MÄLARDALEN UNIVERSITY
SCHOOL OF INNOVATION, DESIGN AND ENGINEERING
VÄSTERÅS, SWEDEN

Thesis for the Degree of Bachelor of Science in Computer Network Engineering
15.0 credits

MINIMIZING THE CLOCK DRIFT IN PARTIALLY SYNCHRONIZED HETEROGENEOUS TSN NETWORKS

Balqis Yusuf
byf19002@student.mdu.se

Examiner: Mohammad Ashjaei
Mälardalen University, Västerås, Sverige

Supervisor: Daniel Bujosa Mateu
Mälardalen University, Västerås, Sverige

2023-05-24

ABSTRACT

The new generation of embedded systems will increase interaction between the environment, people, and autonomous devices. This will increase their need for communication, particularly in meeting real-time requirements. To address the real-time requirements of embedded systems, a communication network capable of providing high bandwidth, low latency, and deterministic behaviour is necessary. Time-Sensitive Networking (TSN) was developed by the IEEE 802.1 TSN Task Group and is a set of standards providing deterministic service over standard Ethernet and is an attractive option for achieving this. TSN leverages the advantages of IEEE Ethernet standards, including low hardware cost, high bandwidth, and deterministic behaviour. TSN uses time synchronization, traffic shaping, strict priority, and resource reservation mechanisms to provide a reliable and deterministic network environment suitable for real-time applications. However, for these mechanisms to work and TSN to achieve high performance, the network must be fully synchronized. In this thesis, we aim to integrate existing legacy devices into a TSN network without incorporating TSN functionality into them, as implementing all TSN standards requires significant investments in time, financial resources, and infrastructure upgrades. However, as the legacy devices don't have TSN capabilities and cannot implement TSN synchronization protocols, they cannot synchronize with the TSN switches, which causes negative adverse such as clock drift between the TSN switches and the legacy end-stations. In this thesis, we aim to minimize the clock drift in the partially synchronized heterogeneous network, allowing researchers and organizations to take advantage of the benefits of adopting TSN into a legacy network without facing those issues. To solve the clock drift that occurs between the legacy end-stations and the TSN switches, we implemented one solution by combining those proposed solutions in the previous work [9] by using the Drift Detector (DD) and the Centralised Network Configuration element (CNC). This will be resolved by DD measuring and calculating the difference between the expected and actual reception of the messages from the receiver end-station. The CNC later uses the variation values detected by the DD to modify the TSN schedule and updates the network with the new period. In this way, we could minimize the negative consequences caused by partial synchronization in the network.

Table of contents

1.Introduction 1
1.2 Thesis outline 2
2.Background 3
2.1 Real-Time Systems 3
2.2 Real-time communication 3
2.2.1 Ethernet 3
2.2.2 Switched Ethernet 3
2.2.3 Audio Video Bridging 4
2.3 Time-Sensitive Networking (TSN) 5
2.3.1 Best Master Clock Algorithm (BMCA) 5
2.3.2 Propagation Delay Measurement (PDM) 6
2.3.3 Time Synchronization Information (TTI) 7
2.3.4 Scheduling Mechanisms 7
3.Related work 10
3.1 Description of previous work 11
4.Problem formulation 12
5.Research method 14
6.Ethical considerations 15
7.Description of experiments 16
7.1 Hardware description 16
7.2 Sending traffic and data analyzing 17
7.3 Experimental setup 18
7.3.2 Scenario with and without the TSN schedule 18
7.3.3 Scenario with the solution 19
8.Results 23
9.Discussion 25
10.Conclusions 27
11.Future work 28
Reference 29
Appendix A The codes for experiments 1 & 2 (Listener.py and Talker.py) 31
Appendix B The codes for experiment 3 (Listener.py and Talker.py) 35
Appendix C The code to the cnc.py 39
Appendix D Cnc.sh 40
Appendix E Configuration of the TSN Switches 41

Appendix F Setting up Docker Desktop environment 44
Appendix G Config.json 46

1.Introduction

An embedded system is a system that was created to perform a specific task or function and is often a component of a larger system. [1]. These systems are used in a wide range of applications and can be found all around us in the modern world, e.g., in the car we drive, in industries or in the smartphone we carry in our pocket. Many embedded system application's functionalities and advanced features transmit data in real-time and need to respond quickly to changes in their environment. Therefore, they require high bandwidth and rely increasingly on real-time capabilities. Real-time systems (RTS) are described as computer systems that respond to external events and perform computer tasks based on these events. RTSs return the response within a specified timeframe measured in milliseconds or microseconds [2]. RTSs are necessary for embedded systems and other applications where errors or delays might lead to consequences, particularly in critical sectors such as healthcare, infrastructure, and transportation [3]. The end-to-end latency must be predictable and bounded for RTSs to guarantee the output and ensure the system meets its real-time requirements. End-to-end latency refers to a system's time to handle input and generate output [4]. Several predictable approaches have been considered to improve reliable time-sensitive network communication, such as Controller Area Network (CAN), a network communication protocol well suited for real-time systems [5]. However, CAN has a relatively low bandwidth and does not provide deterministic communication, meaning there is no guaranteed time frame for message delivery. This can have consequences in real-time systems that require predictable response time. Therefore, Ethernet is considered an alternative to CAN with lower adoption costs and higher bandwidth [6]. However, Ethernet has limited Quality of Service (QoS) capabilities and misses predictability in data delivery, i.e., it cannot provide deterministic behaviour or guarantee the performance needed for real-time systems that depends on accurate and deterministic communication.

To meet the RTSs requirements, a standard providing reliable data transfer and high throughput is required. IEEE 802.1 TSN Task Group constructed TSN, which offers low-latency capabilities and deterministic communication to support real-time applications [7]. TSN uses switches to calculate the network's needs, ensuring minimal packet loss, jitter, and low latency to support real-time communication. The switches can perform this using different traffic classes such as Audio-Video Bridging (AVB), Time-Triggered (TT) and Best-Effort (BE). The high-priority traffic is the TT traffic, which is time-triggered control traffic, followed by AVB, which allows low-latency streaming services to be delivered over the Ethernet network and BE, which has the lowest priority. TSN is an extension of the conventional Ethernet enabling time synchronization, traffic shaping, strict priority, and resource reservation. TSN uses Time-Aware Shaper (TAS) and Credit-Based Shaper (CBS) as traffic shapers to prioritize traffic, ensuring that time-sensitive traffic is given priority over other types of traffic and is delivered deterministically [2]. Time synchronization is used for TSN to perform the characteristics required in real-time systems. However, without proper implementation of time synchronization, other mechanisms, such as scheduling, and traffic shaping, would not work correctly as they rely on the accurate timing of network events. TSN uses time synchronization to synchronize the different clocks within a network to a standard time domain. However, implementing time synchronization takes significant resources and time; in many cases, it is not even practical. In this thesis, we will integrate existing legacy devices in a TSN network without incorporating TSN functionality into them. In this thesis, we will implement previously proposed solutions for a partially synchronized network to help companies or researchers benefit from integrating TSN-capable end-stations with TSN devices in a network and benefit from the reduced time and resources needed to adopt TSN devices.

This thesis is an extension of Nguyen and Nasiri's thesis [8] and Johansson's work in [9], analyzing the performance of a partial and out-of-synchronization heterogeneous TSN network. Authors in [8] and [9] have detected adverse consequences when integrating TSN with legacy devices, as the network lacks full synchronization. Therefore, authors in [9] proposed solutions to detect adverse effects. In this thesis, we aim to implement the previously proposed solutions in [9] and then will test and evaluate them. This is because the authors in previous work [9] did a proof of concept but not a functional implementation of the proposed solution. The implementation presented in this work will allow researchers and organizations to take advantage of the benefits of adopting TSN into a legacy network without facing those issues.

1.2 Thesis outline

The rest of this work is organized as follows: Section 2 contains foundational information, including key terms and concepts, that is necessary to know for this thesis. Section 3 presents related work covering a variety of related topics. In Section 4, the problem formulation is described, while Section 5 briefly provides the chosen research method for this thesis. Section 6 describes ethical considerations, and Section 7 defines the description and implementation of the experiments performed in this thesis. Section 8 presents the results from the experimental scenarios performed, and Section 9 summarizes and discusses all observations and evaluations gained through this thesis. Finally, section 10 summarizes this work, followed by ideas for future investigations in Section 11.

2.Background

Ethernet is the most widely used networking medium. Due to its high cost, limited scalability, and limited QoS, switched Ethernet is an alternative to standard Ethernet since it provides an efficient and convenient way to increase network bandwidth. However, switched Ethernet exhibits limitations when utilized with real-time traffic, which can be solved with AVB or TSN. AVB is a set of standards that allow the transmission of real-time audio and video streams over Ethernet networks. Meanwhile, TSN is based on Ethernet and leverages Ethernet advantages such as scalability and bandwidth while providing the deterministic behaviour required for real-time systems. This section explains the concepts and techniques for achieving goals, such as real-time systems, Ethernet, switched Ethernet, AVB and TSN.

2.1 Real-Time Systems

A real-time system is a type of computer system designed to handle external events, perform computer tasks based on these events, and returns the response within a specified time [2]. A response can consist of doing calculations or responding to the occurrence. A deadline is the earliest possible time the system or any of its processes must respond. The real-time system's accuracy depends on returning the correct answer and providing it within a time. For example, if we take an MP3 player, the music will not sound as it should if the signal is not delivered to the headphones in time. In real-time systems, determinism refers to the capacity to predict and ensure the exact timing of an event or task in a system. To guarantee that deadlines are met, determinism is required, and Jitter indicates how deterministic a system is. Jitter is the variation in the time between packet transmission or arrival in a network [8]. Removing jitter in real-time systems is to avoid dangerous events caused by missed deadlines, such as a delay in an airbag inside a vehicle during an emergency. The primary classification of real-time systems is either soft, firm, or hard, based on the impact of the missed deadline.

- Soft real-time systems are those for which missing one of their deadlines does not mean the system fails [11]. Missed deadlines in soft real-time systems affect only the output quality, i.e., if a digital camera's shutter is slower than it should be, it affects the image, but no disastrous consequences occur.

- Hard real-time systems can tolerate less jitter than soft real-time systems. However, hard real-time systems are those where a single missed deadline can cause the entire system to fail or have catastrophic consequences. A hard real-time system is stricter regarding time and scheduling to ensure that all tasks must be completed on time. [11] Some hard real-time systems are an airbag in a car, aircraft equipment, and anti-lock braking systems in modern vehicles.

- In firm real-time systems, it is acceptable to miss a deadline [2]. If the process is not completed by the deadline, the outcome is useless, and the assignment should be abandoned. A financial trading system is an example of a firm's real-time system. While failing to meet a deadline may not cause the system to fail, it might cause significant financial losses.

When components must communicate through a network, the network's end-to-end delay must be limited to ensure that the critical messages can be transferred within the real-time constraint [2]. However, most traditional networks cannot follow the timing constraints required, so they cannot be applied to real-time systems. Therefore, to meet the requirement significant for real-time systems, Ethernet, which is covered in more detail in the next section, may be an option to meet those requirements.

2.2 Real-time communication

2.2.1 Ethernet

Ethernet is adopted as the most common local area networking technology in homes and offices. Ethernet was developed in 1976 by the Xerox Palo Alto Research Center (PARC), commercialized in 1980, and first standardized as IEEE 802.3 in 1983 [2]. Ethernet is a set of network technology standards used for communication in Local Area Networks (LAN), and it has become a widely adopted standard for wired LANs. Ethernet contains both the physical and data link layers of the OSI model. The data link layer consists of the logical link control (LLC) layer and the media access control address (MAC) layer. MAC handles the hardware with access to the transmission medium, while LLC provides error control of the data link layer, clock synchronization, and flow control. Each node in the network has a unique 48-bit MAC address. High-speed local area networks can be built using Ethernet, a reliable, scalable, and affordable technology. Like radio systems and CAN functions, Ethernet networks have been designed to use a shared medium for all linked nodes and support various network topologies. Ethernet is built on Carrier Sense Multiple Access/Collision Detection (CSMA/CD) and is the arbitration mechanism used in the Ethernet network to control access to the shared transmission medium, commonly a coaxial cable or twisted pair cable. Each network device listens to the network through an Ethernet hub to check if the transmission medium is available for use or not in CSMA/CD. The devices can transfer data if no transmission has been discovered. However, when multiple devices attempt to transmit data at the same time., collisions occur. If a collision occurs, all devices involved in the collision discontinue transmitting and pause for a random period before they try to retransmit data again. However, the collision occurring in CSMA/CD can introduce variable and unpredictable delays, making it unsuitable for real-time performance. Ethernet using a best-effort delivery mechanism makes Ethernet also unsuitable for real-time systems. Resulting in difficulty in guaranteeing the delivery and timing of data packets because the data is sent as soon as possible. Still, there is no guarantee of when they will be delivered. Therefore, a more suitable solution that works better for real-time systems is switched Ethernet. Switched Ethernet can help eliminate collisions occurring in Ethernet using switching technology, which will be described in further detail in the next section.

2.2.2 Switched Ethernet

Switched Ethernet is a type of network technology that connects devices on a network using Ethernet switches. Switched Ethernet is often combined with other network technologies, such as Wi-Fi, to deliver a complete network solution that fits the needs of modern applications and devices. Each device in a switched Ethernet network is linked to a dedicated port on the Ethernet switch, establishing a specific point-to-point connection between the devices [10]. Switched Ethernet takes advantage of the scalability, high bandwidth, and cost-effectiveness of an Ethernet network while providing limited latency and time reliability. Switched Ethernet makes it possible to achieve full-duplex communication, allowing network nodes to send and receive messages at the same time. The significant difference between switched Ethernet and conventional Ethernet is how they handle data transmission, as switched Ethernet uses a switch instead of a hub. A switch establishes dedicated point-to-point connections between devices, where each device is linked to a dedicated port on the switch, and data is sent directly between the transmitter and receiver. Meanwhile, a hub is a simple device that broadcasts incoming data to all connected devices. In traditional Ethernet, a collision may occur when two or more devices transmit traffic simultaneously, resulting in data loss and retransmission. This problem is solved by switching Ethernet, which only sends data from sources to destinations using the MAC addresses of all linked interfaces. However, switched Ethernet performance is still influenced by network congestion, packet size, and QoS algorithms. Therefore, switched Ethernet may not be suitable for all real-time applications, even though it can increase

Ethernet's applicability for real-time systems. The following section will provide further information on AVB, which can be a better solution to provide real-time support than switched Ethernet does.

2.2.3 Audio Video Bridging

Switched Ethernet supports the creation of several independent broadcast domains, known as virtual LANs (VLANs). However, since switched Ethernet performs best-effort delivery; it is not considered suitable for real-time systems. The IEEE AVB Task Group established AVB, where prioritizing traffic according to its importance and time sensitivity is one of AVB's primary characteristics. AVB is a set of protocols that improve switched Ethernet's capabilities for supporting real-time, high-bandwidth audio and video streaming applications [8] [9]. For AVB to guarantee low latency and high-reliability delivery of real-time traffic, it integrates several different technologies, including QoS, synchronization, traffic management, and stream reservation algorithms. The main standards that the task group introduced are the following:

❖ IEEE 802.1AS: Used by AVB for time synchronization, which provides all devices on the network with an exact and stable clock. This enables the synchronization of audio and video streams across all network devices, which is essential for real-time applications.

❖ IEEE 802.1Qat: This standard is used by AVB and is a standard for stream reservation protocol (SRP). This guarantees that the bandwidth for sending audio and video streams is ensured and not affected by other network traffic.

