Delft University of Technology

Master of Science - Computer Science
Master Thesis

Time-Sensitive Networking IEEE 802.1CB: Security and Reliability

Author:
Adriaan de Vos (4422643)

June 15, 2022

Thesis committee

Thesis Advisor
Prof. Dr Mauro Conti
Full Professor of Cybersecurity
University of Delft, the Netherlands
University of Padua, Italy

Daily Supervisor
Prof. Dr Apostolis Zarras
Assistant Professor of Cybersecurity
University of Delft, the Netherlands

Daily Supervisor
Prof. Dr Koen Langendoen
Full Professor of Embedded and Networked Systems
University of Delft, the Netherlands

External Supervisor
Dr Alessandro Brighente
Postdoc researcher of Cybersecurity
University of Padua, Italy

This research has been conducted in cooperation with the University of Padua, the University of Delft, and the SPRITZ Security and Privacy Research Group.

Preface

From a young age, I was already interested in Cyber Security. Because of my broad interest, it was hard to pinpoint a specific topic to research. Finding a research gap and contributing to science is a different challenge than liking to read blogs and articles. This thesis that lays before you is my accumulated work on the security of Time-Sensitive Networking, specifically IEEE 802.1CB. My first introduction to this topic was during the master's course "Computer and Network Security: Advanced Topics", lectured by Prof. Dr Mauro Conti. This course consists of a good overview of contemporary and exciting topics. It offers students the choice to research and to specialize in one of the available topics.

Personally, I found the "Cyber-Physical Systems" topic the most interesting. This decision brought me in contact with my supervisor Dr Alessandro Brighente. After discussing possible research opportunities, he introduced me to Time-Sensitive Networking. This topic is about an upcoming standard that will be widely used in the near future. Therefore it was an excellent field to add my contribution. With great help from my supervisors, I was able to find a research gap by first studying the literature in search of existing work and solutions. This survey showed that there was not yet a fitting solution to the identified attack vectors. The next step was showing that these attack vectors are exploitable and this was done on a hardware test-bed to show that these vulnerabilities are not just theoretical.

For me, creating the hardware test-bed was the most challenging part of the research. I did have some prior experience with Linux hardware and operating systems. Still, it was a huge learning curve to become knowledgeable at low-level stuff such as compiling, booting, networking, configuring and setting everything up. Luckily, the final version works well and is fully reproducible for other researchers. As a result of this research, we aim to make a publication of the literature study of this research at the ESORICS conference as a workshop paper. If this is accepted, we might follow up by publishing the entire research as a second paper.

I would like to express my gratitude to Dr Alessandro Brighente for his continuing support during my research. In addition to weekly meetings to discuss progress and challenges, he also helped with proofreading my thesis chapters and provided helpful feedback. I am also thankful to my thesis committee members: Prof. Dr Mauro Conti, Prof. Dr Apostolis Zarras, and Prof. Dr Koen Langendoen, for allowing me the opportunity to do this research. Especially Prof. Dr Mauro Conti and Prof. Dr Apostolis Zarras have been of great help during the first stage presentation and green light presentation by asking critical questions about essential parts of the research.

Then, I like to mention the support of Dr Roland Kromes and Dr Chhagan Lal while creating the hardware test-bed at the university. They provided me with a place where I could do my research and leave the hardware running while working from home. Especially Roland provided support as we brainstormed about each other's research topics allowing for inspiring new insights. I would also like to thank the company ASG Nederland for allowing me to lend several Raspberry Pi's that I could use in my hardware test-bed. Finally, I would like to thank MSc Suzanne Maquelin, MSc Peter Elgar, and MSc Sanne van der Steur for their continued personal support during this big project.

Adriaan de Vos
Delft, June 15, 2022

Abstract

The upcoming IEEE 802.1CB standard aims to solve performance and reliability issues in Time-Sensitive Networking (TSN). Mission-critical systems often use these standards for communication in automotive, industrial, and avionic networks. However, researchers did not sufficiently investigate the security risks and possible mitigation solutions to this introduced standard. This limited knowledge is a problem as Cyber-Physical Systems (CPS) are mission-critical and time-sensitive, and any unexpected failure of these systems could endanger lives.

To attain a complete overview of the security risks of IEEE 802.1CB, we use an improved STRIDE model for Cyber-Physical Systems. We then design and create a hardware test-bed for Time-Sensitive Networking to prove the feasibility of the identified security risks. Finally, we implement attacks for the identified security risks in IEEE 802.1CB and analyze their impact by running experiments on the hardware test-bed. The results show that some of the identified security risks significantly impact the network's reliability as we successfully execute a Replay attack and a Denial of Service attack. However, the hardware switches provide only limited functionality of the IEEE 802.1CB specification. Therefore, we could not verify the attacks against all identified security risks.

These results show that this networking standard is not ready to be used in Cyber-Physical Systems as the impact of the identified security risks is too significant. On this basis, we recommend additional research and improvements to the IEEE 802.1CB standard and mitigation solutions.

Keywords

Time-Sensitive Networking (TSN), Network Security, IEEE 802.1CB (FRER), Mission-Critical Applications, Hardware Test-Bed, STRIDE, Scapy

Contents

Chapter 1

Introduction

This chapter gives a brief background on the research topic and an overview of the direction and guidelines for this research. The scope, context, problem statement and research goal are all explained in this chapter. Then, we explain the scientific contributions and finally, we give an outline of the thesis.

Background
This research focuses on an upcoming networking standard that aims to be widely used in the near future, called Time-Sensitive Networking (TSN). This standard will be implemented in modern Cyber-Physical Systems (CPS). These systems cover the edge between the digital and the real world and include industrial applications such as flood defence, smart grids, transportation networks, automotive vehicles, etc. With these systems, insufficient security could cause serious adverse effects as people trust their lives to the correct workings of these systems. In recent years these systems are becoming more connected to the internet and are increasingly often the target of malicious actors [1] [2] [3] [4], so they need to be well secured.

Existing techniques used by mission-critical applications such as automotive, avionics, and industrial networks cannot fulfil the requirements for delivering real-time communication as complexity keeps increasing with the growing interconnectivity of sensors and actuators. In addition, many existing network solutions are incompatible with each other, which complicates the development and deployment of real-time networks. As a result of this problem, the IEEE has published the specification of a new networking standard called "Time-Sensitive Networking" (TSN). This standard extends the Ethernet data link layer to ensure that time- and safety-critical traffic is guaranteed to have an extremely low packet loss rate and a finite, low, and stable end-to-end latency.

One specific standard within TSN is IEEE 802.1CB. This standard focuses on providing increased redundancy to network communications. This standard is also called "Frame Replication and Elimination for Reliability" (FRER), and it is often combined with IEEE 802.1Qca to configure multiple disjoint paths within a network. This technique increases frame delivery reliability by using sequence numbering and sending duplicate frames over disjoint paths within IEEE 802.CB compatible switches and endpoints. These replicated frames will protect against hard and soft errors of the underlying links and nodes. Finally, these duplicate frames need to be detected and eliminated at their destination as the standard defines that only the first arrived frame should propagate further. This functionality, however, introduces a security issue which we will explore further in Section 2.3, as an attacker could modify or delay frames to spoof the sequence numbers, tamper with the payload, or execute Denial of Service (DoS) attacks.

Thus, we arrive at the research question as stated below. Chapter 3.1 contains further explanation and decomposition of the research question.

RQ: What is the impact of the security risks on the IEEE 802.1CB networking standard?

Problem Statement & Relevance
The networking standard for Time-Sensitive Networking is a recently developed technique that will see widespread usage in the coming years as mission-critical systems will implement it. Ideally, using this technique should ensure at least the same security level as the older standards it extends. However, as shown by the identified risks in Section 2.3 and the existing solutions in Section 2.4, there has been insufficient research done into the security of the added functionality and protocols that this new networking standard uses. This lack of knowledge will become a problem if these mission-critical systems decide to use this networking standard and unknowingly

introduce significant security flaws that could impact performance and reliability. Especially these types of systems require good security. In addition, there is often a long implementation phase, and it is expensive to make major changes after being put into production.

Therefore, before the widespread roll-out of this new technique, there should be extensive research into the security risks and the possible solutions to mitigate them. With this research, we aim to identify the possible attack vectors that should be taken into account, and we show that the existing solutions are insufficient to prevent them. Furthermore, we then prove that these security issues are exploitable, and thus additional work is required to ensure secure usage of this networking standard. Finally, it is relevant to think about the security of Time-Sensitive Networking as systems such as automotive and avionics networks will be using it. This lack of security might cause human casualties if the network is unreliable or tampered with.

Thesis Scope and Context
For the context of this research, we will only focus on the security aspects of the IEEE 802.1CB standard, which provides increased redundancy to network communications. The research should identify the attack vectors based on reasonable adversary capabilities. The related work section should consider all published works related to providing mitigation solutions to this problem. Developing practical implementations should prove their vulnerability in a real-world environment to show that these attacks are not just theoretical. Therefore, the research should provide proof of concept implementations of these attacks on a hardware test-bed and do experiments that prove the impact of these attacks. These implemented attacks will show that the system's effects and feasibility are realistic and should show more valuable results than just showing a theoretical attack. Unfortunately, the scope of this research does not allow for testing the proof of concept implementation on any of the existing mitigation solutions as there is no source code available, and it is time-intensive to recreate this research.

Research Goal
The main goal of this research is to investigate and prove the feasibility of possible security risks to the IEEE 802.1CB standard. This research consists of describing the possible attack vectors and writing a proof of concept that shows the feasibility of these attacks. In addition, we perform experiments to measure the impact on the key performance indicators. Finally, this research aims to show the severity of the issue, and we hope that this motivates further researchers to provide a fitting solution.

Contribution
The deliverables for this research are:

- A Master Thesis describing all the steps of the research process.
- Source code that can be used to perform the attacks and test against mitigation solutions.
- Hopefully, the related work chapter is published as a workshop paper at the ESORICS 2022 conference.

Additionally, there might be an additional publication of the thesis outcome if the first paper is accepted.

Thesis Outline
Chapter 2 gives an overview of the current research field and provides any additional knowledge required for the following chapters. It shows information about this networking standard and existing mitigation solutions and identifies possible security risks. Then, chapter 3 provides information about the research question and subquestions and gives an overview of the design phase wherein the requirements of the hardware test-bed are specified. Chapter 4, provides information on how the hardware test-bed is created and configured. In addition, it also describes the implementation of the attacks. Then, chapter 5 provides information on the performance of the attacks and their effects on the key performance indicators. It provides a discussion of the experiments and places the results into context. Finally, chapter 6, contains the conclusion and recommendations for future work.

Chapter 2

Related Work

In this chapter, we start by introducing Time-Sensitive Networking and the IEEE 802.1CB standard. Then it will identify and explain the possible security risks and show all current mitigation solutions in published literature. It then aims to show the effectiveness of these mitigation solutions on the identified security risks. Finally, more knowledge is provided about man in the middle attacks and network topologies as this information was required to determine the requirements within the design phase in chapter 3.

2.1 Time-Sensitive Networking

The IEEE 802.1 Time-Sensitive Networking (TSN) standard extends the IEEE 802.1 Audio Video Bridging (AVB) standards released in 2011. These AVB standards provide some extended features for IEEE 802.1 networks in regards to low-latency traffic flows, bandwidth reservation, and synchronization [5]. While these features improved the feasibility of using it for mission-critical applications, it was not extensive enough to support a wide variety of applications as it only focused on audio and video streams. While modern automotive, avionics, and industrial networks might transfer these kinds of streams, they will also require other types of streams. Moreover, these applications require a wide variety of sensors and actuators that need reliable real-time communication to ensure good operation. For this, there has been quite some development in the last decade by different companies to create their proprietary network solutions [6]. However, these solutions are not compatible with each other. Therefore, IEEE released the TSN standard to ensure that all components of different networks can efficiently work together while achieving the requirements for such mission-critical applications.

For this section, we will take IEEE 802.1Q-2018 as the central reference document and describe the TSN related features released within. This central reference document bundles the latest features and updates once every 3 to 6 years. The standard focuses on providing a deterministic service with the following Key Performance Indicators (KPI) [7]:

- Guaranteed delivery with bounded latency
- Low delay variation (jitter)
- Low packet loss

For guaranteed delivery with bounded latency, the protocol makes sure that some capacity is reserved for specific data streams to prevent congestion throughout the network. The bounded latency provides a guarantee about the worst-case delay for packet delivery. The low delay variation reduces the likelihood that delivered packets arrive in an incorrect order. Furthermore, the low packet loss reduces the likelihood that no message is received at all. Combining these KPIs provides a very reliable and deterministic network that would fit well for mission-critical systems.

The IEEE 802.1Q-2018 standard focuses on delivering wired network communication in a network of switches/bridges and some end devices that are each connected to this network through separate cables. These end devices could be workstations, sensors, actuators, or other devices requiring network communication. The basic functionality of this standard is to provide Quality of Service (QoS) and virtual LANs to an Ethernet network. In addition, this standard contains many optional features, of which only a subset has to be implemented based on the application's requirements for network communication. As each feature focuses on a different improvement, we can divide them up into the following categories according to [7]:

- Timing and Synchronization
- Bounded Low Latency
- Resource Management
- High Reliability

Timing and Synchronization ensures that all components within the network (both the bridge and end-devices) have synchronized clocks. This synchronization is necessary for mission-critical systems such as fully automatic driving as they require a common notion of time for sensor fusion.

Bounded Low latency ensures configuration within the network to reserve capacity for certain types of messages or allow time-critical messages to interrupt nontime-critical messages. This traffic shaping ensures minimal delay for critical messages within the network, which is needed for mission-critical systems to quickly act upon their received sensors.

Resource Management provides algorithms and configuration options to divide the available network bandwidth into reserved streams by establishing and enforcing bandwidth contracts between network components. These reservations ensure a deterministic network where no packet loss due to congestion occurs as each application has a maximum throughput they need to adhere to.

