Delft University of Technology
Master of Science Thesis in Computer Engineering

Protecting CAN XL Protocol from Denial-of-Service Attacks

Fanyuan Li

TU Delft Delft University of Technology

Protecting CAN XL Protocol from Denial-of-Service Attacks

Master of Science Thesis in Computer Engineering

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Fanyuan Li

August 1st, 2023

Author
Fanyuan Li (f.li-2@student.tudelft.nl)
(fanyuan.li@nxp.com)
Title
Protecting CAN XL Protocol from Denial-of-Service Attacks
MSc Presentation Date
August 30th, 2023

Graduation Committee

Georgi Gaydadjiev	Delft University of Technology
Mottaqiallah Taouil	Delft University of Technology
Przemyslaw Pawelczak	Delft University of Technology
Rolf van de Burgt	NXP Semiconductors

Abstract

Over development of the past three decades, control systems in automobiles have undergone a significant transformation, shifting from mechanical devices to a multitude of interconnected computers that oversee sensors, drivers, and passengers. In today's modern vehicles, 50-80 independent computers, commonly referred to as Electronic Control Units (ECUs), form complex in-vehicle networks to facilitate communication among themselves. A widely adopted communication protocol in modern in-vehicle networks is the Controller Area Network (CAN). Security wasn't considered when the CAN protocol was designed in the mid-1980s, leaving it susceptible to cyber-attacks. In 2015, Charlie Miller and Chris Valasek successfully demonstrated a remote attack against a Jeep Cherokee, compromising critical components such as the steering wheel and braking system. This event prompted significant concern about the security of CAN networks in both academic and industrial circles.

In reaction to these challenges, researchers have formulated a variety of attacks targeting the CAN bus, encompassing Denial-of-Service (DoS) attacks, spoofing attacks, and more. Furthermore, significant efforts have been directed towards enhancing the security of CAN networks. These endeavors encompass the design of security architectures and intrusion detection systems. NXP developed the TJA115x secure CAN transceiver family, which offers protection against various types of attacks for Classical CAN and CAN FD communication without resorting to cryptography. Nonetheless, all existing researches are directed towards the Classical CAN and CAN FD protocols, leaving a notable gap in the exploration of security aspects concerning the latest CAN XL network.

This thesis focuses on enhancing the design of the TJA115x secure transceiver to support CAN XL protocol. The research comprises two main aspects: flooding detection and flooding prevention. To achieve flooding detection, a leaky buckets tree is introduced, enabling differentiation between normal frames and flooding frames, along with dynamic traffic control for each frame type. Additionally, the leaky bucket parameters are derived in formulaic form. For flooding prevention, two distinct strategies are devised to prevent flooding frames from dominating the bus. These strategies are compared in terms of throughput, latency, and availability. Finally, the proposed secure CAN XL transceiver is validated through software simulations and hardware experiments. The results demonstrate the exceptional performance of the CAN XL secure transceiver in blocking flooding attacks and safeguarding communications. These accomplishments are achieved with minimal memory overhead, and notably, without introducing any additional latency or bandwidth demands.

Acknowledgement

After one year's hard work, my graduation project at TU Delft and internship at NXP Semiconductors is coming to an end. I have learned a lot from the experience in this year. It is a highly challenging project for me, but I managed to overcome it to the best of my ability. During this project, I encountered numerous tricky problems that troubled me both technically and emotionally. Therefore, I know I could not have finished this thesis by myself and I would like to express my gratitude to all the people that have helped me.

First, I would like to thank my supervisor Rolf van de Burgt from NXP. I appreciate his supervision which taught me industrial knowledge that I can never acquire in the university. I also feel thankful for his patience and generous support that guided me the right way when I felt frustrated and lost.

Second, I would like to thank my daily supervisor Mottaqiallah Taouil from TU Delft. I appreciate the discussions and weekly meetings I had with him, his helpful assistance in thesis writing, and arrangement of the final defense.

Thirdly, I would like to thank Pascal van Leeuwen, Chinghsuan Chou, Brecia Nurastu Sasongko, and all other colleagues who have helped with my project.

Last but not least, I would like to thank my parents and friends that kept me motivated all the time.

Thank you all!

Fanyuan Li

Nijmegen, The Netherlands
25th July, 2023

Contents

Chapter 1

Introduction

This chapter briefly introduces the topic addressed in this thesis, its importance, related work and the main contributions. In the end, the thesis organization is presented. Section 1.1 presents the motivation and relevance of the CAN security, describes a popular CAN bus hacking procedure, and explains new challenges that CAN XL brings about. Section 1.2 reviews recent studies on security of CAN bus. Section 1.3 summarizes the contribution of this thesis. Section 1.4 finally outlines the thesis organization.

1.1 Motivation

Nowadays, modern automobiles have become an essential part of our daily lives. With the emergence and widespread use of electronic components, cars have transformed into more than just mechanical devices. Electronic control units (ECUs) connected through different buses across the vehicle are utilized to control and monitor vehicle conditions[1]. The integration of buses and ECUs in the in-vehicle network has made vehicle design, repair, and customization easier, bringing convenience to manufacturers, drivers, and after-sales services.

As the in-vehicle network expands in size, a large number of ECUs are connected together to form an automobile Electrical/Electronic (E/E) system which plays a key role in managing the vehicle's functionality, safety, comfort, and overall performance[2]. As vehicles have become more sophisticated, the complexity and importance of E/E systems have increased significantly. Typically, approximately 50 ECUs can be found in a small- and medium-sized vehicle[3]. Cutting-edge vehicles with rich new functionalities may contain up to 80 ECUs[4]. These ECUs together control many critical components of automobiles that people drive every day. As an example of automotive ECUs, body control modules are responsible for controlling various functions such as doors, seats, car locks, and airbags. On the other hand, Powertrain ECUs manage critical systems like the anti-lock brake system (ABS), engine control unit, and transmission control unit. Especially in recent years, driven by the desire for convenience, safety, efficiency, and a more enjoyable driving experience, along with the rapid advancement of technology in the automotive industry, the development trend of automotive industry is providing connectivity within an automobile and with the outside world[5]. Thus, these safety critical components are communicating

large amount of critical data every second over the in-vehicle network.

However, some ECUs with wireless connection to the outside world, such as telematics and radio data system, might be exposed to cyber attack[6]. The adversary can at first hack one of these surface ECUs, and then control or compromise many other safety critical ECUs through the in-vehicle network. As a consequence, the security threat of cyber attacks on the in-vehicle is worth the attention of the automotive industry.

Controller Area Network (CAN) is a widely used communication protocol and bus system in modern in-vehicle networks[7]. It provides reliable, high-speed, and low cost communication solution between various ECUs within an automobile. At present, CAN has become de facto standard for in-vehicle communication. Fig. 1.1 depicts a typical CAN bus consisting of three nodes. Every node is linked to one another via a traditional two-wire bus (CAN H and CAN L). The wires consist of a twisted pair with a nominal characteristic impedance of 120 Ω. The communication on this bus employs differential wired-AND signals. Note that ECUs are not directly connected with the CAN bus. Instead, a CAN transceiver is put between ECU and the bus. The main function of a CAN transceiver is to convert the digital signals from the CAN controller (local host) into the appropriate electrical signals suitable for transmission over the CAN bus, and vice versa. A CAN transceiver uses TXD and RXD to exchange information with its host – the ECU. Normally, a CAN transceiver doesn't contain any logic decoding CAN frames.

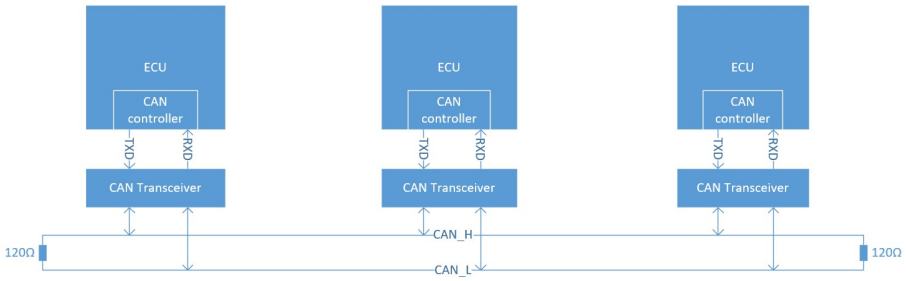

Figure 1.1: **Wiring of a typical CAN bus**

However, CAN is readily susceptible to attacks. Adversaries can easily get access to a variety of ECUs provided that they hack one of them on the same CAN bus[8]. If automobile manufacturers fail to defend ECUs from cyber attacks, the security of all nodes on the same CAN bus will be severely threatened. In 2015, Dr. Charlie Miller and Chris Valasek performed a remote attack against a 2014 Jeep Cherokee car[9]. The exploit chain they followed is listed here:

1. Identify target. The hacker scans the IP address in the range of where the target vehicle is known to reside, and use the result to identify one or more vehicles.

2. Exploit the Open Multimedia Applications Platform (OMAP) chip of the head unit. Once the hacker have an IP address of a vulnerable vehicle, he can run code on the radio and GPS module.

3. Flash the microcontroller (MCU) connected to the CAN bus with modified

firmware. The reflashed MCU provides now full access to the CAN bus from remote.

4. Reverse engineer formats of many normal and diagnostic CAN packets such as steering, braking, etc.

5. Perform cyber physical actions. The hacker can inject specially crafted CAN packets utilizing the modified firmware.

As a result, critical components of the car, including the braking system, were disabled. Additionally, the hacker turned the steering wheel 180 degrees while the car was driving in traffic, irrespective of the input of the driver[10]. Fig. 1.2 shows that hackers drove the Jeep car down to the ditch. These incidents have drawn the attention of security researchers to the security of CAN buses. As a result, Chrysler has issued a recall for 1.4 million vehicles as a result of Miller and Valasek's research[11]. Therefore, it is crucial to prioritize vehicle security and propose a comprehensive approach to secure the CAN bus and provide long-term protection within in-vehicle networks.

Figure 1.2: **Jeep car led astray by hackers[10].**

As a countermeasure against attacks on CAN buses, NXP launched the secure TJA115x CAN/CAN FD transceiver family that provides a hardware-based solution for securing classical CAN and CAN FD communication. The integrated filter enables hardware-based authentication for the local CAN communication and offers flooding protection (DoS prevention) from the host of TJA115x [12]. Different from software-based approaches, hardware-based TJA115x offers security solution with minimum system impact.

In December 2018, Controller Area Network Extra Long (CAN XL) was proposed as a new CAN variant following CAN FD[13]. CAN XL supports higher data rate up to 10Mbit/s and larger data fields with up to 2048 bytes. It also allows tunneling of Ethernet frames which is popular in high-speed in-vehicle networking. A broad spectrum of chip vendors, such as NXP, Bosch, Infineon, STMicroelectroincs, etc., are working on the research and development of CAN XL products[14].