❖ IEEE 802.1BA: This standard for traffic management by AVB enables more efficient use of network resources while ensuring that real-time traffic is provided with high reliability and low latency.

❖ IEEE 802.1Qav: Defines forwarding and queueing rules to ensure messages are received immediately.

However, AVB still fails to fulfil the demands of hard real-time systems since it cannot handle channel congestion, and messages can be delayed because Credit-Based Shaper (CBS) can’t guarantee zero jitters, which makes it not completely deterministic [8] [9]. CBS is used in TSN to handle the traffic flow and ensure that the necessary traffic is transmitted by allocating credits to each stream so they can be prioritized. The IEEE 802.1 AVB Task Group was renamed the TSN Task Group to solve this issue with AVB. Additional approaches have been introduced to transmit time-sensitive traffic via Ethernet. TSN is the most current Ethernet extension and the successor of IEEE AVB, which will be explained in detail in the next section.

2.3 Time-Sensitive Networking (TSN)

TSN was developed to solve the increasing demand for deterministic, high-bandwidth, low-latency communications in automotive, industrial automation, and other sectors where real-time communications are essential. TSN is a set of Institute of Electrical and Electronics Engineers (IEEE) standards. The 802.1 standards were developed by the TSN Task Group to simplify the transport of time-sensitive communications over Ethernet networks [11]. The TSN operates at the OSI data link layer, ensuring that information transmitted between two devices arrives in a specified and predictable time frame. For TSN to ensure that the network is deterministic and meets the real-time requirements of the system, time-synchronization and scheduling mechanisms are used. TSN uses two scheduling mechanisms, TAS and CBS, to fulfil the varying real-time requirements of different applications. Both scheduling mechanisms are needed to guarantee that high-priority traffic is carried out with minimal delay and jitter while allowing the transmission of traffic with lower priority. TSN uses time synchronization to synchronize the different clocks within a network to a standard time domain. To ensure the network's schedule is maintained correctly, TSN requires that all switches be clock-synchronized using 802.1AS. IEEE 802.1AS consists of several mechanisms used to synchronize clocks. The three main mechanisms are the Best Master Clock Algorithm (BMCA), Propagation Delay Measurement (PDM), and Transport of Time Synchronization Information (TTI) [12]. The forthcoming sections will describe the mechanisms used to synchronize the clocks and the scheduling mechanisms in TSN.

2.3.1 Best Master Clock Algorithm (BMCA)

BMCA is a mechanism that enables deterministic and low-latency communication over Ethernet networks and is a crucial component of the TSN standard. In TSN networks, BMCA decides which clock source is the most accurate and should be the network's reference clock [12]. The BMCA determines the hierarchy between the different TSN devices and selects the grandmaster clock. The BMCA defines a time-synchronization-spanning tree, where the grandmaster assumes the root’s role, as shown in Figure 1. As shown in the figure, each system can operate as a grandmaster time-aware system or a slave time-aware system [8]. BMCA selects the grandmaster clock in a TSN network by combining clock quality evaluation and priority ranking. To determine the hierarchy between the devices in the network, each BMCA system periodically sends a broadcast message called an announce message. After identifying the grandmaster, all other systems or nodes become slaves and synchronize their clocks with the grandmaster. The figure illustrates the four different types of port roles, namely Master ports (M), Slave ports (S), Passive ports (P), and Disabled ports (D). BMCA determines the hierarchy of TSN devices based on the information provided by each port role and chooses the most accurate clock source to serve as the grandmaster clock.

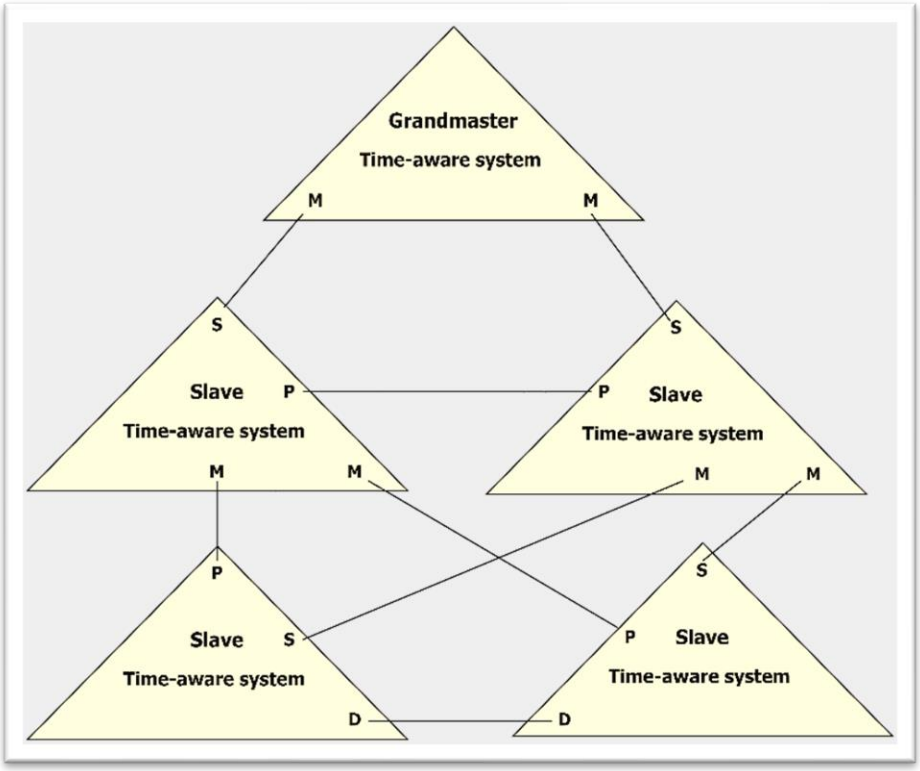

Figure 1: Best Master Clock Algorithm diagram

2.3.2 Propagation Delay Measurement (PDM)

After the hierarchy has been established in BMCA, the PDM mechanism calculates the propagation delay between systems. PDM is a crucial component of TSN, enabling precise synchronization among the devices in a network by compensating for signal propagation delays [12]. PDM begins with one system transmitting a latency request (Delay_request) to another system through its slave port, whether it is another slave time-aware or the grandmaster, and it keeps track of when the message was transmitted (T1), as shown in Figure 2. The time when the message was received is sampled by the receiver (T2) after receiving it through its master port, which returns T2 to the initiator, where the time the message was sent will be noted. After the initiator receives T2, the time the message was received (T4) will be recorded too. Lastly, to calculate the delay, the responder transmits T3 to the initiator. The following formula is used to calculate the delay:

$$\text{Delay measurement} = \frac{(T4 - T1) - (T3 - T2)}{2}$$

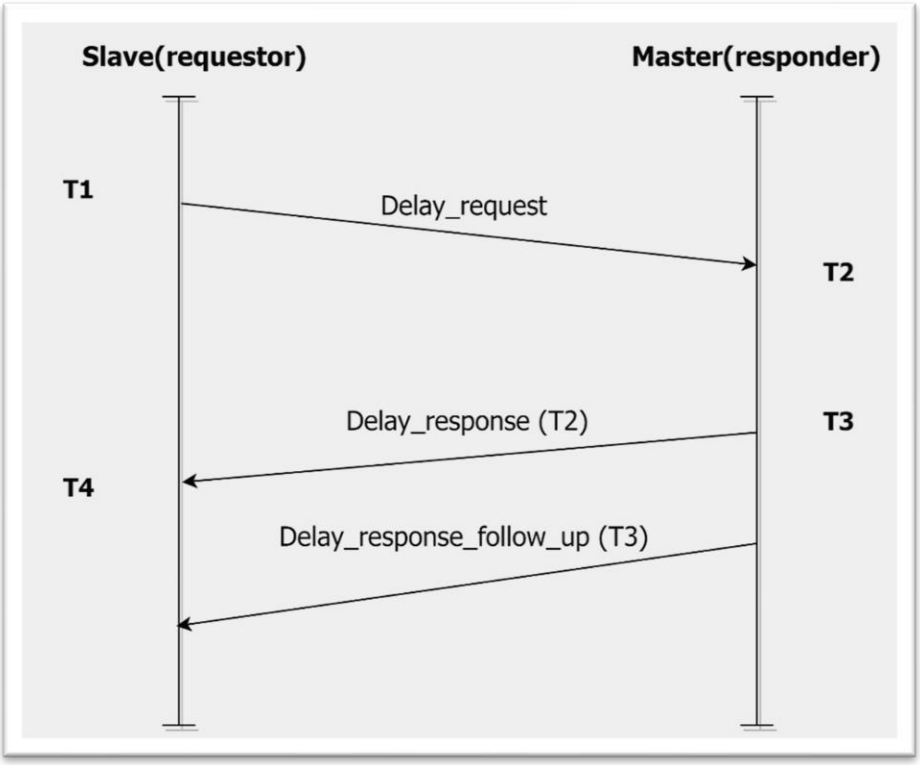

Figure 2: Propagation Delay Measurement diagram

2.3.3 Time Synchronization Information (TTI)

The TTI algorithm can synchronize other time-aware systems by forwarding the grandmaster time [12]. The execution of TTI occurs after establishing the spanning tree with the root as the grandmaster and the latency measurement by the slave time-aware systems. During the TTI procedure, nodes broadcast their current time via their allocated master ports. Systems that receive messages through their slave ports add the calculated delay and modify their local time accordingly.

2.3.4 Scheduling Mechanisms

To avoid congestion and ensure that all traffic can be sent without dropping packets, shaping algorithms regulate the traffic rate on a network in TSN. TSN uses two traffic shaping algorithms, TAS and CBS, to distribute network resources across various classes. The sections that follow provide detailed descriptions of the TAS and CBS mechanisms.

2.3.4.1 Time-Aware Shaper

TAS is an essential mechanism of TSN and is specified in the 802.1Qbv standard, it determines when different traffic classes can transmit on a network [2]. In TAS, TT traffic is prioritized over other traffic using a scheduling algorithm to keep the network from overloading. In Figure 3, we can see an overview of the several possible priority classes in TSN. The first class, ST, is used for network control traffic and has the highest priority. Later, we have the AVB A class [13], which is frequently used for video traffic and has a high priority, while AVB B is used for audio traffic and has a medium focus. Lastly, there is the BE traffic, which is used for "best effort" traffic and has the lowest priority. The Gate Control List (GCL) is a parameter in TAS and is the gate driver that changes the gate state in TAS. Figure 3 shows the GCL table, with a value of 1 denoting an open gate and a value of 0 indicating a closed gate. When a gate is in the open state, frames can be transmitted, and when the gate is closed, there is no transmission. If several gates are open simultaneously, the traffic with higher

priority takes precedence and is transmitted, causing lower-priority traffic to be delayed. When TAS finishes scheduling the last queue in the GCL, it starts over again from the beginning.

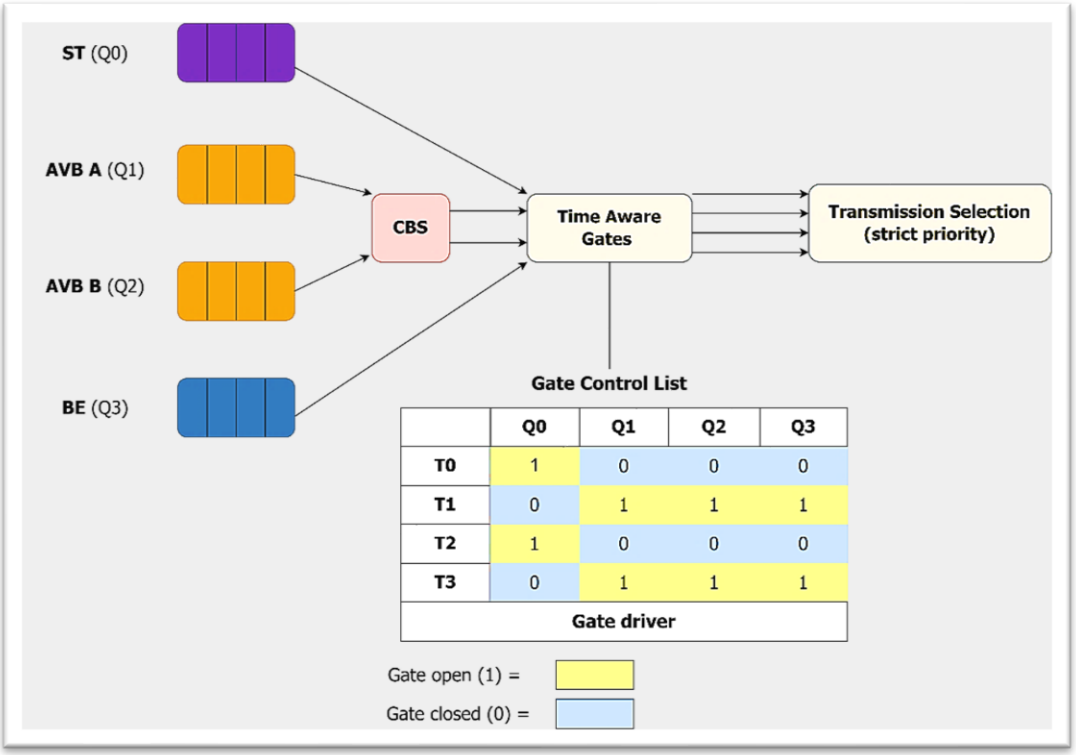

Figure 3: Time-Aware Shaper (TAS) diagram

2.3.4.2 Credit-Based Shaper

CBC is used to limit the traffic rate on a network using a credit-based approach. CBS is another scheduling mechanism that can be applied with TAS, defined in 802.1Qav. The shaping mechanisms complement each other [2]. TAS attempts to reduce the time that high-priority traffic cannot transmit due to the presence of low-priority traffic. In contrast, CBS ensures that low-priority traffic, such as BE traffic, can send traffic without being starved by high-priority traffic [8]. Class A and Class B are the two TSN traffic classifications that CBS typically operates, with Class A having a higher priority than Class B. CBS makes sure that the necessary traffic is transmitted using a variable called "credit," so it can be scheduled in a prioritized way. Figure 4 displays an illustration of a CBS diagram. AVB A credit is represented by the purple line, whereas the orange line represents AVB B credit. If the credit becomes negative after transmission or another queue is transmitting, and frames are waiting in the queue, the credit value increases. When all transmissions in that queue have ended and the queue is empty, the CBS algorithm will reset positive credit to zero. The rectangles with letters shown in the figure indicate the transmission of the frames, and the arrows indicate when the frame has arrived. Finally, the lines show the credit changes over time.