High-Reliability provides methods to improve the reliability of packet delivery within the network by using Quality Of Service (QoS), non-shortest network paths, or redundant packet transmission. As mission-critical systems often have real-time applications, they cannot tolerate delays due to re-transmissions of lost frames.

2.2 IEEE 802.1CB

This section will delve deeper into the High-Reliability category of the TSN standards and specifically into 802.1CB-2017 Frame Replication and Elimination for Reliability (FRER). This standard specifies procedures and protocols for network components that provide identification and replication of packets for redundant transmission and identification/elimination of duplicate packets. However, it does not describe how these disjoint paths should be created and configured. This feature provides an increased probability that a given packet will be delivered. However, it is highly suggested to use it in cooperation with other means to increase the probability of correct delivery further. Research has shown that while this standard does a great job in improving the reliability, there are still some difficult challenges that have to be resolved to increase the reliability of this feature even further [8] [9].

We give a short description of the various functions and explain the inner workings below:

Frame Replication provides the generation of packet sequence numbers for a given stream and encoding it in each packet. This sequence generation function adds an IEEE 802.1CB specific header to provide packet identification. This header allows other network components to detect duplicate packets. After adding the sequence number to a packet, the packet propagates through multiple network paths and, if configured, multiple streams on the same path.

Frame Elimination provides the elimination of duplicate packets. It keeps track of the received sequence numbers and only relays the first packet for each received sequence number. This functionality ensures that no loops or exact duplicates will be relayed and delivered to the next component along the path. After elimination, each network component can replicate the packet again on separate paths if configured.

Latent Error Detection provides a detection mechanism for an unexpected number of packets either due to network failure, invalid network configuration, or an attacker. This detection assumes that the number of discarded packets per sequence number should always stay the same if everything works well. A configurable threshold ensures that there is some leeway for naturally occurring packet loss, which is very rare [10], but should not cause an alarm when this event occurs. However, if it detects that a significantly lower number of packets is received suddenly, it raises an alarm to indicate that a network link has gone down. In contrast, if it detects a significantly higher number of packets, there is a possibility that an attacker is spoofing packets.

Implementation of this IEEE 802.1CB standard can be gradually rolled out within a network as it is backwards compatible with non-supporting systems. Different network configurations provide different guarantees and loss rates depending on their support for this standard and the actual topology [11]. For example, an existing ring topology network with the end devices connected to a ring of switches can already upgrade reliability by only updating the switches. This partial upgrade will ensure that the message will go both clockwise and counterclockwise, resulting in a higher resilience against link failure (hard error). Another example is if we only

update the end devices in this topology. This upgrade will cause the packets to be sent twice through the same route and eliminated at the end device. While this does not protect against link failure, it does protect against soft errors such as a CRC mismatch. Partially upgrading a combination of switches and devices will already result in a much more reliable delivery, even if not all devices support this feature. See Figure 2.1 for a graphical overview of the variations.

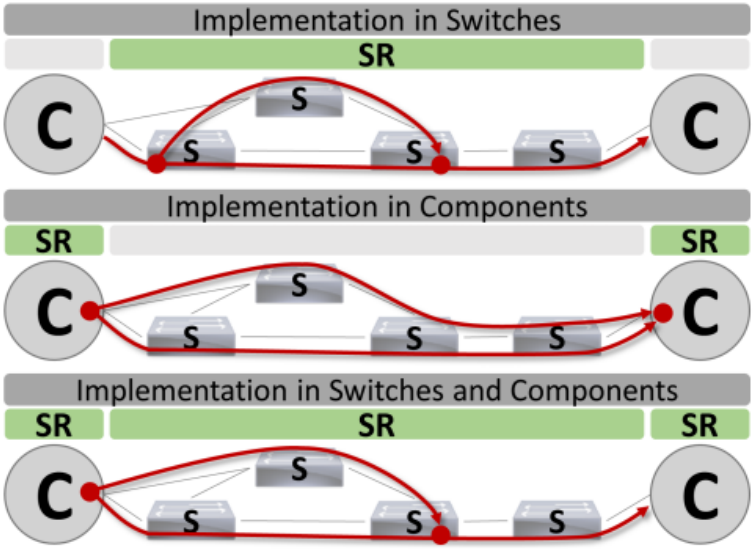

Figure 2.1: Different implementation configurations showing support for seamless redundancy by enabling 802.1CB (from [10])

2.3 Possible Security Risks

The threat modelling framework STRIDE [12] is used to analyse the possible security attacks and effects on IEEE 802.1CB. This framework covers **S**poofing, **T**ampering, **R**epudiation, **I**nformation disclosure, **D**enial of service, and **E**levation of privilege threats against system components. As IEEE 802.1CB has no built-in security, the protection against possible threats is non-existent. For example, there is no mitigation against the misuse of the elimination function and the latent error detection function. Therefore, an adversary can target the network communication to disrupt it. In addition, an incorrect network configuration could also prevent it from working well. An overview of possible attacks against TSN is given by [13]. We describe these threats below:

2.3.1 Sequence Numbering

As the elimination of packets is done based on the sequence number, changes to this could have adverse effects on the reliability. For these identified attacks we will be focusing on the sequence number part of the FRER header as shown in Figure 2.2:

Figure 2.2: IEEE 802.1CB header format (from [14])

If an attacker can intercept packets to modify them, or if the attacker can create new packets within the network, the following attacks can happen:

- The attacker uses **spoofing** to create new packets within the network with existing sequence numbers that arrive earlier than the correct packets. This attack causes the elimination function to drop the original packets resulting in a **denial of service**.
- The attacker uses **spoofing** to create new packets within the network with existing sequence numbers that arrive later than the correct packets. This attack causes the Latent Error Detection function to trigger a warning signal as too many packets are delivered, resulting in the mission-critical system taking unnecessary precaution measures. In addition, if the network has a failure, the attacker can spoof enough packets so that the latent error detection function does not notice this, and it generates no warning signal, creating the illusion that the system is reliable.
- The attacker uses **tampering** by modifying existing packets to have random sequence numbers. This attack causes unexpected packets to drop and delivery of out-of-order packets. This effect will result in a **denial of service**.
- The attacker uses **tampering** by modifying the sequence number of replicate packets. This attack causes the same packet to arrive multiple times at the end destination without a way of detecting it. An adversary can use this to perform a replay attack.

2.3.2 Path Configuration

As some parts of the network are configurable during run-time, protocols exist that enable the configuration of redundant paths and streams that could be abused. While this is not caused by IEEE 802.1CB as it does not provide this configuration, it will affect the performance and reliability, and therefore these attack threats are described below:

- The attacker changes the network configuration to add multiple paths of redundant streams on the same link or multiple redundant streams on different links. This attack causes extra bandwidth usage and possibly higher latency due to this increased computing and throughput that is now required. Additionally, this attack will cause degradation in QoS and could lead to a **denial of service**.
- The attacker changes the network configuration to add intersecting paths. All packets will now go through switches that have received packets with the same sequence number earlier. These packets will be dropped and never delivered to the destination. This attack will result in a **denial of service**.

2.4 Existing Solutions

Some proposals exist that try to resolve the issues shown in the previous section. While not all of these solutions are designed explicitly for TSN, as this is a very recent technology, they are all designed for automotive networks and other related applications. In this section, we will describe how these solutions work and their effects on the KPIs required for TSN.

2.4.1 MACsec - 802.1AE-2018

MACsec is an IEEE standard that works at the medium access control layer. This layer means it works just below the IEEE 802.1CB standard as they both provide functionality to the data link layer. There has been research on the specific application of this standard to automotive Ethernet backbones and their performance and reliability [15]. This solution does not explicitly describe the security improvements of IEEE 802.1CB. However, it does ensure the confidentiality, integrity and authenticity of data within Ethernet frames resulting in mitigation of most attack threats described above. It replaces the existing Ethernet frames and encapsulates them into MACsec-compliant ones. The content is then encrypted and decrypted with symmetric keys by using AES-GCM. This solution depends on IEEE 802.1X for discovering network nodes and configuring and distributing the encryption keys and cryptographic parameters.

Ethernet frames consist of the destination address, source address and user data. These MACsec frames make three modifications to this frame, namely:

- It adds a **SECTag** between the source address and the user data, which provides recognition of the MACsec frame and contains security information such as packet numbering, key length, and replay protection data. This section is 8 to 16 bytes long.

- If the packet requires confidentiality, the user data is optionally encrypted. The length of this section will be equally long as the original section.
- After the user data, it adds a 16 bytes long section for the Integrity Check Value. The **ICV** cover the integrity of the destination and source addresses and the integrity of the user data.

This paper provides detailed descriptions of the actual hardware implementations and some design choices they have made regarding the automotive network environment. They have also included performance tests of their implementation and concluded that their latency is smaller than 350 nanoseconds. This added latency is due to the increased packet size and the required calculations. Finally, they conclude that for a car driving 100 km/h, the physical delay will be less than a millimetre. Therefore this can even be used in safety-related systems such as a braking system.

2.4.2 MACsec - TSN-MIC

Another take at implementing MACsec for time-sensitive networking is called TSN-MIC [16]. This solution differs from other MAC-layer security schemes, such as the 802.1AE solution above, as it only adds checking of the message integrity and no encryption of the payload data. Authors of [16] have first researched the performance of various lightweight cryptography solutions available. They decided on using Chaskey-12 as this is among the fastest algorithms available and is 7 to 15 times faster than AES-CMAC. In addition, this lightweight cryptography is provably secure, patent-free, and provides better key agility than using a key schedule. For the configuration of encryption keys, they have decided on using a modified version of IEC 11770 over IEEE 802.1X. They conclude that their method is more efficient than IEEE 802.1X and more secure than IEC 11770.

This solution also works just below the data link layer, and it would require no changes to the IEEE 801.1CB layer. They have implemented their solution and simulated the network to gain insight into the performance. The absolute added delay would be between 200 and 800 microseconds depending on the Ethernet frame size. This increase would cause a 35% delay to short frames and just a 1% delay to long frames. They conclude that their proposed security schema has a less significant impact on the delay than the frame payload size. Therefore, this will be a feasible solution to time- and mission-critical applications.

2.4.3 Chaos Cipher

The following solution provides a different approach as it implements a physical layer encryption method instead of a medium access control layer. The paper [17] proposes their Chaotic Cipher solution that ciphers the network traffic to hide the complete Ethernet traffic pattern without introducing overhead and throughput loss. For this, they use a stream cipher in combination with symmetric keys that are known to both end devices of a single network link. One big gap in this paper is that they have no recommendation as to how keys should be shared and exchanged.

Their implementation works on the Physical Coding Sublayer (PCS) by directly encrypting the 8b10b symbol flow. This method provides physical layer encryption and obfuscates the traffic pattern as the control symbols such as start/end are also obfuscated. This layer consists of 256 data symbols and 12 control symbols, containing 268 possible symbols. It uses a symmetric key to generate a mapping of the original symbols to the ciphered symbols. This mapping can easily be reversed if the symmetric key is known. The keystream generation is based on the chaotic map method called Skew Tent Map (STM), which provides chaoticity and no periodic windows.

Finally, they conclude with a performance comparison related to other physical layer solutions. They show that their solution has the highest encryption throughput compared to other algorithms. Moreover, this is sufficient to support a Gigabit Ethernet connection without introducing additional delays.

2.4.4 KD & SC

The following solution proposes an application layer Key Distribution and Secure Communications module in [18]. This solution cannot prevent the security issues of IEEE 802.1CB as it works on a higher layer. However, it can detect possible attacks and encrypt the data so that attackers can not eavesdrop.

The Key Distribution Module works as a gateway during the start-up phase of the system by distributing

the asymmetric keys to all legitimate end devices. The gateway has a database of identities and keys for each end device used to exchange keys securely. Each end device has a hard-coded asymmetric key used only for this key exchange. This method ensures that an eavesdropping attacker cannot gain information about the encryption keys used for the subsequent communications.

Each supported end device should implement the Secure Communication Module, and it should provide the ability to encrypt, decrypt, and authenticate messages. For this, it uses DES and HMAC-MD5. In addition, it uses a sequence number to prevent replay attacks.

Finally, the paper provides a real-time performance evaluation of their proposed solution for both the key distribution and the impact on secure communication. This start-up delay is negligible because the key distribution is only done once on boot. However, the communication response time increases by 2 to 6 milliseconds depending on the CPU clock rate.

2.5 Evaluation of attacks and solutions

Table 2.1 shows a comparison of the different existing solutions in regards to the TSN KPIs and the discussed attack threats. Each attack scenario from Section 2.3 is shown with an indication if it can be prevented, detected, or if it is unaffected by the proposed solutions. As some solutions can prevent attacks from happening, they can improve the packet loss due to mitigating these attacks.

	802.1AE-2018 [15]	TSN-MIC [16]	Chaos Cipher [17]	KD & SC [18]
Latency	<350ns	<800μs	0	<6ms
Jitter	-	-	-	-
Packet Loss	improve	improve	improve	-
Spoof DoS	prevent	prevent	prevent	unaffected
Spoof Error	prevent	prevent	prevent	unaffected
Tamper RND	prevent	prevent	prevent	detect
Tamper Replay	prevent	prevent	prevent	detect
Duplicate Paths	unaffected	unaffected	unaffected	unaffected
Intersect Paths	unaffected	unaffected	unaffected	unaffected

Table 2.1: Comparison of effectiveness for all reviewed solutions.

All solutions did not provide information about the Delay Variation (jitter) KPI. However, we think including these measurements is essential to ensure the correct order of packet delivery within mission-critical systems.

2.6 STRIDE Methodology

We use the STRIDE model to identify the security threats to the IEEE802.1CB standard. The STRIDE security threat model was published in 1999 by Microsoft [19], and it conducts a proactive security analysis by identifying various types of threats during the design phase. As our research focuses on the official IEEE 802.1CB specification and not on any manufacturer-specific implementation, this framework is a good match. However, as the original STRIDE model is quite outdated, we use an updated version that is designed explicitly for Cyber-Physical Systems and was released in 2017 [20].