Currently, these products with CAN XL protocol have not been released for mass production yet. Therefore, even though the field of CAN network security has gained significant attention from researchers worldwide, research on the security of CAN XL remains an uncharted territory. This emerging research area, where in-vehicle network security applied in CAN XL, has received limited attention from researchers thus far. CAN XL brings about several new functionalities that not only make some defense measures infeasible, but also enable new countermeasures against attacks. Moreover, TJA115x secure transceiver's approach to flooding defense isn't as attractive if its host is a zonal gateway. Hence, this thesis focuses on the defense against flooding attack on CAN XL networks. Exploring the security of CAN XL network presents an exciting opportunity to push forward the application of this new protocol.

1.2 Recent Studies on Security of CAN Buses

This section focuses on recent studies in academia on security of CAN buses, including ways to attack CAN network and the defense against them.

1.2.1 Attacks on CAN Network

Security issues were not adequately considered when the CAN protocol was designed over 30 years ago, a time when the Internet and cellular networks were not as ubiquitous as they are today. However, in recent times, several significant papers have initiated research in this field.

Published in 2007, the first paper about attacks on CAN bus networks was published[15]. Authors performed sniffing and replay attacks on automotive control systems. Hoppe et al. explored and classified security threats and countermeasures on the CAN bus in 2008[16].

The work of Stephen Checkoway et al. systematically analyzed the external attack surface of a modern automobile[17]. This paper identifies and examines various vulnerabilities in ECUs of an automobile. Fig. 1.3 illustrates the attack surfaces of a typical automobile that adversaries can exploit to their advantage. Authors pointed out that threat models requiring prior physical access are costly and not scalable. Hence, attacks being carried out remotely are more likely to be utilized by attackers. The authors classified attack surfaces into three categories: indirect physical acesss (CD, PassThru), short-range wireless access (Bluetooth), and long-range wireless access (Cellular).

In 2010, K Koscher et al published the first paper that comprehensively analyzed vulnerabilities of CAN network and ECUs experimentally[18]. Authors conducted a range of experiments both in the lab and in road tests, demonstrating that an adversary is capable of maliciously controlling a wide range of automotive functions while entirely ignoring driver input. This includes actions

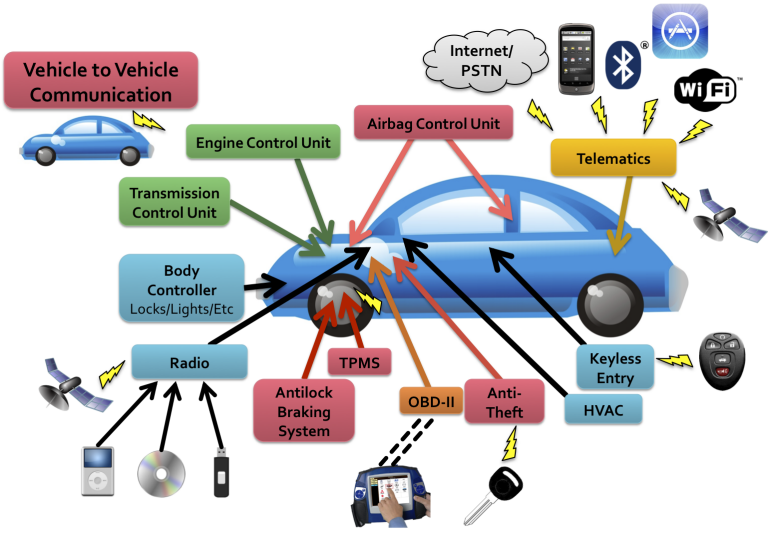

Figure 1.3: **Digital I/O channels appearing on a modern car. Colors indicate rough grouping of ECUs by function[17].**

like disabling the brakes, selectively braking specific wheels on demand, stopping the engine, and more. The adversary can even ensure that all traces of its presence are completely wiped out after a crash by implanting harmful code into a car's telematics unit.

Authors pointed out that CAN protocol has many inherent weaknesses, including:

1. **Sensitive to DoS.** The priority-based arbitration scheme of CAN protocol makes messages with high-priority IDs win the arbitration over low-priority IDs. An important type of attack is Denial-of-service (DoS) attack in which an adversary floods the bus with consecutive high-priority junk messages.

2. **Broadcast Nature**. Each CAN frame is broadcast over the CAN network. Possible attacks include snooping (eavesdropping), reverse-engineer, gather ID/information to inject a new package.

3. **No Authenticator Fields.** It is impossible to know which node sends a certain message frame. The adversary may send frames with anyone's ID. Possible attacks include data injection attacks because any single compromised component can be used to control all the other components on that bus.

Following [18], more researches on CAN attacks was conducted. One key research result was [19] in 2013. Miller et al demonstrated attacks on various components in two well-known cars: Ford Escape and Toyota Prius. For example, Fig. 1.4 depicts one of experiments which alters the speedometer to display a false value. The study showcased that the ECUs of the vehicle can enable remote execution of code through different interfaces like Bluetooth and telematics devices. In 2016, a book titled "The Car Hacker's Handbook: A

Guide for the Penetration Tester" summarized all known attack methodologies and attack processes that hackers can make use of[20].

Figure 1.4: **The compromised speedometer can be controlled to display any value[19]**

Since 2016, several attack strategies were found against CAN network. In 2016, a bus-off attack was proposed that takes advantage of the error-handling mechanism of the CAN protocol[21]. The bus-off attack can put the victim node into bus-off mode by injecting just a few messages at the same time of the victim's transmissions. It deceives an uncompromised ECU into thinking it is defective and eventually forces itself to shut down. In 2017, Palanca et al. proposed a selective denial-of-service attack against vehicles communicating via CAN[22]. This attack leverages the error management functionality of CAN protocol. Additionally, authors highlighted that such an attack cannot be detected at the frame level. The authors provided a practical demonstration of this concept by conducting experiments on vehicles and proved the simplicity of executing such an attack. In 2021, WeepingCAN was proposed that further improved the work of [21] which left detectable traces while carrying out the attack[23]. The new scheme is a stealthier bus-off attack that leaves no sign for intrusion detection.

1.2.2 Defenses Against CAN Attacks

After K Koscher et al initially studies the CAN vulnerabilities in a systematic way in 2010[18], countermeasures against such attacks gradually gained attention of researchers. Defenses against CAN attacks can be roughly classified into two categories: Security Architecture and Intrusion Detection System (IDS).

Security Architecture

Defense strategies against CAN attacks based on security architecture generally use cutting-edge solutions to protect against security threats with cryptographic message authentication code (MAC). Van Herrewege et al. implemented a backward compatible message authentication protocol called CANAuth on the CAN bus[24]. As the name suggests, they adopted a hash-based MAC to achieve

message authentication and replay attack resistance. Since the authentication information can't fit in the 8-bit payload of a classical CAN frame, the authors invented the CAN+ protocol which extends the available bits. Likewise, authors of [25], [26], and [27] utilized cryptography to authenticate each frame, tackling the weakness of CAN protocol. These strategies improved security of communication at the expense of extra data bits.

Intrusion Detection System

Intrusion detection system (IDS) is a security technology designed to monitor and detect unauthorized or malicious activities within a computer network or system. In the context of CAN security, the primary function of IDS is to monitors the traffic of CAN bus and identify suspicious or abnormal behavior that may indicate that one or more ECUs are compromised[28]. When an IDS detects a potential intrusion, it can generate alerts, send notifications to the driver, or take automated actions to block or mitigate the detected threat.

Miller Charlie and Valasek Chris proposed not only several attack strategies in [19], but also a mechanism to detect attacks on CAN network. In normal operation, the frequency of CAN packets is very predictable. The authors believe that if the frequency of frames with a particular ID vary drastically from well-known value, conclusion can be drawn that something strange is happening in the vehicle. In 2011, Muter Michael et al. suggested an intrusion detection method for the CAN bus based on the information theory[29]. The IDS calculate the entropy of CAN messages to identify the compromised ECU. In 2016, Song Hyun Min et al. measures the time intervals of CAN messages to detect an injection attack[30]. Due to its simplicity, the IDS has little impact on the operation of the existing system.

1.3 Contribution

In this thesis, our primary goal is to extend the design of the NXP TJA115x secure transceiver to CAN XL. Although the countermeasure against Denial of Service (DoS) attacks adopted by TJA115x performs well in Classical CAN and CAN FD protocol, it isn't suitable for CAN XL gateways in future zonal architecture. To address this limitation and better secure CAN XL networks, this thesis proposes a novel countermeasure against DoS attacks on CAN XL networks and implements it on a secure transceiver.

The main contributions of the thesis are:

1. Considering the structure of the future zonal IVN architecture, as well as important new fields of CAN XL, We devised a leaky buckets tree to differentiate between normal frames and flooding frames and dynamically control the traffic occupation of each type of frames. Moreover, we calculated all parameters of the leaky bucket based on a few inputs. Finally, we performed error analysis and estimated the implementation cost of the leaky buckets tree.

2. Two flooding prevention measures were proposed: disconnection measure and error flag measure to stop flooding frames from taking the bus,

ensuring the availability of communication. Division of scenarios is discussed including the origin of attack traffic and if error signalling of CAN XL protocol is enabled. We designed the hardware architecture and its state transition for flooding prevention. Finally, we compared these two schemes with regard to a set of metrics.

3. We built a special validation platform and associated Python script to demonstrate the defense against DoS attacks. Various types of normal and flooding CAN traffic are generated to validate the secure transceiver prototype.

1.4 Thesis Organization

The rest of this report is organized into seven chapters. The first four chapters are providing a background and an overview of the topics related to the area of research in this thesis. The middle two chapters explains the methodology of our proposed design of secure transceiver for CAN XL and validates its functionality with simulation and experiments. The last chapter concludes this thesis. The following list provides a more detailed description of the topics discussed in each chapter:

Chapter 2: It gives an overview of in-vehicle networking and highlights the concept of zonal architecture.

Chapter 3: It explains how CAN protocol works, including its physical layer, message frame format, etc. New functionalities specific to CAN XL are highlighted.

Chapter 4: It gives a brief introduction to the NXP TJA115x secure CAN transceiver and its security features, which lays a foundation for the new design proposed in this thesis.

Chapter 5: It explains the design of the secure transceiver for CAN XL, including the system architecture, flooding detection measures, and flooding prevention measures.

Chapter 6: It validate the design of this thesis in two aspects: software simulation and hardware validation.

Chapter 7: it provides a summary of this thesis and discusses potential future work.