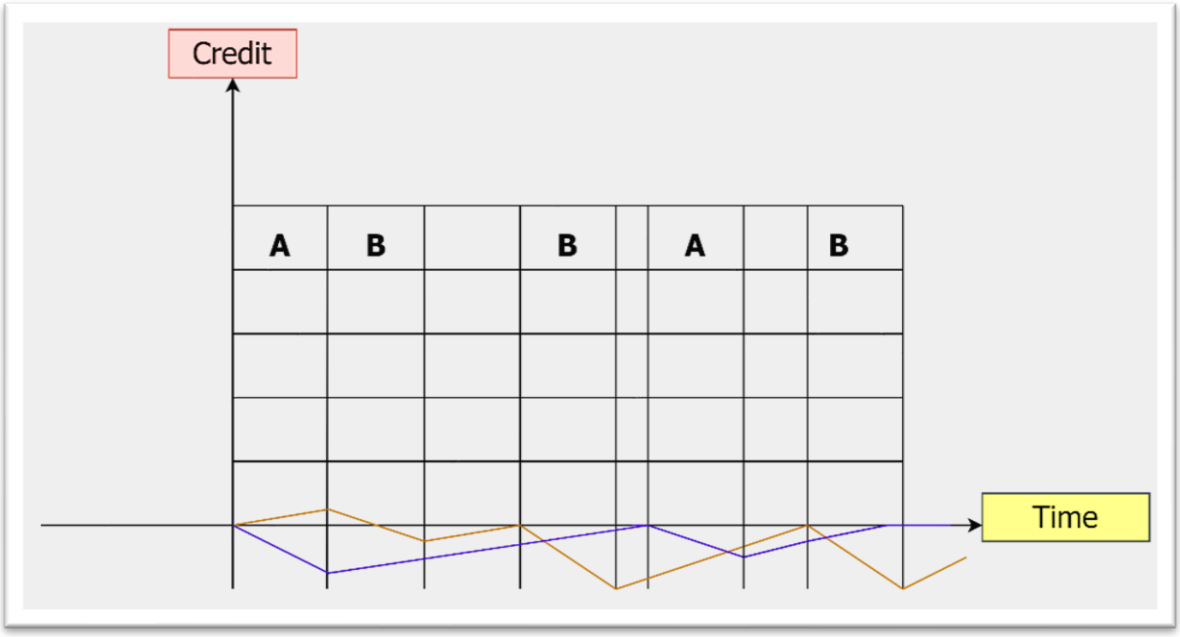

Figure 4: Credit-Based Shaper Diagram

3.Related work

There has been an increasing focus on facilitating the implementation of TSN networks since TSN gained more popularity. Researchers have focused on several TSN-related topics, such as time synchronization, traffic scheduling, and shaping, to improve the effectiveness and reliability of real-time communication in industrial and automation applications. Time synchronization is the main subject in this work, as this thesis aims to evaluate and implement solutions for the outcomes that a network or system experiences if specific network components lack full synchronization.

One of the most critical aspects of TSN is clock synchronization. TSN uses the generalized Precision Time Protocol (gPTP) to synchronize the system's device's clocks to a reference time. gPTP and PTP are similar. However, gPTP has more features compared to PTP. The Network Time Protocol (NTP) is used in this thesis as a time synchronization protocol to synchronize the legacy end-stations with each other. In this paper [14], the authors compare PTP to the NTP and describe the principles and operation of PTP. Paper [15] presents a work similar to this thesis. According to the authors, unscheduled high-priority emergency event traffic can cause significant jitter and delay in time-sensitive traffic. The authors suggested improving the TAS scheduling algorithm by adding a protection band that would allow rapid transmission of emergency event traffic with little impact on the scheduled traffic, and they evaluated the effectiveness of their proposed scheme using the OMNeT++ simulator. A thorough understanding of how TSN works is required to assess the impact of a lack of synchronization. To overcome the inaccuracy caused by packet collisions, the authors in [16] introduce an algorithm combining time-slot-based synchronization and priority scheduling. Their proposed solution is built on the OMNet++ simulation platform and uses a standard in-vehicle network topology. The authors stated in [17] that it is frequently unrealistic for systems such as legacy devices to synchronize with the TSN network and have TSN capabilities. Therefore, they reduced the synchronization standards for end-stations. Later, they developed a solution to offer time-sensitive traffic real-time guarantees. Zimmermann et al. [18] proposed various ways to reduce synchronization start-up time for future Ethernet-based automotive networks. They claimed that the simulation-based performance analysis reveals that the proposal can lower start-up times by a factor of 40 compared to the conventional IEEE 802.1 AS without significantly influencing synchronization inaccuracy.

Despite the different studies related to TSN and several TSN models presented by researchers, there has been limited research into the performance of partially synchronized heterogeneous TSN networks. Therefore, this thesis aims to evaluate and implement the proposed solutions in [9] to solve the problems occurring when adding legacy devices into the partially synchronized network, as there is no research suggesting a general working solution applicable to most legacy end-stations. Based on previous work in [8], [9], we found adverse consequences when adding TSN switches to a legacy network. However, these effects are solvable by implementing the solutions proposed in the previous thesis [9]. This will allow the industry to adopt TSN without facing those side effects. Implementing and evaluating the solution in [9] is necessary, as there is no functional implementation of the solutions proposed in previous related works. The implementation presented in this thesis is also considered more practical than the one previously proposed, as the authors in [9] were modifying the TAS windows manually. Meanwhile, the implementation performed in this thesis provides the opportunity to adjust the size of the TAS window automatically and enables the update of those TAS windows dynamically. The implementation proposed in the thesis can also be adopted by industries seeking to leverage the benefits of integrating legacy devices into TSN networks, as this implementation enables such integration.

3.1 Description of previous work

The previous work in [8], [9] was an evaluation, as in the first thesis [8], the authors discovered that the lack of synchronization in the network causes negative impacts between the computer systems and the TSN device. Meanwhile, the second thesis [9] has also discovered that partially synchronization had adverse effects on the network, and they proposed solutions to minimize those consequences in the network. Lastly, in this thesis, we aim to implement the proposed solutions in [9] in a more functional manner and then will test and evaluate them. This is because the authors in previous work [9] did a proof of the solution but not a functional implementation of the proposed solution. A further description of how the solution will be implemented more practically is provided in the thesis.

The authors of the first thesis [8] investigated the effects of a lack of synchronization in a heterogeneous TSN network regarding reception time drift and jitter. They implemented a network where TT traffic was sent between two laptops running Linux. In their work, they carried out an experiment where the data was sent between the laptops without the TSN switch in the first scenario. In another scenario, the TSN switch was added to the network when the traffic was sent. Their study discovered that devices cannot synchronize their clocks with each other if the network is out of synchronization. Consequently, this will cause different clock speeds (drift) between the devices. The TSN switch in this work has the fastest clock, the receiver node is in the middle, and the sender computer has the slowest clock. The resulting drift can be either positive or negative. The positive reception time drift results in messages being received later and later, which means losing deadlines. Meanwhile, the negative reception time drift results in the messages being received earlier and earlier, which causes some tasks to be performed earlier than expected, such as the airbag opening earlier than expected, resulting in a catastrophic condition. The authors also detected a jitter in the network caused by the Raspberry Pi's lack of a hardware clock and less precise data.

In the second thesis [9], the authors investigated the impact of synchronizing legacy end-stations via their traditional synchronization mechanisms in heterogeneous TSN networks. They also implemented a network using two Raspberry Pis and a TSN switch to observe the network's behaviour. They performed three experimental scenarios to observe the network behaviour when sending TT traffic between two nodes. As a conclusion of their work, they revealed adverse effects when the network is partially synchronized, such as the network experiencing clock drift between the TSN switch and the Raspberry Pis, where the clock drift can either be positive or negative. If the receiver node experiences negative reception time drift, the packets miss their TSN window; meanwhile, if the receiver node experiences positive reception time drift, packets are queued until packet drops occur from the buffer in the switch. Lastly, the authors proposed several solutions to minimize the clock drift experienced in the partially synchronized TSN network. Their first proposed solution is to measure the packet reception drift at the receiver node and apply the measured value to the TSN switch. Meanwhile, their second solution involves periodically calculating the drift in intervals by timestamping packets in the receiver node. Then the receiver node dynamically updates the sending period in the sender node. In this way, they decreased the clock drift by dynamically modifying the transmission interval at the sender node in their second solution. To implement the solution, the authors have modified the legacy end-stations and applied the solutions to them. Aside from investigating the impact of having partial synchronization in the network, the authors also analyzed the jitter occurring in the network.

4.Problem formulation

Embedded systems are all around us in the modern world and continue to develop over time [3]. The advanced features and functionality of many embedded system applications rely heavily on real-time capabilities; as a result, it is imperative to supply mechanisms and capabilities to support these real-time needs. TSN offers several capabilities that enable applications with strict real-time conditions to communicate in a deterministic, low-latency manner. Time-sensitive data is delivered on time and with minimal delay because of the TSN's standards, which include features like traffic shaping, time synchronization, and priority-based forwarding. However, implementing these standards can be costly and time-consuming for companies. Therefore, integrating TSN into existing legacy networks and providing them with real-time capabilities can be cost-effective for companies, but it also increases the time and resources required to implement the TSN environment. However, for TSN to function effectively and for time-sensitive traffic to be delivered with low latency and high reliability, the network requires full synchronization.

Therefore, in this thesis, we will integrate existing legacy devices in a TSN network without incorporating TSN functionality into them. However, because legacy devices do not have capabilities and cannot implement TSN synchronization protocols, they cannot synchronize with the TSN switches. Therefore, the network is partially synchronized as the end-stations are synchronized with each other and the TSN switches have their synchronization protocol. The partial synchronizations in the network cause reception time drift between the legacy devices and the TSN switches. The resulting drift can be positive or negative, each with different consequences in the network. When a significant amount of the drift is collected in the case of the negative drift, the frames miss the transmission timeslot in which they are scheduled, leaving a period with no messages. Figure 5 illustrates the indicated behaviour where the receiver end-station has a faster clock than the other devices. In Figure 6, the opposite case is considered i.e., when the receiver end-station has a slower clock. As shown in Figure 6, the positive drift causes the frames to be lost at a linear rate. Since the frames arrive at the switch faster than it forwards them, leading the frames to get stuck in the buffer. However, the buffer has a limited amount of space as, when the buffer reaches its limit, the frames will be lost. As outlined in subsection 3.1, the authors in [8] and [9] detected the consequences resulting from the reception time drift and are carried out in this work to give the readers a better understanding of what we aim to solve in this thesis.

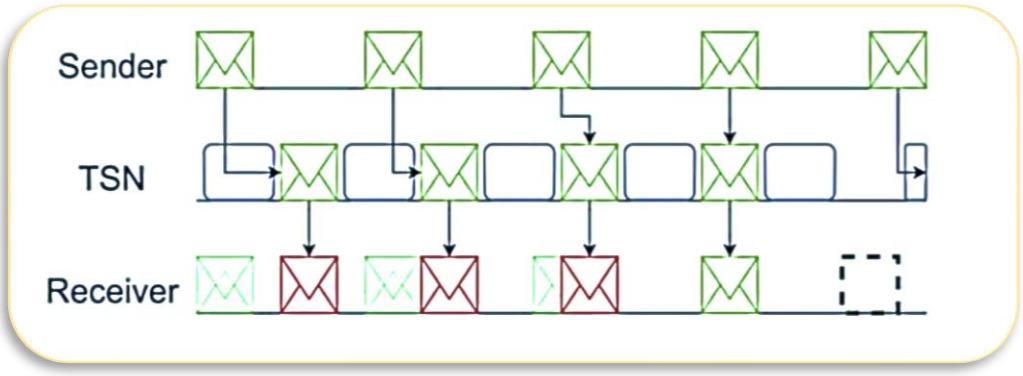

Figure 5: Negative reception time drift in the receiver end-station

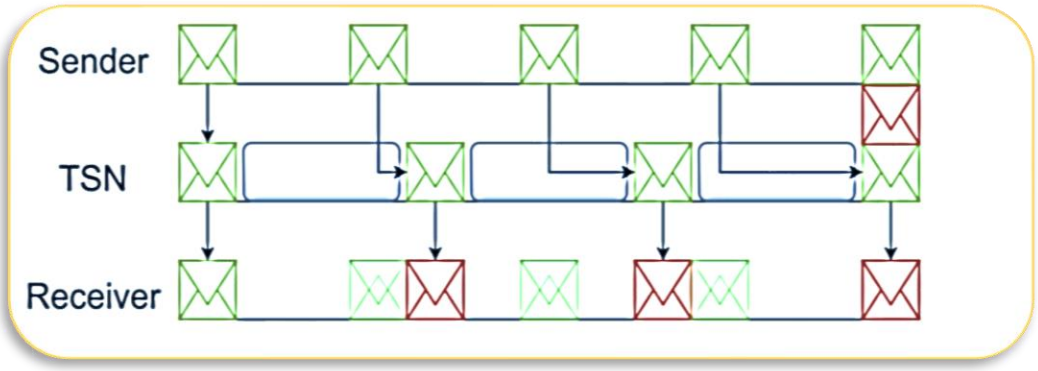

Figure 6: Positive reception time drift in the receiver end-station

Therefore, partial synchronization in the network is a problem because of the previously mentioned consequences and must be solved. Solving the reception time drift is critical to allow the industries and companies to adopt TSN into the legacy network without facing those side effects. This thesis aims to evaluate and implement the previously proposed solutions in [9] for a partially synchronized heterogeneous TSN network regarding reception time drift.

In this thesis, we aim to investigate and address the research questions outlined below:

• RQ1 - What are the effects of the proposed solutions to mitigate the effects of clock drifts?

• RQ2 - What would be the suitable implementation of the proposed solutions in a real network and how does it differ from the experimental model?

To address the research questions (RQ1 and RQ2), existing literature about TSN will be reviewed. Based on the knowledge gained through this process, the proposed solutions will be implemented in a lab environment featuring various scenarios for experimentation and analysis. The RQ will then be answered with the conclusion of the experiments.

5.Research method

To gain knowledge and a foundation in the topic and insight into the equipment involved in this thesis, we study the mechanisms of TSN involved in or related to clock synchronization. An experimental research method is appropriate for this thesis because the purpose of the thesis is to implement the previously proposed solutions in [9], which will be tested and evaluated. According to Amaral [19], an experiment consists of the *evaluation* and *exploratory phases*. The evaluation phase refers to the questions that will be attempted to be answered, while the exploratory phase refers to a body of knowledge that helps determine what questions to ask about the system being evaluated. The experiments that will be carried out are based on the implementation of the solutions proposed by the authors in the previous thesis [9], while the results will be analyzed and evaluated.

The system development research process proposed by Chen and Nunamaker [20] is suitable for this purpose; it is a multi-method approach to conducting information systems research. The research method consists of five steps, as shown in Figure 5.

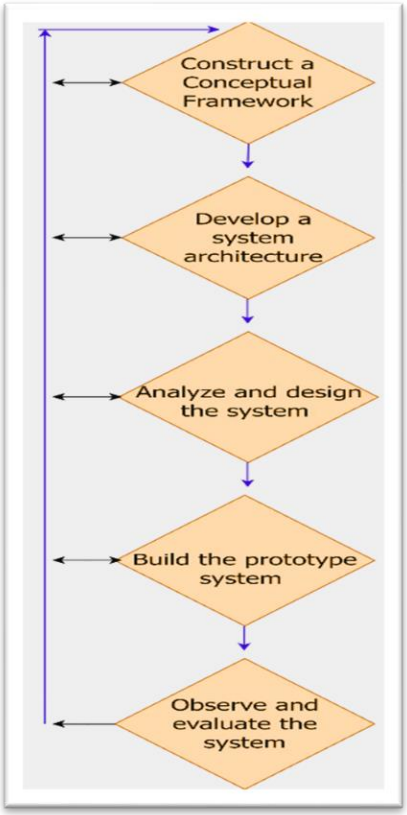

Figure 5: The Process of System Development Research

The following steps will be taken to answer the research questions carried out in this thesis:

➢ Investigation of the TSN-related literature and research on the TSN mechanisms

➢ Review the previous related thesis and gain knowledge to prepare for the experiment.

