TT to GM time using the rate ratio found in the gPTP messages. The DS-TT then adds the calculated residence time and removes the TSi information from the payload suffix field. The resulting gPTP messages exit the 5GS system towards the destination downstream TSN node. [21], [26]

The subsequent section delves further in providing the significant interactions and processes between the TSN Translators and the AF.

3.2.2 The elementary procedures between the AF and DS-TT

In TSN, Ethernet port is a technology through which multiple TSN devices connect to each other and the internet. At these ports, there are some parameters that need to be managed and this process is known as Ethernet port management. An example of such parameters are txPropagationDelay, traffic classes, link layer discovery protocol (LLDP) values (lldpV2PortConfigAdminStatusV2 and lldpV2MessageTxInterval among others). These parameters dictate the settings necessary at each individual port, and dictate for example how often the LLDP messages should be sent and what happens when the time-to-live (TTL) of such messages expires. An advantage of this can be useful in avoiding loops in packets, which make it faster for time convergence between TSN devices and endpoints.

Both the UE and network might offer support for the transfer of standardized and deployment-specific Ethernet port management information between the DS-TT at the UE and AF. The support is essential as it manages the Ethernet ports used at the DS-TT to establish PDU sessions. The port management messages are included in IE's information management container and transported through PDU session establishment and modification procedures.

The Ethernet port management procedure enable the AF to support the following features:

- to get knowledge of all port management parameters that the DS-TT supports.
- obtain the current port management parameter values from the DS-TT Ethernet port.
- setting of the DS-TT Ethernet port, port management parameters.
- subscribe to changes to certain port management parameters from the DS-TT Ethernet port.
- unsubscribe from being notified of changes to certain port management parameters from the DS-TT Ethernet port.

In an attempt to initiate the network-requested Ethernet port management procedure, the AF firstly does the encoding of the information. This encoding involves the different port management parameters values (e.g., those requiring reading, setting, and (un)subscribing to). This port management procedure also includes the list of port management parameters supported by the DS-TT Ethernet port in the Ethernet port management list IE. Secondly, the DS-TT sends the [MANAGE ETHERNET PORT COMMAND] message via the Policy Control Function (PCF) and Session Management Function (SMF) to the UE specified in 3GPP 23.502. The DS-TT finally starts the T100 timer, which is a timer instruction for the start of ethernet port management sent by the TSN AS to the NW-TT. Ethernet ports depending on whether they are being used or not, need to be managed so that resources are utilized.

When the DS-TT receives the [MANAGE ETHERNET PORT COMMAND] message, it considers each of the operations included in the Ethernet port management list IE. Firstly, if the nature of the operation code is "get capabilities," all the Ethernet port management parameters that the DS-TT supports are included in the Ethernet port management capability IE of the [MANAGEMENT ETHERNET PORT COMPLETE] message.

Secondly, for operation codes denoted as "read parameter," an attempt is made to read the parameter values located at the DS-TT Ethernet port. Upon a successful read, the parameters read and their current values are included in the Ethernet port status IE of the [MANAGEMENT ETHERNET PORT COMPLETE] message. In situations where the reading was considered unsuccessful, the parameter and the associated cause value of the Ethernet management service are included in the Ethernet port status IE of the [MANAGE ETHERNET PORT COMPLETE] message.

Thirdly, for operation code denoted as "set parameter," an attempt will be made to set the stated parameter at the DS-TT Ethernet port. When set successfully using the specified value, the parameter and its current value are included in the Ethernet port update result IE of the [MANAGE ETHERNET PORT COMPLETE] message. When a failure occurs and setting the parameter value is not possible, the parameter together with the associated Ethernet "port management service cause value" is included in the Ethernet port update result of the IE message.

Fourthly, for operation code denoted as "subscribe-notify for parameter," the AF requests notifications for when the given parameter values change. When the "unsubscribe for parameter" is the operation code, AF deletes the said parameter and prevents sending more notifications. Finally, the AF sends the [MANAGE ETHERNET PORT COMPLETE] message via the PCF and SMF.

The DS-TT can also initiate the Ethernet port management procedure. This procedure aims to tell the AF of changes to the values to the different Ethernet port management parameters for which the AF had exactly requested notifications. The process starts with the DS-TT creating an [ETHERNET PORT MANAGEMENT NOTIFY] message. After making the message, the DS-TT includes the different port management parameters to report to the AF. These are included in the Ethernet port status IE of the message. The DS-TT, in addition, starts the T200 timer and

sends the message towards the AF. The T200 is a timer instruction for Ethernet port management consisting of notifications and acknowledgments, sent between the DS-TT and TSN AS. The main difference between the T100 and T200 is the area in which they operate, with T100 used in managing ports between DS-TT and TSN AS and T200 between the Nw-TT and TSN AS.

On receiving the message, the AF creates an ACK and sends the ACK message back to the UE. The DS-TT on the receipt of the message stops the T200 timer and makes an [ETHERNET PORT MANAGEMENT NOTIFY COMPLETE] message, which it then sends to the AF.

Again abnormalities can occur in this process. An example can be a transmission failure when transmitting the [ETHERNET PORT MANAGEMENT NOTIFY ACK] message from other layers. The AF in such cases is not required to diagnose such errors but rather consider the port management procedure initiated by the DS-TT as complete.

Abnormal cases prone to occur in the DS-TT are the expiration of the T200 timer. The DS-TT will, in such cases, re-transmit the message and reset the timer T200. The DS-TT tries retransmission a maximum of four times, and on the fifth attempt, aborts the procedure.

The Ethernet port management capability procedure, when initiated by the DS-TT, serves to provision of DS-TT supported port management capabilities for PDU session establishment to the AF. To initiate the Ethernet port management capability procedure, the DS-TT creates the [ETHERNET PORT MANAGEMENT CAPABILITY] message. The DS-TT then includes the port management capabilities in the Ethernet port management capability IE of the created message. The message is then sent to the AF. [21]

3.2.3 The elementary procedures between the AF and NW-TT

Both the AF and NW-TT support the transfer of standardized and deployment-specific Ethernet port management information necessary for managing the Ethernet ports at the NW-TT. The Ethernet messages related to port management are in the "PortManagementContainer" datatype, Port Management Information Container IE, and the Bridge management messages are in the "BridgeManagementContainer" datatype. The N4 is the interface between the Session Management Function (SMF) and User-Plane Function (UPF). Since the NW-TT exists at the network-side and connects to the UPF, there needs to be a procedure to notify SMF, when the path between the UPF and gNB goes down. The N4 Session Level Reporting Procedure and the SM policy association modification procedure are functional when transporting Ethernet port management and Bridge management messages specified in the 3GPP TS 23.503.

Generally, the AF-requested Ethernet port management procedure serves to enable the AF to obtain the port management parameters supported by the NW-TT. It additionally provides the current port management parameter values at the NW-TT, to set up the current values of the port management parameters at the NW-TT Ethernet port and for the AF to be able to unsubscribe to the notifications by the

NW-TT concerning one or more port management procedures. [21]

When initiating the port management procedure, the AF does the following procedures:-

- encoding the information in regards to port management parameters that need reading, those that need setting, port management parameters that need to (un)subscribe, and, whether in some cases a request of the list of the different port management parameters supported by the NW-TT
- sends to the NW-TT a [MANAGE ETHERNET PORT COMMAND] via the PCF and the SMF.
- starts a timer T100. These procedures are as illustrated in the Figure 11.

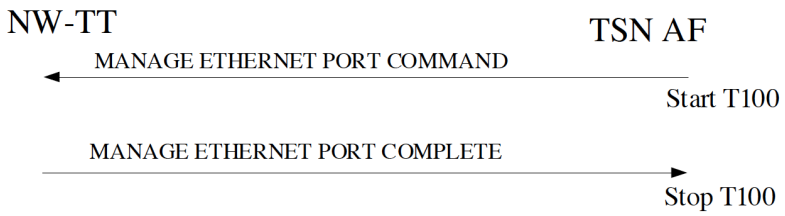

Figure 11: Ethernet port management procedure requested by TSN-AF

Upon receiving the [MANAGEMENT ETHERNET PORT COMMAND] message, the NW-TT for every operation included in the port management list of the UE does the following:-

- for operation codes denoted as "get capabilities," a list of the Ethernet port management parameters that the NW-TT supports is included in the Ethernet port management capability of the [MANAGEMENT ETHERNET PORT COMPLETE] message.
- for operation codes denoted as "read parameter," an attempt is made in reading the values of the parameters at the NW-TT Ethernet port then:
  - if the value of the parameter is read successfully at the NW-TT port, the parameter, and its current value is then included in the Ethernet port status IE of the [MANAGEMENT ETHERNET PORT COMPLETE] message
  - if the NW-TT cannot read the parameter's value successfully, the parameter and its associated Ethernet port management service are then included in the IE of the [MANAGEMENT ETHERNET PORT COMPLETE] message.

- in cases when the "set parameter" is the operation code, an attempt is made to set the parameter's value at the NW-TT Ethernet port to the value specified in the "set parameter." Additionally:
  - if at the NW-TT Ethernet port, there is a success in setting the value specified, the Ethernet port update result of the [MANAGEMENT ETHERNET PORT COMPLETE] includes the parameter and its current value.
  - if at the NW-TT Ethernet port, there is a failure in setting the value specified, the parameter and associated Ethernet port management service and cause of failure value are included in the Ethernet port update result IE of the [MANAGEMENT ETHERNET PORT COMPLETE] message.
- in cases where the operation code is "subscribe-notify for parameter," the request from the AF is stored and notifies of any changes related to the corresponding parameter values.
- in cases where the operation code is "unsubscribe for parameter," the request is deleted from the AF and notifies of any changes relating the corresponding parameter values, and
- sending the [MANAGEMENT ETHERNET PORT COMPLETE] message via the SMF and PCF to the AF.

There are no guarantees, and in abnormal cases in the AF, the T100 can expire. On first-timer expiry, the AF initiates a retransmission of the [MANAGEMENT ETHERNET PORT COMPLETE] and the timer reset and a new start timer T100. The AF tries retransmission a maximum of four times and, on the last time, issues an expiry timer T35xx, after which the AF aborts the procedure.

Abnormal cases occurring in the NW-TT revolve around transmission failures in delivering the [MANAGEMENT ETHERNET PORT COMPLETE] message indicated from lower layers. Upon experiencing this problem, the NW-TT does not need to diagnose the error but rather consider the AF-initiated procedure as relatively complete. This behavior does not cause a reversion of the policies included in the [MANAGEMENT ETHERNET PORT COMPLETE] by the NW-TT. [21]

3.2.4 Port management procedure as initiated by the NW-TT

NW-TT-initiated port management procedure serves to notify the AF of changes to the Ethernet port parameter values. These values are those requested by AF for notifications when changes occur. For the procedure of NW-TT-initiated Ethernet port management, the first process occurs when the NW-TT creates the [ETHERNET PORT MANAGEMENT NOTIFY] message, which:-

- includes all the necessary Ethernet port management parameters that need to be reported to the AF together with their current values. These current Ethernet

port status IE values are included in the [ETHERNET PORT MANAGEMENT NOTIFY] messages.

- initiates a start timer for the T300 which signals the start of the transmission of [ETHERNET PORT MANAGEMENT NOTIFY] message, and
- sends the [ETHERNET PORT MANAGEMENT NOTIFY] message to the AF via the SMF and PCF.

When the AF receives the [ETHERNET PORT MANAGEMENT NOTIFY] message, it creates an acknowledgment (ACK) of the message and then sends the ACK message back to the NW-TT. When the NW-TT receives the ACK message, it stops the timer T300.

In an abnormal situation where the T300 expires, the [ETHERNET PORT MANAGEMENT NOTIFY] message is re-transmitted by the NW-TT. This process resets the timer and starts a timer T300. The re-transmission repeats only a maximum of four times. When the T300 expires for the fifth time, the NW-TT aborts the procedure altogether.

The NW-TT can initiate the bridge management procedure, the purpose of which is to notify the AF of changes in the parameter values to which the AF has requested notifications in case of changes via the AF-initiated bridge management procedure. The procedure starts with the NW-TT creating the [BRIDGE MANAGEMENT NOTIFY] message. The NW-TT then includes the different bridge management parameters to be reported to the AF plus their current value in the bridge status IE of the message. The NW-TT starts a timer T350, which is a timer tasked with distributing the [BRIDGE MANAGEMENT NOTIFY] messages and sends the message to the AF.