Chapter 2

An overview On In-vehicle Networking

This chapter overviews in-vehicle networks and introduces the latest development of in-vehicle network architectures. Section 2.1 explains the importance of in-vehicle networks and highlights several major advantages. Section 2.2 explains the development of in-vehicle networking architectures and gives an detailed introduction of the future zonal architecture

2.1 Introduction to In-vehicle Networks

In the past 30 years, the underlying control systems of passenger automobiles have been rapidly developing. Mechanical devices are replaced by a large number of interconnected computers which monitor sensors, the driver, and passengers[18]. For instance, a typical luxury sedan now contains over 100 MB of data spreading across 50-70 independent ECUs. Those ECUs are communicating over in-vehicle networks[1].

In-vehicle networks refer to the interconnected systems and communication protocols used within modern vehicles. Thousands of electronic components and systems in a vehicle use in-vehicle networks to communicate with each other and exchange data and control signals [31]. The data exchanged can include sensor readings, control commands, diagnostic information, and multimedia content. Key advantages of in-vehicle networking include:

1. Reduced wiring complexity: traditional point-to-point wiring systems are complicated and costly. In-vehicle networks simplify the structure of wiring system so that the harness in vehicles can be lighter and more efficient[32].

2. Improved vehicle performance: In-vehicle networks allows optimized performance and coordination between different ECUs in a vehicle, such as engine control, transmission, suspension and anti-lock braking system (ABS). With exchange of data, the fuel efficiency and driving experience can be improved[33].

3. Scalability and upgradability: In-vehicle networks provide a foundation for incorporating new ECUs as they emerge, thus allowing the vehicle to evolve to support new technology, e.g. advanced driver assistance system (ADAS).

In recent years, the in-vehicle network plays a crucial role in vehicle systems. There are many key components and protocols in a typical in-vehicle network. Some of the most used ones include Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay and Ethernet[34]. These protocols and networks enable different electronic components and systems to communicate and share information, creating a connected ecosystem within the vehicle.

2.2 Recent Development of In-vehicle Networking

Amidst the ongoing evolution of automobile electronic/electrical (E/E) systems, a continuous stream of new-generation vehicles is being introduced, bringing forth plenty of innovative features. In addition, the electronic E/E systems rely more and more on software. The total number of lines of code in ECUs has increased from 4000 in 2000, to 200 million at present[35]. A large number of ECUs are added, which are wired into a larger system where they communicate with each other across the vehicle. This fact adds to the ever-severe wiring burden.

The wiring harness is second heaviest subsystems in many vehicles—as much as 150 lbs in highly contented vehicles—and it's very typical for the average vehicle to have 100–120 lbs of wire harness in the vehicle. Only the internal combustion engine is heavier. The harness comprises many thousands of individual cables, and the length can reach 1.5 miles.

Domain controller architecture is the current mainstream solution as the cost and complexity associated with the wiring harness becomes untenable for the industry. Designers of in-vehicle network are working towards combining ECUs so that fewer ECUs are able to control more of the vehicle's functions. As shown in Fig. 2.1, domain controller architecture rearranges functions of the vehicle system into a hierarchy where ECUs of each function are connected with a domain controller[35].

To further develop the capability of the in-vehicle network, and further reduce the cost associated with wiring harness needed to connect ECUs without compromising the key requirements of their functionalities, the industry is moving towards the next step of evolution: zonal architecture. As shown in Fig.2.2, zonal architecture is a novel architectural solution of vehicle E/E systems. In contrast to a domain architecture in which vehicle systems are grouped by function, zonal architecture offers a more efficient solution where functions within a vehicle are grouped by location into several zones. Each zone is responsible for the devices that are installed in a particular section of the vehicle and are connected to a locally installed zonal controller or gateway[36]. The zonal gateway serves as a local connectivity hub by combining several low-speed interfaces and transmitting data via a single high-speed Ethernet link to the backbone. In this way, the physical hardware and wiring in a vehicle is reduced.

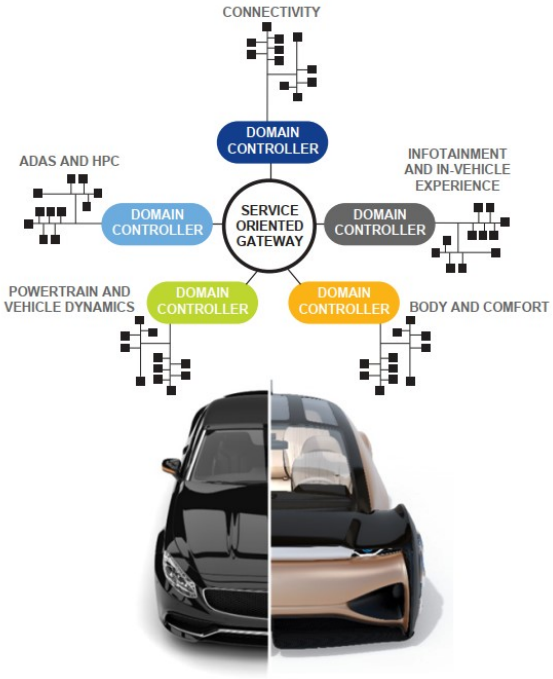

Figure 2.1: **Structure of the domain controller architecture[35]**

Additionally, the zonal gateway can conduct edge processing and transfer of pre-process data for the centralized computer, effectively reducing the workload and bandwidth requirements[37]. In the future, the vehicle's computation will be moving towards a software-defined service-oriented architecture (SOA) in which software-configurable hardware is widely adopted[38]. The central computer in the center of zonal architecture simplifies the functionality of ECUs by offloading their heavy computation.

In conclusion, zonal architecture reduces the number of ECUs, resolving bottlenecks in the automotive supply chain. The secure CAN XL transceiver proposed in this thesis is designed for nodes in future zonal architecture.

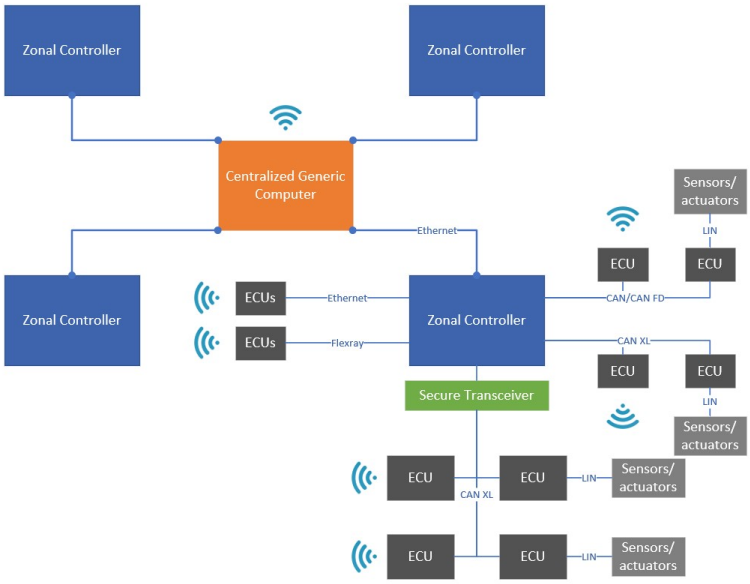

Figure 2.2: **CAN XL bus in Zonal architecture**

Chapter 3

Introduction to CAN Protocol

In this section, one important protocol in zonal architecture – Controller Area Network (CAN) as well as its upgrade versions: Controller Area Network Flexible Data-Rate (CAN FD) and Controller Area Network Extra Long (CAN XL) are briefly explained, including physical layer, frame format, bus arbitration, etc. Section 3.1 provides general information about the CAN protocol, including its development and main features. Section 3.2 explains the physical layer of CAN protocol. Section 3.3 explains frame formats of Classical CAN, CAN FD, and CAN XL. Section 3.4 discusses the transfer layer of CAN protocol. Section 3.5 explores three steps on the transition to CAN XL. Section 3.6 explains the object layer of CAN protocol.

3.1 General Information

Controller Area Network (CAN) is a multicast-based communication protocol proposed by Bosch GmbH in the mid-1980s [7]. It has characteristics of deterministic resolution of contentions, low cost, and simple implementation. CAN intends to provide a cost-effective communications bus for automotive applications but is today widely used also in factory and plant controls, medical devices, robotics, and drones.

Due to its low cost, light protocol management, deterministic resolution of the contention, and built-in features for error detection and re-transmission, CAN is an attractive solution for embedded control systems. Controllers supporting the CAN communication standard are today widely available as well as sensors and actuators that are manufactured for communicating data over CAN. CAN networks are today successfully replacing point-to-point connections in many application domains. The CAN network protocol comes with the following features:

1. Possibility of assigning priority to messages and guaranteed maximum latency times.

2. Multicast communication with bit-oriented synchronization

3. System-wide data consistency.

4. Multimaster access to the bus. This allows to add new nodes without reconfiguration of the already present nodes.

5. Error detection and signalling with automatic retransmission of corrupted messages

6. Detection of possible permanent failures of nodes and automatic switching off of defective nodes.

CAN FD (Flexible Data Rate) and CAN XL (Extra Long) are enhanced versions of the original CAN protocol (CAN 2.0). They provide increased data rates and improved performance compared to traditional CAN, enabling more efficient and faster communication in modern automotive ECUs. CAN FD was developed in 2011 and released in 2012 by Bosch, and now has been widely adopted in industry. Even though CAN XL is still at the stage of paper design, companies in automobile industry are developing CAN XL components for future market.

Like many networking protocols, the CAN protocol can be decomposed into 4 abstraction layers:

1. Physical layer: Bit encoding, Bit decoding, and Synchronization

2. Transfer layer: The majority of the CAN standard relates to the transfer layer, which functions as an intermediary between the physical layer and the object layer. Its main role involves receiving messages from the physical layer and forwarding them to the object layer. Within the transfer layer, various essential functions are performed, including message framing, arbitration, acknowledgment, error detection and signalling, and fault confinement.

3. Object layer: Message filtering and Message buffer arbitration

4. Application layer: Specific to applications

In the following sections, layers except for the application layer are explained in detail.

3.2 Physical Layer

CAN physical layer has two logical states: recessive and dominant. According to ISO-11898-2, CAN bus uses differential voltage to represent these two states, as shown in Fig. 3.1. CAN bus comprises two wires: CAN H and CAN L. In the recessive state, the differential voltage on CAN H and CAN L is less than the minimum threshold (< 0.5V receiver input or < 1.5V transmitter output). In the dominant state, the differential voltage on CAN H and CAN L is greater than the minimum threshold.