➢ Implementation of the solutions previously proposed

➢ Perform the experiment and analyze the output

➢ A conclusion will be made based on the comparison and evaluation of the result

6.Ethical considerations

Our thesis does not gather any personally identifiable information and does not address any ethical issues or concerns. The tools resulting from this work are said to be used responsibly. This is a purely scientific experiment conducted in a simulated environment with consideration for societal aspects, as the purpose of this thesis is to explore the possibility of adopting TSN in a network with legacy end-stations. Through this work, we consider providing others with the benefits achieved when integrating legacy devices into TSN networks without implementing the TSN functionalities into them. This integration also works effectively without requiring complete network synchronization for the components to function correctly. Another advantage of legacy devices is that the devices used in the previous system version (before adding the TSN devices to the network) can be reused even if the network subsystem has been changed. In this way, previous configurations and work saved on these devices can be reused, saving time and resources.

7.Description of experiments

We conducted three experiments to answer the research questions in the problem formulation. In the following section, the experiments conducted and the implementation of the solution are described. This thesis aims to implement the proposed solutions in [9] for a partially synchronized heterogeneous TSN network. The network is partially synchronized as the end-stations are synchronized and the TSN switches have their synchronization protocol, see Figure 6. This means that the end-stations have no knowledge of the TSN switches and do not synchronize with them. The evaluation in this thesis will consist of three experimental scenarios. In the first scenario, we are studying how the network behaves when sending traffic between two end-stations without the influence of TSN switches. In the second scenario, the data is also transmitted between the end-stations, but the TSN switches are introduced in the network. The second scenario performed in this thesis is also one of the experiments the authors in the previous work [9] have performed, as mentioned in section 3. Lastly, in the last scenario, we aim to implement one solution by combining those proposed solutions in the previous work [9] to solve the adverse effects caused by partial synchronization. This thesis applies the proposed solutions to the TSN switches without modifying the legacy end-stations.

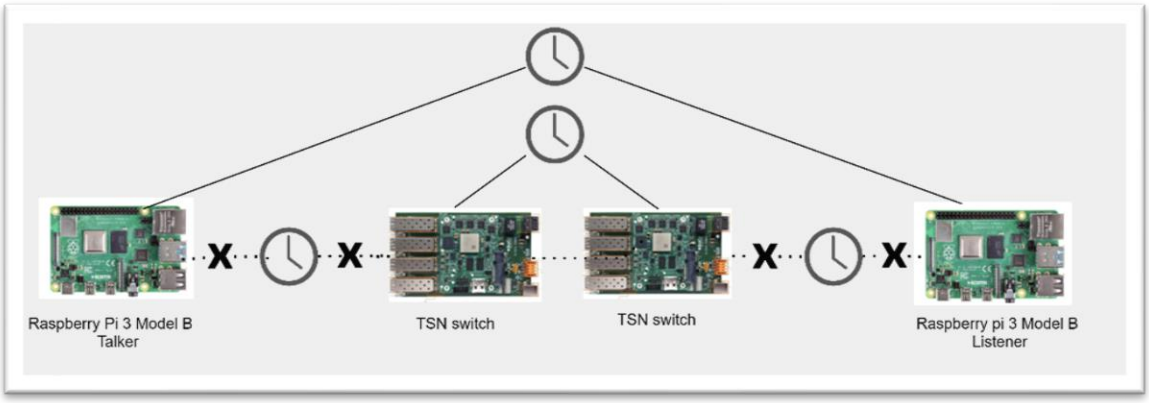

Figure 6: Synchronized end-stations and synchronized TSN switch with no synchronization in between.

7.1 Hardware description

In this thesis, a lab environment consisting of two legacy end-stations (Raspberry Pi 3 Model B) [21] with the Raspberry Pi operating system (OS), two multiport TSN switches System-on-Chip Engineering (SoC-e) [23], see Figure 7. Lastly, a laptop running the Linux OS is used. Several synchronizations protocols can be used to synchronize end-stations if incompatible with the gPTP protocol used by the TSN switches. However, in this thesis, NTP was used to synchronize the end-stations with each other, so their software clocks share the same time domain. NTP is a widely used time synchronization software protocol designed to synchronize the clocks of computers on a network to a highly accurate time reference, for example, a GPS receiver or an atomic clock [22]. On the other hand, the TSN switches use a hardware clock, which can be referred to as a "real-time clock" thanks to its high accuracy.

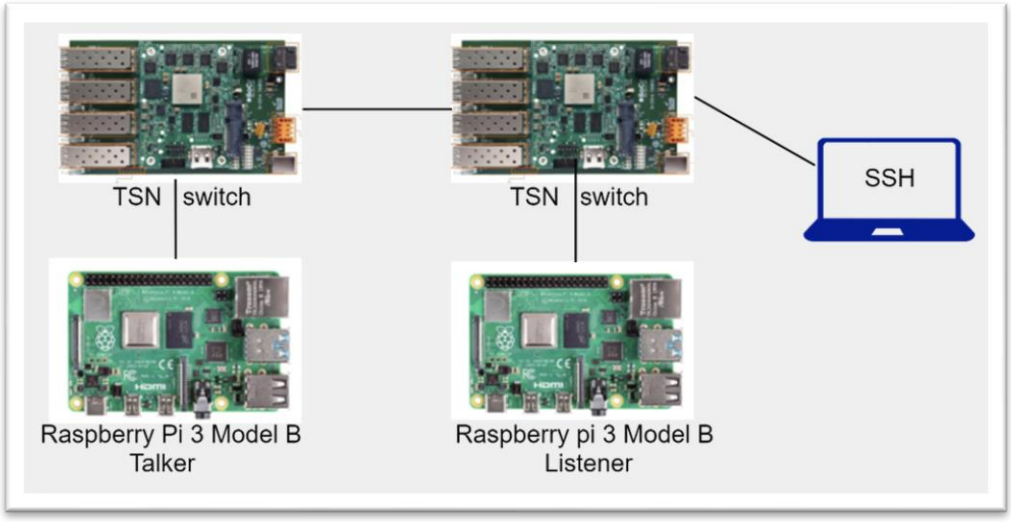

Figure 7: Experimental topology

7.2 Sending traffic and data analyzing

To perform the three experimental scenarios, one Raspberry Pi was the Talker node sending the TT traffic and the other Raspberry Pi, i.e., the Listener node receiving the data. In specific experiments, the TSN switches were also introduced between the end-stations when the data was sent. The experiments were conducted by executing the Python scripts mentioned in *Appendix A: The codes for experiments 1 & 2 (Listener.py and Talker.py)* for the first and second experiments, and the scripts provided in *Appendix B The codes for experiment 3 (Listener.py and Talker.py)* and *Appendix C The code to the cnc.py* for the third experiment. The authors in [9] have implemented the Python scripts, which have been adjusted to suit the experiments conducted in this thesis. One method used by TSN to handle priority traffic is VLAN tagging. In the scripts that transmit TT traffic between the end-stations, we added an 802.1Q header to the Ethernet frame used for periodic sending. TT traffic was selected in this experiment because the partially synchronized network significantly impacts TT traffic more than other traffic. In all the experiments, Q7 was removed from the ports without a schedule applied to ensure that traffic was only sent to the appropriate port. Q7 is the highest priority queue that can be given to TT in the TAS configuration. By doing this, there is no indication that the TT mechanism used in the experiments would be influenced by other network traffic as TT traffic is isolated in its respective queue. We modified the transmission rate of the sender end-station by adjusting the period after every 100 messages to generate more significant negative and positive reception time drift in the receiver end-station. This was performed by the Talker sending every new period to the receiver end-station. However, the TSN switches cycle was kept at one second. The messages were transmitted from the Talker node towards the Listener node in an interval of 1 second for negative drift and an interval of 0.3769 seconds for positive drift. Those seconds were selected as the clock drift caused an impact on the period, leading to the period differing between those values.

After the experiments were executed for 25 minutes, the "samples.txt" files were generated and saved by the Listener script. The data stored in the "samples.txt" files were later used to illustrate the results gathered from the experimental scenarios. The Listener saves data in “samples.txt” files about when the messages were scheduled to be received and the actual reception of the messages. Later, with this information, we could compare and evaluate the different results gathered from the experiments. To illustrate the results, no mathematical calculation was required. This is because the way we presented the results provides all the information needed to conclude and evaluate how the

effects caused by the clock drift affect the various experiments performed, as we can see in section 8. To determine when the data was sent and received between the end-stations, we analyzed the results from the experiments in Microsoft Excel. In this work, we sampled 2000 messages per experiment to get an exact result of the reception time drift. To graph the reception time drift, the scheduled time of the messages was represented as the y-axis, and the reception time of the messages was defined as the x-axis. Later, with the values of the y-axis and the x-axis, we could graph a graph illustrating the three experimental scenarios performed in this thesis.

7.3 Experimental setup

7.3.2 Scenario with and without the TSN schedule

In the first scenario, the traffic is sent between the end-stations, and the switches do not have any influence in this scenario. In the second scenario, the traffic is transmitted between the end-stations and the TSN switches are added to the network. However, the TSN switches forward packets only when their corresponding gates are opened. The *Scapy* Python Module [24] was applied in the Python scripts to enable the transmission of the periodic traffic through TSN. We had to determine the window size so the prioritized packets could pass through and not allow several packets to traverse together in one TAS window. To do this and find the exact window size, the time window for ST was measured using the trial-by-error method. We needed to modify twice the size of the TAS-cycle time in the switches in this experiment and choose either 1000000000 ns or 376900000 ns. This is because the clock drifts influence the period, which causes it to vary all the time between those values. The reason for trying this scenario is to understand how partial synchronization affects the network's performance. As previously mentioned, the previous authors already performed the second scenario in [9]. This is carried out in this work to give the readers a better understanding of what we aim to solve in this thesis. For a better understanding of how the proper schedule was performed on both switches, see *Appendix E: Configuration of the TSN Switches*. In all experiments, the traffic was sent when the end-stations and the TSN switches had different perceptions of time as we synchronized the end-stations with NTP, and the TSN switches also had their notion of time using gPTP.

Figure 8 illustrates the behaviour of the experimental network before implementing the solution. The figure shows two legacy devices, one Talker and one Listener communicating through TSN switches with one second period. The TSN switches share synchronization, and the legacy end-stations are synchronized using NTP. However, as shown in the figure, there is no synchronization between TSN switches and legacy end-station. In this manner, as the messages transmitted by the Talker traverse through the schedule of the TSN switches, the transmission rate changes from the one-second schedule of the legacy end-station to the one-second schedule of the TSN switches. This results in the detection of drift by the Listener between the transmission of the TSN switches and the legacy end-stations schedule. This is due to the period being adjusted by either increasing or decreasing it by 5% compared to the scheduled period.

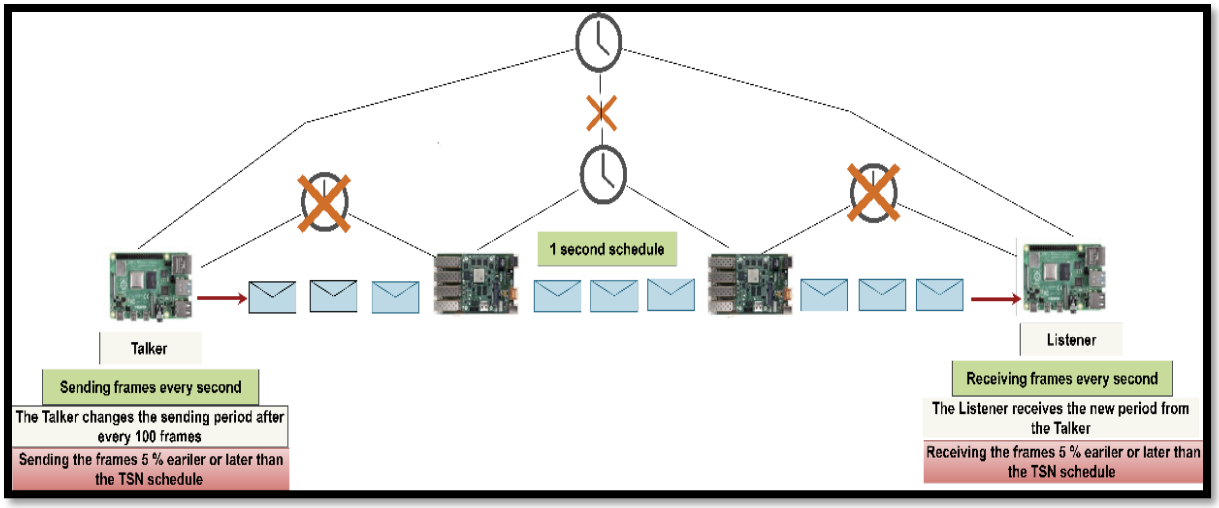

Figure 8: The network behaviour before the solution implementation

7.3.3 Scenario with the solution

The third experiment is based on implementing the proposed solutions in [9] to prevent the clock drift that occurs when sending traffic between the end-stations. The proposed solution consists of measuring the detected clock drift between the end-stations and the TSN switches and modifying the TAS schedule to avoid the adverse effects of partial synchronization. The solution proposed by the authors can be implemented in several ways. In this thesis, we decided to implement the solution using the CNC together with the DD for the solution to function effectively. CNC is a device that controls a whole network by, for example, reconfiguring the TSN networks. In this thesis, a simplified version of the CNC was used with the function of automatically reconfiguring the period of the switches to match the end-station transmission based on the saved configuration of the schedule. The DD is a component with the function of detecting any deviations or drifts in the network, and it can be implemented in several positions in the network. In a real network and as best practice, the optimal place for the DD is in the TSN switches and compare the Talker transmission with the TSN schedule. This is because of not interfering with legacy systems and maintaining them unchanged, just like in the previous system. However, as we could not access the TSN switches and were not allowed to modify them in this thesis, we implemented the DD in the Listener script. Instead, we compared the transmission of the TSN switches with the Listener's schedule. However, whether you have the DD in the Listener script or the TSN devices, it will provide the exact solution to the clock drift, as the position of the DD does not matter. Docker desktop has also been used in this scenario, an application that is easy to install on PCs [26], among other things, to build and share containerized applications and microservices. Docker Desktop is the environment where the CNC was implemented; see Figure 9. The CNC in the Docker environment was implemented by a former PhD student [25] but has been adjusted to fit this scenario. To understand how to install and set up the Docker Desktop environment to run the solution, see *Appendix F Setting up Docker Desktop environment.*

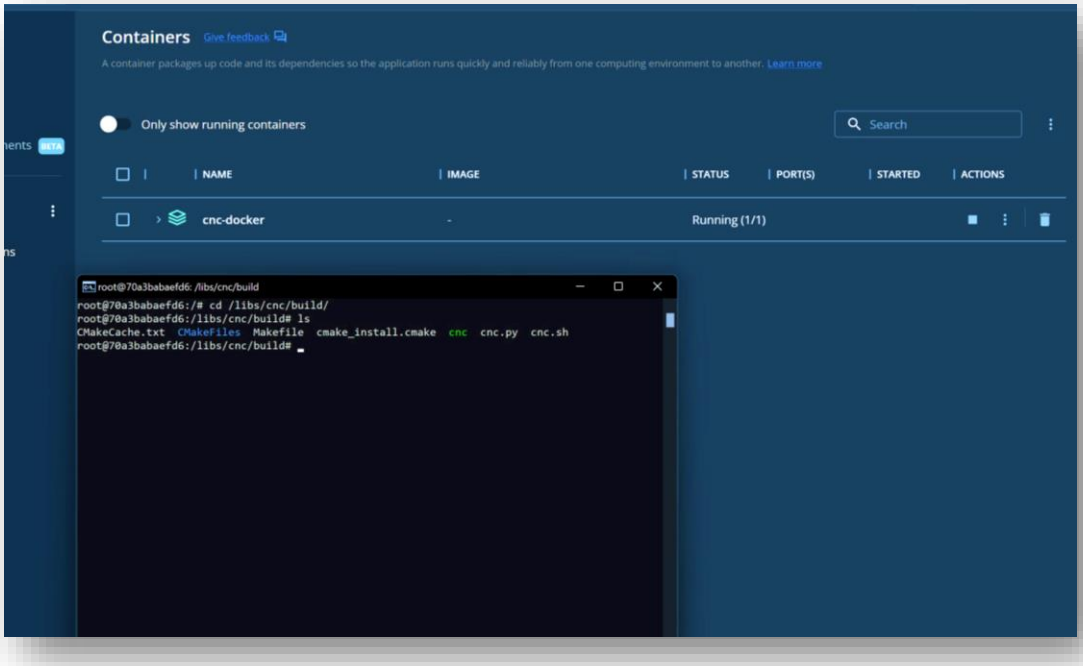

Figure 9: The CNS container created in the Docker Desktop

Figure 10 illustrates the implementation of the solution in the partially synchronized network. To implement the proposed solutions in [9], we decided to apply their solutions to the TSN switches without modifying the legacy end-stations, as shown in Figure 9. This figure exhibits similarity to the previous Figure 8, as we also have in this figure one Talker and one Listener communicating through TSN switches with one second period. However, in this network, we added the DD to the Listener end-station, with the function of detecting the variation between the reception of the end-station (i.e., the transmission of the TSN switches) and the schedule of the end-station. The DD performs this by sampling the traffic by packet sniffing with Scapy and checking if the slope value is within the predefined range to consider whether the drift is stable or not; see the Listener code in *Appendix B The codes for experiment 3 (Listener.py and Talker.py).* After the DD detects the variation between the expected and actual reception of the messages, it sends the value of the variation to the CNC. The CNC is responsible for updating the network and implements an automatic reconfiguration method to update the configuration of the TSN schedule according to the drift value measured by the DD, see Appendix G: Config.json. The solution is deployed within the system once the required processes have been completed. As a result, the schedule of the end-station and the TSN switches will share the same period, and the network will experience less drift. To inform the CNC about the drift value detected, one connection was established between the DD and the CNC, *see Appendix C The code to the cnc.py.* An example will be provided in the next section illustrating how the clock drift affects the partially synchronized network and how the solution with the CNC and DD solves this issue.