STRIDE consists of a systematic approach for analyzing threats against system components. It provides a clear understanding of the impact of each threat and helps ensure system security. The name of this method, STRIDE, represents an acronym for the six different categories of threats, namely:

- **S**poofing;
- **T**ampering;
- **R**epudiation;
- **I**nformation Disclosure;

- **D**enial of Service (DoS);
- **E**levation of Privilege.

We describe the complete analysis of the IEEE 802.1CB standard below:

Decompose System into Components
The first step is to decompose the system into components. For our analysis we use the most simplistic scenario available. The system contains a source component, a destination component, and multiple communication lines between them as shown in Figure 2.3.

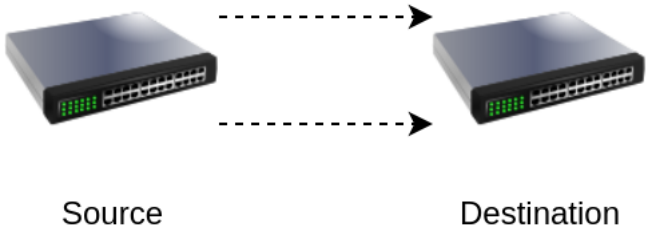

Figure 2.3: Components used for the STRIDE analysis

Plot DFD for System Components
The next step consists of creating a Data Flow Diagram (DFD) for the system components. As described in Section 2.2 the networking standard consists of three parts: the Frame Replication, the Frame Elimination, and the Latent Error Function. For this analysis, we define the Frame Replication function in the source component, and the Frame Elimination function and the Latent Error Function in the destination component.

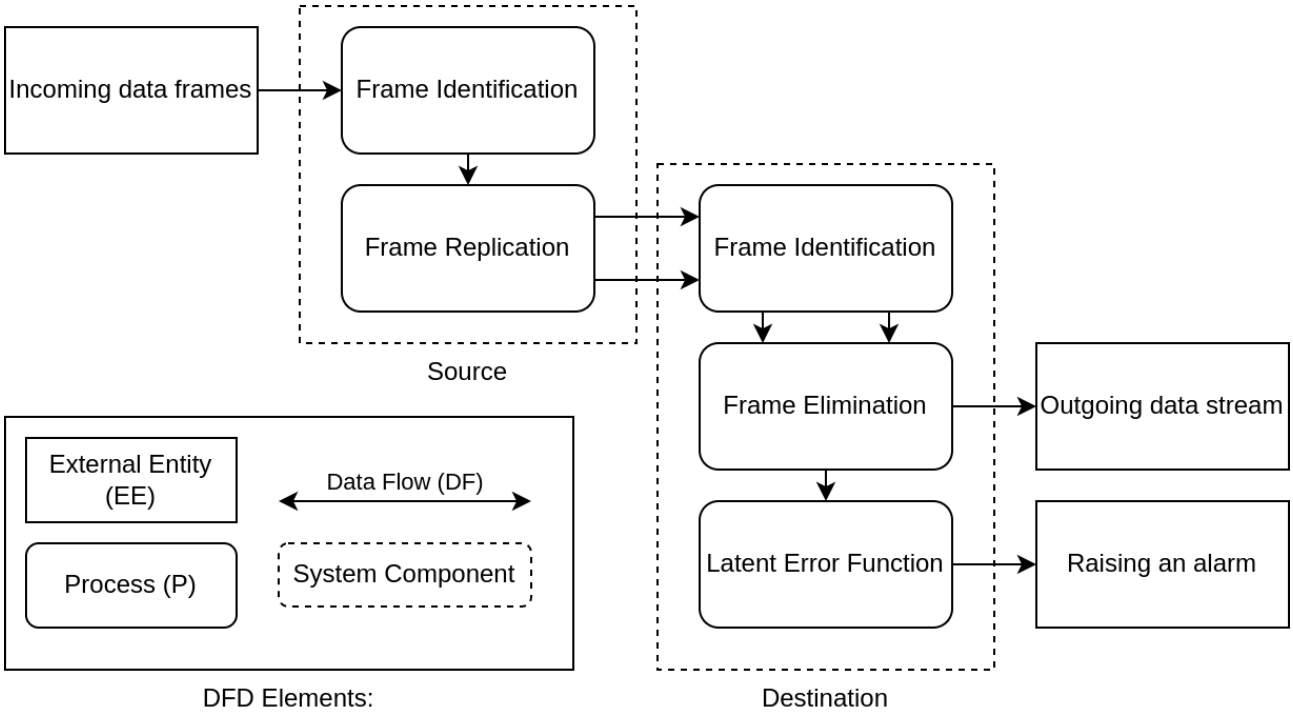

Figure 2.4: Data Flow Diagram used for the STRIDE analysis

Analyze Threats in the DFD
This step is performed by selecting a STRIDE Threat and a DFD Element and then identifying all possible

threats. We summarize the outcome of this STRIDE-per-element approach in Table 2.2 and briefly explain it in the following.

1) Spoofing: For this research, we define spoofing as the ability to introduce new information within the system. An adversary can abuse this threat by creating new frames between the source and destination system components. They could also trick the frame identification methods to incorrectly identify the new frames as legitimate. As a result of these threats, the frame elimination function can decide only to relay the first arriving spoofed packet instead of the legitimate packets, resulting in a spoofed outgoing data stream. If the adversary decides to create frames that always seem to arrive late, they could trigger the latent error function to raise an alarm.

2) Tampering: For this research, we define tampering as the ability to modify existing information within the system without authorization. An adversary can abuse this threat by modifying existing frames between the source and destination system components. They could also modify frames to be incorrect, thus causing the latent error function to raise an alarm. In addition, if the adversary can tamper with the frame replication function, they could introduce additional or duplicate paths, causing an increase in bandwidth usage. Along the same line, the adversary could also introduce intersecting paths in a more complex system, which will be an even bigger problem.

3) Repudiation: We define this threat as the denial of a specific action executed within the system. This threat would be helpful for an adversary if they perform a malicious operation and want to deny their involvement with this. However, this threat is not relevant to this system as the latent error function raises alarms independently of who is causing the issue.

4) Information Disclosure: We define this threat as unintentionally revealing data to unauthorized users. This threat would only be the case within this system if the adversary can tamper with the frame replication function to create a new path that routes the frames to an unauthorized user.

5) Denial of Service: We define this as a disruption of the service for legitimate users. The adversary can abuse this threat in several ways; one example is that the adversary fast forwards the sequence number to the maximum allowed value so the frame elimination function's sequence number counter will speed up. If the adversary stops the attack, the frame elimination function will drop all legitimate packets until the frame sequence numbers reach the allowed sequence number range again. Another way that requires a more active attitude of the attacker is to make sure he always sends his packets with a higher sequence number than the original packets. This attack is similar to the Spoofing threat as it ensures that only the adversary packets will arrive and that legitimate packets are dropped.

6) Elevation of Privilege: As this system does not make a distinction between users and privileges, this threat is not an issue.

Element	**S**	**T**	**R**	**I**	**D**	**E**
Frame Identification	X					
Frame Replication		X		X		
Frame Elimination	X	X			X	
Latent Error Function	X	X			X	

Table 2.2: Summary of the analyzed threats in the data flow diagram.

Identify Vulnerabilities

Section 2.3 described the identified vulnerabilities based on this STRIDE analysis. As shown by Table 2.2, the most vulnerable components are the Frame Elimination function and the Latent Error Function. Therefore, this research focuses on these components as they are purely based on the inner workings of the specification. In comparison, the frame identification and frame replication components are most vulnerable to malicious configurations, and those are not configurable by IEEE 802.1CB, but by IEEE 802.1Qca.

2.7 Man In The Middle Attacks

This section gives a short overview of man in the middle attacks and the types of attacks considered for this research. Based on existing research [21], there exist four different types of man in the middle attacks. Spoofing-based MITM attacks, SSL/TLS MITM attacks, BGP MITM attacks, and false base station MITM attacks. As this research focuses on the IEEE 802.1CB standard, which works on the link layer of the OSI model, we explain the spoofing-based MITM attacks further and ignore the other type of attacks as they are

not relevant. The adversary can physically be present between the source and destination devices, or it can perform a spoofing-based attack. We describe these different attacks below:

- **Link-Based attack:** The attacker physically places his device on a network cable between two devices. The attacker executes this attack by disconnecting the cable at one end, connecting it to the attacker's device, and placing a new cable between the disconnected device and the attacker. This way, the attacker's device acts as a bridge where he can modify traffic. Another option is to wiretap the cable, and there exist several methods for this based on the physical medium used.
- **Host-Based attack:** The attacker has access to one of the existing hosts that lie between the source and destination device. As traffic flows through this device, the attack can read and modify it.
- **ARP Spoofing:** The attacker modifies the ARP cache table of the source and destination device by sending malicious ARP responses to these devices. This method does not require the attacker to be physically between the devices. Instead, they are now voluntarily routing their traffic to the attacker where he can tamper with it and relay it to the correct device.
- **DNS Spoofing:** The attacker executes a DNS cache poising attack to modify the resolving of hostnames to his device. This method is not applicable for this research as it does not use hostnames.
- **DHCP Spoofing:** The attacker runs a DoS attack on the legitimate DHCP server and then acts as the legitimate DHCP server. This method is not applicable for this research as it uses static network configurations.
- **IP Spoofing:** The attacker spoofs network packets by replacing his own IP with the destination or source IP address. This method works if he is knowledgeable or can guess the correct sequence numbers being used.

In addition to these attacks, Man On The Side Attacks also allows an attacker to read traffic and insert messages but not to tamper or drop packets. While this was a serious option we considered, we opted to use a full man in the middle attack as this allows exploiting most of the identified vulnerabilities.

For this research, we use the Host-Based man in the middle attack as it is the most reliable form of MITM attacks and is easy to implement.

Adversary Models

In addition to choosing the type of attack used for this research, there also exist different adversary models that describe the attacker's abilities. Some of these adversaries have only limited access to the network communication of systems. In contrast, other models have full access to everything that happens within the network and even the processing and memory of all the devices [22]. We will not go too much into detail here as that paper describes it very well and extensively. However, for our research, we decided to use the first formalized adversary model called "Dolev-Yao" model [23]. This model is one of the most well-known adversary models as it is essentially acting as a man in the middle attacker. This adversary can listen to all the traffic within the network, and it can freely tamper with existing packets or spoof new packets. These abilities mean that it has complete control of the communication network.

We also consider the Bellara-Rogaway [24] adversary as it allows for the modelling of adversaries with varying levels of power. Depending on the specified capabilities, it can be even stronger than the Dolev-Yao model as it allows corrupting and revealing the devices' internal state. We feel that these additional capabilities are not necessary to exploit the identified vulnerabilities. Therefore, we decide to use the Dolev-Yao model as it is widely known and straightforward for most researchers to understand and get up to speed.

2.8 Network Topology

There is limited information available in the literature about the network topologies used for Cyber-Physical Systems and specifically automotive networks. Research shows that there are significant differences between a traditional Ethernet network and an Automotive Ethernet network [25]. The most significant difference is that there is often one central gateway that combines different technologies' networks, such as CAN/FlexRay. The introduction of Time-Sensitive Networking could solve this combination of different bus networks. However, migrating all these different networking systems in a car will take significant effort and time. Also, there is much more of a hierarchical structure apparent in automotive networks, which means that some devices relay a

lot of information from other devices and could therefore be a good point of attack for an adversary. Based on a performance study of in-car switched Ethernet networks [26] we see that different topologies are being used such as:

- **Bus Network:** This topology connects multiple components to the same single cable and thus enables all components to hear and talk to each other. This topology is unreliable as it depends on a single link for multiple communication paths, and it is not secure as all components can hear everything. However, it is cheap to implement and widely used within CAN networks.
- **Ring Network:** This topology connects components in a ring-like structure, with some components able to reach other networks outside the ring. This topology provides a loop, meaning that the data can travel either way and still reach the rest of the network. This loop ensures the network's reliability, for if a host or link goes down, all data can still be sent and received.
- **Star Network:** This topology connects multiple components to a single component which is responsible for processing or relaying the information from these other components. While this is less reliable, there is an increase in security as all these individual components cannot read or tamper with the messages.

We use a Ring Network for this research as it allows the IEEE 802.1CB standard to have separate disjoint paths. Herein lies the strength of the standard, and we use a minimalistic version of this to keep it affordable and still ensure improved reliability.

Chapter 3

Design

3.1 Methodology

This section describes the approach used in this research to answer the main research question. We divide the main research question into several sub research questions relating to several aspects of the research. Combining the outcome of these questions enables us to answer the main research question.

First, we review existing literature to analyze possible threat vectors to this standard and provide a good overview of the current research field and its mitigation solutions. Then, we create a proof of concept to show that we can exploit these threat vectors. Finally, the effects of the proof of concept attack on the system should be measured. By combining the answers to these sub research questions, it should be possible to give a detailed analysis of the impact of all the security risks of IEEE 802.1CB.

RQ1.1: What threat vectors exist that might compromise the performance and reliability of IEEE 802.1CB
First of all, we aim to discover the possible security risks of this standard. We can answer the main research question quickly if we find no security risks. However, if we find any risks, they should be correctly described and identified.

To answer this sub research question, we study existing literature to get a complete grasp of the networking standard and be knowledgeable of the current research related to this topic. In addition, we study the inner workings of the IEEE 802.1CB standard to make sure we know how it works precisely. Combining this knowledge allows us to identify possible threat vectors that could impact any key performance indicators of Time-Sensitive Networking using the STRIDE method.

RQ1.2: Can we make a proof of concept to exploit these threat vectors?
Secondly, after identifying the threat vectors, we want to show that they are exploitable. For this research, we do this on a hardware test-bed instead of a software simulation. Doing this on actual hardware gives insight into the practicality of the attack. It proves that it is not just a theoretical attack but can affect real-world performance.

For this, we design and configure a hardware system that simulates a TSN supported network on which we can perform our attacks. This network should provide a minimal but realistic implementation that allows for running the proof of concepts and experiments. Making a minimalist network ensures that the costs are affordable and simple to replicate while allowing a real-world experiment. Finally, for each threat vector, an attack should be implemented to see if they work.