The signal type is encoded with Non-Return to Zero (NRZ), which gives the CAN bus high resilience to external disturbance. Typically, 0 is associated with the dominant state and 1 is with the recessive state. If multiple masters try to drive the bus state, the "dominant" configuration always prevails on the

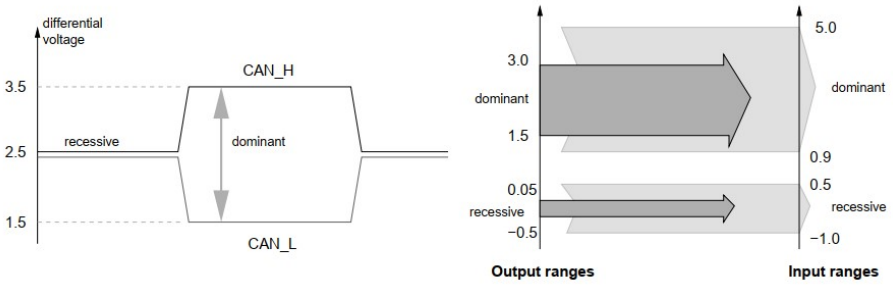

Figure 3.1: **Encoding of dominant and recessive bits[39]**

"recessive". If at least one node outputs a dominant bit, the status of the bus changes to 'dominant' regardless of recessive bit outputs from other nodes. Thus, the CAN bus acts as a wire AND gate to all nodes. This fact is the foundation of the nondestructive bitwise arbitration of CAN (see Section 3.4.2).

3.2.1 Bit Stuffing

To make every node agrees on the value of the bus, each node implements a synchronization protocol that uses falling edges to resynchronize nodes. Hence, long sequences of 0 or 1 that make the CAN bus lack bit transitions should be avoided to prevent drifts in the node bit clocks. This is the reason for the so-called "bit stuffing" rule, which forces a complemented bit in the stream after 5 bits of the same polarity such as "00000" or "11111" are transmitted. This is called "Dynamic bit stuffing". Stuffing bits are automatically inserted by the transmission node and removed at the receiving side before processing the frame contents. For instance, "000000" and "111111" will change to "0000010" and "1111101", respectively.

To limit the length of the CRC with the same hamming distance as Classical CAN, a new stuffing scheme called "Fixed bit stuffing" is introduced in CAN FD and CAN XL. In the CRC field of FD Frames, the stuff bits shall be inserted at fixed positions after each fourth bit of the CRC field. The value of such a fixed stuff bit shall be the inverse value of the bit preceding the fixed stuff bit. In CAN XL, fixed bit stuffing shall be applied in the Control Field, Data Field, and FCRC sequence. The stuff rate shall be 11, which means a fixed stuff bit shall be inserted after each 10th frame bit, counted from the DL1 bit in the ADS.

3.2.2 Switchable Operating Modes of CAN FD and CAN XL

In CAN FD and CAN XL, bit rate within a frame may optionally change between fast mode and slow mode. There are the following two operating modes in the frame of CAN FD and CAN XL:

1 Arbitration Phase, in which the Nominal bit time is used and bit rate is up to 1 Mbit/s

2 Data Phase, which can be either FD Data Phase or XL Data Phase depending on the protocol used.

- 2a FD Data Phase, in which the FD Data bit time is used. The bit time is shorter than the Normal bit time, thus accelerating the bit transmission. Bit rate can reach up to 5 Mbit/s
- 2b XL Data Phase, in which the XL Data bit time is used. The bit time is shorter than the Normal bit time, and in addition, the encoding of bit '0' and '1' on the bus has changed. Bit rate can reach up to 20 Mbit/s

Fig. 3.2 shows the transition between normal bit time and XL data bit time. The CAN XL transceiver of the transmitter sends the bus state level_1 (L1) instead of recessive and the bus state level_0 (L0) instead of dominant during the Data TX Mode. Note that the bus states level_1 and level_0 cannot overwrite each other like dominant overwriting recessive. If two nodes are driving concurrently level_1 and level_0, different receiving nodes will unpredictably detect either level_1 or level_0. Because of the feature of zero-threshold and perfect symmetry for both bit levels, the bit rate of XL data phase can be dramatically faster without compromising the robustness to interference.

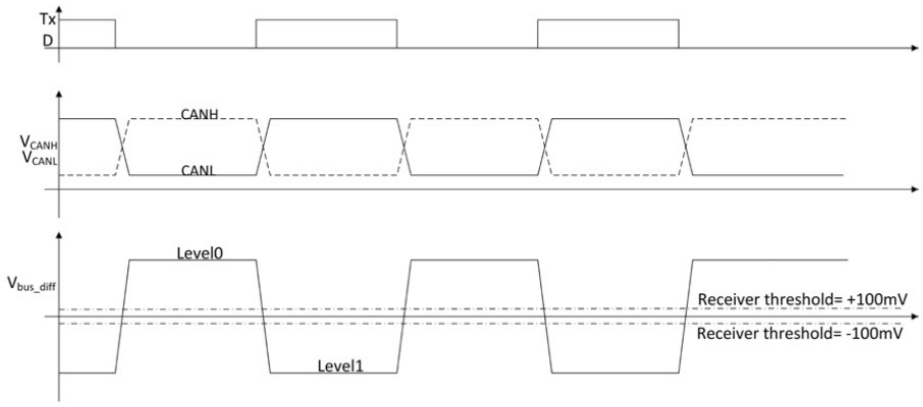

Figure 3.2: **CAN XL data phase level scheme**[**40**]

3.3 Message Frame Formats

CAN frames are used for CAN bus communication. Fig. 3.3 provides an overview of frames in Classical CAN, CAN FD, and in CAN XL formats. classical CAN has three versions: classical Base Frame Format (CBFF), CBFF remote frame and classical Extended Frame Format (CEFF); CAN FD has FD Base Frame Format (FBFF) and FD Extended Frame Format (FEFF), whereas CAN XL doesn't support remote frame and extended frame at all. In this thesis, we mainly focus on XLFF frames. We ignore remote frames and extended frame format and assume all CAN frames discussed are base format.

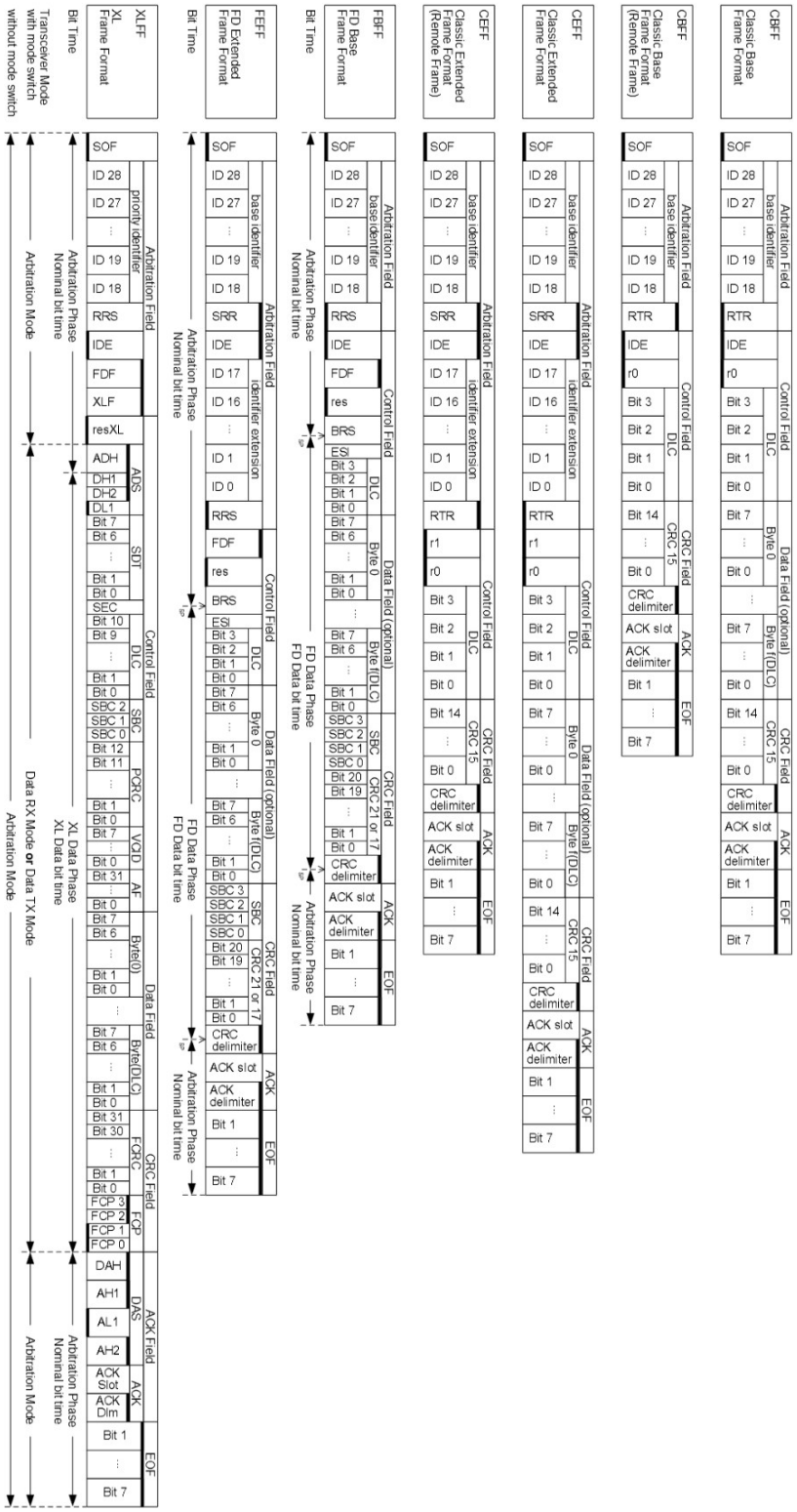

Figure 3.3: **Overview of frames in Classical CAN, CAN FD, and in CAN XL formats[39]**

3.3.1 Fields in CBFF

The explanations of all fields in a CBFF are listed as follows:

1. SOF: The Start of Frame. A 'dominant 0' indicates to other nodes the beginning of a frame.
2. ID (4 bits): Identifier is the priority of this frame in bus arbitration. Lower ID values have higher priority. It also contains information about the source and type of this frame.
3. RTR: Remote Transmission Request. It reveals whether a node sends data or asks another node for dedicated data.
4. Control (6 bits): The Control contains the Identifier Extension Bit (IDE) which is a 'dominant 0' for an 11-bit ID. FDF (also known as r0) is the bit indicating whether this is a FD frame (recessive) or classical frame (dominant). Additionally, Control field includes the 4-bit Data Length Code (DLC), which is the number of data bytes in the data payload (0 to 8 bytes)
5. Data (up to 64 bits): The payload, or data bytes, is made up of CAN signals that can be extracted and decoded to provide information. The width depends on the value of Control field which ranges from 0 to 8.
6. CRC: Contains CRC and CRC delimiter. CRC is 15-bit Cyclic Redundancy Check ensuring data integrity. CRC delimiter is a recessive bit marking the end of CRC field.
7. ACK: Acknowledgement. The ACK slot shows whether the node has acknowledged and correctly received the data. ACK delimiter is a recessive bit marking the end of ACK field.
8. EOF (7 bits): End Of Frame consisting of 7 recessive bits. The CAN frame comes to an end at the EOF. Moreover, if a CRC error or ACK error is detected, error flags are sent in the place of EOF.
9. IFS (3 or 11 bits): Inter-Frame Space consisting of 3 recessive bits. The bus becomes idle and accessible for all nodes after IFS. Note that an error-passive node (see 3.4.7 fault confinement), which has been transmitter of the previous frame, shall suspend the start of further frame transmissions for 8 bit times following intermission, which means the total IFS has 11 recessive bits in this case.