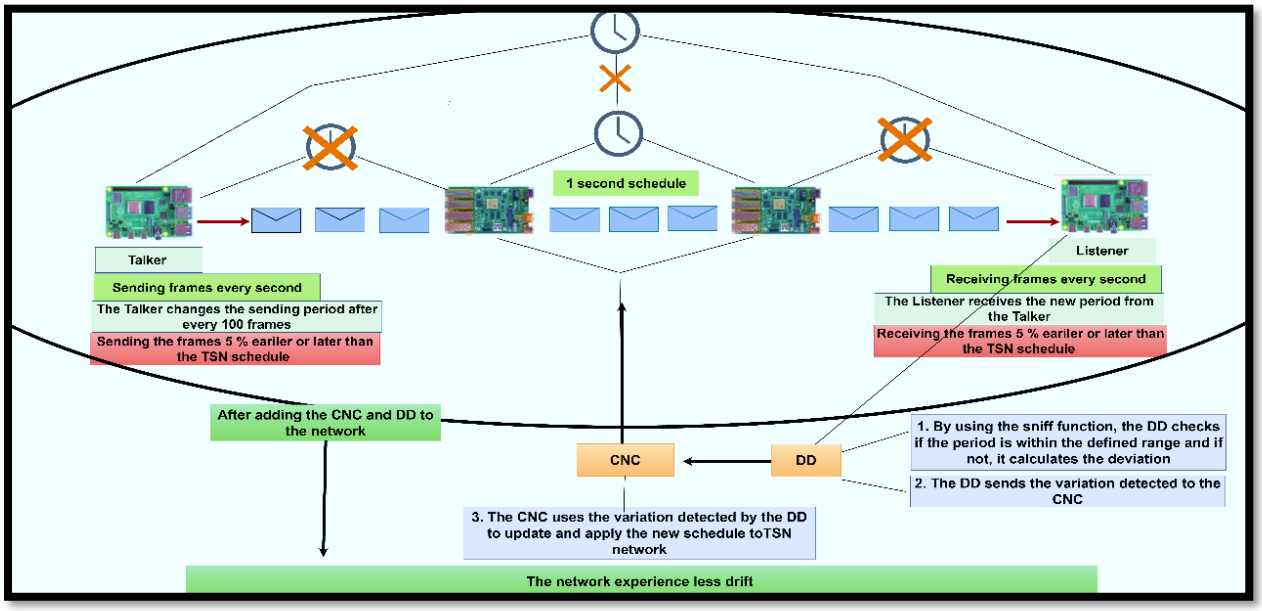

Figure 10: The network behaviour after adding the solution implementation

In Figure 11, we can see that initially, the Listener receives messages without drift. However, when the new period is applied, the messages differ from the one-second schedule. The DD detects the variation between the schedule and reception of the messages and sends this information to the CNC. After the CNC updates the period, the drift begins to reduce as the TSN switches adjust their transmission period to match the periodicity of the legacy end-stations.

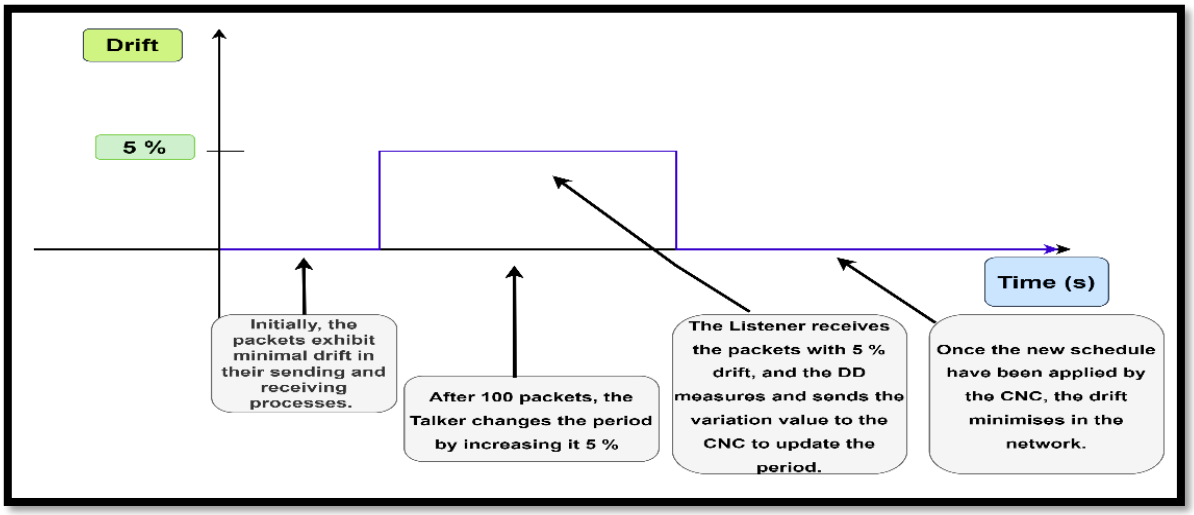

Figure 11: illustrates a simple example explaining before and after the solution implementation

The authors proposed solutions in [9], including adjusting the network's period to ensure all devices share the same period; we implemented one solution by combining their solution using the CNC and the DD. However, the implementation of the solution performed by the authors had several issues. The author's implementation was not correctly implemented because they lacked in their study the tools and resources necessary to carry out a functional implementation that could benefit others, like companies. In the previous author's proof of the solution, they were modifying the end-stations in a way that they were not designed by changing the application behaviour, to solve the varying clock drift in the network. Changing the application behaviour can, in some cases, cause a system to fail if the application is critical to the operation, as it is for the end-station in our case. Apart from not meeting their time requirements due to their modifications in the end-stations, these modifications may not be applicable due to hardware limitations. Due to these characteristics, the industry cannot benefit and adopt the author's implementation.

We believe applying the DD together with the CNC is the best method to implement those modifications to reduce the clock drift in the network. In other words, the CNC and the DD perform this process automatically without interfering with the legacy end-stations. Another benefit of implementing the solution with CNC is that instead of manually configuring the TAS window, the CNC performs this automatically and applies the new configuration to the entire network. Companies or researchers can also benefit from the implementation proposed in this work, as it provides environmental benefits, as the legacy devices used in the previous system can be reused even if the network subsystem has been changed. In this way, previous configurations and work saved on these devices can be reused, saving time and resources. However, the problem with clock drift can be solved in several ways, such as by adding an extra synchronization protocol to synchronize the TSN switches with the legacy systems. But this would require adding a new protocol to all new and old devices in the network, which isn't feasible because of the idea of not modifying or interfering with the legacy devices. It is also considered to be time- and resource-consuming. This scenario was built the same way as the second experiment, with the same switch schedule and the addition of the CNC on a computer running a Linux operating system.

8.Results

In this paragraph, we present two graphs as the conclusions of the three experimental scenarios carried out in this thesis. The graphs presented in this section are the receiver nodes, showing how the different experiments performed affect the partially synchronized network. The results of all three scenarios are presented in one graph, using three different colours to represent each experiment. The grey colour presents the results from the scenario without the TSN schedule. Meanwhile, the orange colour represents the results from the scenario with the TSN schedule. Lastly, the blue colour represents the results from the solution scenario. However, in this section, we present two graphs, as one is focused on showing the positive reception time drift while the other is focused on the negative reception time drift. We present both negative and positive reception time drift as they impact the partially synchronized network differently, as previously mentioned in this work.

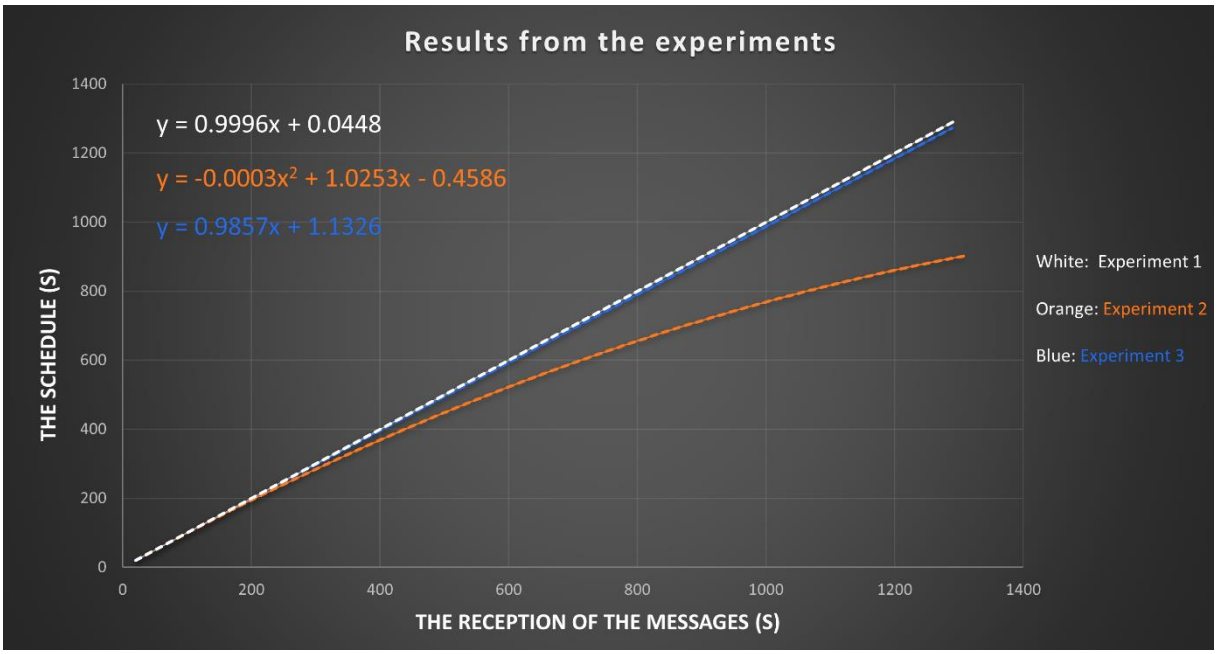

Figure 10: The graph of the Listener node (negative drift)

Figure 3: We can see the graph of the Listener node showing the results of the scenarios previously mentioned in three trendlines. The y-axis of the graph shows the scheduled time (ST), and the x-axis shows the reception time (RT) of each frame in seconds. The graph shows that the expected time matches the reception time when the resulting graph is linear and has a slope of one. The schedule time matching the reception time indicates no drift in the network, as the messages arrived when they were scheduled. Otherwise, if the graph is not linear and has no slope of one, it indicates that the messages sent between the end-stations differ, and the graph can exhibit other slopes and curve functions. We can see that the blue and grey lines match the scheduled time with the reception of the messages, as the slope value is one in both lines. However, the orange line differs from the previous lines and does not fit the schedule. The grey line is linear because the scenario without the TSN schedule is the baseline experiment, and the traffic is sent between two synchronized legacy end-stations using NTP. However, the opposite scenario is applicable for the orange line because the network experiences clock drifts in this scenario due to legacy end-stations and TSN devices not sharing the same time domain. Adding TSN devices to the legacy network generates many benefits, such as reducing jitter; however, it also introduces clock drift in the partially synchronized network.

Therefore, the orange line shows the variation between the schedule and the actual reception of the messages and differs from the other lines. The graph also illustrates the non-linear curves below the other lines because the lack of synchronization results in negative drift. The blue line represents the solution implemented to reduce the clock drift occurring in the partially synchronized network. Due to the implementation of the solution using the CNC and DD to reschedule the switches, the network experiences less drift. The solution solves the problem of the transmission not fitting with the schedule, which generates the blue linear line.

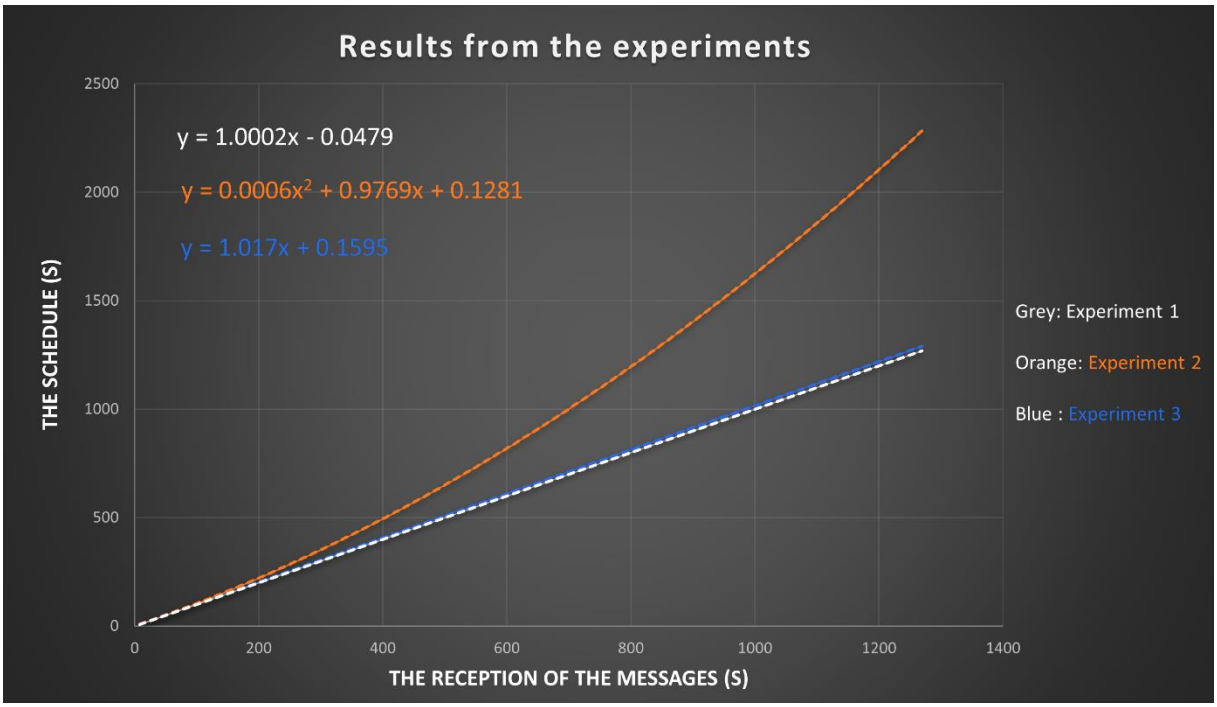

Figure 11: The graph of the Listener node (positive drift)

Figure 11: This graph is similar to Figure 10 since it shows three trendlines. The previous graph showed the Listener node experiencing negative drift. Meanwhile, this graph shows the Listener node experiencing positive drift. This graph illustrates a non-linear curve above the other lines instead because the reception time drift results in positive drift.