RQ1.3: What are the effects of these attacks on the system?
Finally, after implementing the proof of concepts, we run experiments to measure if they are successful and what their effects and impact are.

For this, we design and implement experiments that determine the amount of impact on the key performance indicators of the networking standard, such as packet loss and packet delay. In addition, we support the tuning of

configurable values and parameters within the network to get a better insight into the resilience or vulnerability of the system.

RQ: What is the impact of the security risks on the IEEE 802.1CB networking standard?
The results of all the subquestions should contain enough information to give a complete overview of the possible security risks and their impact on the system. In addition, the results should show precisely the impact on the key performance indicators when using this networking standard within a mission-critical system.

3.2 Hardware

During the design phase of the research, the main focus lies in purchasing the required hardware to do the experiments and implementation. This section describes the requirements to perform this research. It also contains information about the vendors and products that have been contacted and looked into. Then, it gives a product description of all the hardware components. Finally, it extensively describes the hardware test-bed used for this research.

Hardware Requirements
For this research, we specifically need switches instead of routers. There is a clear distinction between them as routers are used to connect multiple networks, often have support for Border Gateway Protocol (BGP), and include a strict firewall. This research needs a switch as we connect multiple network devices to create a network. More specifically, these switches should support Time-Sensitive Networking and especially include support for the IEEE 802.1CB standard. However, this requirement is quite rare as it is an upcoming standard and no consumer-grade devices exist.

Each of the two switches should have at least 3 RJ45/Ethernet ports to allow one incoming/outgoing connection and to allow the creation of two disjoint paths between the switches. Two types of devices fit this requirement. Either the device is readily delivered with these ports, or they support inserting SPF modules, thus supporting different physical mediums. While it allows for more flexibility, it comes with a higher price tag.

In addition, the device should have sufficient documentation and software that support configuring the TSN configuration, the networking paths and the IEEE 802.1CB parameters. Ideally, the devices allow direct access to the operating system so we can fine-tune everything we need.

Finally, these switches should be affordable within the budget of € 2.000, and they should be able to deliver quickly. Unfortunately, it turns out that these last two requirements are the hardest to meet as most devices are very expensive or have problems due to difficulties with the global supply chain.

Hardware Vendors
Searching for hardware that fits these requirements was done with the following search query in google: `("TSN" OR "Time-Sensitive Networking") AND "switch"`. The first few pages contain mixed results of general information about Time-Sensitive Networking and several vendors that produce these devices. Because these vendors are business-oriented and not consumer-facing, they often do not show the price tag on their website. Therefore, we requested quotes from all these companies. In addition, we had to do pricing negotiations and discuss the exact specifications of the devices. Because TSN consists of various sub-standards, for each hardware device, we had to check the support for IEEE 802.1CB specifically.

Table 3.1 provides information about the vendors that are found during the search for fitting TSN products with (partial) support for IEEE 802.1CB:

Based on the aggregated information from Table 3.1, the best fitting solution is the Layerscape LS1028A Reference Design Board from NXP, also called LS1028ARDB. This device supports IEEE 802.1CB, is the cheapest solution, and is readily available to be delivered. Phoenix and Innoroute indicate that they are working on supporting the IEEE 802.1CB standard in the near future.

Interestingly, all companies are based in western countries as no eastern companies showed up in the Google search results. Of course, this does not guarantee that the west is the only provider of these products. However, it shows either a language barrier in finding suitable products or that eastern countries are not actively

Product	Company	IEEE 802.1CB Support	Price	Delivery Time
Layerscape LS1028A Reference Design Board	NXP	✓	€1.714	1 day
RELY-TSN-BRIDGE	Relyum	✓	€2.500	2 weeks
SMARTmpsoc Brick	Soc-E	✓	€3.000	2 weeks
Time Sensitive Networking Evaluation Kit	Fraumhofer IPMS	✓	€5.000	4 weeks
FL Switch TSN 2316	Phoenix	×	-	2 months
RealTimeHat	Innoroute	×	€1.100	6 months

Table 3.1: Comparison of available hardware vendors.

researching Time-Sensitive Networking.

Hardware Products
In addition to the NXP LS1028ARDB, we require additional hardware devices for running the experiments, such as the Raspberry Pi 4B and the TP-Link UE300. Section 3.3 will give an extensive overview of how these components interact. Here, we describe the capabilities of each of the devices.

NXP LS1028ARDB
The NXP Layerscape LS1028A Reference Design Board is an evaluation and development platform that supports industrial networking and Real-Time applications. It is a system on a chip (SoC), and it provides two Ethernet switches with 5 Ethernet ports in total. Each Ethernet switch runs a different hardware IP (Felix / ENETC), and only the Felix ports have support for IEEE 802.1CB and other TSN standards. Internally, there is some switching and configuration required to make the ports of all the switches compatible with each other.

In addition, the LS1028ARDB is delivered with a pre-installed Open Industrial Linux distribution on its internal flash memory. This operating system would be a good solution for some research goals, but for researching IEEE 802.1CB, we require the latest version as the standard is unsupported in the base image. The hardware provides an SDHC port in which we can place an SD card for our operating system. This functionality does, however, require that it has to be compiled and formatted according to the hardware specifications.

Finally, serial ports are available to manage and configure the hardware device. These ports allow the researcher to see the full boot logs, make changes in the bios, configure the network connections and enable SSH access. We did not specify this feature in our requirements, but it helped considerably by providing more insight into how the device works.

Raspberry Pi 4B
The Raspberry Pi 4B is also a system on a chip, but it is more customer-focused than business-oriented. It has a tiny form factor, the size of a hand, and it can be used for various purposes such as a desktop computer, smart home hub, media centre, factory controller and much more. It provides only a single Ethernet port, and it does not have support for TSN and IEEE 802.1CB. However, we need multiple Ethernet ports for this device so it can act as a bridge (more information in Section 3.3). Therefore we use USB to Ethernet dongles in one or multiple USB ports to extend the functionality of this system. In addition, this device provides WiFi support in both connecting to a network and hosting a WiFi hotspot. Finally, the device does not provide a pre-installed operating system. This operating system has to be downloaded and placed on a microSD card by the researcher.

TP-Link UE300
This component is an essential part of the hardware test-bed as the Raspberry Pi 4B's only have a single Ethernet port while the test setup requires 2 or 3 ports per device. The TP-Link UE300 is an off-the-shelf plug-and-play USB device that provides an additional Ethernet port. No custom drivers are required as modern Linux distributions fully support the device.

3.3 Hardware Test-bed Overview

This section gives a textual and graphical overview of the hardware components of the test-bed and how they interact with each other. We show a graphical overview in Figure 3.1. The system consists of 2x LS1028ARDB, shown in green, and 3x Raspberry Pi 4B, shown in red. In addition, multiple TP-Link UE300 are being used

to extend the number of available Ethernet ports in the Raspberry Pi.

The left-most LS1028ARDB is named the Source-Switch, as this component will start the IEEE 802.1CB process by identifying and replicating the packet streams onto separate paths. The right-most LS1028ARDB is named the Destination-Switch as this component will perform the identification and elimination of duplicate packet streams. It should also keep track of counters for the Latent Error function.

The top-most Raspberry Pi purpose is where the adversary sits in this hardware test-bed. This device causes the top path to be compromised, and this could, in practice, happen if any link or node along the path is in the hands of the attacker. Every packet that travels through this path can be read, modified, and dropped according to the attacker's will. This device will thus be the location where we implement the proof of concept attacks. Then, the middle Raspberry Pi acts simply as a bridge between the two switches. It can optionally introduce a configurable amount of artificial delay to ensure that the adversary is present on the quickest path between the switches. Finally, the bottom-most Raspberry Pi acts as our experiment and evaluating device. This device is responsible for initiating and sending the original packet stream. It is also the device that receives the packet stream after the Destination-Switch executes the IEEE 802.1CB elimination function. This device acts both as a source and a destination endpoint and is, therefore, able to accurately monitor each packet's timing differences and possible content differences.

Finally, the arrows in the diagram show the Ethernet cables between the components. The direction of the arrow shows how the packet stream moves through the system during our experiments.

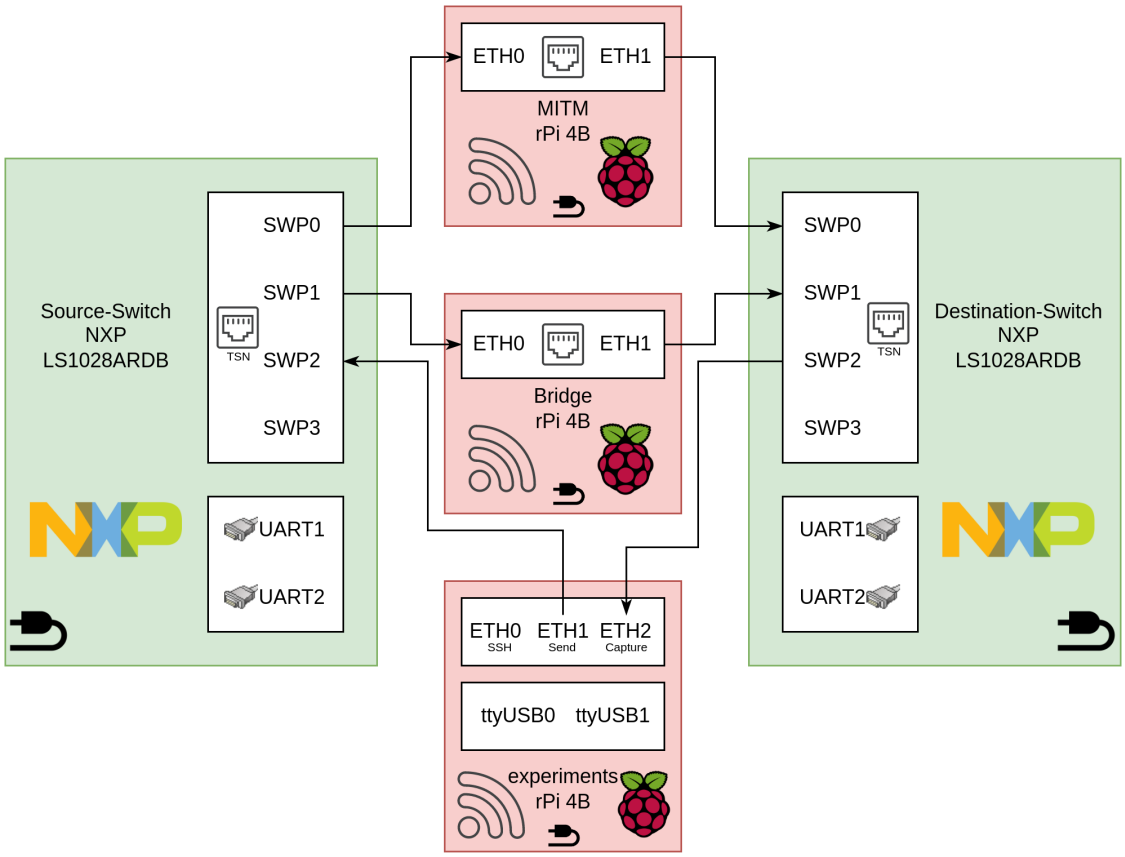

Figure 3.1: Overview of the test-bed setup

3.4 Software

In addition to the hardware overview, this section describes the software components that are part of the hardware test-bed and how they work together. Finally, we briefly explain available options and explain why we made certain decisions.

Operating System
Initially, the aim was to use Open Industrial Linux as this is a popular open-source project that includes support for both Time-Sensitive Networking and support for the NXP LS1028ARDB. In addition, the device is delivered with this operating system pre-installed, and there is quite some documentation and presentations that reference this operating system. However, we found several challenges during the implementation phase, and we describe them below.

- The pre-installed Open Industrial Linux version is old and does not support the newest Time-Sensitive Networking standards required for this research.
- The operating system and open-source project are deprecated in favour of a newer version called Real-Time Edge Software, so it has not received an update in over a year.

We lost some valuable research time by using this operating system as a starting point as its documentation did not show that it is a deprecated project. More details on this will be given in Section 4. After contacting the original developers, they recommend using their new project called Real-Time Edge Software. This operating system uses an updated build system and contains support for the latest standards.

Real-time Edge Software is an evolved version of Open Industrial Linux (OpenIL) for real-time and deterministic systems. The key technology components include Real-time Networking, Real-time System, and support for a wide range of protocols:

- The Real-time Networking functionality includes TSN technology, TSN standards, and related management and configuration applications. This functionality is partially implemented as drivers and support in the kernel, and some userspace applications are provided to support a wide range of networking and redundancy features.
- The Real-time System features include PREEMPT_RT Linux, Jailhouse, and U-Boot based BareMetal framework. The U-Boot functionality is used for this research and configured as described in Section 4.2.

Network Configuration
Initially, the idea was to fully automate the network configuration based on NETCONF and YANG. This is a network configuration protocol (NETCONF) with a recommended modelling language that describes the configuration changes (YANG). The LS1028ARDB switches, by default, enable this functionality and listen on a pre-defined port without a password for any incoming configuration changes. This functionality is not enabled by default for the Raspberry Pi's, but it can be installed and configured if needed. The big advantage of using this method is that it scales well for big networks as it does not require executing manual commands or making file changes on all connected devices. However, while this solution sounds promising, several issues prevent us from using this:

- The YANG formatting for IEEE 802.1CB has not been finalized yet, and currently, only an IEEE draft exists. While we can use this, it could hinder the reproducibility of the research if the formatting of the final IEEE specification significantly differs.
- NETCONF requires all network devices to be reachable by their IP address. There is no DHCP server within the hardware test-bed. This means that there is no automated way for each device to retrieve its basic network configuration and become reachable to other devices.

Due to these issues, we decided to use a manual configuration method. This method does not introduce problems with reproducibility as it is well documented in this thesis document and on the open-source repository from this research. It does, however, introduce issues related to scalability as it is not feasible in large networks to prepare each device manually.