When the AF receives the message, it creates an ACK. It also sends the message ACK back to the NW-TT, which upon receiving the ACK, stops the T350 timer. [21], [27]. The T350 timers calculates the time that is taken for the messages to be transmitted between the AF and NW-TT.

3.3 TSN prototypes for LAN and 5G networks

In building a LAN network for testing the TSN prototype, the TSN Evaluation Kit is one of the tools used. The kit provides TSN gateway functionality that enables a quick assessment of the different TSN features, which helps better understand how TSN works. The Innovasic's TSN kit can utilize the REM Switch chip (Innovasic's fido5000) to provide a TSN solution to different applications.

Figure 12 shows the architecture and different components making the kit usable for deploying and testing TSN functionality. When implemented using the equipment, the various possible architectures are partitioned and integrated.

By using the TSN gateway functionality, it is possible for a non-TSN device to participate in the TSN network without the native support for the TSN-specific features. The TSN gateways support different IEEE specifications such as 801.1AS, AS-REV (Time synchronization), 802.1Qbv (Scheduled Traffic), 802.1Qci (Ingress

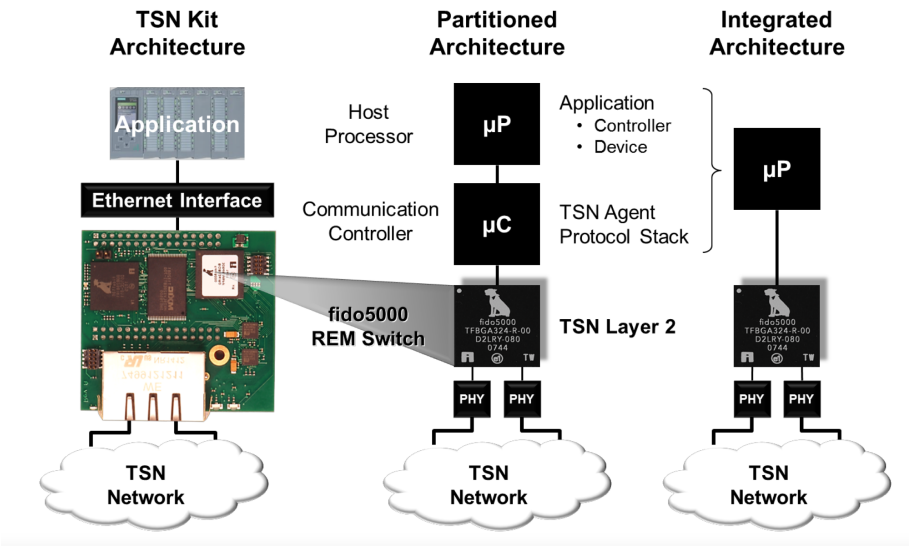

Figure 12: TSN Evaluation Kit architecture

Policing), 802.1CB (Seamless Redundancy), 802.1Qcc (Stream Reservation Protocol), and 802.1Qbu/802.3br (Preemption). Out-of-the box TSN gateway is pre-installed with software that offers support for 802.1AS, 802.1Qbv, 802.1Qcc, and stream translation. The GUI currently provides the support for the 801.1Qcc, with future implementations looking to support gateway configuration through a Central Network Controller (CNC).

When connected through a gateway, network topologies with more than one TSN kit automatically perform time synchronization. In the absence of a GM, one of the kits becomes the GM, the rest adopts the role of slaves. The preinstalled GUI offered as a web server helps to tune the gateway configurations of the kits to either switch between different settings for a GM or slave.

When the synchronization completes, the user can define the different streams from the Ethernet device that needs translating into the TSN network through the webserver. The GUI offers configuration for translating packets from the talkers to a format routable through the TSN network. Such packets need not be gPTP packets. The stream translation process assigns VLAN tags and priorities to different TSN streams. The relevant kits handling the streams insert them into queues with time windows dictating at what stage such streams can egress out of the network. The whole process is done through the web server GUI. The figure 13 shows the topology that will be used for LAN networks when trying to test and measure E2E synchronization together with the other components involved, such as talkers and listeners.

As for the TSN integration with the 5GS, the network is no longer just wired but rather wireless. The 5GS is a logical TSN bridge and consists of the 5G network components and the TSN translators. Figure 14 shows a prototype used to test and measure the integration and time synchronization between 5GS and TSN.

Both prototypes used for testing TSN time synchronization need three TSN

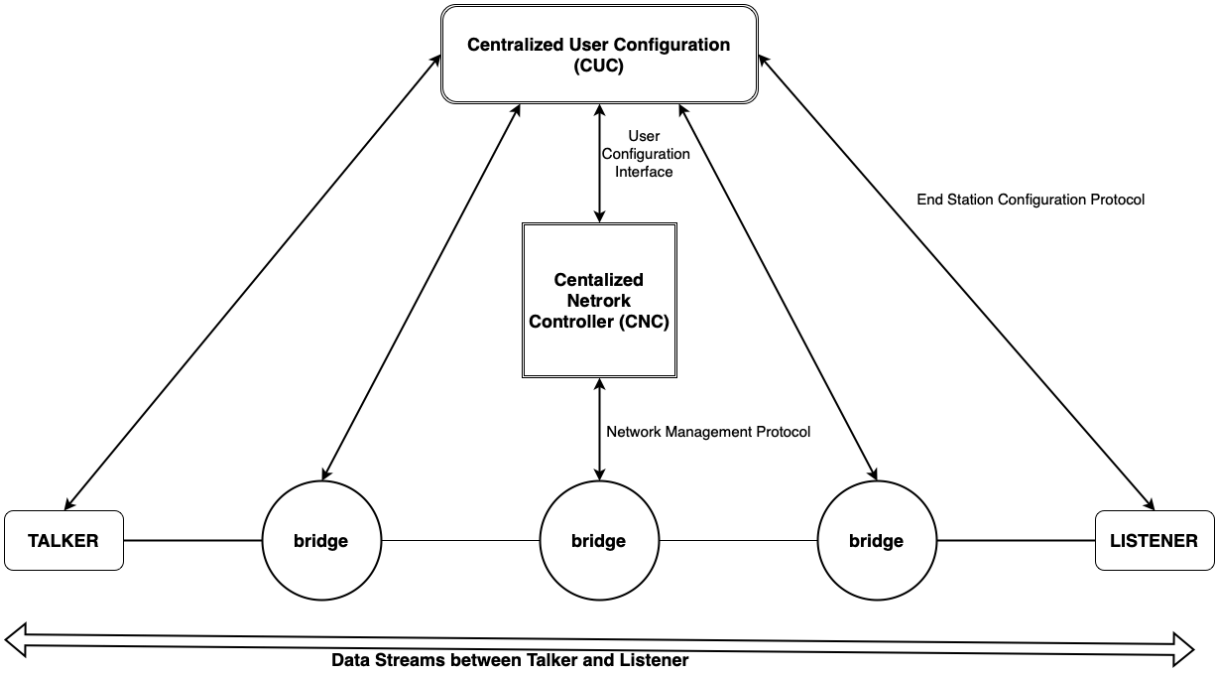

Figure 13: TSN prototype for Ethernet Network

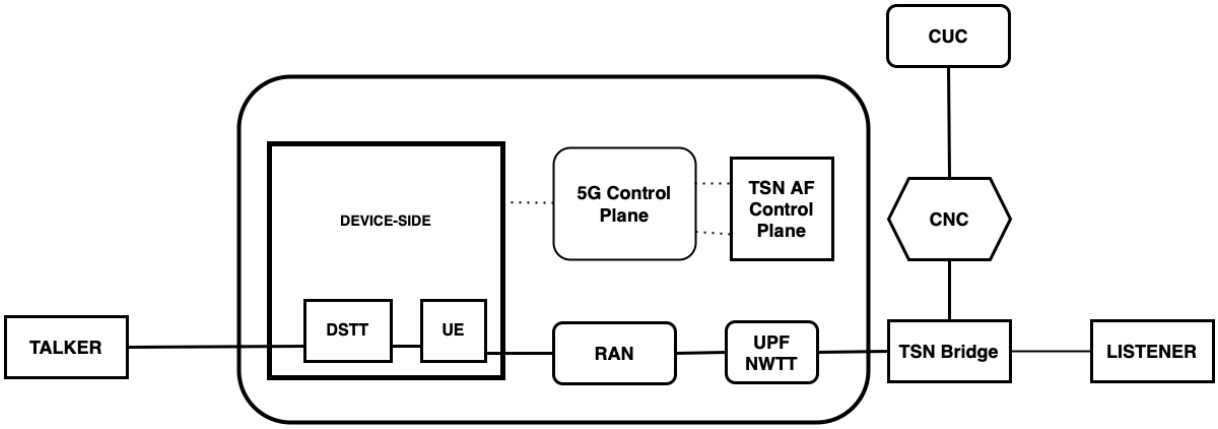

Figure 14: 5GS-TSN Integration Prototype

evaluation kits. The topology can be further enhanced/modified as applications demand to get the best testing and measurement scenarios of time synchronization in TSN networks. [27]

4 Environment requirements

This chapter describes the technical specification of the needed equipment used in the setting, testing, and debugging of the 5GS-TSN. Section 4.1 highlights the reasons for the need and the benefits offered by hardware timestamping. At the same time, section 4.2 covers the discussion regarding the choice of L3 test switches to be used in forwarding and switching gPTP packets. Section 4.3 discusses how to theoretically separate the gPTP/TSN traffic between two nodes to achieve control in packet handling and modification. Lastly, in Section 4.4, a short description is given of the open-source gPTP implementation, which is used as the GM and to synchronize to the other TSN modules in the test configuration.

4.1 Hardware timestamping

According to the IEEE definition of PTP, the fundamental component at the core is the exchange of messages between clocks mostly, master and slave. For a simple setup consisting of a message sent between the master and slave, and monitoring its arrival time. It is possible to record different time instances, for example, firstly, when the Sync message is sent (T1) by the master, secondly, when the same message is received by the slave (T2) based on the Follow_Up message received by the slave. Thirdly, is the T3 which involves the DelayRequest message sent to measure the delay and the resulting response contained in the DelayReponse from the master (T4).

In most scenarios, a symmetric underlying path is assumed in delay measurements. As long as the delays are large but similar, there is no cause for concern because the offset will reflect the difference when gPTP nodes exchange the messages between the master and slave.

In networks, queues are usual and negatively impact different networks' performance to a considerable extent. Network queues are virtually present in any networking equipment, ranging from basic routers, switches, and even end devices such as PCs. In ideal scenarios, the different messages from various protocols spend the least time in the queues.

On the other hand, when that is not the case, such messages spend a majority of time waiting to be processed from the queues. Reasons for this clog in handling packets can stem from an intermittent internet connection, connections centered around shared ports, which then have to give precedence to one group of messages over the others. The precedence is useful when considering the usefulness of the packets in the network. Depending on the scenario, this queue problem can either result in better timings or, in the worst case, cause considerable delays in the manner of milliseconds or even seconds. [28]

Hardware timestamping is then the solution for the problems caused by a software implementation of different entities. In hardware timestamping, anytime a message arrives to or departs from a given port on a network, a particular type of hardware is tasked with generating a timestamp by using the local clock, a media-independent interface lying between the MAC and PHY layers of the protocol stack.

The benefit offered by using hardware timestamping is the removal of the unpredictability in responses by software and operating systems [29] [30]. The higher timestamping accuracy needed for the 5GS-TSN requires the use of hardware timestamping. Hardware timestamping has its limits as when too many messages are sent over a port at any given time its performance degrades. In non-symmetric paths, the use of reverse packets can be used to track delays in both directions. The disadvantage is that the process is a little complicated to accomplish.

4.2 L3 TSN Switches

Testing the TSN functionality and their incorporation to the TSN modules (NW-TT and DSTT) needs L3 switches. The choice in this regard was the TSN Evaluation Kits (TEK) from Innovasic. These kits possess the capabilities useful in testing emergent IEEE 802.1 TSN standards such as the 802.1Qbv, 802.1AS among others.

The switches have capabilities necessary for experiments involving time synchronization, frame preemption, stream reservation protocols, seamless redundancy, scheduled traffic, and ingress policing. The kits offer plug-and-play functionalities, making it easier to test the different TSN functionalities. By simply connecting the TEKs, it is possible to see the packet flow and reporting of the different timing offsets. These are, for example, the delay between sending and receiving nodes. This timing information is the basis for measuring TSN metrics such as delay. [27]

In measuring networks delays, verifying and reporting the given TSN modules (NW-TT and DSTT) syncing accuracy levels, the project uses L3 Switches from Innovasic.