9.Discussion

This section summarizes the observations and conclusions related to the performance of a partially synchronized heterogeneous TSN network regarding reception drift. We evaluated and tested the previously proposed solutions in [9] to prevent clock drift in the partially synchronized network since TSN requires full synchronization to function correctly. To investigate this, several experimental scenarios were performed to respond to the questions we intend to answer in this thesis. The network is partially synchronized as we transmit TT traffic through a network that consists of two synchronized Raspberry Pis and two TSN switches, each with its perception of time.

The experiments carried out in this thesis contained three scenarios. In the first scenario, the traffic was sent between the end-stations without the TSN switches having any effect on the network, and the end-stations were synchronized using NTP. This scenario was utilized as a baseline experiment compared with the remaining experiments when traffic was sent between the end-stations. Since TSN switches were not introduced in this scenario, the messages were received without variations in reception. However, as the Raspberry Pi has low quality and can't provide good periodic transmission, this causes high jitter in the network. The Raspberry Pis not having hardware clocks was a limitation in the first experiment due to increased jitter and less precise data. However, in this work, we decided not to focus on the jitter occurring in the first scenario as it is outside the scope of this thesis and has been proven in [8], [9] the previous thesis. Instead, we focus on implementing the proposed solutions to solve clock drift.

The second scenario was performed similarly to the initial scenario, and the TSN switches were introduced into the network. Introducing TSN devices to the legacy network generates many benefits, such as reducing jitter since TSN switches have more accurate clocks since they use hardware clocks in contrast to Raspberry Pis. However, after including the TSN switches in the network, the results showed a clock drift between the expected and the real reception of the messages. The messages were scheduled in the switches to have a period of one second. However, the period will either decrease or increase because of the real reception of the messages in the Listener node and the schedule of the switches differing. The reason for performing those experiments is to understand the partial synchronization impact on the network and the consequences that occur due to clock drift, which was previously mentioned in Section 3.1. If the mentioned consequences occurred, for example, in an airbag inside a vehicle during an emergency, those consequences can be severe and pose a significant threat to human life.

The focus of this thesis was to implement proposed solutions in [9] to solve the problems caused by the clock drift occurring in the second scenario, and to perform this, a scenario with the solution took place. According to the graphical representation presented in Section 8, the solution successfully detected and eliminated the clock drift while maintaining the same schedule used in the second experiment. This was performed using the DD in the Listener script to detect the variation between the expected and real reception of the messages. After detecting the deviation in the network using the DD, the variation values are later used to modify the TSN schedule. We change the TSN schedule dynamically to remove the clock drift using the CNC according to the variation detected by the DD. The results of this step can be confirmed and observed from the graph in Section 8, as the third experiment has the same appearance as the first experiment since lines with a slope of one represent both. The third experiment, which results in a linear line with a slope of one and does not exhibit the same appearance as the second experiment, indicates that the solution implemented for the clock drift works in the network consisting of two end-stations not sharing the same synchronization as the TSN switches.

However, the solution proposed in [9] can be implemented differently; for instance, how the authors implemented the solution is described in Section 3.1. The author's proof implementation modified the end-station period to match the communication subsystem, i.e., the TSN switches. The issue in their implementation was that they were modifying the end-stations in a manner that changed the application behaviour or performance, which might result in a system failure or significant financial losses. Their implementation has other drawbacks, such as hardware limitations due to their solution demands on modifying the end-stations and their static implementation to modify the TSN devices. As rescheduling the switches becomes an increasingly complex problem in a company, it is not feasible to reschedule the switches manually. The way the authors implemented the solution was to demonstrate that the solution works but is not practical and cannot be adopted by industries or companies. Other ways to implement the solution include adding an extra synchronization protocol to synchronize the TSN switches with the legacy systems. But this would require adding a new protocol to all new and old devices in the network, which isn't feasible because of the idea of not modifying or interfering with the legacy devices. Which is also considered time-consuming and resource intensive. Therefore, we believe that the most efficient approach for implementing the proposed solution is to utilize the DD and the CNC to implement those modifications to reduce the clock drift in the network. Implementing the solution with CNC provides the user with the benefit of automatic reconfiguration of the TSN switches.

With this solution, we expect the heterogeneous TSN system to work endlessly even when exposed to environmental changes, such as pressure or temperature, which can affect the different clocks. The limitation of the CNC is that it introduces a time delay in applying the solution since it operates outside of real-time. However, the time required for the CNC to implement the solution is always shorter than the time it takes for the drift to cause a significant impact on the network. This is because the drift variation affects the behaviour of the legacy devices very slowly since we apply a 5% drift after every 100 packets, which takes a more extended period before the drift problems become noticeable. Therefore, in a real-case scenario, the CNC will address the issue before the variation increases or decreases by 5%, as the CNC applying the solution is always faster than the rate at which the drift increases or decreases to reach 5%. Having the DD implemented in the Listener script is also considered a limitation, as the optimal place for the DD is in the TSN switches and instead compares the Talker transmission with the TSN schedule. This is because of not interfering with legacy systems and maintaining them unchanged, just like in the previous system. Therefore, while including the DD in the TSN switches may require additional effort from the industry, it is considered best practice in a real network to avoid having the DD in the Listener end-station. However, as we could not access the TSN switches and were not allowed to modify them, we implemented the DD in the Listener script in this thesis.

Through this summarization of all the observations and conclusions, we have explained what happens when TT traffic is sent in a partially synchronized network using two different synchronization protocols. We have also addressed and explained the reason for the change in the behaviour of the specified performance metrics and the impacts that can occur when we send traffic between devices running different synchronization clocks. The implementation presented in this work allows the industry to integrate existing legacy devices within a TSN network without implementing the TSN functionalities into them. In this way, previous configurations and work saved on these legacy devices can be reused, saving time and resources.

10.Conclusions

The purpose of this study is to implement previously proposed solutions in [9] for partially synchronized heterogeneous TSN networks. Since this thesis aims to integrate TSN switches into legacy networks without integrating TSN functionality into legacy devices, However, such integration presents other challenges, especially from the synchronization point of view, as the legacy systems are not able to fully integrate with the TSN networks because they lack compatible synchronization mechanisms with the TSN switches. This partial synchronization causes clock drift in the network. This clock drift, in turn, has consequences on the network, as previously mentioned in Section 3.1. To solve this issue, we performed three experimental scenarios to evaluate and implement solutions for the clock drift that occurs in the network when the end-stations are integrated with TSN switches.

We set up a small network with two nodes sending TT traffic from the Talker node to the Listener node. Traffic is sent between end-stations in the first experiment without affecting the network from the TSN switches. As the first scenario has the base behaviour, the results confirmed that the messages arrived when they were scheduled to be received. A second experiment took place to evaluate the effects of adding two TSN switches between these end-stations. The cost of the end-stations not being synchronized with the TSN switches is that the clock drift increases in the network. Furthermore, clock drift can have different effects on the network depending on whether it is a positive or negative drift, as mentioned in Section 3.1. A third experiment took place to solve the clock drift problem occurring in the network, which also responded to our first research question: "What are the effects of the implemented proposed solutions on the clock drift?" According to the graphical result presented in Section 8, we can see how the implemented solutions impacted the clock drift. We can observe in Section 8 that in the solution scenario (the blue graph), the reception shows the same behaviour as in the first scenario, although the solution scenario maintains the same schedule used in the second scenario. This indicates that the solution implemented for the clock drift works effectively on the partially synchronized network. This was possible due to our implementation of the proposed solution in [9] using DD together with the CNC to match the period of the end-station with the TSN switch schedule, which is explained in detail in Section 7.3.3.

The second question in this thesis was: "What would be the suitable implementation of the proposed solutions in a real network, and how does it differ from the experimental model?" A suitable implementation of the solution proposed in [9] in a real network will be to have the DD directly implemented in the TSN network and instead compare the Talker transmission with the TSN schedule. This is because of the idea of not modifying the legacy systems and instead maintaining them unchanged, just like they were in the previous system. However, as we did not have access to TSN switches and were not allowed to modify them in this thesis, we implemented the DD in the Listener script and instead compared the transmission of the TSN switches with the Listener's schedule. Whether the DD is in the listener code or the TSN devices, it will provide the same solution for the clock drift. In conclusion, since we had the DD in the listener code and not in the TSN devices, this chosen method is not considered a best practice, but despite this, we get the same solution.

11.Future work

In future work, we plan to get access to TSN switches and implement the solution as designed, i.e., having the drift detector in the TSN devices. Another interesting consideration would be investigating what happens in a heterogeneous network when introducing different end-stations that use other synchronization protocols or methods.

Reference

[1] Q. Jabeen, F. Khan, M. Hayat, H. Khan, S. Jan, and F. Ullah, *A Survey: Embedded Systems Supporting By Different Operating Systems,* May 2016, arXiv:1610.07899.

[2] A. Bergström, “Automatic generation of network configuration in simulated time-sensitive networking (TSN) applications,” M. S. thesis, Mälardalen University, Västerås, Sweden, 2020.

[3] R. Rännar and M. Mustaniemi,” Designing a lab assignment for studying real-time embedded systems,” BSc thesis, Mälardalen University, Västerås, Sweden, 2019.

[4] H. Suljic and M. Muminovic, “Performance Study and Analysis of Time Sensitive Networking,” M. S. thesis, Mälardalen University, Västerås, Sweden, 2019.

[5] K. Tindell, A. Burns and A. J. Wellings, “Calculating controller area network (can) message response times,” *Control Engineering Practice*, vol. 3, no. 8, pp. 1163-1169, 1995.

[6] J. D. Decotignie, “Ethernet-based real-time and industrial communications,” in *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1102–1117, 2005.

[7] H. Hassani, P. J. Cuijpers and R. J. Bril, “Work-in-Progress: Layering Concerns for the Analysis of Credit-Based Shaping in IEEE 802.1 TSN,” in *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, 2020.

[8] D. Nguyen and Z. Nasiri, "Performance analysis of a non-synchronized heterogeneous TSN network," BSc thesis, Mälardalen University, Västerås, Sweden, 2022.

[9] A. Johansson, “Get in sync with TSN: A Study of Partially Synchronized TSN Networks,” BSc thesis, Mälardalen University, Västerås, Sweden, 2022.

[10] K. C. Lee and S. Lee, “Performance evaluation of switched Ethernet for real-time industrial communications,” *Computer Standards & Interfaces*, vol. 24, no. 5, pp. 411–423, 2002.

[11] A. Magnusson and D. Pantzar, “Integrating 5G Components into a TSN Discrete Event Simulation Framework,” M. S. thesis, Mälardalen University, Västerås, Sweden, 2021.

[12] D. B. Mateu, D. Hallmans, M. Ashjaei, A. V. Papadopoulos, J. Proenza and T. Nolte, "Clock Synchronization in Integrated TSN-EtherCAT Networks," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2020.

[13] B. Houtan,” Configuring and Analysing TSN Networks Considering Low-priority Traffic” Doctoral dissertation, Mälardalen University, Västerås, Sweden, 2021.

[14] S. Waldhauser, B. Jaeger and M. Helm, “Time Synchronization in Time-Sensitive Networking,” *Network Architectures and Services,* 2020.

[15] M. Kim, J. Min, D. Hyeon and J. Paek, “TAS Scheduling for Real-Time Forwarding of Emergency Event Traffic in TSN,” in *2020 International Conference on Information and Communication Technology Convergence (ICTC),* 2020.

[16] H. Zhu, K. Liu, Y. Yan, H. Zhang and T. Huang, “Measures to Improve the Accuracy and Reliability of Clock Synchronization in Time-Sensitive Networking,” in *IEEE Access,* vol. 8, pp. 192368- 192378, 2020.

[17] M. Barzegaran, N. Reusch, L. Zhao, S. Craciunas and P. Pop, *Real-Time Guarantees for Critical Traffic in IEEE 802.1Qbv TSN Networks with Unscheduled and Unsynchronized End-Systems*, 2021.

[18] A. Diarra, T. Hogenmueller, A. Zimmermann, A. Grzemba and U. A. Khan,“Improved clock synchronization start-up time for switched ethernet-based in-vehicle networks,” in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA),* 2015.

[19] J. N. Amaral, “About computing science research methodology.” Citeseer, 2011.

[20] J. F. Nunamaker, M. Chen and T. D. Purdin, “Systems development in information systems research, “*Journal of management information systems*, vol. 7, no. 3, pp. 97-101, 1990.

[21] “Raspberry Pi 3 Model B,” raspberrypi.com. https://www.raspberrypi.com/products/raspberry-pi-3-model-b/ (accessed Sep. 20, 2022).

[22] E. Nilsson, “Evaluation of how clock synchronization protocols affect inter-sender synchronization of live continuous multimedia”, M. S. thesis, Umeå University, Umeå, Sweden, 2018.

[23] “1G MTSN – Multiport TSN Switch IP Core,” soc-e.com. https://soc-e.com/mtsn-multiport-tsn-switch-ip-core/ (accessed Sep. 25, 2022).

[24] “Welcome to Scapy’s documentation!,” scapy.readthedocs.io. https://scapy.readthedocs.io/en/latest/index.html (accessed Nov. 19, 2022).

[25] I. Álvarez, A. Servera, J. Proenza, M. Ashjaei and S. Mubeen, "Implementing a First CNC for Scheduling and Configuring TSN Networks," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA),* 2022.

[26] Docker “Docker Desktop,” docs.docker.com. https://docs.docker.com/desktop/ (accessed Nov. 19, 2022).