3.3.2 Fields Specific to FBFF

As depicted in Fig. 3.3, FBFF and XLFF introduces several new fields in addition to CBFF. New fields associated with CAN FD are:

1. RTR changes to RRS: In CAN FD, remote frames are not supported at all. Therefore, the Remote Request Substitution (RRS) is always dominant (0).
2. r0 changes to FDF: In Classical CAN, r0 is reserved and dominant (0). In CAN FD, it changes to FDF and recessive (1).

3. r1: This is the new reserved bit that now distinguishes between FD and XL. Always dominant (1) in FD. When an FD enabled receiver detects the res bit to be recessive instead of the expected dominant it shall, when protocol exception handling is enabled, detect a protocol exception event; when protocol exception handling is disabled, it shall treat this as a form error.

4. BRS: The Bit Rate Switch (BRS) can be dominant (0), meaning that the CAN FD data frame is sent at the arbitration rate (i.e. up to max 1 Mbit/s). Setting it to recessive (1) means that the remaining part of the data frame is sent at a higher bit rate (up to 5 Mbit/s).

5. ESI: The Error Status Indicator (ESI) bit is by default dominant (0), i.e. 'error active'. If the transmitter becomes 'error passive' (see 3.4.7 fault confinement) it'll be recessive (1) to indicate it is in error passive mode.

6. SBC: The Stuff Bit Count (SBC) precedes the CRC and consists of 3 gray-coded bits and a parity bit. The gray-coded bits is the number of stuff bits modulo 8. The SBC is added to improve communication reliability.

7. DLC / Data: Now a FDFF can support up 64 bytes data payload. To maintain a 4-bit DLC, CAN FD uses the remaining 7 values from 9 to 15 to denote the number of data bytes used (12, 16, 20, 24, 32, 48, 64).

3.3.3 Fields Specific to XLFF

New fields of CAN XL are:

1. ID moves to Priority ID & AF: In CAN XL, ID only represent priority of this frame. Information about the source and type of this frame moves to Acceptance Field (AF).

2. res changes to XLF: XL Field Identifier is the reserved bit of FD. Setting to recessive (1) indicates this frame is XL format.

3. rex XL: The resXL bit shall be dominant. It is reserved for future expansion of the protocol.

4. ADS and DAS (4 bits): Arbitration to data switch (ADS) and Data to arbitration switch (DAS) are places for proper switching between arbitration and data bitrate and back. ADS and DAS consist of fixed bit pattern.

5. SDT (8 bits): Service Data unit Type (SDT) is an essential feature for Zone Architectures, comparable with EtherType in Ethernet. It indicates the type of the protocol embedded in the data field. CAN XL frames can tunnel many IVN protocols, such as classical CAN and Ethernet in its payload, depending on the value of the SDT. The correspondence between SDT and other fields is shown in Table 5.7.

6. VCID (8 bits): Virtual CAN Network ID is comparable with VLAN in Ethernet which separate the CAN network into virtual networks.

7. DLC / Data: Now a XLFF can support up 2047 bytes data payload. CAN XL lengthens DLC to 11 bits to denote the number of data bytes used, which ranges from 0 to 2047.

8. PCRC (Preface CRC, 13 bits), FCRC (Frame CRC, 32 bits), FCP (Format Check Pattern, 4 bits): these are extra error detection fields safeguarding the integrity of the frame.

3.4 Transfer Layer

3.4.1 Bus Idle and Integration

The bus shall be considered to be idle by receivers and by error active transmitters when the third bit of intermission is seen recessive; by error passive transmitters (see 3.4.7 fault confinement) when the eighth bit of suspend transmission time is seen recessive; When the bus is idle, any node may access the bus for transmission. A frame pending for transmission during the transmission of another frame shall be started in the first bit following intermission.

CAN nodes shall enter the bus integration state after starting the protocol operation, during bus-off recovery, or (for nodes that are FD tolerant or FD enabled) after detecting the protocol exception state. CAN nodes shall leave the bus integration state when the idle condition (see 4.28) is detected.

3.4.2 Bus Arbitration

The bus arbitration of CAN protocol is both priority-based and non-preemptive, which means a message that is being transmitted cannot be preempted by messages with higher priority that has to wait until the transmission has started. A node seeking to transmit waits for the next available slot and initiates an arbitration phase by sending a start-of-frame bit if it finds the CAN bus to be in an idle state.

By serially transmitting the message's identifier (priority) bits in the arbitration slots, each node with a message to transmit can then begin competing for access to the CAN bus. Fig. 3.4 shows an example: assume node A and node B are transmitting frames with ID 000,1010,0000 and 000,1001,0000 respectively. At the 6th bit of the arbitration slot, node A sends recessive (1) and node B sends dominant (0). Due to the logical AND nature of the CAN bus, node A will detect dominant (0) on the bus at this bit time. At this time, node A realizes it loses the arbitration and thus gives up its transmission and becomes the receiver of that frame. Transmitters losing the arbitration will retry when idle is detected.

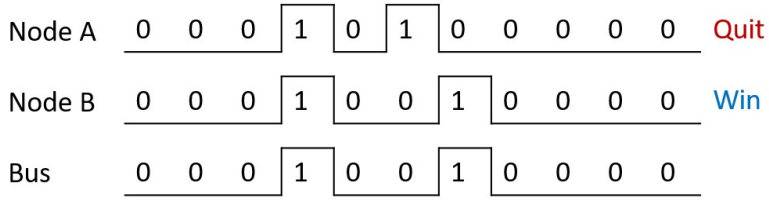

Figure 3.4: **Example of a bus arbitration process**

3.4.3 Acknowledgement

The ACK field of the CAN frame shown in Fig. 3.3 consists of 2 bits: ACK slot and ACK delimiter. In the ACK field, the transmitting node sends 2 recessive bits. The receivers are required to acknowledge the reception of the message by overwriting the ACK field from recessive to dominant if a correct message (after CRC checks) is received. When the transmitter detects recessive bit at RXD, an ACK error should be signalled with an error flag, if error signalling is enabled. Then the transmitting node may attempt to retransmit the message based on the retry strategy.

3.4.4 Error Management

The CAN protocol offers mechanisms for error detection, signalling, and self-diagnostics, including fault confinement techniques that stop malfunctioning nodes from impairing network performance.

The basic methods for error detection rely on each node's ability to monitor broadcast transmissions over the bus, whether it is a transmitter or a receiver, and to signal error circumstances coming from various sources. Any node identifying an error will flag corrupted communications. Such communications are automatically retransmitted after being aborted. If the CAN controller support XL format, due to its long data payload and existence of error management on higher levels, the number of automatic retransmissions of frames that are not transmitted successfully by any reason, except the loss of bus arbitration, shall be configurable to a dedicated number. The configured number shall be in the range

- from 0 (no retransmission)
- to at least 6 (6 retransmission attempts)
- and the highest number in this range may allow unlimited retransmission attempts, while unlimited means that retransmissions are only limited by the error counting mechanisms.

Error Types

There are 5 different error types, which are not mutually exclusive:

1. Bit error: Nodes also monitor the bus when it is sending a bit on the bus. When the bit value monitored is different than the bit value sent, a bit error is detected.
2. Stuff error: In light of the bit stuffing rule which forces a complemented bit in the stream after 5 bits of the same polarity, a stuff error is detected at the 6th consecutive occurrence of the same bit that is subject to stuffing.
3. CRC error: If the CRC computed by the receiver is different from the one sent by the transmitter in the message frame.
4. Form error: When a fixed-form bit field contains illegal bits.
5. Acknowledgement error: When a transmitter detects a recessive bit on the ACK slot.

Whenever one of this error is detected, An error flag should be transmitted if error signalling is enabled. The transmitting node should retransmit the message based on its retry strategy.

3.4.5 Frame Validation

Frame validation is the process by which a transmitting or receiving node examines incoming frames on the CAN bus to determine if they are valid. Transmitters should retransmit invalid frames. Receivers should ignore invalid frames, processing only valid frames further. The behaviour is different if error signalling is enabled or is disabled.

Behaviour When Error Management is Enabled

The frame shall be valid for receivers if there is no error until the last but one bit of EOF. The frame shall be valid for a transmitter if there is no error until the end of EOF.

Behaviour When Error Management is Disabled

Error management is optional in CAN XL. If it is disabled, the frame shall be valid for receivers, if there is no error detected until the ACK Slot and the ACK Slot is sampled dominant. The frame shall be valid for a transmitter, if the ACK Slot is sampled dominant.

3.4.6 Error Signalling

In the previous section, 5 types of errors were explained. Whenever one of these errors is detected by any node, this node shall transmit an error flag on the bus at the next bit (except for CRC error in which the error flag starts at the bit following the ACK delimiter). For an error active node, it is an active error flag consisting of 6 dominant bits. For an error passive node, it is a passive error flag consisting of 6 recessive bits. Fig. 3.5 depicts the format of the active and passive error flag. The first signal is an active error frame and the second one is a passive error frame.

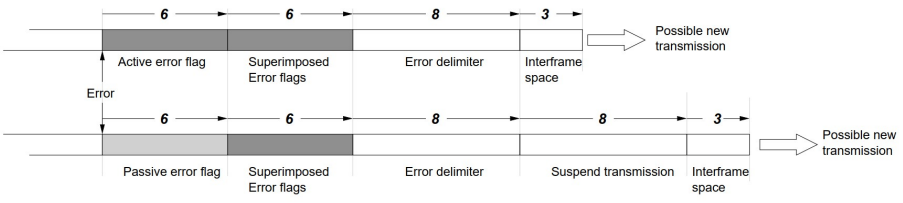

Figure 3.5: **The format of active and passive error frame [39]**

The active error flags violate the bit-stuffing rules. As a result, all other nodes on the bus detect a stuff error which also triggers the transmission of another error flag. Therefore, the error flag that can be monitored on the bus is the result of the superposition of different error flags transmitted by individual nodes. The total length of this sequence can vary from 6 to 12 bits. The passive

error flag sent by the error passive node actually has no impact on the bus, so it can't trigger the stuff error of other nodes.