4.3 TSN traffic separation between networks

Figure 15 shows the NetGear switches used in separating the traffic between different TSN devices. The traffic into SWITCH-1 consists of synchronization messages between the two TSN Evaluation Kits (TEKs), the Grandmaster and the NW-TT. The packets entering this switch are those that have been captured by the NW-TT,modified and eventually forwarded via the switch towards the DS-TT.

The other network consisting of SWITCH-2 receives packets from the NW-TT. The packet exchange between the SWITCH-2 and the DS-TT will be the basis of the overall synchronization between the GM, NW-TT and DS-TT. This separation of traffic into different networks will be helpful later in testing the synchronization accuracy and reporting.

4.4 Open-Source GM Configurations

Most of the PTP solutions that exist and are widely popular such as PTPv1 and PTPv2 do not seamlessly work well with gPTP. To ensure the higher accuracy levels needed for TSN synchronization, a PTP solution supporting the optimal configuration of the GM beforehand is necessary.

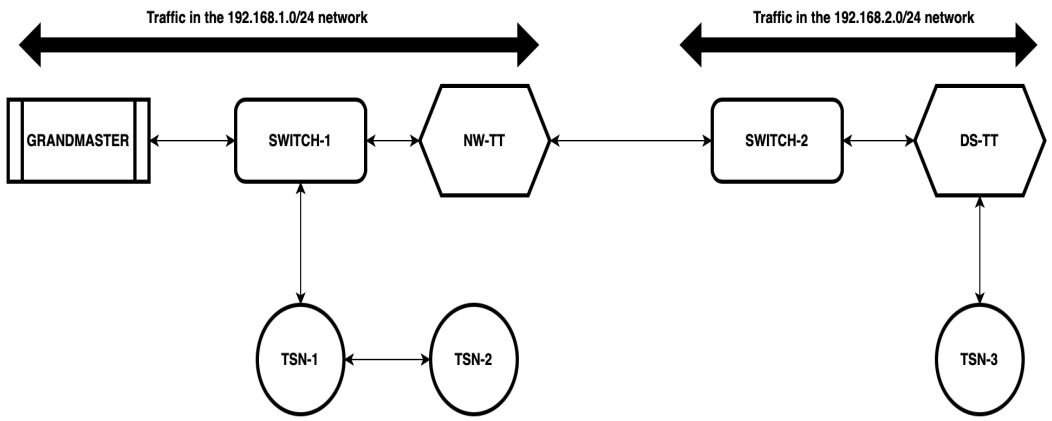

Figure 15: Separation of traffic into two networks using switches

Many open-source solutions have varying configuration efforts needed to make them work. One particular choice was the Excelfore's gPTP considered production-worthy, gPTP Ethernet stack. Excelfore's gPTP implementation is a well-known solution applied in the synchronization of autonomous cars, making it a solution to provide higher accuracy levels regarding timing synchronization. Additionally, the configuration and setup tend to be straightforward as the same company supplies two required packages on GitHub with more detailed documentation regarding the necessary configuration changes.

A little configuration change is needed to ensure that the Excelfore's gPTP software is successfully installed and acts as the GM. After building the software, it involves changing the configuration file parameters, local master priority 1 and 2 to be lower (default = 255), compared to those that are to be configured for other TSN switches (slaves) that will make the larger TSN network have only one GM with the rest of the TSN switches synchronizing to that GM.

4.5 Summary of necessary equipment

The table shows the summary of the different hardware equipment needed for the project. Chapter 5 discusses the TSN modules C language software implementation.

Index	Item Name	Quantity	Description
1	Linux Desktop PCs	2	Linux PCs with peripherals attached
2	Excelfore GM Software	1	GitHub sourced C++ Software
3	Innovasic's TSN Switches	3	Switches to form TSN network
4	Linux Mini PCs	2	NW-TT and DSTT hosting

The technical specifications needed (such as RAM) for all the PCs are left out of the project, as the most vital requirement is that the NIC supports hardware timestamping. This caveat leaves room for even microcomputers to be used in this implementation to execute the software and provide the needed hardware interactions necessary for accurate timestamps.

4.6 Methods used in thesis

This section introduces the different methodologies used in coming up with the relevant information used in prototyping the TSN modules. Through the IEEE 802.1AS-WG, the primary purpose is to develop the feature specifications relating to the 802.1 projects. Most of the technical requirements specifying how to achieve the integration and synchronization accuracy levels exist in IEEE articles and publications on 5GS.

The readings related to the technical specifications for the 5GS-TSN synchronization specify the changes needed to the packets and the performance requirement levels to be up with acceptable limits. These articles, as such, provided a starting point in coming up with ideas necessary to develop the TSN modules. [27]

Again the WG has different revisions based on discussions held at the IEEE. These were important, too, as many technical specifications are often updated, and others are frequently reviewed and revised. So to a more considerable extent, the core methodology used in implementing the TSN modules was from the documentation and articles provided by the IEEE.

Secondly, many organizations are focused on implementing and optimizing the specifications documented by the IEEE. In this regard, they take the standards and implement them in their way. In most cases, they present results that might be open source but complicated as their documentation might not be updated regularly.

In this case, by reading the company's marketing articles related to the product advertisements, it was possible to get some vital information and indication regarding what they considered when coming up with TSN-related products. A case example is Excelfore, which has a widespread gPTP implementation used in autonomous cars, which needs the timing accuracy to be in nanoseconds range.

Thirdly, previous PTP daemon(PTPd) [6] implementation served as a promising starting point for indicating how the TSN worked in previous technologies. Taking this and trying to see the needed additions served as an indication, moving forward, and helped develop a better way of implementing the TSN modules. Most of the reading in this section involved going through the Linux ptpd implementation and perusing the code to gain significant insights into what is happening in the PTP implementation.

At the opening phases of the project, it involved running the ptpd software and synchronizing it with the other machines that were either configured to be master(s) or slave(s). Earlier on, this preliminary testing helped further, as a realization came from these experiments that the PTP did not synchronize well with gPTP. In most

[6] git@github.com:ptpd/ptpd.git

cases, the results were in seconds accuracy instead of nanoseconds. This provides a useful insight that gPTP time synchronization accuracy is much better compared to PTP.

Lastly, some oral discussions with the thesis supervisor at different stages provided essential insights. These conversations and discussions served as helpful indications, either delivering valuable insights or pointing to the right directions. These discussions involved advice on the implementation options necessary to provide/ensure better time synchronization between the TSN modules. Additional talks with experts from companies who have tested TSN implementations were helpful too. These offered positive and valuable information useful in implementing and testing TSN modules.

5 Implementation and results

This chapter of the thesis involves the design and implementation of the TSN Translators based on the recommendations specified by the IEEE 802.1 WG. The language of choice for this was C since it provided the different Linux libraries vital when writing code that interacts with the kernel.

Linux offers multiple ready-to-use libraries and modules that make writing sockets that handle traffic between different nodes straightforward. The advantage provided is knowing what is needed and plugging most functionalities to generate the desired behavior.

This thesis chapter forms the basis upon which the 5GS (emulated using switches) interacts with the other time-aware devices available in the TSN network. The IEEE provides recommendations on what to program. These recommendations consist of changes in PTPv2 to ensure that 5GS can be implemented as a bridge and ensure that 5GS can route TSN traffic successfully.

The switches replace the actual 5GS and separate the gPTP packets depending on their origins. This emulation is helpful as it serves to measure the delay between the two TSN modules. The advantage of this approach is that when the TSN modules are integrated into the 5GS eventually, there are no additional modifications required to measure synchronization accuracies.

Section 5.1 discusses the theoretical changes needed in the 5GS to ensure the support of TSN. Section 5.2 details the NW-TT functionalities implemented in C and provides code samples of sections considered necessary. Section 5.3 presents the DS-TT functionalities implementation.

5.1 Theoretical changes in 5GS to support TSN

Under the IEEE Release 16, there is a need for the 5GS to be time-aware and support gPTP. While generically, the 5GS delivers gPTP messages over the user plane, TSN translators need to cater to ingress timestamping. This involves monitoring the ingress ports and upon detecting gPTP packets, timestamps are inserted to the relevant ones. One of the notable features of 5GS synchronizations relates to achieving ingress time signaling. Ingress time is time it takes for packets to move from the NW-TT to the DS-TT.

The 5GS has different components (such as the UE), which might be the source/destination of traffic signaled throughout the 5GS. Depending on the origin and destination, this information can be considered downlink or uplink. It is worth noting that the whole 5GS does not need to support IEEE 802.1AS operations as only NW-TT and DS-TT need them. Additionally, the different 5GS components such as gNB, DS-TT, and NW-TT need synchronization to the 5G internal clock [31].

Generally, in an attempt to achieve synchronization, the 5GS calculates the residence time, which the TSN modules add to the correctionField of the different TSN packets. The modified packets are eventually forwarded to the end devices such as UEs.

5.2 NW-TT functionalities implementation

The upstream packets that come from different TSN bridges (such as TSN evaluation kits used in this setting or GM) on arrival at the NW-TT need modifications to insert ingress timestamping into the PTPv2 packets. These timestamps are either inserted into the SYNC messages (one-step) or the exact time the SYNC messages are sent, is inserted into the subsequent FOLLOW_UP messages (two-step).

Secondly, the upstream delay (stored in either the correctionField of SYNC or FOLLOW_UP packets) needs updating to a newer value, depending on the rateRatio inserted in the correctionField of SYNC or FOLLOW_UP packets.

The devices in the test setting form a small and detached network with only one interface at the GM connecting to the internet for time synchronization purposes. The other components have a local test network to which they connect. Based on the small nature of the topology with a small number of devices, having the rateRatio modified and inserted to either the SYNC or FOLLOW_UP does not have a huge impact since there are a few time TSN synchronization devices with preset configuration. This ensures that devices do not cause huge time offsets which might always impact the synchronization and necessitate the need for the rateRatio. The closeness of the devices ensures that the rateRatio rarely changes and in this project, though assumed at first to be 0, does not change much [31], [32]. The rateRatio value of 0 forms a basis that the two local clocks used do not have the same time, with the one at the GM lagging behind the one at the NW-TT. The synchronizations based on packets exchanges, will function to ensure that the times are corrected a new and updated rateRatio value is calculated.

In TSN modules implementation, packet capture and modification is a necessary process and as such, considerations are made regarding what packets to capture and in what manner. The preferred solution in this regard is the use of Linux raw sockets. The reason for the choice of Linux raw sockets is that they provide all the headers related to a particular protocol, allowing more straightforward modifications.

5.2.1 Structure of gPTP packet headers

The TSN modules implementation first need to have a template of the data structures of the different PTPv2 headers (the standard headers) shared by all the other PTPv2 messages. After traffic capture, the structures help to point to the required packet header modifications.

All the gPTP messages share a standard gPTP header. In addition to the common header there are message-specific parts defined for the SYNC and FOLLOW_UP messages. The following listings show the templates for both the SYNC and FOLLOW_UP packet:-

```
struct sync_msg
{
    struct ptp_header hdr;
    struct Timestamp originTimestamp;
};
```

```
struct ptp_header
{
    uint8_t messageType: 4;
    uint8_t transportSpecific: 4;
    uint8_t ptp_version: 4;
    uint8_t reserved0: 4;
    uint16_t messageLength;
    uint8_t domainNumber;
    uint8_t reserved1;
    struct gptp_flags flags;
    int64_t correctionField;
    uint32_t reserved2;
    struct GPTPPortIdentity sourcePortIdentity;
    uint16_t sequenceId;
    uint8_t control;
    int8_t logMsgInterval;
};
```

Listing 1: gPTP common header (PTPv2)

Listing 2: SYNC message header

```
struct follow_up_msg
{
    struct ptp_header hdr;
    struct Timestamp preciseOriginTstamp;
    struct follow_up_info_tlv tlv;
};
```

Listing 3: FOLLOW_UP message header

The gPTP messages (SYNC and FOLLOW_UP) are the only ones that need modifications. However, the implementation uses all the headers for the other gPTP messages, such as ANNOUNCE, PEER_DELAY_REQ messages to copy the message contents from a receiving to a sending buffer. The NW-TT ingress port/interface needs to capture all the incoming gPTP messages from the GM and store them into a receive buffer. The socket has all the necessary options to ensure that it captures only gPTP messages, which will undergo modifications by the NW-TT. The NW-TT will copy the modified packets into the transmit buffer, and will send them to the DS-TT module for synchronization.