Appendix A The codes for experiments 1 & 2 (Listener.py and Talker.py)

The codes Lisenter.py and Talker.py were used to send data between the Raspberry Pis in this thesis's first and second experiments. The Talker code sends data to the Listener to update with the new period. Despite using the same codes for both experiments, it has still been adjusted to suit the different scenarios performed. The codes were created by authors in [9] but have been modified to adapt to the devices used in this thesis. To execute the codes on the end-stations, it is necessary to install the *Scapy* package by entering the following command in the terminal: *sudo apt-get install scapy*. Because the *sudo* command is often run as root and is required when running the previous command, enter sudo python3 Listener.py or Talker.py in the terminal to execute the code and the experiment.

```
# The Listener code
# Modified by: Daniel Bujosa Mateu
# Author: Andreas Johansson


from scapy.all import *
from scapy.utils import *
import time, sys, os, socket, threading
import numpy as np

pkt_count=20
slope=0
p = 1.0

#Set up a simple TCP server
server_address = ('192.168.4.10', 1338)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(server_address)
sock.listen(1)

#Function to run TCP server in a separate thread
def data_receiver():
        global p
        connected = False
        while True:
                if connected == False:
                        connection, client_address = sock.accept()
                        print("Client has connected")
                        connected = True
                else:
                        data = connection.recv(1024)
                        p=float(data.decode())
                        print("New period:" + str(p))
##### Set the priority of the script to real-time and to the highest priority
#####
#sudoPrio = os.popen("ps -aux | grep Listener.py | awk '{print $2}' | head -
1").read()
```

```
#prosPrio = os.popen("ps -aux | grep Listener.py | awk '{print $2}' | awk
'NR==2'").read()

#os.system("sudo chrt -p 99 " + str(sudoPrio))
#os.system("sudo chrt -p 99 " + str(prosPrio))
#os.system("sudo renice -20 -p " + str(sudoPrio))
#os.system("sudo renice -20 -p " + str(prosPrio))

#Start data receiver thread
thread_0 = threading.Thread(target = data_receiver)
thread_0.start()

## Setup sniff, filtering for IP traffic
n = pkt_count-1
pkt = sniff(iface="eth0",filter='vlan 1',count=pkt_count)
wrpcap('file.pcap',pkt,append=True,nano=True)
nanopacket_init = rdpcap('file.pcap')
x = [0.0] * pkt_count
y = [0.0] * pkt_count
base=nanopacket_init[0].time
for i in range(1, pkt_count, 1):
        x[i] = i*p
        y[i] = float(nanopacket_init[i].time-base)

try:
        while True:
                pkt = sniff(iface="eth0",filter='vlan 1',count=1)
                wrpcap('file.pcap',pkt,append=True,nano=True)
                wrpcap('file_temp.pcap',pkt,nano=True)
                nanopacket = rdpcap('file_temp.pcap')
                x.pop(0)
                y.pop(0)
                y.append(float(nanopacket[0].time-base))
                rest=y[n]-y[n-1]
                mult=np.around(rest/1.0)
                x.append(x[n-1]+max(p,(p*mult)))
                sum_x=sum(x)
                sum_y=sum(y)
                sum_xy=sum(np.multiply(x,y))
                sum_squared_x=sum(np.multiply(x,x))

                slope = (n*sum_xy - sum_x*sum_y) / (n*sum_squared_x -
sum_x*sum_x) #This is the trendline formula for the slope only
                print("Slope of last 20 packets: ", str(slope))

                if 0.95 <= slope <= 1.05:
                        print("Drift is stable")
                else:
                        print("Sending drift to CNC")
```

```
                file = open('samples.txt', 'a')
                file.write(str(x[n])+";"+str(y[n])+";\n")
                file.close()

except for KeyboardInterrupt:
        print('\033[2DBye.')
        thread_0.join()
```

```
# The Talker code
# Modified by: Daniel Bujosa Mateu
# Author: Andreas Johansson


import time, sys, os, socket, threading
from scapy.all import *
from scapy.utils import *

server_address=('192.168.4.10',1338)

#Traffic period in nsec = x
p=1000000000

##### Set the priority of the script to real-time and to the highest priority
#####
sudoPrio = os.popen("ps -aux | grep Talker.py | awk '{print $2}' | head -
1").read()
prosPrio = os.popen("ps -aux | grep Talker.py | awk '{print $2}' | awk
'NR==2'").read()

os.system("sudo chrt -p 99 " + str(sudoPrio))
os.system("sudo chrt -p 99 " + str(prosPrio))
os.system("sudo renice -20 -p " + str(sudoPrio))
os.system("sudo renice -20 -p " + str(prosPrio))

try:
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error:
        print("Socket failed. Exiting..")
        sys.exit()

print("Socket Created, connecting to " + str(server_address[0]) + " port " +
str(server_address[1]))
try:
        client.connect(server_address)
except ConnectionRefusedError:
        print("Could not connect to server. Exiting..")
```

```
        sys.exit()
print("Connection established.")

#Forge ethernet frame used for periodic sending
frame = (Ether(dst='b8:27:eb:01:a2:b4',src='b8:27:eb:88:ab:96')
        / Dot1Q(prio=7)
        / Dot1Q(vlan=1, prio=7)
        / IP(dst='192.168.4.10',src='192.168.4.30')
        / ICMP())

#Main thread, periodic sending of traffic in x seconds adjusted with slope
value from receiver
q = time.time_ns()
s = time.time_ns()+p
i = 0
try:
        while i<=2022:
                # print("p " + str(p) + " q-k " + str(q+k)) #Used to validate
that period is updating when receiving new slope (k) value from receiver node
                while q < s:
                        q = time.time_ns()
                sendp(frame, iface='eth0')

                time.sleep(p/(2*(1000000000.0**2)))
                i+=1
                if i % 100 == 0:
                        p = p*0.95
                        client.sendall(str(p/1000000000.0).encode())
                s+=p

except KeyboardInterrupt:
        print('\033[2DBye.')
```

Appendix B The codes for experiment 3 (Listener.py and Talker.py)

The codes Lisenter.py and Talker.py defined below were used to send traffic between the Raspberry pies in the third experiment. The Talker sends the perceived clock drift periodically to the listener. The listener samples the TSN messages according to the updated period and sends the drift value to the CNC after been measuring it. To run the third experiment, see the necessary information in Appendix A and cnc.py in Appendix C, which also establishes a connection with the Listener code as a second client.

```
# The Listener code
# Modified by: Daniel Bujosa Mateu
# Author: Andreas Johansson


from scapy.all import *
from scapy.utils import *
import time, sys, os, socket, threading
import numpy as np

pkt_count=20
slope=0
#p = 1.0
p=0.3769

#Set up a simple TCP server
server_address = ('192.168.4.10', 1338)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(server_address)
sock.listen(1)

#Function to run TCP server in separate thread
def data_receiver(connection):
        global p
        while True:
                data = connection.recv(1024)
                p=float(data.decode())
                print("New period:" + str(p))

c = 0
while c < 2:
        connection, client_address = sock.accept()
        print("Client has connected")
        if client_address[0] == '192.168.4.60':
                cnc = connection
                c+=1
        elif client_address[0] == '192.168.4.30':
                thread_0 = threading.Thread(target = data_receiver,
args=(connection,))
                thread_0.start()
                c+=1
```

```
## Setup sniff, filtering for IP traffic
n = pkt_count-1
print("test")
pkt = sniff(iface="eth0",filter='vlan 1',count=pkt_count)
wrpcap('file.pcap',pkt,append=True,nano=True)
nanopacket_init = rdpcap('file.pcap')
x = [0.0] * pkt_count
y = [0.0] * pkt_count
base=nanopacket_init[0].time
for i in range(1, pkt_count, 1):
        x[i] = i*p
        y[i] = float(nanopacket_init[i].time-base)

try:
        i = 0
        e = 0
        while True:
                pkt = sniff(iface="eth0",filter='vlan 1',count=1)
                wrpcap('file.pcap',pkt,append=True,nano=True)
                wrpcap('file_temp.pcap',pkt,nano=True)
                nanopacket = rdpcap('file_temp.pcap')
                x.pop(0)
                y.pop(0)
                y.append(float(nanopacket[0].time-base))
                rest=y[n]-y[n-1]
                mult=np.around(rest/p)
                x.append(x[n-1]+max(p,(p*mult)))

                #sum_x=sum(x)
                #sum_y=sum(y)
                #sum_xy=sum(np.multiply(x,y))
                #sum_squared_x=sum(np.multiply(x,x))
                #slope = (n*sum_xy - sum_x*sum_y) / (n*sum_squared_x -
sum_x*sum_x) #This is the trendline formula for the slope only
                slope = np.mean(np.divide(np.diff(x),np.diff(y)))
                print("Slope of last 20 packets: ", str(slope))

                i += 1
                if 0.97 <= slope <= 1.03 or 0.97 <= (x[1]-x[0])/(y[1]-y[0]) <=
1.03:
                #if 0.95 <= slope <= 1.05:
                        print("Drift is stable")
                elif e + 40 < i:
                        print("Sending drift to CNC")
                        cnc.send(str(slope).encode())
                        e = i

                file = open('samples.txt', 'a')
```

```
                file.write(str(x[n])+";"+str(y[n])+";\n")
                file.close()

except KeyboardInterrupt:
        print('\033[2DBye.')
        thread_0.join()
```

```
# The Talker code
# Modified by: Daniel Bujosa Mateu
# Author: Andreas Johansson

import time, sys, os, socket, threading
from scapy.all import *
from scapy.utils import *

server_address=('192.168.4.10',1338)

#Traffic period in nsec = x
#p=1000000000
p=376900000

##### Set the priority of the script to real-time and to the highest priority
#####
sudoPrio = os.popen("ps -aux | grep Talker.py | awk '{print $2}' | head -
1").read()
prosPrio = os.popen("ps -aux | grep Talker.py | awk '{print $2}' | awk
'NR==2'").read()

os.system("sudo chrt -p 99 " + str(sudoPrio))
os.system("sudo chrt -p 99 " + str(prosPrio))
os.system("sudo renice -20 -p " + str(sudoPrio))
os.system("sudo renice -20 -p " + str(prosPrio))

try:
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error:
        print("Socket failed. Exiting..")
        sys.exit()

print("Socket Created, connecting to " + str(server_address[0]) + " port " +
str(server_address[1]))
try:
        client.connect(server_address)
except ConnectionRefusedError:
        print("Could not connect to server. Exiting..")
        sys.exit()
print("Connection established.")
```

```
#Forge ethernet frame used for periodic sending
frame = (Ether(dst='b8:27:eb:01:a2:b4',src='b8:27:eb:88:ab:96')
        / Dot1Q(prio=7)
        / Dot1Q(vlan=1)
        / IP(dst='192.168.4.10',src='192.168.4.30')
        / ICMP())

#Main thread, periodic sending of traffic in x seconds adjusted with slope
value from receiver
q = time.time_ns()
s = time.time_ns()
i = 0
try:
        while i<=2022:
                # print("p " + str(p) + " q-k " + str(q+k)) #Used to validate
that period is updating when receiving new slope (k) value from receiver node
                while q < s:
                        q = time.time_ns()
                sendp(frame, iface='eth0')

                #time.sleep(p/(2*(1000000000.0**2)))
                i+=1
                if i % 100 == 0:
                      #p = p*0.95
                        p = p*1.05
                        client.sendall(str(p/1000000000.0).encode())
                        #client.sendall(str(0.3769).encode())
                s+=p

except KeyboardInterrupt:
        print('\033[2DBye.')
```

Appendix C The code to the cnc.py

The cnc.py acts as second client to establish a connection with the listener code (in the third experiment) by also using socket communication. The cnc.py receives the perceived clock drift from the listener code. After the cnc.py receives the drift, it will update the period by reconfiguring the config.json in Appendix G to update the switches with the new schedule and then apply it on the TSN network. Lastly, *"cnc.py"* is used to run this client.

```
import time, sys, os, socket, threading, io, subprocess
import numpy as np

server_address_a=('192.168.4.10',1338)
#server_address_b=('192.168.4.20',1338)

sem =threading.Semaphore()

def reconfigure(file_name, line, period):
    lines = open(file_name, 'r').readlines()
    lines[15] = "                    [" + str(int(np.around((period -
30000.0)/10000))) + "0000, 01111111]\n"
    lines[47] = "                    [" + str(int(np.around((period - 30000.0
- 40000.0)/10000))) + "0000, 01111111]\n"

    out = open(file_name, 'w')
    out.writelines(lines)
    out.close()
    #sp = subprocess.call("bash cnc.sh", shell=True)
    #time.sleep(5)
    #sp.terminate()
    #os.system("expect -c \"spawn ./cnc; expect \"Password:\"; send -- \"soc-
e\r\"; interact\"")
    os.system("bash cnc.sh")
    #child = pexpect.spawn('./cnc')
    #child.expect('Password:')
    #child.sendline('soc-e:\r')

def listener_management(server_address,line):
    try:
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error:
        print("Socket failed. Exiting..")
        sys.exit()

    print("Socket Created, connecting to " + str(server_address[0]) + " port "
+ str(server_address[1]))
    try:
        client.connect(server_address)
    except ConnectionRefusedError:
        print("Could not connect to server. Exiting..")
```

```
        sys.exit()

    print("Connection established.")
    period = 376900000.0

    #period = 1000000000.0
    data = client.recv(1024)
    while len(data) > 0:
        drift=float(data.decode())
        print("Drift " + str(server_address[0]) + ":" + data.decode())
        period = period * drift
        sem.acquire()
        reconfigure("../../../shared/config.json", line, period)
        sem.release()
        data = client.recv(1024)

thread_0 = threading.Thread(target = listener_management,
args=(server_address_a,28,))
thread_0.start()
#thread_1 = threading.Thread(target = listener_management,
args=(server_address_b,9,))
#thread_1.start()

try:
    while True:
        time.sleep(10)

except KeyboardInterrupt:
    thread_0.join()
    #thread_1.join()
```

Appendix D Cnc.sh

This line of code is used to automatically enable passwords in the Docker Desktop terminal when running experiment 3. For example, cnc.sh should be moved to the folder *"cd /libs/cnc/build/" to participate in this function.*

```
expect -c "spawn ./cnc; expect \"Password:\"; send -- \"soc-e\r\"; interact;
expect \"Password:\"; send -- \"soc-e\r\"; interact"
```

Appendix E Configuration of the TSN Switches

The following steps were taken to configure the first TSN device:

- The connection from the switch to the PC is done through Ethernet, but to do that, the PC needs to be in the same subnet as the switch, in this case, we use IP address 192.168.4.60 with subnet mask 255.255.255.0, see Figure B-1.
- The PC was connected to port 2 on the switch and the switch was by default, accessible on IP address 192.168.4.68, which can be reached via a web browser by entering the username and password that requires, see Figure B-2.
- Port 1 was enabled in the switch by modifying the tab Advanced Network, see Figure B-3 in the TSN menu; the other ports were left by default unmodified.
- As shown in Figure B-4, the cycle time was set to one second since the experiment used periodic traffic in one-second intervals. The list length is set to two, as seen in the figure, in slot 0, ST traffic which is Q7, is sent with a time interval of 30 000 ns. 30 000 ns was closed by using the trial error method as the exact window size for the ST time window. Other traffic traversing the network is allowed to be sent in Q6-Q0 outside the 30 000 ns interval.