CAN error signalling enables CAN nodes to invalidate incorrect messages and retransmit the erroneous frame. This guarantees that data will not be contaminated by transient local disturbances, such as noise. The transmitter instead tries to transmit the message again.

During XL data phase, however, as discussed in Section 3.2, it is impossible to overwrite the voltage of L0 and L1 like classical CAN and CAN FD. As a result, error frames are forbidden during XL data phase. In addition, considering the fact that the payload of CAN XL can be enormously long (up to 2047 bytes), sending error frames is significantly time-inefficient when it goes back to dominant/recessive levels at the end of CAN XL frames. Therefore, it is mandatory to disable error signalling and management if a CAN XL physical layer is used. It is optional for CAN XL nodes if a CAN FD physical layer is used (the frame format can still be XLFF).

3.4.7 Fault Confinement

The CAN protocol provides a fault confinement protocol that detects faulty nodes and prevents them from affecting the bus with their outputs. If error signalling is enabled, the protocol assigns each node two error counters: transmit error counter (TEC) and receive error counter (REC). According to the value of TEC and REC, each node has these three states:

1. Error active: Nodes in this state have both TEC and REC less than 128. Since they are considered to function normally, error active nodes can transmit on the bus and signal errors with an active error flag.

2. Error passive: Nodes in this state have TEC or REC more than or equal to 128. Error passive nodes are suspected of faulty behavior. They can still transmit on the bus, but their error signalling ability is restricted to passive error flags which have no effect on the bus.

3. Bus off: Nodes in this state have TEC more than 255. They are considered to be corrupted and disconnected from the bus.

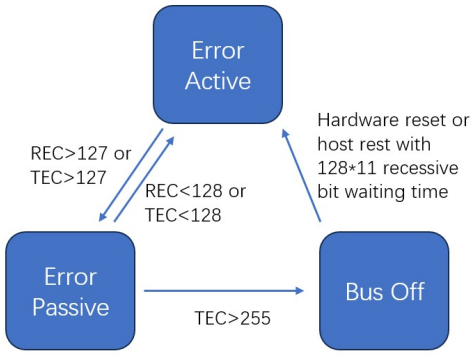

Figure 3.6: **The transitions between three states**

The transitions between the three states are shown in Fig. 3.6. Note that nodes that have disabled error signalling do not enter error-passive and bus-off state.

The complete rules of managing TEC and REC are rather complicated and thus omitted here. The most used rule is that if a transmitter sends an error flag its TEC increases by 8; If a receiver detects an error its REC increases by 1. Any successful transmission or reception of a message decreases TEC or REC by 1, respectively.

3.5 Transition to CAN XL Network

As the 3rd generation of CAN protocol, CAN XL has not supported by MCU vendors yet. There is no MCU available on the market that supports CAN XL. Therefore, in the process of the transition to pure CAN XL network, 3 stages should be experienced as more CAN XL compatible ECUs and transceivers are used in automobiles. They are explained chronologically as follows:

CAN XL Protocol Engine with FD Transceivers

As shown in Fig. 3.7, this is the first stage of the transition in which only part of ECUs are CAN XL compatible. Original Equipment Manufacturers (OEMs) don't need to redesign all modules at this stage of transition. Signal Improvement Capability (SIC) transceiver allows building more complex network topologies, such as stars and long stub lines, with significantly faster bit rates than conventional transceivers[41]. The network doesn't support XL data rate due to the lack of XL transceiver. The data bit rate is up to 5 Mbit/s.

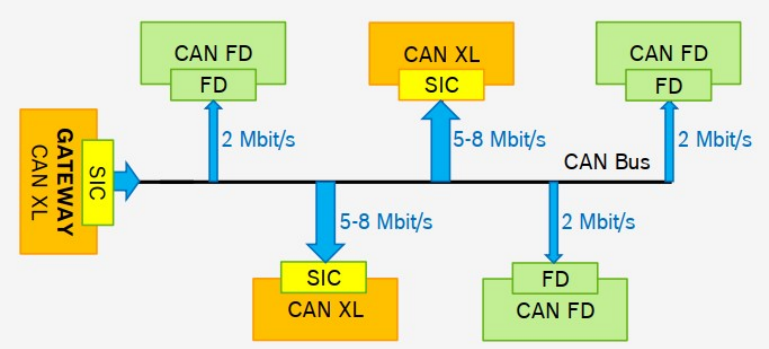

Figure 3.7: **CAN XL protocol engine with FD transceivers[42]**

Mixed Topology Full Speed CAN XL with CAN FD

As shown in Fig. 3.8, this is the second stage of the transition in which all transceivers are CAN XL compatible. XL frames and XL data rate are supported by part of ECUs, whereas CAN FD ECUs ignore XL frames and enter bus-integration state. The data bit rate is up to 20 Mbit/s in XL data phase and up to 5 M bit/s in FD data phase.

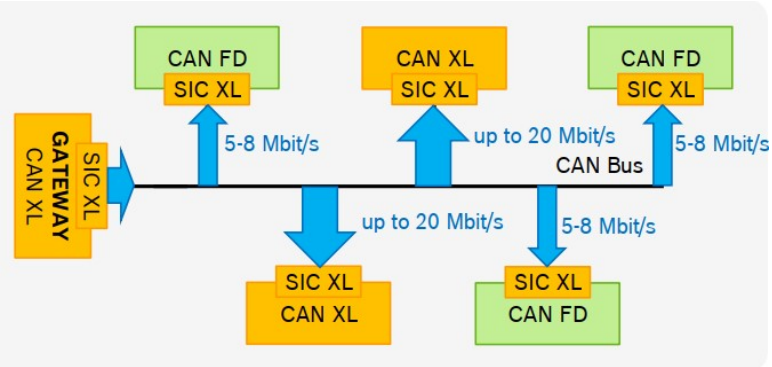

Figure 3.8: **Mixed topology full speed CAN XL with CAN FD[42]**

Pure CAN XL Network

As shown in Fig. 3.9, this is the final stage of the transition in which all components support CAN XL. The data bit rate is up to 20 Mbit/s in XL data phase. Since all nodes use XL transceivers, error signalling and fault confinement has to be disabled.

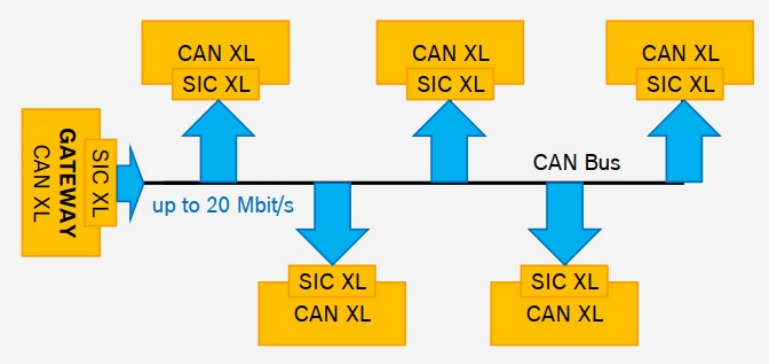

Figure 3.9: **Pure CAN XL network[42]**

3.6 Object Layer

MCUs in ECUs are equipped with CAN controllers (also called protocol engine) to interface with CAN transceivers. The CAN controller interfaces between MCU and CAN transceiver. It transmits and receives CAN frames under the control of the MCU. The transmit process and receive process are discussed as follows:

3.6.1 Transmit Process

Fig. 3.10 depicts how a CAN controller transmits CAN frames to a transceiver.

1. MCU loads message pending transmission into one of TX Mailboxes (MBs) or Tx First In, First Out (FIFO).

2. The TX arbitration process selects the next message to be sent to the TX Serial Message Buffer (SMB), depending on the configurable arbitration rule.

3. The message in SMB is sent out at TX. If the transmission fails, due to either loss of arbitration or transmit errors, retransmission of SMB shall start immediately. During retransmission of the message in SMB, all messages in Tx MBs and Tx FIFO should wait for the SMB to become empty. This implies that a low priority message continuously looses arbitration can stall high priority messages that are in the MB's. If the transmission is successful, SMB loads the next message from one of MBs or Tx FIFO.

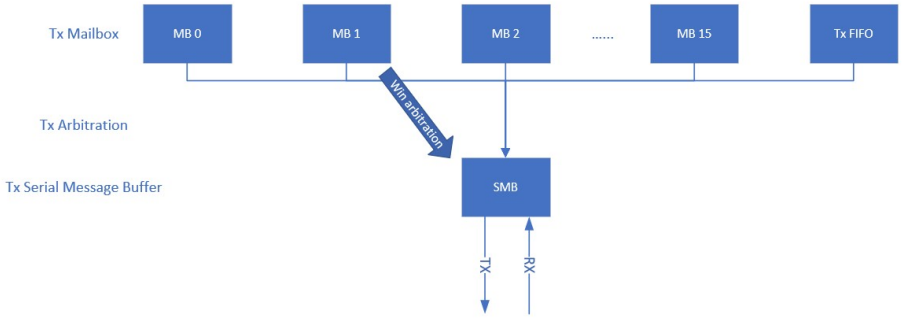

Figure 3.10: **Frame transmission in a CAN controller**

3.6.2 Receive Process

Fig. 3.11 depicts how a CAN controller receives CAN frames from a transceiver.

1. The ECU assigns priorities to each Rx MB and the Rx FIFO.

2. The CAN controller receives a frame on the bus and stores it into the Rx SMB.

3. The matching process scans the MB memory looking for Rx MBs and Rx FIFO programmed with the same ID as the one received from the CAN bus. Optionally, they may be configured with acceptance filters where the received ID is compared with a range of IDs.

4. The frame in the Rx SMB is copies to the matching Rx MB or Rx FIFO.

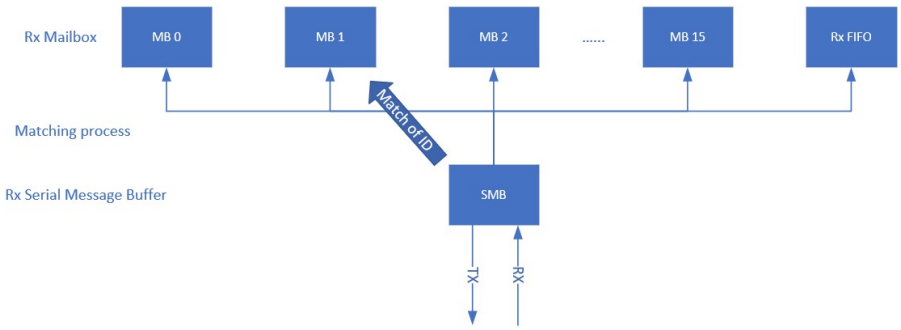

Figure 3.11: **Frame reception in a CAN controller**

Chapter 4

Protecting CAN Communication with Secure Transceiver

To better secure classical CAN and CAN FD communication without cryptography, NXP developed TJA115x CAN and CAN FD transceiver family[12]. This chapter explains its functionalities and advantages over current solutions. Section 4.1 discusses the system impacts with state-of-the-art solution. Section 4.2 introduces security features of the secure transceiver, encompassing flooding prevention, spoofing prevention, and tamper protection.