As our hardware test-bed requires manual interaction to configure the basic network configuration, we decided to also do the advanced network configuration and Time-Sensitive Networking configuration in the same

way. We put extra attention into making it easy to reproduce, so for most devices, the manual configuration consists of only a single file that the research has to put on the device. We provide an extensive description for this in Section 4.2.

Endpoints
Initially, the idea was to have two endpoint devices that are used to create and receive the packet streams. However, this solution is sub-optimal as it is easier to compare the performance and content of messages if a single device creates and receives the packet streams. This decision requires some changes compared to the initial design presented in the first stage presentation. In addition to having more accurate results, the cost of the hardware test-bed is also lower as it requires fewer devices for the experiment. It uses a stream of UDP packets for running the experiments as this is one-directional traffic that is also widely used in Time-Sensitive applications. The experiments should monitor the quality of the connection according to the KPIs specified in Section 2.1.

Artificial Delay
To ensure that the adversary has enough time to analyze the incoming packets and possibly modify them, we introduce an artificial delay on one of the disjoint paths between the switches. The configurable delay depends on the processing speed of the attacker. As an existing python library is used that is known for not being the most performant, the configurable delay is measured in milliseconds instead of nanoseconds. In a real-life example, this would mean having a different number of hops on each disjoint path. Where the attacker resides on the path with the shortest hops to ensure he can get his packets to arrive the quickest.

Proof of Concept Attacks
For all identified Proof of Concept attacks described in Section 2.3, this research will focus on the IEEE 802.1CB specific attacks. It will not implement attacks related to IEEE 802.1Qca. All attacks will be implemented in the MITM Raspberry Pi and use the Scapy Python framework. This framework contains powerful packet manipulation functionality to send, sniff, dissect, and forge network packets on different OSI model layers. It supports a wide range of protocols and contains many helper functions that will significantly speed up the implementation phase of this research. The downside is that both the Python language and the Scapy framework are not optimized for speed. Therefore, this framework will influence the research results related to the performance. For the implementation, each attack should individually be toggle-able, and we should also implement a passive mode to test against.

Requirements
In short, we determined the following requirements for the hardware test-bed:

- There should be a configurable and working IEEE 802.1CB implementation on the switches.
- The network should be configured to have two separate disjoint paths that use the IEEE 802.1CB functionality.
- One of the disjoint paths should have an artificial delay to allow the attacker to act first. This delay should also be configurable.
- The experiments should be configurable and give insight into the contents and performance of each packet.
- The MITM device should be able to receive all packets, print them to the console and then send them forward.
- The Scapy framework should be extended to give insight into the header introduced by IEEE802.1CB.
- The Proof of Concept attacks should be able to modify the sequence numbers of existing packets.
- The Proof of Concept attacks should be able to create new packets with "strange" sequence numbers.
- The creation of the hardware test-bed should be fully reproducible for other researchers.

Chapter 4

Implementation

This chapter first details how we create and implement the hardware test-bed. Then, we describe the optional extension used in this research to work from home. The following section will describe how we implement the different proof of concept attacks. Finally, we give an evaluation of the hardware test-bed, and we describe some pitfalls and setbacks that occurred during the research so it will prevent other researchers from making the same mistakes. In addition to the textual description and graphical overviews in this chapter. There is also accompanying code and documentation available on GitHub at the following address: https://github.com/AdriaanDeVos/tsn-hardware-testbed-802.1CB

4.1 Preparation

This section describes in detail the preparation steps required for creating the hardware test-bed. It describes how to prepare the Raspberry Pi 4B and the NXP LS1028ARDB.

Raspberry Pi 4B
First of all, we use three Raspberry Pi's for the experiments, the artificial delay, and the man in the middle attack. To prepare these devices, their BIOS needs to be updated. We then need to place the operating system on a MicroSD card. Furthermore, we transfer some files onto the MicroSD card to configure these devices. For this, the following things are needed:

- 3x Raspberry Pi's 4B with power adapter
- 3x microSD card of at least 4GB
- 1x microSD Adapter
- Raspberry Pi Imager software available at: https://www.raspberrypi.com/software/
- Network configuration files available at: https://github.com/AdriaanDeVos/tsn-hardware-testbed-802.1CB

The following steps are required to prepare the devices:

1. To update the BIOS, run the Raspberry Pi Imager program. Next, select the Bootloader and write it to a microSD card. Finally, put this microSD card in the Raspberry Pi and connect the power for one minute. The BIOS update is now finished.
2. To install the operating system, again use the Raspberry Pi Imager program, but select Raspberry Pi OS Lite (64-bit), which is based on Debian Bullseye. Put this on the microSD card and continue to the next step.
3. On the microSD card, open the `BOOT` partition and create an empty file called `ssh` in the root directory.
4. On the microSD card, open the `SYSTEM` partition and copy the network configuration from the GIT repository and place it at `/etc/network/interfaces`. Make sure to select the correct configuration file for the device.

5. On the microSD card, open the `SYSTEM` partition and copy the `wpa_supplicant.conf` file from the GIT repository and place it at `/etc/wpa_supplicant/wpa_supplicant.conf`.

6. Now, the microSD card can be put in the Raspberry Pi. Connect it to power, and it is ready for the experiment.

NXP LS1028ARDB
While the device comes ready to be used out of the factory with an outdated version of Open Industrial Linux (OpenIL), we recommend using Real-Time Edge Linux instead. More information on why we do not recommend OpenIL and why it is unofficially deprecated can be found in Section 3.4 and Section 4.6.

There are no readily available images provided for this distribution by NXP. However, they provide everything in several open-source repositories and there exists some documentation in various sources that aim to help get it to run. The lack of a distributed image means that we have to compile the operating system and kernel from scratch, which might be a big challenge for people without much experience. Therefore, after much trial and error, this section will provide complete instructions on compiling everything and formatting it correctly on the SD card. The following things are needed:

- A Debian-Based Linux system with networking available. If preferred, it can be native, virtualized (VirtualBox), or containerized (Docker).
- At least 50 GB of available disk space on the build machine
- An SD card of at least 8GB
- Real-Time Edge Linux source code available at: https://github.com/real-time-edge-sw/yocto-real-time-edge

The following instructions have carefully been determined based on the official documentation from Yocto [27], Repo [28], and NXP [29]. Every item consists of a single command but is splitted along several new lines due to formatting.

1. Run the following command to install dependencies:
```
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential
chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping
python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3 xterm curl
```

2. Install Repo dependency:
```
mkdir -p  /.bin && PATH="$HOME/.bin:$PATH" &&
curl https://storage.googleapis.com/git-repo-downloads/repo >  /.bin/repo &&
chmod a+rx  /.bin/repo
```

3. Create a new working directory:
```
mkdir real-time-edge && cd real-time-edge
```

4. Configure git with a placeholder identity:
```
git config --global user.email "researcher@security.com" && git config --global user.name
"Researcher"
```

5. Pull the source code from a variety of repositories:
```
repo init -u https://github.com/real-time-edge-sw/yocto-real-time-edge.git
-b master -m real-time-edge-2.1.1.xml && repo sync
```

6. Setup configuration and environment variables:
```
DISTRO=nxp-real-time-edge MACHINE=ls1028ardb source real-time-edge-setup-env.sh -b build
```

7. Build the kernel and operating system (this will take a while!):
```
bitbake nxp-image-real-time-edge
```

8. Unzip and copy the image:
```
unzip ./tmp/deploy/images/ls1028ardb/nxp-image-real-time-edge-ls1028ardb.wic.bz2
```

9. Make sure with `df -h` that `/dev/mmcblk0` is the correct SD card and then run:
```
sudo dd if=nxp-image-real-time-edge-ls1028ardb.wic of=/dev/mmcblk0 bs=1M conv=fsync
```

4.2 Configuration

This section describes the configuration steps required to create the hardware test-bed. It describes the boot configuration, the network configuration, the TSN configuration, and finally, the user-space configuration.

Booting
The Raspberry Pi's will correctly boot, configure themselves and enable SSH access after following the steps in the Raspberry Pi paragraph above. However, the NXP LS1028ARDB defaults to the onboard outdated Open Industrial Linux distribution. Therefore, the following steps need to be done to ensure it will run Real-Time Edge Linux.

1. Insert the SD card with Real-Time Edge Linux into the NXP switch.
2. Connect the included USB-to-serial cable to UART1 and your computer.
3. Run the following commands to open a serial interface to the LS1028ARDB:
   `sudo chmod 777 /dev/ttyUSB0`
   `screen /dev/ttyUSB0 115200`
4. Connect the power cable and interrupt autoboot by holding down any button on your keyboard.
5. Run the following commmands on the U-Boot CLI to configure SD booting:
   `setenv bootcmd "mmc rescan; mmc dev 0; qixis_reset sd;"`
   `saveenv`
   `reset`
6. The boot configuration has been successful if you are greeted with a `NXP Real-time Edge Distro 2.1` message. You can now login by entering the user `root`.

If the default boot settings have to be restored, for example, if other researchers want to use this hardware device, the following command will do that:
`setenv bootcmd "run sd_bootcmd; run emmc_bootcmd"`
`saveenv`

Network and TSN settings
The next step in creating the hardware test-bed is configuring the network settings. For the Raspberry Pi's this is already done by including the `/etc/network/interfaces` file from this research GIT repository on the microSD card. However, for the NXP switches, this has to be done through the serial interface when the device is running Real-Time Edge Linux. Creating a working network turned out to be very difficult due to limited knowledge and many low-level details not obvious to beginners. In Section 4.6, we describe various things that we tried but turned out to not work well. In this paragraph, we discuss only the final working solution. We made this guide with great help from the Debian Bridge Network Connections documentation on: https://wiki.debian.org/BridgeNetworkConnections

To configure the network settings on the NXP switches, make sure to execute the following steps:

1. Make sure to establish a serial connection to the device and log in.
2. Open the network settings file: `vi /etc/network/interfaces`
3. Delete all current lines by pressing `99dd`
4. Press `i` to start editing and paste the network configuration from the GIT repository. Make sure to select the correct configuration file for your device.
5. Press Escape and then `:wq`
6. Reboot the device: `reboot`

Application / User Space
While the NXP switches have the `tcpdump` package included for monitoring the traffic, the Raspberry Pi 4B devices need additional packages for running the attacks and getting insight into the traffic. Run the following commands on the Raspberry Pi's to make them ready:

1. `sudo apt install tcpdump tshark python3-scapy`
2. `sudo pip3 install --pre scapy[complete]`

4.3 Description of Settings

This section will explain each of the network and TSN settings used during the configuration of the hardware test-bed. We compiled the information from the Debian Networking documentation, the Linux Foundation [30], and the NXP Documentation about tsntool [31]. All the following configuration snippets are from this research's open-source repository on: https://github.com/AdriaanDeVos/tsn-hardware-testbed-802.1CB

Raspberry Pi Configuration
First off, we describe the settings from the Raspberry Pi that is adding an artificial delay to the bridge:

```
auto eth0
iface eth0 inet manual
    up ifconfig eth0 up

auto eth1
iface eth1 inet manual
    up ifconfig eth1 up
# Add a delay to this interface to make sure the other/MITM route arrives first.
    post-up tc qdisc add dev eth1 root netem delay 50ms
```

This section is responsible for `auto`matically bringing up the `eth0` and `eth1` interface on the device. `eth0` is the native Ethernet port included in the hardware, and `eth1` is the USB to Ethernet dongle that we added to the device. Both interfaces are `manual`ly configured by the followed settings instead of having a static IP or using a DHCP server. The `ifconfig dev up` command brings the interface up so it can send and retrieve packets. Please note that no IP address, netmask, or gateway address is configured or required as this works on layer 3 of the OSI model and is thus not required for this research as IEEE 802.1CB focuses on layer 2. For the `post-up` command, the Linux traffic controller uses the `netem` utility to introduce a configurable amount of delay to any packets traversing this interface. In this case, we configured it to 50 milliseconds.

```
auto br0
iface br0 inet manual
    pre-up ip link add name br0 type bridge && ip link set eth0 master br0 && \
           ip link set eth1 master br0
    post-down ip link delete br0 type bridge
```

This section is responsible for enabling bridging between the two network interfaces. Again, this interface is `manual`ly configured and is called `br0` instead of `ethX` to indicate it is a bridging interface. The `pre-up` phase contains three commands that will first create a new interface with the name `br0` that is of `bridge` type. Then, we link both Ethernet interfaces to the bridge in a master-slave connection. Finally, if the device is shutting down or the network needs to restart, the `post-down` command ensures the correct deletion of the bridge interface.

```
auto wlan0
iface wlan0 inet dhcp
        wpa-ssid adriaanhotspot
        wpa-psk adriaandevosisastudent
```

This section is responsible for enabling a Wi-Fi connection to the experiments Raspberry Pi so we can easily manage all devices from home. More details in Section 4.4. For this, we use `DHCP` as the Raspberry Pi running the hotspot will provide a DHCP server for easy network configuration. Then, we provide the network credentials required to authenticate and make a connection.