5.2.2 Hardware timestamping in Linux kernel

The necessary precondition in implementing the TSN clock synchronization is the ability to obtain the gPTP messages timestamps freely. In calculating the timing offsets needed to correct/synchronize times between the master and the slaves, the transmit and receive timestamps are needed and should be present in the relevant messages. [33]

The benefit of using hardware timestamping stems from its ability to eliminate jitters and delays when the gPTP messages pass the network protocol stack. [33]

After preparing the headers, a receive socket needs to be defined, depicting from what protocols the traffic should be captured, among other things. The socket domain, type, and protocol are AF_PACKET, SOCK_RAW, and ETH_P_1588. The capturing socket needs an Ethernet interface (such as eth0) from a computer with NIC supporting hardware timestamping. The first implementation is the receive hardware timestamping() function, which generates receive timestamps for gPTP messages. When the Linux kernel detects the first gPTP message header, the process generates timestamps for the gPTP messages.

```
void set_rx_hw_timestamping(struct GPTP *GPTP)
{
    struct ifreq ifr;
    struct hwtstamp_config config;
    int timestamp_flags = 0;
    struct timespec timeout;
```

```
if ((GPTP->socket1 = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_1588))) < 0)
    bail("socket creation failed");
memset(&GPTP->ifreqIndex, 0, sizeof(struct ifreq));

strncpy(GPTP->ifreqIndex.ifr_ifrn.ifrn_name, NW-TT_RX_INTERFACE,
    GPTP_IF_NAME_SIZE - 1);

if (ioctl(GPTP->socket1, SIOCGIFINDEX, &GPTP->ifreqIndex) < 0)
    bail("failed to retrieve device index");

memset(&GPTP->ifreqMac, 0, sizeof(struct ifreq));
strncpy(GPTP->ifreqMac.ifr_ifrn.ifrn_name, NW-TT_RX_INTERFACE,
    GPTP_IF_NAME_SIZE - 1);

if (ioctl(GPTP->socket1, SIOCGIFHWADDR, &GPTP->ifreqMac) < 0)
    bail("failed to retrieve MAC address of the device");

memset(&GPTP->ifreqHW, 0, sizeof(struct ifreq));
memset(&config, 0, sizeof(struct hwtstamp_config));
strncpy(GPTP->ifreqHW.ifr_ifrn.ifrn_name, NW-TT_RX_INTERFACE,
    GPTP_IF_NAME_SIZE - 1);
config.flags = 0;
config.tx_type = HWTSTAMP_TX_ON;
config.rx_filter = HWTSTAMP_FILTER_ALL;
GPTP->ifreqHW.ifr_ifru.ifru_data = (char *) &config;

if (ioctl(GPTP->socket1, SIOCSHWTSTAMP, &GPTP->ifreqHW) < 0)
    bail("hardware timestamping failed");
else
    printf("HW Timestamp enabled. tx_type = %d and rx_filter = %d\n",
          config.tx_type, config.rx_filter);
timestamp_flags |= SOF_TIMESTAMPING_RX_HARDWARE;
timestamp_flags |= SOF_TIMESTAMPING_TX_HARDWARE;
timestamp_flags |= SOF_TIMESTAMPING_RAW_HARDWARE;
timestamp_flags |= SOF_TIMESTAMPING_OPT_CMSG;
timestamp_flags |= SOF_TIMESTAMPING_OPT_ID;
timestamp_flags |= SOF_TIMESTAMPING_SYS_HARDWARE;

if (setsockopt(GPTP->socket1, SOL_SOCKET, SO_TIMESTAMPING,
    &timestamp_flags, sizeof(timestamp_flags)) < 0)
    bail("error setting socket options for timestamping");

timeout.tv_sec = 1;
timeout.tv_nsec = 0;
if (setsockopt(GPTP->socket1, SOL_SOCKET, SO_RCVTIMEO, &timeout,
    sizeof(timeout)) < 0)
    bail("error setting socket timeout options");

strncpy(ifr.ifr_ifrn.ifrn_name, NW-TT_RX_INTERFACE, GPTP_IF_NAME_SIZE -
    1);
ioctl(GPTP->socket1, SIOCGIFFLAGS, &ifr);
ifr.ifr_ifru.ifru_flags |= IFF_PROMISC;
ioctl(GPTP->socket1, SIOCGIFFLAGS, &ifr);
```

```
    timestamp_flags = 1;
    if (setsockopt(GPTP->socket1, SOL_SOCKET, SO_REUSEADDR, &timestamp_flags,
        sizeof(timestamp_flags)) < 0)
        bail("failed to set socket address reuse");

    if (setsockopt(GPTP->socket1, SOL_SOCKET, SO_BINDTODEVICE,
        NW-TT_RX_INTERFACE, GPTP_IF_NAME_SIZE - 1) < 0)
        bail("failed to bind to device");

    transmit_address(GPTP);
    receive_address(GPTP);
}
```

Listing 4: Receive hardware timestamp generation

Listing 4 shows the process of establishing an index of the interface that supports hardware timestamping after socket creation. Given the interface name, it is possible to retrieve the MAC address associated with that interface. The socket declaration process automatically retrieves the MAC address, and this is the exact MAC address that the socket will use to receive incoming packets.

The process of enabling the hardware timestamping is taken care of by the hwtstamp_config. The needed configuration flags are the transmit_type, which turns on the relevant timestamps bits. The receive filter configuration describes what kind of gPTP messages need timestamping. The given configuration will timestamp all the gPTP messages since the NIC does not offer individual-level packet timestamping.

It is worth noting that timestamping all the gPTP messages might increase complexity as irrelevant gPTP messages not involved in calculating the delay are all timestamped, resulting in increased delays and resource consumptions.

To enable the defined hardware timestamping, the flag option parameter undergoes compound bit-wise OR (|=) operation for a combination ranging from hardware timestamp on receive hardware and hardware timestamp on transmit hardware, among other options. These tell the NIC how and where to apply timestamps at ingress and egress interfaces.

The subsequent processes in the code involve:- setting the socket options to enable the accuracy level of reporting timestamps (nanoseconds in this case), the timeout of receiving packets, re-use of the socket address multiple times, and even binding the device to the interface. The final step in configuring the timestamping is the configuration of the addresses from which the socket will receive packets and send packets.

The configuration for the transmit timestamping to a more considerable extent is similar to the receive_timestamping. However, the significant difference is the need to receive the packets from a given interface and forward them to another one. The packets are thus obtained from the NW-TT_RX_INTERFACE and delivered to NW-TT_TX_INTERFACE. Listing 5 shows the sample configuration code used in depicting the significant differences between the two timestamping implementations.

```
void set_tx_hw_timestamping(struct GPTP *GPTP)
{
struct ifreq ifr;
struct hwtstamp_config config;
int timestamp_flags = 0;
struct timespec timeout;

if ((GPTP->socket2 = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_1588))) < 0)
    bail("socket creation failed");
memset(&GPTP->ifreqIndex, 0, sizeof(struct ifreq));

strncpy(GPTP->ifreqIndex.ifr_ifrn.ifrn_name, NW-TT_TX_INTERFACE,
    GPTP_IF_NAME_SIZE - 1);

if (ioctl(GPTP->socket2, SIOCGIFINDEX, &GPTP->ifreqIndex) < 0)
    bail("failed to retrieve device index");

memset(&GPTP->ifreqMac, 0, sizeof(struct ifreq));
strncpy(GPTP->ifreqMac.ifr_ifrn.ifrn_name, NW-TT_TX_INTERFACE,
    GPTP_IF_NAME_SIZE - 1);

if (ioctl(GPTP->socket2, SIOCGIFHWADDR, &GPTP->ifreqMac) < 0)
    bail("failed to retrieve MAC address of the device");

memset(&GPTP->ifreqHW, 0, sizeof(struct ifreq));
memset(&config, 0, sizeof(struct hwtstamp_config));
strncpy(GPTP->ifreqHW.ifr_ifrn.ifrn_name, NW-TT_TX_INTERFACE,
    GPTP_IF_NAME_SIZE - 1);
config.flags = 0;
config.tx_type = HWTSTAMP_TX_ON;
config.rx_filter = HWTSTAMP_FILTER_ALL;
GPTP->ifreqHW.ifr_ifru.ifru_data = (char *) &config;

if (ioctl(GPTP->socket2, SIOCSHWTSTAMP, &GPTP->ifreqHW) < 0)
    bail("hardware timestamping failed");
else
    printf("HW Timestamp enabled. tx_type = %d and rx_filter = %d\n",
           config.tx_type, config.rx_filter);
timestamp_flags |= SOF_TIMESTAMPING_RX_HARDWARE;
timestamp_flags |= SOF_TIMESTAMPING_TX_HARDWARE;
timestamp_flags |= SOF_TIMESTAMPING_RAW_HARDWARE;
timestamp_flags |= SOF_TIMESTAMPING_OPT_CMSG;
timestamp_flags |= SOF_TIMESTAMPING_OPT_ID;
timestamp_flags |= SOF_TIMESTAMPING_SYS_HARDWARE;

if (setsockopt(GPTP->socket2, SOL_SOCKET, SO_TIMESTAMPING, &timestamp_flags,
    sizeof(timestamp_flags)) < 0)
    bail("error setting socket options for timestamping");

timeout.tv_sec = 1;
timeout.tv_nsec = 0;
if (setsockopt(GPTP->socket2, SOL_SOCKET, SO_RCVTIMEO, &timeout,
```

```
        sizeof(timeout)) < 0)
        bail("error setting socket timeout options");

    strncpy(ifr.ifr_ifrn.ifrn_name, NW-TT_TX_INTERFACE, GPTP_IF_NAME_SIZE - 1);
    ioctl(GPTP->socket2, SIOCGIFFLAGS, &ifr);
    ifr.ifr_ifru.ifru_flags |= IFF_PROMISC;
    ioctl(GPTP->socket2, SIOCGIFFLAGS, &ifr);

    timestamp_flags = 1;
    if (setsockopt(GPTP->socket2, SOL_SOCKET, SO_REUSEADDR, &timestamp_flags,
        sizeof(timestamp_flags)) < 0)
        bail("failed to set socket address reuse");

    if (setsockopt(GPTP->socket2, SOL_SOCKET, SO_BINDTODEVICE,
        NW-TT_TX_INTERFACE, GPTP_IF_NAME_SIZE - 1) < 0)
        bail("failed to bind to device");

    transmit_address(GPTP);
    receive_address(GPTP);
}
```

Listing 5: Tansmit hardware timestamp generation

5.2.3 Necessary packet modifications

As stated in Section 2.1, the main objective is to enable hardware timestamp from the interface on the Linux machine, knowing that all the interfaces support hardware timestamping (NIC support). The ingress_timestamping function does the packet modifications on the NW-TT module, starting with the definition of the MAC address of the GM. The defined MAC address filters the messages from the GM, which are then captured and stored in the rxBuffer from which the txBuffer copies them for the relevant gPTP messages.

The rxMessageHeader tests whether the defined socket captures gPTP packets. For the subsequent phases to continue, there must be gPTP packets in the traffic, or else this will be the final test after which the socket will fail. When the socket successfully detects the presence of gPTP messages, a verification is done to check whether the requested gPTP type and level of timestamping conforms to those supported by the NIC. Timestamps generation and insertion is the next step when a match exists for the necessary packets.

When the socket captures all the relevant gPTP packets, the rxBuffer stores them as depicted in Listing 7.1. The txBuffer copies the messages from the rxBuffer using the memcpy() function, after doing the required modifications and timestamping. As the gPTP process is two-step, the FOLLOW_UP message carries most of the vital information related to timing. SYNC messages are copied from the rxBuffer to the txBuffer without modifications and sent to the destination address with the same sequenceID in the original message.

The FOLLOW_UP messages, on the other hand, firstly, need to be updated with the correctionField on the SYNC messages. The upstream delay is usually present in the correctionField of the received FOLLOW_Up message. Additionally,

the preciseOriginTimestamp needs to be converted into nanoseconds and then added to the correctionField. The final change that needs to be made relates to the TLV header/portion. The cumulativeScaledRateOffset needs updating with the rateRatio, which is initially assumed to be 0 (reported by the GM). The rateRatio assumed value of 0 enables the cumulativeScaledRateOffset that is sent in the gPTP header packets to not be 0. This is based on the formula ((rateRatio - 1.0)(2**41)). The calculated and accumulated rateRatio, after being inserted in the packet header, and measured at the DS-TT, can provide insights on what changed when the packets move from the NW-TT to the DS-TT. This is why the original value of 0 is assumed for the rateRatio.