4.1 System Impacts with State-of-the-art Solution

At present, cryptographic methods like AUTOSAR SecOC are employed to authenticate messages. This involves adding a message authentication code (MAC) and a parameter that indicates the freshness of the message to the original unprotected payload, which is discussed in Section 1.2.2. This process ensures the validity and integrity of the message. Even though some microcontrollers integrate crypto accelerators to support intensive cryptographic calculations, this solution is not always the most efficient for secure CAN communication. Implementing these solutions on CAN network leads to the following system impacts:

1. **Secure communication startup delay**. After system power-on, or ignition, a certain time is required to agree and align the freshness property of the forthcoming authenticated message. This period delays the readiness of the authentication for the first critical message.

2. **Increase of busload and bandwidth**. Increase of the busload to transport additional parameters such as MAC and freshness value leads to additional bandwidth needs.

3. **Increase of processor load and functional latency**. An extra buffer or interrupt handling and communication to the hardware accelerator leads to extra processor load and functional latency.

4. **Need for key management**. Any cryptography-based solution relies on the usage of keys, either symmetric or asymmetric. The effort and the complexity of appropriate key management is sometimes tremendous.

4.2 Security Features of the Secure Transceiver

TJA115x secure transceiver provides prevention of spoofing, tamper, and flooding attacks without bandwidth overhead, major configuration changes, and lifecycle management needs. The biggest advantage of TJA115x is its minimum system impact - as long as no security incidents have been detected, the secure CAN transceiver behaves like a standard CAN transceiver. It can either acts as a standalone solution or work an extra layer of defense in addition to other security solutions such as cryptography. Fig. 4.1 shows the structure of TJA115x secure transceiver. It is located between the MCU host and the CAN bus. The message ID filtering compares frames from TXD with the TX Passlist and frames from the CAN bus with the BUS Blocklist. In addition, the Leaky Bucket monitors transmission rate of the MCU host at TXD.

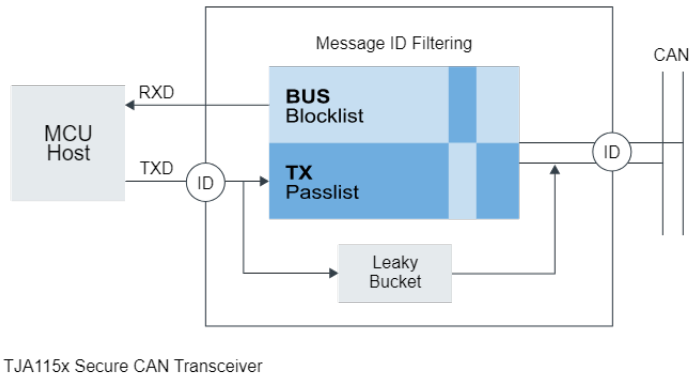

Figure 4.1: **Structure of TJA115x secure transceiver[12]**

4.2.1 Spoofing Prevention on Transmit Side

Spoofing attack means that the compromised ECU pretends to be another normal ECU, transmitting frames that should only originate from that ECU to deceive or manipulate a target system. For example, the compromised radio may transmit CAN frames with speedometer's ID to display false speed value on the dashboard.

A method for the secure CAN transceiver to safeguard the bus against a compromised ECU is by filtering messages based on CAN message IDs in the transmit path. If the ECU attempts to send a message with an ID that is not

originally assigned to it, the secure CAN transceiver can block its transmission to the bus. This action invalidates the message and avoid possible threat it may cause. To accomplish this, the manufacturer can configure a passlist of IDs for CAN message ID filtering.

4.2.2 Spoofing Prevention on Receive Side

As the receiver, the secure transceiver also has the ability to invalidate messages on the bus in case they are received with a CAN message ID assigned for transmission. This means each ECU with a secure transceiver can protect its own IDs if an adversary tries to spoof them. In the event that any spoofed ECU sends a message on the bus, the secure CAN transceiver of the legitimate ECU can actively invalidate that message by writing an active error frame to the bus. This can be done by configuring a blocklist in which all IDs exclusively belong to the ECU host are saved.

4.2.3 Tamper Protection

Tampering is a type of attack which maliciously modify the content of messages on the CAN bus. [21] and [23] are typical examples of tampering attack in which a recessive bit of a frame on the CAN bus is overwritten by a dominant bit from the attacker. Tampering can cause bit errors in a functional ECU and CAN network, and eventually disrupt the communication.

Invalidating messages on the CAN bus by error frames can also be used to prevent tampering. The secure CAN transceiver can check if a legitimate message, where its ECU wins the arbitration but stops its transmission in the data field (due to receiving a dominant bit while sending recessive), is completed by a tampering ECU. This serves as a clear indication that a compromised ECU tempers the transmission. This detection becomes especially valuable when the host ECU's CAN controller is not reporting errors during the error-passive state.

4.2.4 Flooding Prevention

Flooding attack, also known as Denial of Service (DoS) attack, means that the compromised ECU continuously transmits high priority frames without any break, see Fig. 4.2. All other nodes in the CAN bus have no chances to access the bus because their messages never win the arbitration with the flooder.

Figure 4.2: **Flooding attack**

Implementing sender-side measures to restrict an ECU's contribution to the bus load over a specific time interval can effectively prevent flooding the bus.

To implement this strategy, a leaky bucket mechanism is employed. This mechanism involves filling a bucket as messages are transmitted and continuously emptying it. If the bucket becomes full, any additional transmissions are halted, thereby preventing flooding. By implementing this flooding protection measure, the availability of the bus is enhanced, mitigating the risk of denial of service.

Chapter 5

Design of Flooding Protection for CAN XL

This chapter describes the design of flooding protection for the CAN XL secure transceiver, including the system architecture, flooding detection measures, and flooding prevention measures. Section 5.1 discusses why state-of-the-are countermeasures for Classical CAN and CAN FD doesn't work on CAN XL. Section 5.2 explains the system architecture of proposed design. Section 5.3 and Section 5.4 explain the structure of a leaky bucket and how its parameters are determined, respectively. Section 5.5 describes the proposed flooding detection measure and the leaky buckets tree. Section 5.6 proposes two flooding prevention measures, including disconnection measure and error flag measure. In Section 5.7, two flooding prevention measures are compared in terms of throughput, latency, and availability.

5.1 New Challenges of DoS Attack

In the cases of classical CAN and CAN FD, TJA115x secure transceivers simply block the transmitter when a large number of messages are transmitted by the host. This is implemented in a leaky bucket that limits the number of transmitted messages per unit of time. Whenever the leaky bucket is full due to continuously flooding traffic, secure transceivers in classical CAN or CAN FD will set the RXD of the host dominant for some fixed amount of time (e.g., 1 second) to prevent the host from taking part in the arbitration, thus making room for messages from other nodes.

However, in the future zonal architecture, this strategy is less attractive when the secure transceiver has the zonal controller as its host. In this scenario, messages from other buses, such as Flexray and classical CAN, can be forwarded via zonal controller to the CAN XL bus, as shown in Fig. 5.1. When an ECU in another bus is compromised and keep sending trash frames to the local bus via the gateway, if we still adopt the countermeasure of classical CAN or CAN FD, messages from other buses can no longer reach its destination since all gateway messages are blocked by the secure transceiver.

Fig. 5.2 categorizes traffic from the zonal gateway. The red arrow indicates flooding traffic sent by a compromised ECU. The green arrow indicates normal

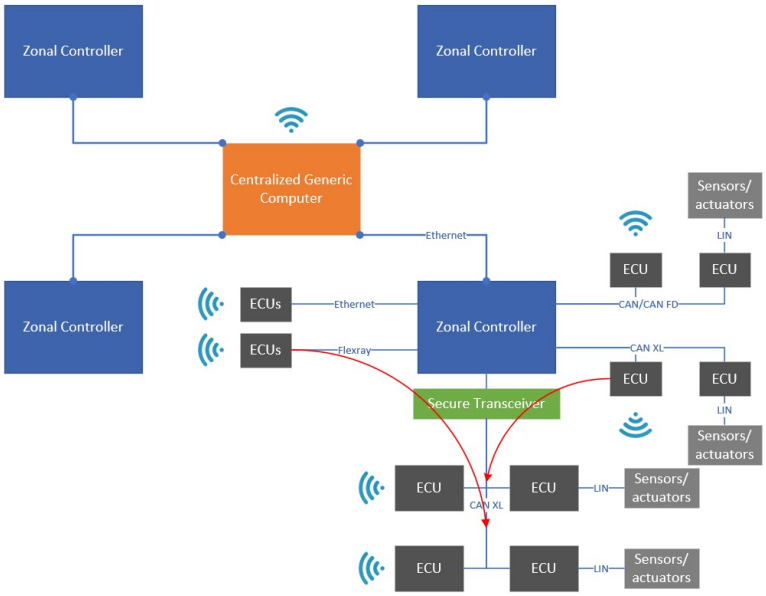

Figure 5.1: **Traffic via a zonal controller in zonal architecture**

traffic from a different ECU group. Our goal is to extend the flooding prevention functionality of TJA115x to CAN XL protocol that stop flooding traffic from attacking the local bus and allow all normal messages coming from innocent ECU groups to pass. Implementation of spoofing prevention and tamper prevention functionalities is beyond the scope of this thesis and is a direction for future work.

5.2 System Architecture

To accomplish our objective of flooding prevention, the secure transceiver should monitor whether the CAN network is under attack. Once a flooding attack is detected, it should take actions to stop the compromised node from occupying the bus, thereby enabling other nodes to communicate. In this thesis proposes a two-fold approach: the flooding detection measure to deal with the former issue, and the flooding prevention measure to deal with the latter.

The diagram in Fig. 5.3 demonstrates the system architecture of the secure transceiver. The protocol engine decodes incoming frames from TXD of the host MCU. When all important fields, including Priority Identifier (PID), Service Data unit Type (SDT), Virtual CAN network ID (VCID) and Acceptance Field (AF) are decoded, the CAN protocol engine selects and enables a leaky bucket in the leaky buckets tree according to identification information in these fields. Then the selected leaky bucket feedback to the protocol engine whether the selected leaky bucket overflows. If yes, the protocol engine will control the Flooding Prevention circuit to stop its transmission by disconnecting the Host MCU from the CAN bus.