NXP source switch Configuration
Secondly, we describe the additional settings for the NXP switch that provides the IEEE 802.1CB replication function. It is responsible for enabling VLAN tagging, which is required by the stream identification function. It also creates an entry in the forwarding database and configures the IEEE 802.1CB settings by using the tsntool application.

```
auto br0
iface br0 inet manual
        pre-up ip link add name br0 type bridge vlan_filtering 1 && ip link set br0 up && \
               ip link set swp0 master br0 && ip link set swp2 master br0 && \
```

The first two lines of the `pre-up` phase are the same as described earlier for the Raspberry Pi. The big difference is the usage of `vlan_filtering 1` which enables the VLAN awareness of this bridge.

```
               bridge vlan add dev swp0 vid 1 pvid && bridge vlan add dev swp2 vid 1 pvid
```

In the final line, the bridge configuration is edited to include automatic tagging of the packets with the correct VLAN ID. The stream identification function uses a combination of destination mac address and VLAN ID to identify matching packets.

```
        post-up bridge fdb add 61:64:72:69:61:6e dev swp0 vlan 1 && \
```

The first line of the `post-up` phase adds a definition to the forwarding database. Any Layer 2 device uses this database to store which port can reach which MAC addresses. In this case, we use it to ensure that even without any TSN configuration, the packets from the experiments device will be sent out on the path through the MITM device.

```
               tsntool cbstreamidset --device swp0 --index 1 --nullstreamid \
               --nulldmac 0x61647269616E --nullvid 1 --streamhandle 1 && \
```

The first part of the TSN configuration is added to ensure this stream is identified and replicated. The identification happens based on the `nulldmac`, which is the destination MAC, and the `nullvid`, which is the VLAN identifier. The "Null Stream Identification" method, `nullstreamid` is the only one out of four identification methods that are fully supported by this implementation on NXP devices.

```
               tsntool cbgen --device swp2 --index 1 --iport_mask 0x04 --split_mask 0x03 \
               --seq_len 16 --seq_num 2048
        post-down ip link delete br0 type bridge
```

The second part of the TSN configuration enables stream replication. By configuring the `iport_mask` we determine which ports are monitored for any incoming packets. The `split_mask` determines to which ports the identified stream needs to be replicated. Finally, we use the `seq_len` and `seq_num` parameters to specify the number of bits used for the sequence number counter and to specify the initial state of the sequence number counter.

NXP destination switch Configuration
Finally, the TSN configuration on the NXP switch responsible for the IEEE 802.1CB elimination function is described here:

```
auto br0
iface br0 inet static
        pre-up ip link add name br0 type bridge vlan_filtering 1 && ip link set br0 up && \
               ip link set swp0 master br0 && ip link set swp1 master br0 && \
               ip link set swp2 master br0 && bridge vlan add dev swp0 vid 1 pvid && \
               bridge vlan add dev swp1 vid 1 pvid && bridge vlan add dev swp2 vid 1 pvid
        post-up bridge fdb add 61:64:72:69:61:6e dev swp2 vlan 1 && \
               tsntool cbstreamidset --device swp2 --index 1 --nullstreamid \
               --nulldmac 0x61647269616E --nullvid 1 --streamhandle 1 && \
               tsntool cbrec --device swp0 --index 1 --seq_len 16 --his_len 31 --rtag_pop_en
        post-down ip link delete br0 type bridge
```

While there are some minor differences for the specified interfaces. The addition is the `tsntool cbrec` function that is called in the last line of the `post-up` phase. This command configures the elimination function, also called "recover". The `seq_len` parameter should match with the other configuration as it specifies the length of the sequence number in bits. The `his_len` determines the length of the sequence counter-history. Finally, the `rtag_pop_en` parameter ensures that the IEEE802.1CB header is removed from the packet when relayed to the final destination as that device does not have TSN support.

4.4 Work@Home

This section contains details on making the hardware test-bed available and fully manageable from a remote location. This functionality was needed as we performed the research during a time of corona lockdowns and corresponding measures. In addition, as there was not a reserved workspace available, the hardware test-bed had to be disassembled and reassembled every day. Therefore, this section will first give an overview of the added functionality based on Figure 4.1. Then it will describe how we configure the Wi-Fi Hotspot and how we provide the external SSH access.

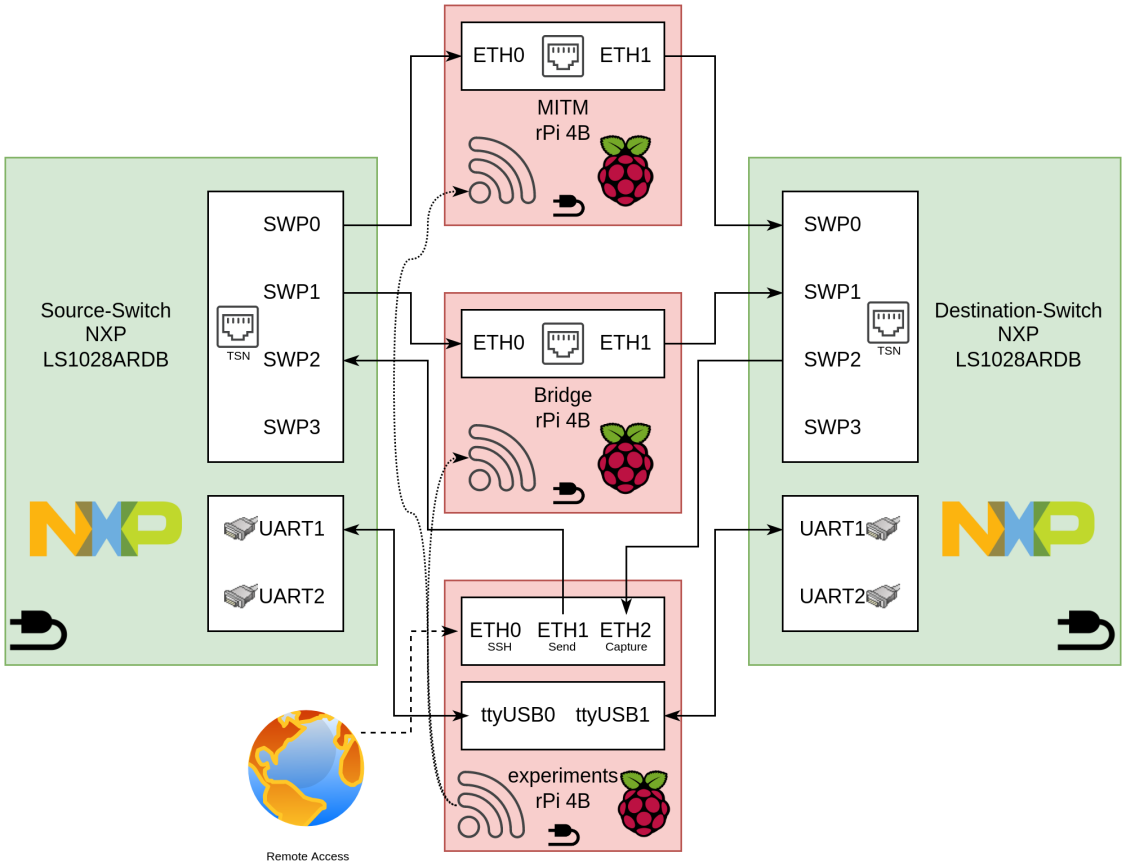

Figure 4.1: Full setup including Work@Home connectivity

Compared to the original hardware overview in Figure 3.1 there are a few minor additions. The most notable addition is the inclusion of the globe, indicating remote access through the internet. This access is provided by connecting the experiments Raspberry Pi to a network outlet in the university. Then, the addition of two arrows between the `ttyUSBX`'s on the experiments Raspberry Pi and the NXP switches. These cables provide serial interfaces that allow connecting to and managing these devices without opening an SSH connection that might influence the performance of the experiments. Finally, there are arrows between the Wi-Fi icons on the Raspberry Pi's as the experiments Raspberry Pi creates a Wi-Fi hotspot. This Wi-Fi connection can then be

used to SSH to the other Raspberry Pi's without sending packets over the network used for evaluating and experimenting. These additions allow full access to all devices from a remote location without interfering with the TSN network within the hardware test-bed.

Wi-Fi Hotspot
To enable the Wi-Fi hotspot on the experiments Raspberry Pi we recommend the following steps (as described in this research's GIT repository):

1. Take the microSD card from the experiments Raspberry Pi and open the `SYSTEM` partition on your computer.
2. Copy the `wifi-configuration/dhcpcd.conf` file from GIT to `/etc/dhcpcd.conf`.
3. Copy the `wifi-configuration/dnsmasq.conf` file from GIT to `/etc/dnsmasq.conf`.
4. Copy the `wifi-configuration/hostapd.conf` file from GIT to `/etc/hostapd/hostapd.conf`.
5. Copy the `wifi-configuration/sysctl.conf` file from GIT to `/etc/sysctl.conf`.
6. Copy the `wifi-configuration/rules.v4` file from GIT to `/etc/iptables/rules.v4`.
7. Run the following command to enable and reboot the Wi-Fi hotspot:
   `sudo systemctl unmask hostapd && sudo systemctl enable hostapd && sudo reboot`

External SSH access
The IT department from the EEMCS faculty has provided information about setting this up. They confirm that most Ethernet outlets in the faculty building are assigned a public-facing IP address. Therefore, the experiments Raspberry Pi has been hardened with public key authentication and firewall rules. However, there is no guarantee that the device has a static IP address as it could change during maintenance, downtime, or other unforeseen circumstances. We resolved this issue by enabling the following cronjob on the experiments Raspberry Pi to ensure it regularly sends a message with its current IP address to a server which keeps logs.

```
* * * * * /usr/bin/curl -I https://adriaandevos.nl/here-i-am/$(/usr/bin/curl ifconfig.me) >/dev/null
```

4.5 Proof of Concept Attacks

As described in the design phase, we use the Scapy Python framework as it speeds up the development process by providing ready to use functions and support for many protocols. However, it does not have support for IEEE802.1CB yet. In this section, we first describe how we implemented this support in the Scapy framework, and then we will describe the implementation of each of the five proof of concept attacks. In general, the MITM program requires the user to select one of the available attacks or select passive mode. It then feeds all incoming packets to one of the functions provided by the proof of concept attacks and finally relays the resulting packets.

IEEE 802.1CB Support
Implementing support for IEEE 802.1CB in Scapy was relatively simple due to all the provided functionality and functions. We provide the resulting code in the GIT repository in `scapy/RedundancyTag.py`. It consists of three parts.

The first part consists of defining the structure of the header within the packet. This structure is described in the official IEEE specification and also shown in Figure 2.2. However, during packet analysis, we noticed that the NXP implementation differs from the official specification as it omits the reserved bytes in the middle of the redundancy tag.

Secondly, the decapsulation of the redundancy tag needs to be specified to allow stripping and correct parsing of packets that contain it. This stripping is done by checking if the packet starts with an Ethernet header and then confirming that a redundancy tag header follows it. It then allows the removal of these bytes from the packet and returning of the original packet without the redundancy tag.

Finally, the newly created redundancy tag layer must be bound to the Ethernet layer with a possible extension to the IP layer. For this, we use the correct EtherType as documented in the official IEE specification,

being `0xF1C1`. However, as found during packet analysis and confirmed by checking the source code, the NXP implementation uses a different placeholder EtherType, namely 0x2345. For this, a pull request has been made on GitHub to change the incorrect hardcoded EtherType: Github Real-Time Edge Software

Spoof Ahead
This proof of concept attack uses **spoofing** to create new packets in the network. These packets arrive **earlier** than legitimate packets with the same sequence number. This attack causes the IEEE 802.1CB elimination function to only relay the spoofed packet, **resulting in a compromised communication channel**. It matches the identification function because it uses the details of the legitimate packet to create new spoofed packets.

This attack is implemented in Python in the `spoof_ahead` method by first checking if the packet has the RedundancyTag, then using the characteristics of the legitimate packet in combination with a spoofed payload to send out the spoofed packet before returning and allowing the original and legitimate packet to continue along the bridge.

Spoof Behind
This proof of concept attack uses **spoofing** to create new packets in the network. These packets arrive **later** than legitimate packets with the same sequence number. This attack causes the IEEE 802.1CB latent error function to detect a sudden increase in identified and eliminated packets, **resulting in a raised alarm**. It matches the identification function because it uses the details of the legitimate packet to create new spoofed packets.

This attack is implemented in Python in the `spoof_behind` method by first checking if the packet has the RedundancyTag, then allowing the original and legitimate packet to continue along the bridge. Finally, the legitimate packet's characteristics are combined with a spoofed payload to send out the spoofed packet.

Tamper Random
This proof of concept attack uses **tampering** to modify existing packets in the network. These packets are modified to have a **random** sequence number instead of their original one. Depending on the random value and the state of the sequence number counter, this could cause either legitimate packets to be dropped, packets to be delivered in a different order, or to raise an alarm. This exploits the IEEE 802.1CB functionality, **resulting in unstable and unpredictable network conditions**.

This attack is implemented in Python in the `tamper_random` method by first checking if the packet has the RedundancyTag, then modifying the sequence number with a random value between 0 and the maximum allowed value dependent on the IEEE 802.1CB configuration. Finally, it allows the tampered packet to continue along the bridge.

Tamper Replay
This proof of concept attack uses **tampering** to modify existing packets in the network. These packets are modified to **replay** the payload of earlier packets while being unnoticed because it uses the sequence number of the legitimate packet. This attack is undetectable by the IEEE 802.1CB latent error function, and it could repeat a single packet or a sequence of packets depending on the needs of the adversary. In an industrial system, this can be used to replay legitimate sensor values, as used in Stuxnet [4]. Alternatively, it can be used to replay actuator commands such as "increase the cruise-control car speed by 5km/h". It **results in a compromised communication channel**.

This attack is implemented in Python in the `tamper_replay` method by first checking if the packet has the RedundancyTag. If it matches, the adversary can make one of two choices based on the packet's content. It can either decide to "record", namely storing a copy of the packet for later use, while letting the original packet continue along the bridge. Alternatively, it can decide to "replay", using one of the recorded packets, with an updated sequence number to make it unnoticed, and then forwarding the tampered packet.

Speedup
This proof of concept attack uses **tampering** to modify the sequence number of existing packets. The sequence number for each packet is **speed up** with a configurable step-size instead of using an incremental counter. The step size should stay below the size of the history length as this functionality acts as a rolling window to check which sequence numbers are expected and should be processed further. This attack causes the sequence number

counter to be sped up by a significant amount for each relayed packet. **Resulting in a denial of service** when the attack is stopped or paused as the IEEE 802.1CB will then drop legitimate packets as they contain sequence numbers that are not within the allowed values determined by the rolling window. This attack will automatically resolve itself when the sender's sequence number counter catches up with the tampered sequence number counter of the receiver.

This attack is implemented in Python in the `speed_up` method by first checking if the packet has the RedundancyTag, then modifying the sequence number by adding the multiplication of the configurable step size and an internal incremental counter. Finally, after tampering with the sequence number, the packet is returned to be relayed along the bridge.