With the changes made to the FOLLOW_UP messages, the next stage is to send the messages to the destination. In this regard, the sendto() function retrieves messages from the txBuffer. It uses the DS-TT MAC address configured in the txSocketAddress to send the packets, which automatically deletes the packets to free memory in the buffers. The process described in this section entails all the needed packet handling at the NW-TT module.

5.3 DS-TT functionalities implementation

Like in the NW-TT module, there are two sockets at the DS-TT: one to receive the packets from a receive interface and the other for forwarding the modified packets to their destination. The functionality regarding generating timestamps and where to insert them is similar to the process discussed for the NW-TT.

After the declared socket captures of the packets, it copies them to the rxBuffer using the egress function. The subsequent process generates the TSe timestamps, calculates the residence time, and updates it accordingly using the rateRatio. To prepare the individual packets before sending them to the final destination (e.g., receiver MAC address or destination PTP agents), the timing information inserted in earlier phases needs removal. This timing information exists in the correctionField of the FOLLOW_UP messages. After deleting the upstream correctionField and updating the value with the downstream correctionField, the messages are ready for delivery to PTP End Stations.

Listing 7.2 details the code for the DS-TT involving all the steps related to the egress timestamping.

5.4 TSN testing topology

Figure 16 shows the topology of the different equipment forming the TSN network. Traffic in this scenario starts from the GM sending synchronization messages (Multicast). The process begins with SWITCH-1 receiving the synchronization messages sent by the GM. At the same time, the TSN-1 sends synchronization messages both to the SWITCH-1 and TSN-2. Since the default set priority in the GM (248) configuration is lower than in all the TSN switches (255), the TSN-1 synchronizes and becomes the slave. The TSN-1 has two ports, the one attached to the switch becomes the slave port, and the one connected to the TSN-2, the master port.

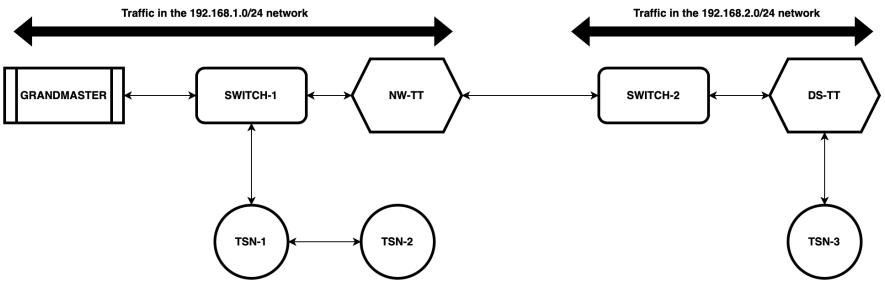

Figure 16: TSN topology for testing

The master port then synchronizes to the TSN-2. When the switch forwards the synchronization messages, they reach the NW-TT from the SWITCH-1 directly attached interface. The interface filters the packets coming from the GM based on the source MAC address of the GM. The NW-TT module then handles the packets by inserting the ingress time and updates the correction_field as necessary. When the NW-TT completes the packet processing, it sends them out to the interface connected to the SWITCH-2 destination MAC address of the DS-TT. This process makes the messages appear as if they were originating from the GM, while in essence, the origin is the NW-TT.

The DS-TT receives the messages from one interface (interface attached to SWITCH-2) and retrieves the TSi inserted at the NW-TT. The DS-TT generates the egress timestamp (TSe) when the messages are received by the socket. Since the network topology is a small one, in terms of the number of components making the TSN network, the rateRatio does not seem to have any impact. For this reason, the original value, inserted from the NW-TT side, when it reaches the DS-TT, has a value of one, based on the calculations. The rate ratio which is the ratio of the delay requester frequency to that of the delay responder clock frequency is assumed to be one. This is because when devices are connected together in close proximity, the clock frequencies tend to be similar. When the ratio of these clock frequencies is compared, the results is one. The important parameters to consider when calculating the delay between the TSN modules are, therefore, the TSe and TSi, together with the correction_field values accumulated from the GM as rate ratio. The rate ratio is ultimately used in time adjustments for those clocks that might be quite far from the GM [34].

When the DS-TT completes all the packet handling, modifications, and forwarding, the expected behavior is for the DS-TT GUI to portray the relevant synchronization information. Through this information, it is possible to judge whether synchronization was possible and the accuracy levels. Additionally, it is possible from the results to find the topology choices based on the synchronization information exchanged in the network.

First, the results achieved by analyzing the DS-TT GUI dashboard indicate that the steps removed (e.g., steps between different gPTP Instances) from the master was 1. This result means that from the DS-TT's perspective, it appears that it is directly attached to the GM and uses those messages from the GM to synchronize its clock. This masquerading of the messages to appear as if they are directly from the GM is beneficial as it ensures that synchronization happens faster and any adjustments at the GM can be detected faster and propagated to TSN devices connected to the GM.

5.5 GM synchronization sequence

The Excelfore gPTP implementation provides a configuration file **gptp2.conf** containing the required parameters to achieve the desired software behavior when considering the GM necessary configurations. Figure 17 shows the GM configurations, which sets the priority as 248, making it lower than any other priorities set in the slave TSN switches (255).

```
#
#  10.3.2 systemIdentity
#  PRIMARY is used for domainNumber=0, SECONDARY is used for the other domains
CONF_PRIMARY_PRIORITY1 248 # 8.6.2.1 priority1, 255 for not GM-capable
CONF_PRIMARY_CLOCK_CLASS 248 # 8.6.2.2 clockClass
CONF_PRIMARY_CLOCK_ACCURACY 0x22 # 8.6.2.3 clockAccuracy, accurate to within 250ns
CONF_PRIMARY_OFFSET_SCALED_LOG_VARIANCE 0x436A # 8.6.2.4 offsetScaledLogVariance
CONF_PRIMARY_PRIORITY2 248 # 8.6.2.5 priority2
#
```

Figure 17: GM base settings

After compiling the source code and starting the GM, an interface that supports hardware timestamping needs to be included in the scripts as shown in Listing 6 with the -c flag representing the configuration file and -d the Ethernet device interface name. It is in the configuration file that all GM parameter settings are declared.

```
$ ./gptp2d -c /home/researcher/excelfore-gptp/gptp2.conf -d eno1
```

Listing 6: Command to start GM

When the GM starts, a priority set comparison occurs between the PTP Instances and the one with the lowest value selected as the GM. The GM has the lowest set priority, and the decision is straightforward. As a result, the MAC address of the GM changes from the default to that of the interface provided in the configuration parameters as shown in Figure 18.

The gPTP synchronization messages are sent periodically after the initial synchronization. This process is helpful since more or newer TSN devices can be introduced or removed from the network. Failures too can occur, which necessitates the selection process of the newer GM.

The Excelfore gPTP package [7] when built/compiled needs additional packages.

[7] https://github.com/xl4-shiro/excelfore-gptp

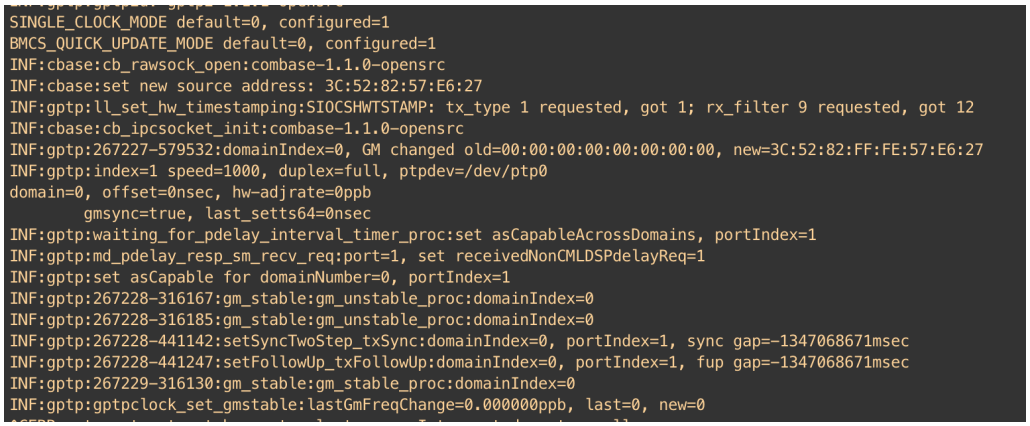

Figure 18: GM console log when started

```
int main()
{
  puts("NW-TT agent started...");
  set_rx_hw_timestamping(&gptp);
  set_tx_hw_timestamping(&gptp);
  ingress_timestamping(&gptp);
  close(gptp.socket1);
  close(gptp.socket2);
  exit(0);
}
```

Listing 7: Main function to start NW-TT

These are linked in the company's GitHub account, namely, xl4combase [8] and xl4unibase [9]. The three packages discussed are all the needed configurations to make the GM predefined and be a source of exact time synchronization and a means through which other devices can synchronize.

5.6 NW-TT synchronization sequence

For the NW-TT module, as shown in Listing 7, the first two commands set the receive and transmit hardware timestamping and reports on what was requested and what is possible. The ingress timestamping is then started on incoming packets, which continues forever or until the interface no longer receives gPTP packets with GM MAC address. Listing 7 shows the two sockets (socket1 and socket2) used receiving and sending packets.

The NW-TT module then captures every packet with the GM MAC address, modifies the relevant ones (SYNC and FOLLOW_UP), and forwards them through the outgoing interface. Figure 19 shows a sample output for sent packets after

[8] https://github.com/xl4-shiro/xl4combase
[9] https://github.com/xl4-shiro/xl4unibase

modifications.

```
rate_ratio == 0.000000
FOLLOW_UP sent
rate_ratio == 0.000000
FOLLOW_UP sent
ANNOUNCE sent
ANNOUNCE sent
PEER_DELAY_RESP sent
PEER_DELAY_RESPONSE_FOLLOW_UP sent
PEER_DELAY_RESP sent
PEER_DELAY_RESPONSE_FOLLOW_UP sent
SYNC sent
SYNC sent
rate_ratio == 1.000000
FOLLOW_UP sent
rate_ratio == 0.000000
FOLLOW_UP sent
SYNC sent
SYNC sent
rate_ratio == 1.000000
FOLLOW_UP sent
rate_ratio == 0.000000
FOLLOW_UP sent
SYNC sent
SYNC sent
rate_ratio == 0.000000
FOLLOW_UP sent
rate_ratio == 0.000000
FOLLOW_UP sent
^Cmake: *** [Makefile:16: run] Interrupt
```

Figure 19: Sample gPTP message sent by NW-TT

The tests and verifications conducted at the NW-TT module denote that it works as anticipated in the configurations.

5.7 DS-TT synchronization sequence

The same configurations in the NW-TT start the DS-TT module. The module is programmed to run until manually stopped or when no gPTP packets are available from the incoming interface. When the DS-TT starts, it reports the TSi retrieved from the Follow_Up message originated from the NW-TT, together with the TSe generated from the HW timestamping. These two are helpful values for calculating the delay and accuracy of the TSN implementation.

The DS-TT, when started, reports on the hardware timestamping types enabled based on the configuration settings for the receiving and transmitting interfaces. The console displays the parameters such as **tx_type** and **rx_filter**, together with the values, which is one because the NIC does not offer per message timestamping but rather support from gPTP protocol messages.

Figure 20 shows the sample configuration depicting that hardware timestamping is enabled and the interface accepted the configuration requests.

The DS-TT continuously monitors messages from the GM MAC address and processes them. The DS-TT then forwards such packets to the TSN switch attached to it (TSN-3), which then reports whether synchronization was possible or not. The

```
root@dstt:/home/dstt/tsn-project/dstt# make c
gcc -g -Wall -Wextra -Werror -O0 dstt.c dstt.h -o dstt.out
maroot@dstt:/home/dstt/tsn-project/dstt# make run
sudo ./dstt.out
DSTT agent started...
HW Timestamp enabled. tx_type = 1 and rx_filter = 1
HW Timestamp enabled. tx_type = 1 and rx_filter = 1
```

Figure 20: Sample console log when DS-TT is started

TSN-3, through the GUI, is used to interpret the synchronization accuracy achieved from the received gPTP packets.

5.8 Thesis results

This chapter of the thesis presents the synchronization results achieved based on the TSN modules implemented. The TSN synchronization setup discussed in Chapter 5 forms the basis upon which the final testing results are retrieved from. The final synhronization results after the messages are finally modified and sent by the DS-TT, are visible from the TEK directly attached to the DS-TT. These synchronization results are visible through the GUI offered by the TEK.