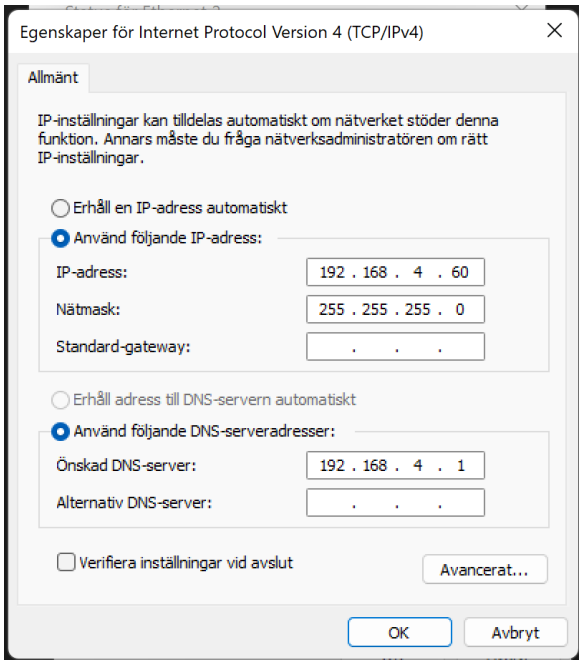

Figure B-1: configuration of the PC

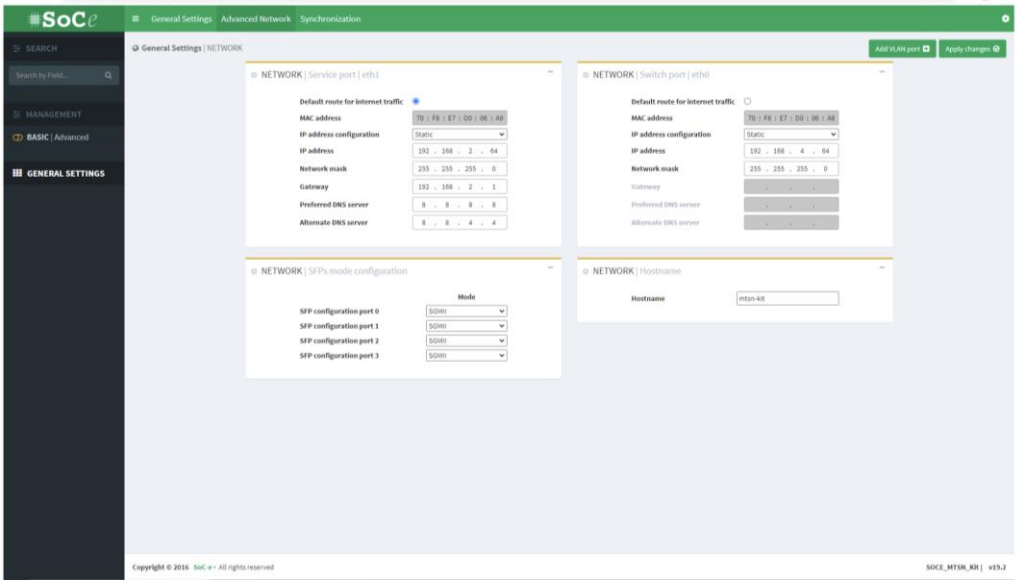

Figure B-2: The web browser of the TSN switch

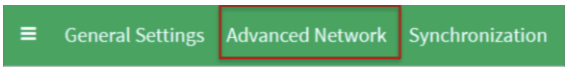

Figure B-3: The TSN menu

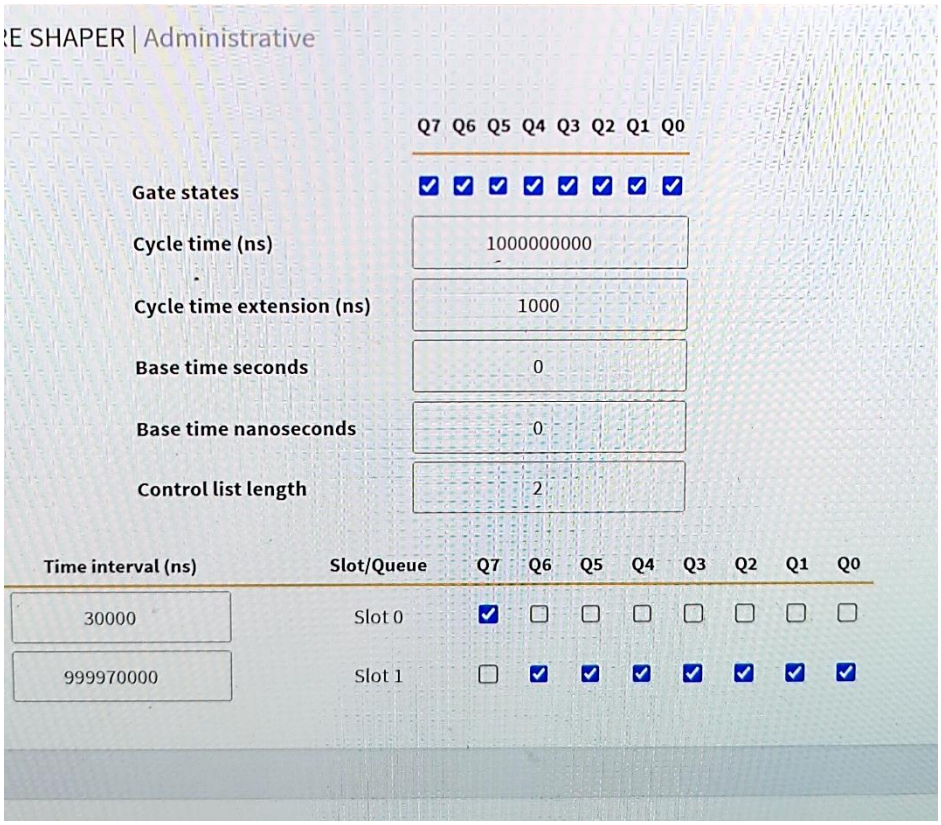

Figure B-4: TAS configuration in the first switch.

The following steps were taken to configure the second TSN device:

- The steps mentioned above are used to configure the second switch except for the IP address 192.168.4.69 with subnet mask 255.255.255.0.
- As shown in the Figure B-5 the cycle time was set to one second since the experiment used periodic traffic in one-second intervals. The list length is set to three, as seen in the figure, in slot 0, the window size 40 000 ns was chosen so the window wouldn't be scheduled in the same position as in slot 1. The traffic is sent between Q6-Q0.
  In slot 1, ST traffic which is Q7 is sent with a time interval of 30 000 ns. 30 000 ns was closed using the trial error method as the exact window size for the ST time window. Lastly, other traffic in slot 2 traversing the network is allowed to be sent in Q6-Q0 outside the 30 000 ns interval.

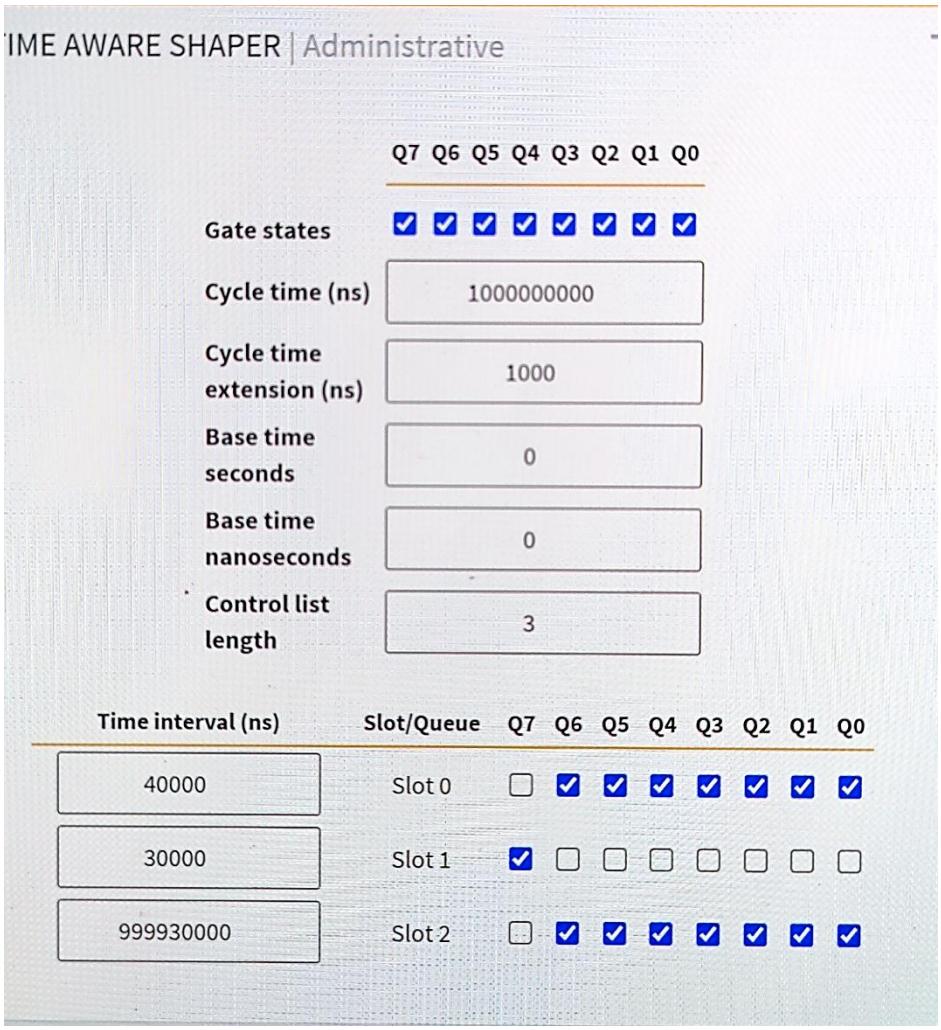

Figure B-5: TAS configuration in the second switch.

Appendix F Setting up Docker Desktop environment

- First step is downloading the Docker Desktop container by clicking on this link https://docs.docker.com/desktop/install/windows-install/ and choosing the version suitable for your operating system as in Figure 4.1. In our case, Docker Desktop for Windows was chosen.

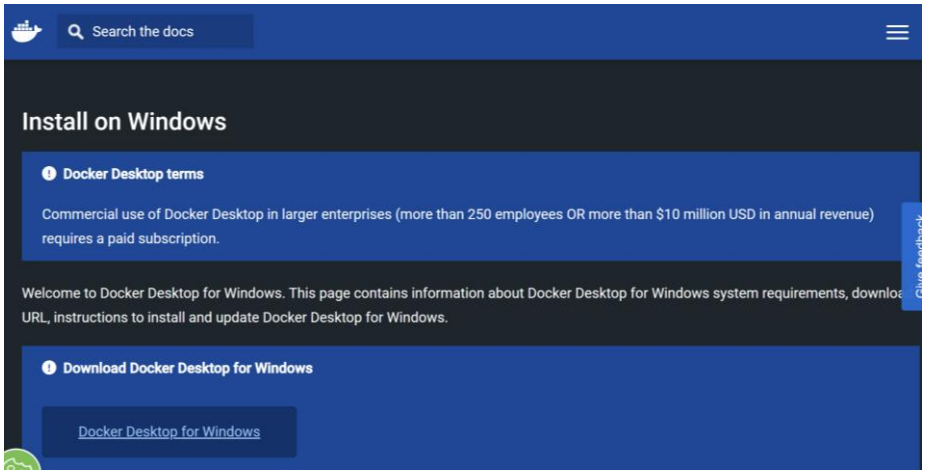

Figure 4.1: Installing Docker Desktop

- After the installation is completed, open the command prompt on your device and follow all the steps that appear in this link. https://learn.microsoft.com/sv-se/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package.

- Download the "cnc-main" file on your device and save it on your computer

- Once this is done, type the commands in your command prompt terminal.

    - *cd "C:\folder\path\to\cnc-main"*
    - *docker build --rm -t aservera/cnc:latest*
    - *cd "C:\folder\path\to\cnc-main\Docker\cnc-docker"*
    - *docker-compose up -d*
    - *docker exec -it cnc ./bin/bash*

- Now that you have access to the Docker Desktop terminal, type these commands

    - Install expect by *sudo apt install expect*
    - Install pip3 by *sudo apt install python3-pip*
    - Install all cnc.py required libraries

- Once all required libraries to run cnc.py are downloaded, navigate to this folder through cd /libs/cnc/build/.
    - In this folder, you should move or copy the cnc.py and cnc.sh.

- When all those steps are completed, you can now use the Docker Desktop as a client to connect with the listener server. Use the command "python3 cnc.py" to run the client, see Figure 4.2.

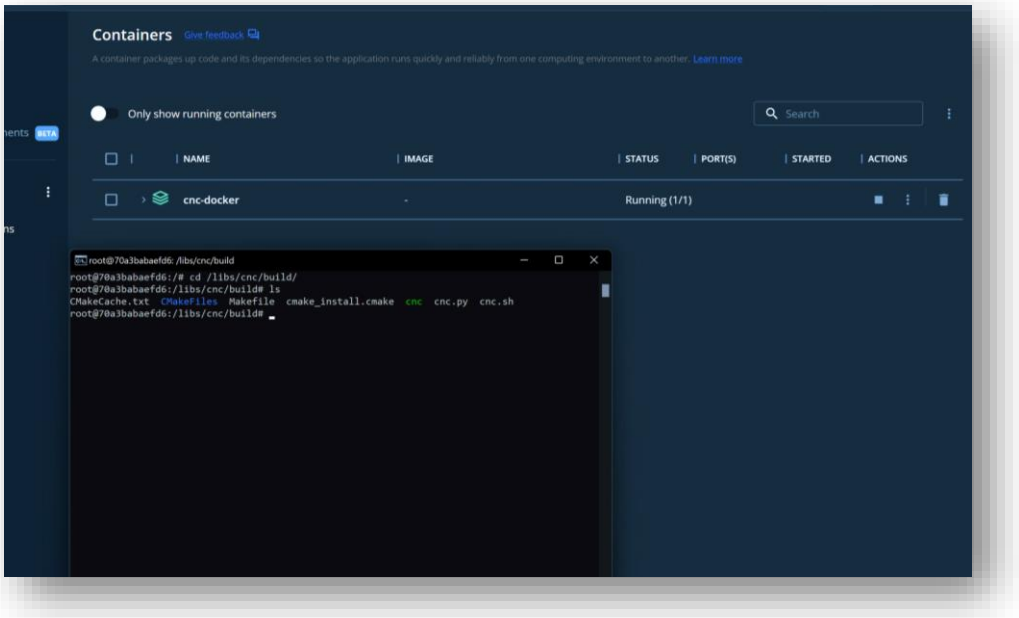

Figure 4.2: The CNS container created in the Docker Desktop

Appendix G Config.json

Config.json file is the switch configuration file containing the TSN switches configuration, ports, and configuration values, as shown below. Config.json files update the switches with the new schedule and then apply it to the TSN network. To use the file, specify the exact port configuration used in the TAS configuration on the TSN switch interface. After running experiments, the config.json file must reconfigure to the default state. Supervisor Daniel Bjuosa Mateu created the codes, but they have been modified by the authors in this thesis to adapt to the devices used in this thesis.

```
Author Daniel Bujosa Mateu
Modified by Balqis Yusuf

[
    {
        "switch": "TEST_68",
        "ip": "192.168.4.68",
        "port_list": [
            {
                "port_number": "PORT_0",
                "values":[
                    [376900000, 01111111]
                ]
            },
            {
                "port_number": "PORT_1",
                "values":[
                    [30000, 10000000],
                    [1000010000, 01111111]
                ]
            },
            {
                "port_number": "PORT_2",
                "values":[
                    [376900000, 01111111]
                ]
            },
            {
                "port_number": "PORT_3",
                "values":[
                    [376900000, 01111111]
                ]
            }
        ]
    },
    {
        "switch": "TEST_69",
        "ip": "192.168.4.69",
        "port_list": [
            {
                "port_number": "PORT_0",
```

```
                "values":[
                    [376900000, 01111111]
                ]
            },
            {
                "port_number": "PORT_1",
                "values":[
                    [40000, 01111111],
                    [30000, 10000000],
                    [999970000, 01111111]
                ]
            },
            {
                "port_number": "PORT_2",
                "values":[
                    [376900000, 01111111]
                ]
            },
            {
                "port_number": "PORT_3",
                "values":[
                    [376900000, 01111111]
                ]
            }
        ]
    }
]
```