4.6 Evaluation

During the implementation of the hardware test-bed, numerous challenges occurred, causing a delay to the research schedule. We will describe some of these pitfalls in the next paragraph. However, we also underestimated the time required for the implementation phase in our schedule. Based on other literature such as [18], the expectation was that the creation of the hardware test-bed required only attaching the cables and making some minor configuration adjustments. However, in practice it turned out to require a lot of preparation and configuration steps, as described earlier in this chapter. It also required a ton of low-level knowledge about the hardware, booting, compiling, and networking. This misjudgement caused some delay in the research schedule, and it was a significant challenge to overcome.
In Figure 4.2, we show photos of the hardware test-bed. The cable management is quite messy, and therefore the schematic overviews have been made, which are shown in Figure 3.1 and Figure 4.1.

Figure 4.2: Photos of the created hardware testbed

Things NOT to do
This section contains abbreviated tips about the pitfalls that other researchers can avoid:

- **Open Industrial Linux**, after spending significant time and effort in getting this distribution to compile and boot, we noticed that it has limited TSN support and that it is actually deprecated by NXP. This deprecation was not clear from their repositories and website. So we have been in contact with employees of NXP, and they suggested using Real-Time Edge Linux instead.
- **NETCONF / YANG**, while this protocol and modelling language can be very helpful in configuring network settings. It does not yet have support for all TSN settings. It only works if either a DHCP server is present in the network or all addresses are hardcoded as NETCONF requires network connectivity to propagate changes.
- **Source MAC and VLAN Stream Identification**, while this type of identification is one of the four officially documented in the IEEE specification, its support of it is somewhat limited on the NXP switches. These hardware devices contain 2 IP switches with different support, and only one of them supports this type of stream identification. As only some ports are available with this functionality and additional bridging and configuration are required, we advise against using this.
- **ARP storm**, the hardware test-bed contains a loop to allow for two disjoint paths between the NXP switches. When ARP is enabled and layer 3 configuration is done, there exists a risk of an ARP storm occurring, causing a denial of service within the network. This issue can be partially be prevented with Spanning Tree Protocol (STP), but we advise not to enable layer 3 configuration for this hardware test-bed.
- **VLAN support**, while it should be possible to configure all devices, including Raspberry Pi's to be aware of VLAN headers, we advise against configuring this. Some progress was made by which the VLAN tagging was done on the experiments device instead of on the NXP switches, but this required a lot of additional configuration. In practice, VLAN tagging is most often done on the switches themselves based on the ports where the traffic is received, and therefore this will be used in the final version.
- **Forwarding Database**, technically, it should be possible to fully manage the MAC and IP forwarding within the network and predefine all available routes by using the forwarding database. However, this becomes rather tedious, and especially with larger networks, we advise against using this method.
- **Unique IP per interface**, for one of the first versions, every interface was configured with a separate IP address. This configuration allowed one-hop communication between devices, but it did not allow for bridging through any devices. Trying this with different IPs that should allow subnet routing was also rather complex to configure. Therefore, we strongly advise using bridging on each device and one IP per device instead of per interface.
- **bridge-utils**, this is a Linux package responsible for Ethernet bridge administration. While it contains most functionality required for configuring Ethernet bridges, it is limited for modern network environments. This package is deprecated in favour of the iproute(2) package in most modern distributions. This hardware test-bed configuration is made with the iproute package installed by default in Real-Time Edge and Raspbian.

Chapter 5

Results and Discussion

This chapter gives an overview of the performed experiments, their results, and an evaluation. First, it provides a short reminder of the Key Performance Indicators and how they can be measured. Then it describes the workflow execution by explaining how the test packets travel through the network. Finally, it describes the steps required to perform the experiments and analysis and the configurable settings.

In the second part, we show the results from performing the experiments on the hardware test-bed and discuss their effects. First, we summarise the results for each attack in a table. Then, we describe the results in detail while reporting the effects of the configuration parameters. Finally, in the last section, we extensively discuss and evaluate the possible causes for the results.

5.1 Experiments

Key Performance Indicators
While Time-Sensitive Networking targets the following Key Performance Indicators [7], IEEE 802.1CB does not affect all these KPIs. This substandard focuses on the reliability of the network connection in regards to packet loss. Compared to other TSN standards, it does not alter Quality of Service (QoS) related bandwidth reservations or latency guarantees.

- **Guaranteed delivery with bounded latency**, this KPI can be affected by a misconfiguration of the communication paths that uses IEEE 802.1CB. As described in Section 2.3, the path configuration of the IEEE 802.1Qca standard is out of scope for this research.
- **Low delay variation (jitter)**, this KPI is unaffected by IEEE 802.1CB as the processing within the functions of this standard is independent of the packet size or other variables. Each replicated and eliminated packet requires the same amount of processing. Therefore there is no impact on the jitter or the order of the received packets.
- **Low packet loss**, this KPI is the focus of the IEEE 802.1CB standard. Because of the duplicate paths, there is a significantly decreased chance that packets are lost. However, some of the implemented attacks abuse the functionality of the elimination function to drop legitimate packets, resulting in detrimental effects on the network communication quality.

In addition to these KPIs, the experiments evaluate incorrectly raised alarms by the latent error function, and it verifies the ability to change the payload content while staying unnoticed. Finally, for the implemented attacks that are timing dependent, some experimenting is done to evaluate the sensitivity, but this is somewhat limited as the Python framework has slow performance.

Workflow Execution
For each analysis, the experiments Raspberry Pi generates a packet stream that travels through the network and returns to this device for analysis. In addition, the modification of some configuration parameters is possible, and some actions are optional. Workflow execution is given below, with a graphical representation shown in Figure 5.1.

1. **Creating a new packet stream**, this is done on the experiments Raspberry Pi in the `scapy/experiments-send.py` script. It creates a packet stream with an Ethernet header, IP header, and a payload. The created payloads alternate between accelerating the car and decelerating the car. This

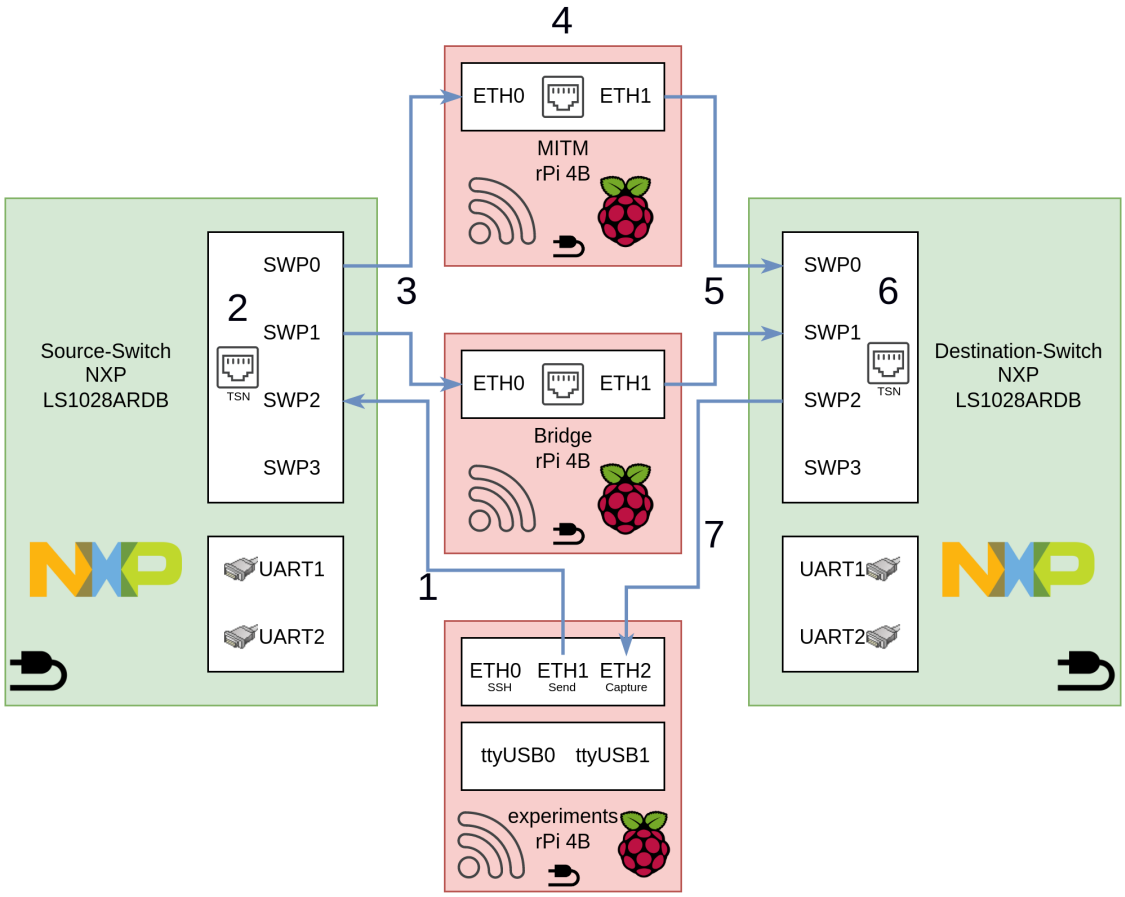

Figure 5.1: Workflow Execution of the Experimental Setup

heuristic gives a more tangible example of a person driving a car. Additionally, the payload contains a counter used to analyse the packets to discern between legitimate packets and tampered with packets as they would otherwise be unnoticeable.

2. **TSN: Frame Identification Function**, this function is configured on the source NXP switch, and it identifies the packet stream based on the VLAN and destination MAC.

3. **TSN: Frame Replication Function**, this function is configured on the source NXP switch, and it makes sure to replicate the identified packet stream onto multiple disjoint paths.

4. **Running experiments:**

   (a) **(optionally) Performing an Attack**, the MITM Raspberry Pi runs the `scapy/mitm-active.py` script to select one of the implemented attacks or to relay all packets passively.

   (b) **(configurable) Artificial Delay**, the Bridge Raspberry Pi introduces a configurable artificial milliseconds delay to its communication path while relaying packets.

5. **TSN: Frame Identification Function**, this function is configured on the destination NXP switch, and it identifies the packet stream based on the VLAN and destination MAC.

6. **TSN: Frame Elimination Function**, this function is configured on the destination NXP switch, and it checks the identified packet stream to make sure it relays the first arriving packet for each sequence number. It drops any additional packets and always removes the redundancy tag header from the relayed

packets. In addition, a latent error function should run here that notices irregularities. Finally, the sliding window for which sequence numbers are allowed is **configurable**.

7. **Receiving and Analysing the Packet Stream**, this is done on the experiments Raspberry Pi in the `scapy/experiments-receive.py` script. First, it receives all packets and parses them to show the payload. Then, it tries to detect any attacks based on the counter included in the payload.

Running the Experiment
First, to verify that the hardware test-bed is working, start the `send` and `receive` scripts on the experiments Raspberry Pi and set the MITM Raspberry Pi's attack mode to passive. The network connection works if the `receive` script shows any packets. Then, testing if the TSN configuration works is accomplished by disrupting one of the disjoint paths and noticing that there are no lost packets. To perform this test, take one of the following actions: disconnect a network cable between the Bridge or MITM Raspberry Pi, restart one of the devices, or bring down one of the interfaces used for either path.

Performing the analysis of the effects of any attack requires running the `send`, and `receive` scripts on the experiments Raspberry Pi and choosing one of several available attacks on the MITM Raspberry Pi by running `sudo python3 ./mitm-active.py 3`. Specify the type of attack with the final numerical argument of this command. Documentation on the options is available in the code and the provided readme. Finally, monitor the output of the `receive` script to see if any attacks are automatically detected.

The artificial delay introduced in the Bridge Raspberry Pi is configurable by editing the `/etc/network/interfaces` configuration file. These modifications to the amount of artificial delay are only necessary for evaluating the effectiveness of the Spoof Ahead attack and the Tamper Replay attack as they are time-sensitive.

The size of the sliding window used for the Frame Elimination Function is configurable in the destination NXP switch by editing the `/etc/network/interfaces` configuration file. These modifications will only affect the Tamper Random and Speedup attacks as they are dependent on the effectiveness of the sequence history functionality.

5.2 Results

Table 5.1 shows a summary of the results for each of the implemented attacks. The delay column contains the required amount of artificial delay for a successful attack. A dash means that the attack works independently of any configured artificial delay.

Name of the Attack	Implemented	Result	Delay	Explanation
Spoof Ahead	Yes	Yes	60ms	Inserting own packets while the original packets are discarded
Spoof Behind	Yes	No	-	NXP implementation doesn't have Latent Error Function
Tamper Random	Yes	Partial	-	NXP implementation sequence history does not follow specification
Tamper Replay	Yes	Yes	6ms	Possible to replay packets without the end device noticing
Speedup	Yes	Partial	-	NXP implementation sequence history does not follow specification

Table 5.1: Comparison of all identified attack vectors.

Each of the attacks, their effects, and the outcome are described below:

- **Spoof Ahead**, This attack is successful in compromising the communication channel by only allowing the payloads of the attacker to arrive at the end destination. The IEEE 802.1CB Stream Elimination Function will drop the legitimate packets. However, a significant artificial delay is required to ensure that the attacker can deliver their packets first. Decreasing the artificial delay results in a less effective attack as only a percentage of the packets within the stream can be affected.

- **Spoof Behind**, This attack is, while implemented, not successful and does not have any noticeable effect on the system or network communications.

- **Tamper Random**, This attack is partially successful in creating an unstable communication network. It causes most legitimate packets to arrive twice at the end destination as it is received twice at the NXP destination switch with different sequence numbers. It relays both the packet with the legitimate sequence

number and the packet with the random sequence number as long as it is not the same or very close to the legitimate sequence number.

- **Tamper Replay**, This attack is successful in replaying legitimate packets and being undetectable by the end device. Furthermore, it requires a significantly shorter artificial delay than the Spoof Ahead attack and could therefore be easier to execute in a real-life situation.
- **Speedup**, This attack is just as successful as the Tamper Random attack in creating an unstable communication network. When executing this attack, both the legitimate and the speedup packet will arrive at the end destination, causing unexpected network behaviour.