To achieve the experimentation results, a simple bash script that scrapes the synchronization results from the GUI is taken into use. The script goes occasionally, to the website page, retrieves the synchronization value reported and stores it in a textfile. The whole bash script is as shown in Listing 8.

```
#!/bin/bash

VALUE=$(curl --silent 192.168.2.50/syncstat.stm | htmlq td | grep timeOffset
    | grep value | awk '{ print $5}' | cut -b 8- | awk '{ print substr( $0,
    1, length($0)-2) }')

if [[ $VALUE -gt 0 ]]
then
    echo $VALUE >> sample.txt
fi
```

Listing 8: Scraping synchronization results

To start the synchonization results scraping, the watch command in Linux is useful as it tells after how many seconds should the command be run. The command is as show in Listing 9.

```
watch --interval 300 bash synchronization_results.sh
```

Listing 9: Scraping after every 5 minutes

To achieve substantial synchronization results test data, the command was run from Friday evening (17:00) to Monday morning (10:00). This resulted in around 637 synchronization results data points.

A summary of the data is denoted in Table 1, which shows the minimum values recorded when synchronization was started as 0.9 μs and maximum to be around 15 milliseconds. The median values are about 328 μs. The summary data provided is just a general overview, to helm understand better about synchronization and how it happens over a period of time, graph plots are more useful.

Table 1: Summary of synchronization results

Summary	**Value**
count	637.000000
mean	343.804504
std	647.191945
min	0.906000
25^{th} percentile	328.899000
50^{th} percentile	328.899000
75^{th} percentile	328.899000
max	15961.430000

Matlab plots of the data during the whole duration of running the TSN modules gives a better trend and helps in further understanding the synchronization. The Figure 21, show the plot using the whole dataset.

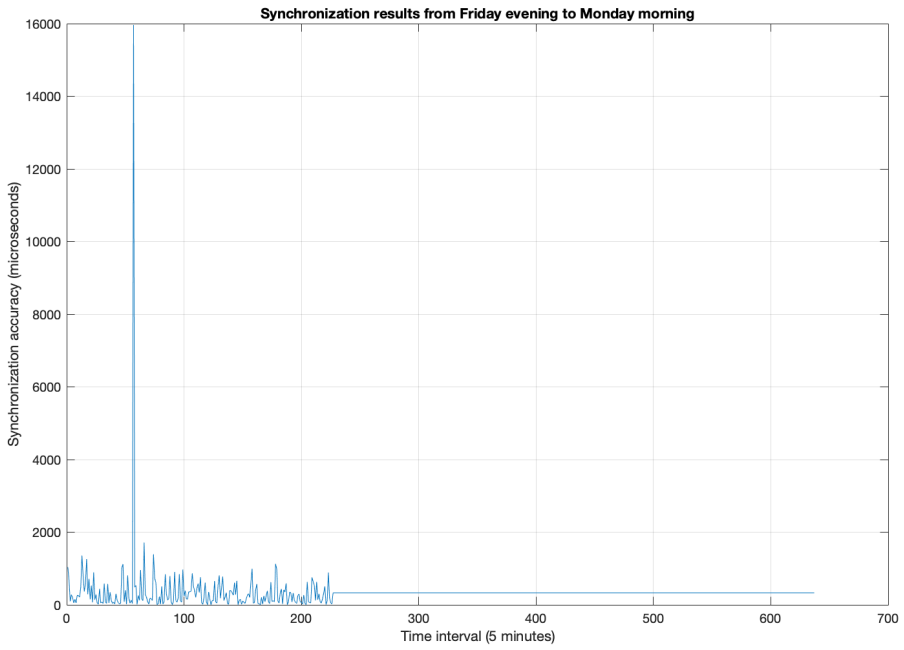

Figure 21: Synchronization trend and eventual stabilization

As can be seen from the Figure 21, during the first parts, when the TSN modules

start, the synchronization accurary fluctuates a lot. This continues for sometime and even reaches a maximum value of 15961 μs.

When the synchronization reaches the datapoint around 240, which is roughly after one day, the synchronization becomes stable with a synchronization accuracy of 328 μs. This is the synchronization value that stays for the rest of the time which is more than one day. This shows that the synchronization accuracy obtained in this the TSN modules implementation becomes stable over a period of time to a constant value.

6 Conclusions

This chapter presents the important findings related to the work conducted in the thesis. It summarizes the project strengths, notes weaknesses, and wraps the project. Section 6.1 presents the important observations and the implications of the results obtained from the thesis work. Section 6.2, states areas in which the thesis work could be improved upon, to obtain better results regarding the timing synchronization in TSN and its eventual integration with 5GS.

6.1 Conclusions regarding thesis work

The project set out to ascertain whether the theoretical 5GS delay of around 900 nanoseconds was attainable. The GM consisted of two interfaces with one configured to chronyc, with a timesource in Europe, which synchronized the GM time in a better way as an alternative to a GPS receiver which was not available in this case.

There was a need for two networks to simulate the gPTP message flow inside the 5GC. The first network consisted of the GM, TSN-1, TSN-2, and NW-TT. It contains synchronization messages from the GM and TSN modules (TSN-1 and TSN-2). On the other hand, the second network consists of NW-TT modified gPTP packets, TNS-3, and the DS-TT. The described network setup made it possible to introduce artificial paths. These paths simulate the traffic inside the 5GC (also known as TSN streams) to a certain extent.

The achieved synchronization accuracies when measured at the end TSN-3, which is the eventual box in which synchronization results are reported showed value of around 328 μs, when the setup was left for longer periods of time. At the beginning, when the synchronization starts and before stabilization, the synchronization accuracy figures fluctuate a lot. The synchronization accuracy attained in this setup seems reasonable considering that the TEKs, used as TSN switches are meant for experimental and not production use. In this regard, the performance achieved experimentally is plausible.

Even though most of the delay measurement values average around 328 μs in some minor cases, outliers of negative values occurred in the measurements at the start. This results when the clocks at the end stations report a different and higher time compared to that at the GM. This however, resolves itself when the GM starts to synchronize to the end devices after a short period of time. To show the general synchronization trend, these outliers are left out of the thesis discussion.

In comparing the TSN WG, 5GS synchronization delay budget of 900 nanoseconds to the results attained in the thesis (328 μs), it is possible to conclude that by integrating TSN modules developed into the 5GC, the Synchronization accuracy can be further improved to levels below the microseconds. For this to be the case, industrial TSN switches and a GPS receiver at the GM are needed. Additionally, code improvements can be made to ensure that packet buffer durations are limited to ensure that delays are limited further. The positive outcome from the experiment was that after more than two days the synchronization accuracy reported had stabilized to a constant value. This value can be taken as a reference point upon which

improvements to the synchronization accuracies can be conducted.

The thesis demonstrated the development, integration and testing of the TSN synchronization modules with the GM. The synchronization result, shows that it is possible to get the TSN modules to synchronize to the GM and between themselves. This synchronization is based on the packet modification and forwarding between the TSN modules. The thesis additionally, provides discussions and recommendations regarding the areas in which improvements could be made to be able to arrive at synchronization accuracies below a microsecond. These in short are for example, through the use of better quality TSN switches, codebase optimizations, GM synchronization to GPS among others.

6.2 Future research and work related to 5G and TSN implementation

The integration of 5GS and TSN is still in the research phases in most projects. More work is still ongoing on prototyping and coming up with an integrated system that meets the set goals regarding the delay inside the 5GS as stipulated by the IEEE WG. Even though successful in prototyping the TSN modules, the project did not include these in the context of 5GS. So, this is one area in which the research could move further. With the integration of the TSN modules, TSN can thus compare measurements to those stipulated by the IEEE 802.1AS WG.

Since the IEEE 802.1AS WG continually provides recommendations regarding areas needing improvement, this offers endless possibilities and better synchronization results considering 5GS and TSN integrations.