5.3 Discussion

The results show that some of these attacks are successful, albeit with a certain artificial delay. In addition, the implementation of other identified attack vectors shows only limited effects on the hardware test-bed. Here we discuss the results in more detail while proposing possible reasons why some attacks do not work as expected.

First of all, as described in the previous section, some of the implemented attacks are shown to be successful. The experiments show that both the Spoof Ahead and the Tamper Replay attacks significantly impact the network. It shows that an attacker with either Tamper or Spoof ability can completely compromise the communication while being transferred redundantly on multiple paths. The attacker only needs to be quick enough on a single path to change the content of the relayed messages or perform a Denial of Service attack. The attack's simplicity and the significant negative effect show that these security risks greatly impact the IEEE 802.1CB standard.

The next thing to discuss is the artificial delay. This delay was introduced within the hardware test-bed to ensure that the adversary is present on the quickest path. In practice, this means that the adversary is on the path with the least hops as each switch along the path introduces an additional delay. Concerning Time-Sensitive Networking, a delay of 60ms, or even 6ms, is very big and therefore not realistic in a real-life situation. However, we think that the cause for these delay requirements is the usage of the Scapy Python framework for our attacks [32]. Research has shown this framework to be very slow compared to solutions written in C or using an FPGA. Switches or devices that support Time-Sensitive Networking require only a few nanoseconds to process these packets and possibly modify them. In our implementation, the Spoof Ahead attack requires 60ms compared to the 6ms from the Tamper Replay attack. This big difference is caused by the Spoof Ahead attack creating new packets that it wants to send out, while the Tamper Replay attack only stores existing packets and reuses these to replay any earlier communication. Therefore, this attack requires significant less parsing and modifications to the packets and thus a more negligible artificial delay.

The next thing to discuss is that the Spoof Behind attack does not affect the hardware test-bed. We think this is caused by the current version of Real-Time Edge Linux only having a partial implementation of IEEE 802.1CB. To be more specific, the Latent Error Function, an essential part of the IEEE 802.1CB specification, has not been implemented. Therefore, it raises no alarms when there is a sudden change in the number of packets that arrive per sequence number. This partial support was not known beforehand, but these experiments have shown the lack of implemented functionality. However, in theory, we expect that our proof of concept attack should be effective and could therefore work on more complete implementations from other vendors.

Then, the Tamper Random and the Speedup attack show only partial results. We think this is also caused by a limited implementation of the IEEE 802.1CB specification. For these attacks, the focus lies on the functionality of the sequence history counter used for the sliding window. The speedup attack does not have the expected result. The goal was to speed up this sequence history counter to automatically discard legitimate packets that arrive outside the sliding window. Unfortunately, this does not work as it allows all packets to go through, causing duplicate packets to arrive at the destination device. Surprisingly, the lack of proper implementation of this functionality increases the effectiveness of the Tamper Random attack. This attack aimed to send random sequence numbers, raise alarms, and irregularly drop legitimate packets. However, due to the sliding window not working, all packets with random sequence numbers will arrive at the destination device resulting in duplicate packets. To conclude, while the sliding window size is configurable, our experiments have shown that they do not affect the functionality, so we assume that this configuration value is currently unused.

While we cannot show the effectiveness of all implemented attacks due to hardware limitations, we show that

the impact of the Spoof Ahead and Tamper Replay is significant enough to require additional security measures or mitigations when using this networking standard. Especially for mission-critical systems, we need guarantees for their security and reliability, which is not the case with IEEE 802.1CB. With these results, we can answer the research question by showing a big impact of the security risks within this standard. This impact prevents usage of this networking standard in the near future as additional research and development are required to adhere to safety standards.

Finally, this research has shown that significant security risks exist to the IEEE 802.1CB standard and that no existing solutions completely mitigate these risks. This outcome provides a good starting point for future research into possible mitigations of the security threats and additional research to validate the additional identified security risks.

Chapter 6

Conclusion and Future Work

In this section, we conclude the outcome of this research by answering the research question. Finally, we provide suggestions for future work as we think this hardware test-bed can be of great use for other researchers.

6.1 Conclusion

Our research uses the STRIDE model to identify several security risks to the IEEE 802.1CB standard. According to our literature research, these security risks are currently unable to be mitigated by any existing solutions. An extensive design and implementation phase has resulted in an advanced hardware test-bed that can simulate a real-life environment to test attacks and mitigations solutions to the functionality introduced by the IEEE 802.1CB standard. Due to a focus on the reproducibility of the research, this hardware test-bed can be used and extended by other researchers interested in testing practical attacks and mitigations against Time-Sensitive Networking standards. We have investigated all identified security risks introduced by the functionality of the IEEE 802.1CB standard by creating proof of concept attack implementations that try to exploit these vulnerabilities. These attack implementations are available in an open-source repository to ensure other researchers can use them to test their mitigation solutions. This research performs the experiments on the Real-Time Edge Linux implementation of the NXP LS1028ARDB switches. Two of the five attacks show promising results in their effectiveness as they can replay existing packets or completely take over the communication channel with their crafted packets. These are significant risks to a Time-Sensitive Networking system as mission-critical applications depend on the reliability of the network. However, the research also shows that three of the five attacks have no effect or only a partial effect. This result is most likely due to the limited IEEE 802.1CB implementation in the Real-Time Edge Linux project. We expect them to be effective on implementations that follow the official specifications. While this research cannot confirm the effectiveness of all the identified security risks and corresponding proof of concept attacks, we show that successful attacks greatly impact the network's reliability. Therefore, we conclude that exploiting these security risks severely impacts the IEEE 802.1CB standard and that further research is needed to determine mitigation solutions.

6.2 Future Work

This research has been done as a master thesis, and therefore the available time has been limited. There are several things that we would like to have done but were not feasible within the time limit. We do, however, hope that we have laid a good foundation for further research, and we advise the following research directions:

- **Attack Against Mitigation Solutions**, while we have done research to find existing mitigation solutions. Unfortunately, the authors of these papers did not respond, or they required us to accept their PhD candidate position if we wanted to receive the source code. Therefore, we could not verify the implemented attacks against existing mitigation solutions. Nevertheless, we have published the source code of the attacks for further research so other researchers can use them to verify their mitigation solutions.
- **Create Working Mitigation Solution**, the existing mitigation solutions that we found all had some limitations. It might be interesting to combine the Chaos Cipher [17], and TSN-MIC [16] solutions as we expect that they complement each other well. Alternatively, research can be done into a new mitigation solution as the current ones are insufficient to mitigate all security risks.

- **Attack Different Implementations**, this research has shown that the Real-Time Edge Linux implementation of IEEE 802.1CB is lacking. It would be interesting to perform the attacks on implementations from different vendors. These implementations might better follow the official specification or produce additional risks by introducing bugs to the functionality.
- **Extend the Hardware Test-Bed**, the current hardware test-bed provides the minimal network topology and configuration for testing IEEE 802.1CB. However, it might also be interesting to include more devices, longer paths, or other advanced network configurations to simulate a real-life example better. In addition, other researchers can easily extend this hardware test-bed to test the functionality of other TSN standards.
- **Attack Optimalisation**, we implemented the current attacks with the Scapy Python framework, which is rather slow. It would be interesting to see these attacks when they are implemented in, for example, C or Rust to ensure good performance. Unfortunately, this will take more development time as the Scapy Python framework uses a higher-level language and several helper functions to ease implementation. However, as a result of increased performance, the hardware test-bed can be more realistically configured by minimizing the artificial delay.

Bibliography

[1] Case, D. U. (2016). Analysis of the cyber attack on the Ukrainian power grid. Electricity Information Sharing and Analysis Center (E-ISAC), 388, 1-29.

[2] Lee, R. M., Assante, M. J., Conway, T., & SANS Industrial Control Systems. (2014). ICS CP/PE (Cyber-to-Physical or Process Effects) Case Study Paper–Media report of the Baku-Tbilisi-Ceyhan (BTC) pipeline Cyber Attack.

[3] Slay, J., & Miller, M. (2007, March). Lessons learned from the maroochy water breach. In International conference on critical infrastructure protection (pp. 73-82). Springer, Boston, MA.

[4] Farwell, J. P., & Rohozinski, R. (2011). Stuxnet and the future of cyber war. Survival, 53(1), 23-40.

[5] Teener, M. D. J., Fredette, A. N., Boiger, C., Klein, P., Gunther, C., Olsen, D., & Stanton, K. (2013). Heterogeneous networks for audio and video: Using IEEE 802.1 audio video bridging. Proceedings of the IEEE, 101(11), 2339-2354.

[6] Nasrallah, A., Thyagaturu, A. S., Alharbi, Z., Wang, C., Shao, X., Reisslein, M., & ElBakoury, H. (2018). Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research. IEEE Communications Surveys & Tutorials, 21(1), 88-145.

[7] Bello, L. L., & Steiner, W. (2019). A perspective on IEEE time-sensitive networking for industrial communication and automation systems. Proceedings of the IEEE, 107(6), 1094-1120.

[8] Ergenç, D., & Fischer, M. (2021, May). On the Reliability of IEEE 802.1 CB FRER. In IEEE INFOCOM 2021-IEEE Conference on Computer Communications (pp. 1-10). IEEE.

[9] Hofmann, R., Nikolić, B., & Ernst, R. (2019). Challenges and Limitations of IEEE 802.1 CB-2017. IEEE Embedded Systems Letters, 12(4), 105-108.

[10] Prinz, F., Schoeffler, M., Lechler, A., & Verl, A. (2018, September). End-to-end redundancy between real-time I4. 0 Components based on Time-Sensitive Networking. In 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA) (Vol. 1, pp. 1083-1086). IEEE.

[11] Pannell, D., & Navet, N. (2020). Practical Use Cases for Ethernet Redundancy.

[12] A. Shostack, S. Hernan, S. Lambert, & T. Ostwald, Uncover Security Design Flaws Using The STRIDE Approach.

[13] Ergenç, D., Brülhart, C., Neumann, J., Krüger, L., & Fischer, M. (2021, June). On the Security of IEEE 802.1 Time-Sensitive Networking. In 2021 IEEE International Conference on Communications Workshops (ICC Workshops) (pp. 1-6). IEEE.

[14] Frame Replication and Elimination for Reliability, IEEE standard P802.1CB, 2017

[15] Carnevale, B., Fanucci, L., Bisase, S., & Hunjan, H. (2018). Macsec-based security for automotive ethernet backbones. Journal of Circuits, Systems and Computers, 27(05), 1850082.

[16] Watson, V., Ruland, C., & Waedt, K. (2021). MAC-layer Security for Time-Sensitive Switched Ethernet Networks. INFORMATIK 2020.

[17] Pérez-Resa, A., Garcia-Bosque, M., Sánchez-Azqueta, C., & Celma, S. (2018, May). Using a chaotic cipher to encrypt Ethernet traffic. In 2018 IEEE

[18] Wang, C. T., Qin, G. H., Zhao, R., & Song, S. M. (2021). An Information Security Protocol for Automotive Ethernet. Journal of Computers, 32(1), 39-52. International Symposium on Circuits and Systems (ISCAS) (pp. 1-5). IEEE.

[19] Kohnfelder, L., & Garg, P. (1999). The threats to our products. Microsoft Interface, Microsoft Corporation, 33.

[20] Khan, R., McLaughlin, K., Laverty, D., & Sezer, S. (2017, September). STRIDE-based threat modeling for cyber-physical systems. In 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe) (pp. 1-6). IEEE.

[21] Bhushan, B., Sahoo, G., & Rai, A. K. (2017, September). Man-in-the-middle attack in wireless and computer networking—A review. In 2017 3rd International Conference on Advances in Computing, Communication & Automation (ICACCA)(Fall) (pp. 1-6). IEEE.

[22] Do, Q., Martini, B., & Choo, K. K. R. (2019). The role of the adversary model in applied security research. Computers & Security, 81, 156-181.

[23] Dolev, D., & Yao, A. (1983). On the security of public key protocols. IEEE Transactions on information theory, 29(2), 198-208.

[24] Bellare, M., & Rogaway, P. (1993, August). Entity authentication and key distribution. In Annual international cryptology conference (pp. 232-249). Springer, Berlin, Heidelberg.

[25] Corbett, C., Schoch, E., Kargl, F., & Preussner, F. (2016). Automotive Ethernet: Security opportunity or challenge?. Sicherheit 2016-Sicherheit, Schutz und Zuverlässigkeit.

[26] Lim, H. T., Weckemann, K., & Herrscher, D. (2011, March). Performance study of an in-car switched ethernet network without prioritization. In International Workshop on Communication Technologies for Vehicles (pp. 165-175). Springer, Berlin, Heidelberg.

[27] The Yocto Project. (2020, April). Yocto Project Reference Manual. Yocto Project. https://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html

[28] Android. (2021, December) Repo Command Reference. Android Source. https://source.android.com/setup/develop/repo

[29] NXP. (2020, August) Layerscape Software Development Kit User Guide for Yocto. NXP Documentation. https://docs.nxp.com/bundle/GUID-E5527A77-2F97-4244-BF9C-D08F068EFD16/page/GUID-2BDBF943-FBB7-46C3-80B3-43A4BCBC6ABB.html

[30] ausmatt. (2021, October) netem. The Linux Foundation. https://wiki.linuxfoundation.org/networking/netem

[31] NXP. (2020, January) Layerscape Software Development Kit User Guide. NXP Documentation. https://docs.nxp.com/bundle/GUID-3FFCCD77-5220-414D-8664-09E6FB1B02C6/page/GUID-E5431AF7-C1EF-4C34-BC70-54A0C473763A.html

[32] D'Agostino, A. M., Ometov, A., Pyattaev, A., Andreev, S., & Araniti, G. (2018). Performance Limitations of Parsing Libraries: State-of-the-Art and Future Perspectives. In Internet of Things, Smart Spaces, and Next Generation Networks and Systems (pp. 405-418). Springer, Cham.

List of Figures

List of Tables