7 Appendix

7.1 NWTT Ingress Timestamping and Packet Modifications

```
#include "../libs/gptp_functions.h"
#include <linux/if_packet.h>
#include <math.h>

#ifndef THESIS_TSN_PROJECT_NWTT_H
#define THESIS_TSN_PROJECT_NWTT_H

_Noreturn static void ingress_timestamping(struct GPTP *GPTP) {
    int level, type;
    double rate_ratio = 0;
    unsigned char mac[6] = {0x3C, 0x52, 0x82, 0x57, 0xE6, 0x27};
    ssize_t count, error;
    struct timespec *ts;
    struct cmsghdr *cm;
    struct ether_header *ethernet = (struct ether_header *) &GPTP->rxBuffer[0];
    struct ptp_header *ptp = (struct ptp_header *) &GPTP->rxBuffer[sizeof(struct
        ether_header)];
    struct sync_msg *sync = (struct sync_msg *) &GPTP->rxBuffer[sizeof(struct
        ether_header)];
    struct follow_up_msg *follow_up = (struct follow_up_msg *)
        &GPTP->rxBuffer[sizeof(struct ether_header)];
    struct delay_req_msg *delay_req_msg = (struct delay_req_msg *)
        &GPTP->rxBuffer[sizeof(struct ether_header)];
    struct delay_resp_msg *delay_resp_msg = (struct delay_resp_msg *)
        &GPTP->rxBuffer[sizeof(struct ether_header)];
    struct announce_msg *announce_msg = (struct announce_msg *)
        &GPTP->rxBuffer[sizeof(struct ether_header)];
    struct ptp_pdelay_resp_follow_up *pdelay_resp_follow_up = (struct
        ptp_pdelay_resp_follow_up *) &GPTP->rxBuffer[sizeof(struct
        ether_header)];

    struct sync_msg *sync2 = (struct sync_msg *) &GPTP->txBuffer[sizeof(struct
        ether_header)];
    struct follow_up_msg *follow_up2 = (struct follow_up_msg *)
        &GPTP->txBuffer[sizeof(struct ether_header)];
    struct delay_req_msg *delay_req_msg2 = (struct delay_req_msg *)
        &GPTP->txBuffer[sizeof(struct ether_header)];
    struct delay_resp_msg *delay_resp_msg2 = (struct delay_resp_msg *)
        &GPTP->txBuffer[sizeof(struct ether_header)];
    struct announce_msg *announce_msg2 = (struct announce_msg *)
        &GPTP->txBuffer[sizeof(struct ether_header)];
    struct ptp_pdelay_resp_follow_up *pdelay_resp_follow_up2 = (struct
        ptp_pdelay_resp_follow_up *) &GPTP->txBuffer[sizeof(struct
        ether_header)];

    for (;;) {
        initialize_transmit_buffer(GPTP);
        initialize_receive_buffer(GPTP);
```

```
count = recvmsg(GPTP->socket1, &GPTP->rxMessageHeader, 0);
if (count < 0) bail("recvmsg() failed");

for (cm = CMSG_FIRSTHDR(&GPTP->rxMessageHeader); cm != NULL; cm =
    CMSG_NXTHDR(&GPTP->rxMessageHeader, cm)) {
    level = cm->cmsg_level, type = cm->cmsg_type;
    if ((level == SOL_SOCKET) && (type == SO_TIMESTAMPING)) {
        ts = (struct timespec *) CMSG_DATA(cm);
    }

    if (ethernet->ether_type == htons(ETH_P_1588) && 0 ==
        memcmp(ethernet->ether_shost, mac, 6)) {
        switch (ptp->messageType & 0x0F) {
            case GPTP_MSG_TYPE_ANNOUNCE:
                memcpy(&announce_msg2->hdr, &announce_msg->hdr,
                    sizeof(announce_msg->hdr));
                memcpy(&announce_msg2->gmPriority1,
                    &announce_msg->gmPriority1,
                       sizeof(announce_msg->gmPriority1));
                memcpy(&announce_msg2->gmPriority2,
                    &announce_msg->gmPriority2,
                       sizeof(announce_msg->gmPriority2));
                memcpy(&announce_msg2->pathTraceTlv,
                    &announce_msg->pathTraceTlv,
                       sizeof(announce_msg->pathTraceTlv));
                memcpy(&announce_msg2->currentUTCOffset,
                    &announce_msg->currentUTCOffset,
                       sizeof(announce_msg->currentUTCOffset));
                memcpy(&announce_msg2->gmClockAccuracy,
                    &announce_msg->gmClockAccuracy,
                       sizeof(announce_msg->gmClockAccuracy));
                memcpy(&announce_msg2->gmClockClass,
                    &announce_msg->gmClockClass,
                       sizeof(announce_msg->gmClockClass));
                memcpy(&announce_msg2->gmClockVariance,
                    &announce_msg->gmClockVariance,
                       sizeof(announce_msg->gmClockVariance));
                memcpy(&announce_msg2->timeSource,
                    &announce_msg->timeSource,
                    sizeof(announce_msg->timeSource));
                memcpy(&announce_msg2->localStepsRemoved,
                    &announce_msg->localStepsRemoved,
                       sizeof(announce_msg->localStepsRemoved));
                memcpy(&announce_msg2->gmClockIdentity,
                    &announce_msg->gmClockIdentity,
                       sizeof(announce_msg->gmClockIdentity));

                announce_msg2->hdr.correctionField = 10U;
                error = sendto(GPTP->socket2, &GPTP->txBuffer, 90, 0,
                             (struct sockaddr *) &GPTP->txSocketAddress,
                             sizeof(struct sockaddr_ll));
                if (error < 0) bail("sendto() failed");
                else
```

```
            printf("ANNOUNCE message sent at %ld\n",
                   (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
                       ts[2].tv_nsec);
        break;
    case GPTP_MSG_TYPE_PDELAY_REQ:
        memcpy(&delay_req_msg2->hdr, &delay_req_msg->hdr,
            sizeof(delay_req_msg->hdr));

        error = sendto(GPTP->socket2, &GPTP->txBuffer, 68, 0,
                     (struct sockaddr *) &GPTP->txSocketAddress,
                     sizeof(struct sockaddr_ll));
        if (error < 0) bail("sendto() failed");
        else
            printf("PEER_DELAY_REQ sent at %ld\n",
                   (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
                       ts[2].tv_nsec);
        break;
    case GPTP_MSG_TYPE_PDELAY_RESP:
        memcpy(&delay_resp_msg2->hdr, &delay_resp_msg->hdr,
            sizeof(delay_resp_msg->hdr));
        memcpy(&delay_resp_msg2->tlv, &delay_resp_msg->tlv,
            sizeof(delay_resp_msg->tlv));
        memcpy(&delay_resp_msg2->rx_tstamp,
            &delay_resp_msg->rx_tstamp,
              sizeof(delay_resp_msg->rx_tstamp));
        memcpy(&delay_resp_msg2->req_port_id,
            &delay_resp_msg->req_port_id,
              sizeof(delay_resp_msg->req_port_id));

        error = sendto(GPTP->socket2, &GPTP->txBuffer, 68, 0,
                     (struct sockaddr *) &GPTP->txSocketAddress,
                     sizeof(struct sockaddr_ll));
        if (error < 0) bail("sendto() failed");
        else
            printf("Peer_Delay_Resp sent at %ld\n",
                   (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
                       ts[2].tv_nsec);
        break;
    case GPTP_MSG_TYPE_PDELAY_RESP_FLWUP:
        memcpy(&pdelay_resp_follow_up2->hdr,
            &pdelay_resp_follow_up->hdr,
              sizeof(pdelay_resp_follow_up->hdr));
        memcpy(&pdelay_resp_follow_up2->responseOriginTimestamp,
              &pdelay_resp_follow_up->responseOriginTimestamp,
              sizeof(pdelay_resp_follow_up->responseOriginTimestamp));
        memcpy(&pdelay_resp_follow_up2->requestingSourcePort,
              &pdelay_resp_follow_up->responseOriginTimestamp,
              sizeof(pdelay_resp_follow_up->requestingSourcePort));

        error = sendto(GPTP->socket2, &GPTP->txBuffer, 68, 0,
                     (struct sockaddr *) &GPTP->txSocketAddress,
                     sizeof(struct sockaddr_ll));
        if (error < 0) bail("sendto() failed");
        else
```

```
        printf("PEER_DELAY_RESPONSE_FOLLOW_UP message sent at
            %ld\n",
               (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
                   ts[2].tv_nsec);
    break;
case GPTP_MSG_TYPE_SYNC:
    memcpy(&sync2->hdr, &sync->hdr, sizeof(sync->hdr));
    memcpy(&sync2->originTimestamp, &sync->originTimestamp,
        sizeof(sync->originTimestamp));
    error = sendto(GPTP->socket2, GPTP->txBuffer, 60, 0,
        (struct sockaddr *) &GPTP->txSocketAddress,
                 sizeof(struct sockaddr_ll));
    if (error < 0) bail("sendto() failed");
    else
        printf("SYNC message sent at %ld\n",
               (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
                   ts[2].tv_nsec);
    break;
case GPTP_MSG_TYPE_SYNC_FLWUP:
    memcpy(&follow_up2->hdr, &follow_up->hdr,
        sizeof(follow_up->hdr));
    memcpy(&follow_up2->tlv, &follow_up->tlv,
        sizeof(follow_up->tlv));
    memcpy(&follow_up2->preciseOriginTstamp,
        &follow_up->preciseOriginTstamp,
          sizeof(follow_up->preciseOriginTstamp));

    int64_t tsi = ts[2].tv_sec * NANOSECONDS_PER_SECOND +
        ts[2].tv_nsec;

    follow_up2->hdr.correctionField +=
        sync->hdr.correctionField;
    follow_up2->hdr.correctionField += tsi;
    follow_up2->tlv.cumulativeScaledRateOffset = 0;

    follow_up2->tlv.type = htons(0x3);
    follow_up2->tlv.length = htons(28);
    follow_up2->tlv.id[0] = 0x00;
    follow_up2->tlv.id[1] = 0x80;
    follow_up2->tlv.id[2] = 0xC2;
    follow_up2->tlv.subtype[0] = 0;
    follow_up2->tlv.subtype[1] = 0;
    follow_up2->tlv.subtype[2] = 1;
    follow_up2->tlv.cumulativeScaledRateOffset += (int32_t)
        ((rate_ratio - 1.0) * ldexp(2,41)); // (Integer32)
        ((RateRatio-1) * 2^41)
    follow_up2->tlv.lastGmPhaseChange.nsec = 0U;
    follow_up2->tlv.lastGmPhaseChange.nsec_msb = 0U;
    follow_up2->tlv.lastGmPhaseChange.subns = 0U;
    memcpy(&follow_up2->tlv.lastGmPhaseChange.nsec, &tsi,
        sizeof(tsi));


    error = sendto(GPTP->socket2, &GPTP->txBuffer, 90, 0,
```

```
                                (struct sockaddr *) &GPTP->txSocketAddress,
                                sizeof(struct sockaddr_ll));
                        if (error < 0) bail("sendto() failed");
                        else
                            printf("FOLLOW_UP message sent at %ld\n",
                                   (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
                                       ts[2].tv_nsec);
                        break;
                }
            }
        }
    }
}

#endif //THESIS_TSN_PROJECT_NWTT_H
```

7.2 DSTT Egress Timestamping and Packet Modifications

```
#include "../libs/gptp_functions.h"
#include <math.h>

#ifndef THESIS_TSN_PROJECT_DSTT_H
#define THESIS_TSN_PROJECT_DSTT_H

_Noreturn static void egress_timestamping(struct GPTP *GPTP) {
    int level, type;
    double rate_ratio = 0;
    unsigned char mac[6] = { 0x00, 0xE0, 0x67, 0x26,0x7A, 0xA5 };
    ssize_t count, error;
    struct timespec *ts;
    struct cmsghdr* cm;
    struct ether_header* ethernet = (struct ether_header*)&GPTP->rxBuffer[0];
    struct ptp_header* ptp = (struct ptp_header*) &GPTP->rxBuffer[sizeof(struct
        ether_header)];
    struct sync_msg* sync = (struct sync_msg*) &GPTP->rxBuffer[sizeof(struct
        ether_header)];
    struct follow_up_msg* follow_up = (struct follow_up_msg*)
        &GPTP->rxBuffer[sizeof(struct ether_header)];
    struct delay_req_msg* delay_req_msg = (struct delay_req_msg*)
        &GPTP->rxBuffer[sizeof(struct ether_header)];
    struct delay_resp_msg* delay_resp_msg = (struct delay_resp_msg*)
        &GPTP->rxBuffer[sizeof(struct ether_header)];
    struct announce_msg* announce_msg = (struct announce_msg*)
        &GPTP->rxBuffer[sizeof(struct ether_header)];
    struct ptp_pdelay_resp_follow_up* pdelay_resp_follow_up = (struct
        ptp_pdelay_resp_follow_up*) &GPTP->rxBuffer[sizeof(struct ether_header)];

    struct sync_msg* sync2 = (struct sync_msg*)&GPTP->txBuffer[sizeof(struct
        ether_header)];
    struct follow_up_msg* follow_up2 = (struct follow_up_msg*)
        &GPTP->txBuffer[sizeof(struct ether_header)];
```

```
struct delay_req_msg* delay_req_msg2 = (struct delay_req_msg*)
    &GPTP->txBuffer[sizeof(struct ether_header)];
struct delay_resp_msg* delay_resp_msg2 = (struct delay_resp_msg*)
    &GPTP->txBuffer[sizeof(struct ether_header)];
struct announce_msg* announce_msg2 = (struct announce_msg*)
    &GPTP->txBuffer[sizeof(struct ether_header)];
struct ptp_pdelay_resp_follow_up* pdelay_resp_follow_up2 = (struct
    ptp_pdelay_resp_follow_up*) &GPTP->txBuffer[sizeof(struct ether_header)];

for (;;) {
    initialize_transmit_buffer(GPTP);
    initialize_receive_buffer(GPTP);

    count = recvmsg(GPTP->socket1, &GPTP->rxMessageHeader, 0);
    if (count < 0) bail("recvmsg() failed");

    for (cm = CMSG_FIRSTHDR(&GPTP->rxMessageHeader); cm != NULL; cm =
        CMSG_NXTHDR(&GPTP->rxMessageHeader, cm)) {
        level = cm->cmsg_level, type = cm->cmsg_type;
        if ((level == SOL_SOCKET) && (type == SO_TIMESTAMPING)) {
            ts = (struct timespec*) CMSG_DATA(cm);
        }
        if (ethernet->ether_type == htons(ETH_P_1588) && 0 ==
            memcmp(ethernet->ether_shost, mac, 6)) {
            switch (ptp->messageType & 0x0F) {
                case GPTP_MSG_TYPE_ANNOUNCE:
                    memcpy(&announce_msg2->hdr, &announce_msg->hdr,
                        sizeof(announce_msg->hdr));
                    memcpy(&announce_msg2->gmPriority1,
                        &announce_msg->gmPriority1,
                        sizeof(announce_msg->gmPriority1));
                    memcpy(&announce_msg2->gmPriority2,
                        &announce_msg->gmPriority2,
                        sizeof(announce_msg->gmPriority2));
                    memcpy(&announce_msg2->pathTraceTlv,
                        &announce_msg->pathTraceTlv,
                        sizeof(announce_msg->pathTraceTlv));
                    memcpy(&announce_msg2->currentUTCOffset,
                        &announce_msg->currentUTCOffset,
                        sizeof(announce_msg->currentUTCOffset));
                    memcpy(&announce_msg2->gmClockAccuracy,
                        &announce_msg->gmClockAccuracy,
                        sizeof(announce_msg->gmClockAccuracy));
                    memcpy(&announce_msg2->gmClockClass,
                        &announce_msg->gmClockClass,
                        sizeof(announce_msg->gmClockClass));
                    memcpy(&announce_msg2->gmClockVariance,
                        &announce_msg->gmClockVariance,
                        sizeof(announce_msg->gmClockVariance));
                    memcpy(&announce_msg2->timeSource,
                        &announce_msg->timeSource,
                        sizeof(announce_msg->timeSource));
                    memcpy(&announce_msg2->localStepsRemoved,
                        &announce_msg->localStepsRemoved,
```

```
            sizeof(announce_msg->localStepsRemoved));
        memcpy(&announce_msg2->gmClockIdentity,
            &announce_msg->gmClockIdentity,
            sizeof(announce_msg->gmClockIdentity));

        error = sendto(GPTP->socket2, &GPTP->txBuffer, 90, 0,
            (struct sockaddr*) &GPTP->txSocketAddress,
                    sizeof(struct sockaddr_ll));
        if (error < 0) bail("sendto() failed");
        else printf("ANNOUNCE message sent at %ld\n",
            (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
            ts[2].tv_nsec);
        break;
    case GPTP_MSG_TYPE_PDELAY_REQ:
        memcpy(&delay_req_msg2->hdr, &delay_req_msg->hdr,
            sizeof(delay_req_msg->hdr));

        error = sendto(GPTP->socket2, &GPTP->txBuffer, 68, 0,
            (struct sockaddr*) &GPTP->txSocketAddress,
                    sizeof(struct sockaddr_ll));
        if (error < 0) bail("sendto() failed");
        else printf("PEER_DELAY_REQ sent at %ld\n", (ts[2].tv_sec
            * NANOSECONDS_PER_SECOND) + ts[2].tv_nsec);
        break;
    case GPTP_MSG_TYPE_PDELAY_RESP:
        memcpy(&delay_resp_msg2->hdr, &delay_resp_msg->hdr,
            sizeof(delay_resp_msg->hdr));
        memcpy(&delay_resp_msg2->tlv, &delay_resp_msg->tlv,
            sizeof(delay_resp_msg->tlv));
        memcpy(&delay_resp_msg2->rx_tstamp,
            &delay_resp_msg->rx_tstamp,
            sizeof(delay_resp_msg->rx_tstamp));
        memcpy(&delay_resp_msg2->req_port_id,
            &delay_resp_msg->req_port_id,
            sizeof(delay_resp_msg->req_port_id));

        error = sendto(GPTP->socket2, &GPTP->txBuffer, 68, 0,
            (struct sockaddr*) &GPTP->txSocketAddress,
                    sizeof(struct sockaddr_ll));
        if (error < 0) bail("sendto() failed");
        else printf("Peer_Delay_Resp sent at %ld\n", (ts[2].tv_sec
            * NANOSECONDS_PER_SECOND) + ts[2].tv_nsec);
        break;
    case GPTP_MSG_TYPE_PDELAY_RESP_FLWUP:
        memcpy(&pdelay_resp_follow_up2->hdr,
            &pdelay_resp_follow_up->hdr,
            sizeof(pdelay_resp_follow_up->hdr));
        memcpy(&pdelay_resp_follow_up2->responseOriginTimestamp,
            &pdelay_resp_follow_up->responseOriginTimestamp,
              sizeof(pdelay_resp_follow_up->responseOriginTimestamp));
        memcpy(&pdelay_resp_follow_up2->requestingSourcePort,
            &pdelay_resp_follow_up->responseOriginTimestamp,
              sizeof(pdelay_resp_follow_up->requestingSourcePort));
```

```
    error = sendto(GPTP->socket2, &GPTP->txBuffer, 68, 0,
        (struct sockaddr*) &GPTP->txSocketAddress,
                  sizeof(struct sockaddr_ll));
    if (error < 0) bail("sendto() failed");
    else printf("PEER_DELAY_RESPONSE_FOLLOW_UP message sent at
        %ld\n", (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
        ts[2].tv_nsec);
    break;
case GPTP_MSG_TYPE_SYNC:
    memcpy(&sync2->hdr, &sync->hdr, sizeof(sync->hdr));
    memcpy(&sync2->originTimestamp, &sync->originTimestamp,
        sizeof(sync->originTimestamp));
    error = sendto(GPTP->socket2, GPTP->txBuffer, 60, 0,
        (struct sockaddr*)&GPTP->txSocketAddress,
                  sizeof(struct sockaddr_ll));
    if (error < 0) bail("sendto() failed");
    else printf("SYNC message sent at %ld\n", (ts[2].tv_sec *
        NANOSECONDS_PER_SECOND) + ts[2].tv_nsec);
    break;

case GPTP_MSG_TYPE_SYNC_FLWUP:
    memcpy(&follow_up2->hdr, &follow_up->hdr,
        sizeof(follow_up->hdr));
    memcpy(&follow_up2->tlv, &follow_up->tlv,
        sizeof(follow_up->tlv));
    memcpy(&follow_up2->preciseOriginTstamp,
        &follow_up->preciseOriginTstamp,
        sizeof(follow_up->preciseOriginTstamp));

    rate_ratio =
        ntohl(follow_up2->tlv.cumulativeScaledRateOffset);
    rate_ratio *= ldexp(2, -41);
    rate_ratio += 1;
    printf("calculated rateRatio = %f\n", rate_ratio);

    int64_t TSi = follow_up2->tlv.lastGmPhaseChange.nsec;
    printf("\tIngress Timestamping\t== %ld\n", TSi);
    int64_t TSe = ts[2].tv_sec * NANOSECONDS_PER_SECOND +
        ts[2].tv_nsec;
    printf("\tEgress Timestamp\t== %ld\n", TSe);
    int64_t residence_time = (TSe - TSi);
    printf("\tResidence Time\t\t== %ld.%09ld\n",
        (residence_time / NANOSECONDS_PER_SECOND),
        (residence_time % NANOSECONDS_PER_SECOND));
    printf("\tResidence Time (sec)\t== %ld\n", residence_time
        / NANOSECONDS_PER_SECOND);
    printf("\tResidence Time (nsec)\t== %ld\n", residence_time
        % NANOSECONDS_PER_SECOND);

    error = sendto(GPTP->socket2, &GPTP->txBuffer, 90, 0,
        (struct sockaddr*) &GPTP->txSocketAddress,
                  sizeof(struct sockaddr_ll));
    if (error < 0) bail("sendto() failed");
```

```
                    else printf("FOLLOW_UP message sent at %ld\n",
                        (ts[2].tv_sec * NANOSECONDS_PER_SECOND) +
                        ts[2].tv_nsec);
                    break;
                }
            }
        }
    }
}

#endif //THESIS_TSN_PROJECT_DSTT_H
```

References

[1] E. Calvanese Strinati, M. Mueck, A. Clemente, J. Kim, G. Noh, H. Chung, I. Kim, T. Choi, Y. Kim, H. K. Chung, *et al.*, "5gchampion–disruptive 5g technologies for roll-out in 2018," *ETRI Journal*, vol. 40, no. 1, pp. 10–25, 2018.

[2] E. J. Khatib and R. Barco, "Optimization of 5g networks for smart logistics," *Energies*, vol. 14, no. 6, p. 1758, 2021.

[3] T. Striffler, N. Michailow, and M. Bahr, "Time-sensitive networking in 5th generation cellular networks - current state and open topics," in *2019 IEEE 2nd 5G World Forum (5GWF)*, 2019, pp. 547–552. DOI: 10.1109/5GWF.2019.8911720.

[4] ——, "Time-sensitive networking in 5th generation cellular networks-current state and open topics," in *2019 IEEE 2nd 5G World Forum (5GWF)*, IEEE, 2019, pp. 547–552.

[5] I. Godor, M. Luvisotto, S. Ruffini, K. Wang, D. Patel, J. Sachs, O. Dobrijevic, D. P. Venmani, O. Le Moult, J. Costa-Requena, *et al.*, "A look inside 5g standards to support time synchronization for smart manufacturing," *IEEE Communications Standards Magazine*, vol. 4, no. 3, pp. 14–21, 2020.

[6] C. Public, *Time-sensitive networking: A technical introduction*, 2017. DOI: C11-738950-00. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/white-paper-c11-738950.pdf.

[7] "Ieee draft standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications," *IEEE P802.1AS-Rev/D8.1, August 2019*, pp. 1–516, 2019.

[8] N. Finn, "Introduction to time-sensitive networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22–28, 2018.

[9] J. Farkas, L. L. Bello, and C. Gunther, "Time-sensitive networking standards," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 20–21, 2018.

[10] L. L. Bello and W. Steiner, "A perspective on ieee time-sensitive networking for industrial communication and automation systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019.

[11] Y. Seol, D. Hyeon, J. Min, M. Kim, and J. Paek, "Timely survey of time-sensitive networking: Past and future directions," *IEEE Access*, vol. 9, pp. 142 506–142 527, 2021.

[12] Q. Li, D. Li, and X. Sun, "Key technologies of time-sensitive networking and its application," in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol. 782, 2020, p. 042 015.

[13] Y. Kang, S. Lee, S. Gwak, T. Kim, and D. An, "Time-sensitive networking technologies for industrial automation in wireless communication systems," *Energies*, vol. 14, no. 15, p. 4497, 2021.

[14] I. W. Group *et al.*, "Draft standard for local and metropolitan area networks–part 16: Air interface for broadband wireless access systems," *IEEE P802. 16Rev2 D*, vol. 1, 2007.

[15] H. Puttnies, P. Danielis, A. R. Sharif, and D. Timmermann, "Estimators for time synchronization—survey, analysis, and outlook," *IoT*, vol. 1, no. 2, pp. 398–435, 2020.

[16] I. S. Association *et al.*, "Ieee standard for local and metropolitan area networks—timing and synchronization for time-sensitive applications in bridged local area networks," *IEEE Std*, vol. 802, 2011.

[17] O. Daniel, G. E. Juan, C. Lua, and O. Roman, "Failure detection in tsn startup using deep learning," in *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, 2020, pp. 140–141.

[18] M. Böhm and D. Wermser, "Multi-domain time-sensitive networks—control plane mechanisms for dynamic inter-domain stream configuration," *Electronics*, vol. 10, no. 20, p. 2477, 2021.

[19] M. Pahlevan and R. Obermaisser, "Redundancy management for safety-critical applications with time sensitive networking," in *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, IEEE, 2018, pp. 1–7.

[20] S. Meier, H. Weibel, and K. Weber, "Ieee 1588 syntonization and synchronization functions completely realized in hardware," in *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, IEEE, 2008, pp. 1–4.

[21] G. T. version 16.1.0 Release 16, "5g; 5g system (5gs); time-sensitive networking (tsn) application function (af) to device-side tsn translator (ds-tt) and network-side tsn translator (nw-tt) protocol aspects; stage 3," *IEEE Std*, vol. Release 16, no. 24.519, 2020-07.

[22] K. B. Stanton, "Distributing deterministic, accurate time for tightly coordinated network and software applications: Ieee 802.1 as, the tsn profile of ptp," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 34–40, 2018.

[23] A. Larrañaga, M. C. Lucas-Estañ, I. Martinez, I. Val, and J. Gozalvez, "Analysis of 5g-tsn integration to support industry 4.0," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, vol. 1, 2020, pp. 1111–1114.

[24] G. T. version 16.5.0 Release 16, "5g; service requirements for cyber-physical control applications in vertical domains," *IEEE Std*, vol. Release 16, no. 24.519, 2020-09.

[25] S. Bhattacharjee, K. Katsalis, O. Arouk, R. Schmidt, T. Wang, X. An, T. Bauschert, and N. Nikaein, "Network slicing for tsn-based transport networks," *IEEE Access*, vol. 9, pp. 62 788–62 809, 2021.

[26] J. W. Won and J. M. Ahn, "3gpp urllc patent analysis," *ICT Express*, vol. 7, no. 2, pp. 221–228, 2021.

[27] M. Gundall, C. Huber, P. Rost, R. Halfmann, and H. D. Schotten, "Integration of 5g with tsn as prerequisite for a highly flexible future industrial automation: Time synchronization based on ieee 802.1 as," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2020, pp. 3823–3830.

[28] A. Mahmood, R. Exel, and T. Sauter, "Impact of hard-and software timestamping on clock synchronization performance over ieee 802.11," in *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, IEEE, 2014, pp. 1–8.

[29] D. Arnold, "Why is ieee 1588 so accurate," *Access on:" http://blog. meinberg-global. com/2013/09/14/ieee-1588-accurate/" in Nov-22-2013*, 2013.

[30] Y. Li, B. Noseworthy, J. Laird, T. Winters, and T. Carlin, "A study of precision of hardware time stamping packet traces," *Researchgate*, pp. 102–107, Nov. 2014. DOI: 10.1109/ISPCS.2014.6948700.

[31] P Rost, D. Chandramouli, and T. Kolding, "5g plug-and-produce," *Nokia*, 2020.

[32] J. K. Ray, P. Biswas, R. Bera, S. Sil, and Q. M. Alfred, "Tsn enabled 5g non public network for smart systems," in *2020 5th International Conference on Computing, Communication and Security (ICCCS)*, IEEE, 2020, pp. 1–6.

[33] Z. Li, Z. Zhong, W. Zhu, and B. Qin, "A hardware time stamping method for ptp messages based on linux system," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 11, no. 9, pp. 5105–5111, 2013.

[34] H. Puttnies, P. Danielis, E. Janchivnyambuu, and D. Timmermann, "A simulation model of ieee 802.1 as gptp for clock synchronization in omnet++.," in *OMNeT++*, 2018, pp. 63–72.