Meanwhile, the protocol engine detects if the host is occupying the bus. As a part of the protocol engine, Fig. 5.4 depicts the Final State Machine (FSM)

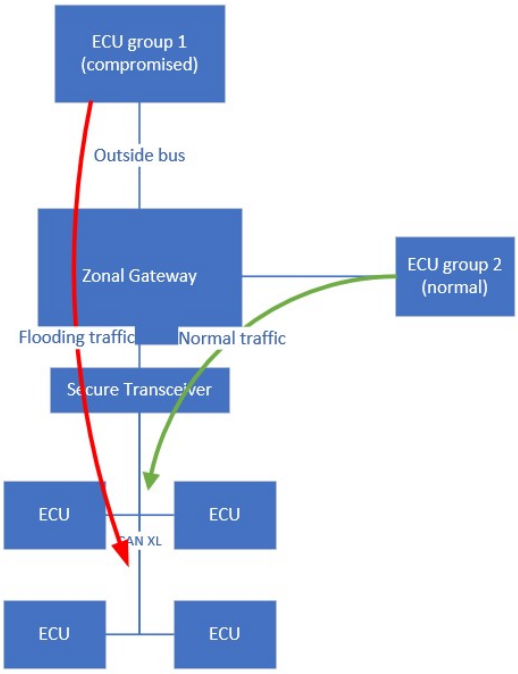

Figure 5.2: **Categories of gateway traffic**

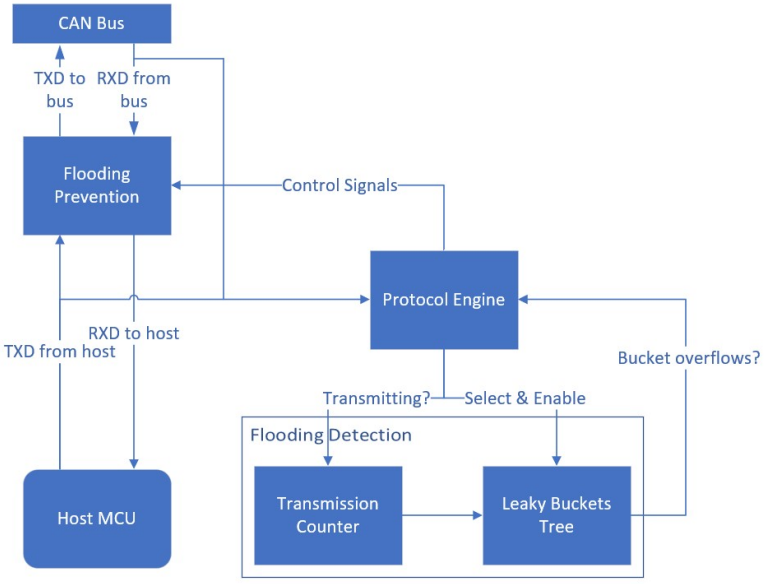

Figure 5.3: **System architecture of the secure transceiver**

for idle detection. In the IDLE state, if a dominant Start Of Frame (SOF) bit on TXD is detected, the FSM will enter the OCCUPY state. It returns to the IDLE state when 11 bits of recessive bits in a row are detected on RXD, which indicates the bus is idle again and the host no longer occupies the bus.

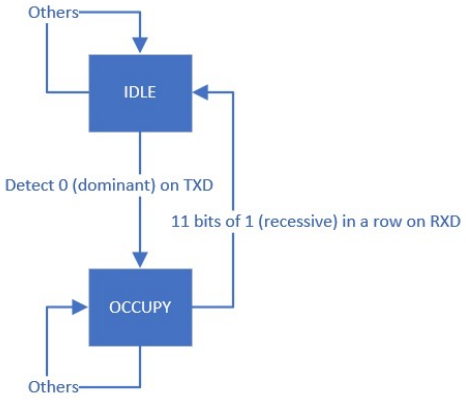

Figure 5.4: **FSM for idle detection**

Fig. 5.5 depicts the structure of the transmission counter. It comprises a cascading counter and the 2nd level counter increment only when the 1st level counter overflows. When the host is occupying the bus, the transmission counter keeps increasing. When the bus becomes idle, the transmission counter is cleared and stays at 0, and the final counter value is output to the leaky buckets tree.

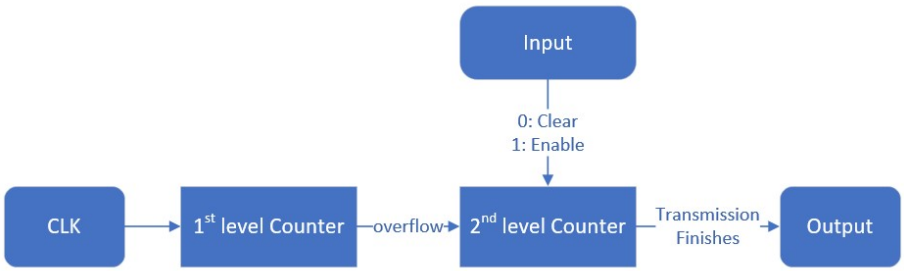

Figure 5.5: **Structure of the transmission counter**

5.2.1 Protocol Engine

To block all flooding traffic without stopping normal transmission of messages from the gateway and nodes outside of CAN XL buses, the secure transceiver should know which bus the frame comes from. This is achieved by the protocol engine. Apart from idle detection, the protocol engine finds out the origin of the gateway frame by buffering and analyzing some control fields including SDT, VCID and AF. Then it enables target buckets in the leaky buckets tree based on the origin of the frame. Buffers of the protocol engine is shown in Fig. 5.6

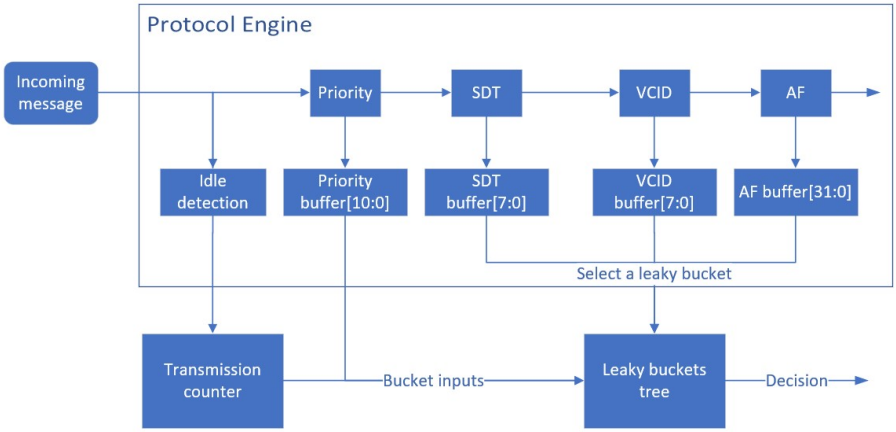

Figure 5.6: **Buffers of the protocol engine**

5.3 Leaky Bucket Structure

Limiting the overall bus load of an ECU per time unit is needed to prevent flooding the bus. The leaky bucket mechanism is implemented at the sender side to prevent flooding in the secure transceiver. The bucket is filled when messages are transmitted and emptied continuously. When the bucket overflows, attack is detected, and further transmissions are stopped by flooding prevention measures. Thus, flooding protection ensures the availability of the bus, preventing DoS of CAN networks. Messages with a low priority can be excluded from filling the bucket, as they would lose arbitration anyway and thus are unsuitable for a flooding attack. This allows diagnostic services such as software upload/download to exchange messages with low priority at a high rate without triggering the flooding protection.

Leaky buckets are basic component of our countermeasure against DoS attack. Even though their parameters might be very different, their basic structures are the same, as shown in Fig. 5.7.

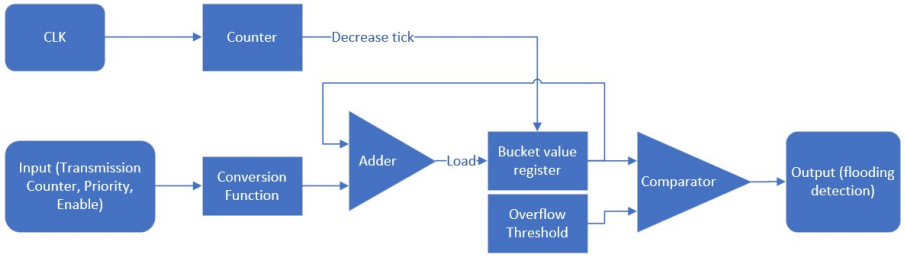

Figure 5.7: **Structure of the leaky bucket**

The leaky bucket receives the transmission counter value of the incoming frame and its priority from TXD of the host MCU. The conversion function then maps the counter value to the number added to the bucket value register. Essentially, this function excludes frames with lower priority from increasing the

counter. Frames with low priority or not enabled are mapped to 0 to enable e.g., firmware update messages which needs traffic of significant bandwidth.

The bucket value register increases by loading the adder value and decreases every incoming decrease tick. The decrease tick is the carry-out of the counter, which can be shared among multiple leaky buckets. The bucket value is compared with the overflow threshold. When it is higher than the designated threshold, the leaky bucket will conclude that flooding attack is happening and cut off further incoming frames.

5.4 Determination of Leaky Bucket Parameters

In this section, we define all parameters needed to build a leaky bucket. First initialization parameters are defined. Then derived parameters are calculated with regard to initialization parameters. Finally, examples are given that shows how parameters of a leaky bucket are determined.

5.4.1 Initialization Parameters

Initialization parameters are used to initialize the system before any further calculations can be performed. Once the initialization parameters are specified, the remaining parameters can be calculated using mathematical or computational methods.

There are two initialization parameters that should be specified: average traffic and time window. Let a be the percentage of bus time allowed for incoming frames in the time window t_w. If the occupation of the bus is equal to a in a time window t_w, the average increase rate of the bucket value will be equal to the decrease rate and the bucket value remains the same. Traffic exceeding a in time window t_w will increase the bucket value over time, and eventually the leaky bucket will overflow. Leaky buckets with higher a suggests the host is allowed to take more bus time on average.

5.4.2 Derived Parameters

Once all initialization parameters are specified, the remaining parameters can be calculated using computational methods.

Let t_b be burstiness, meaning the maximum time allowed for a burst of CAN XL frames before the leaky bucket overflows. As shown in Fig. 5.8, if at time 0 the counter value starts from 0 and the host keeps sending frames continuously, it will reach the overflow threshold at time t_b. We have

$$t_b = at_w \tag{5.1}$$

Then, two key parameters in the implementation of leaky buckets are calculated: increase and decrease rate of the bucket value. Let u denotes the increase rate of the counter when a frame is being transmitted, d is the decrease rate of the counter, and T is the threshold at which the leaky bucket overflows.

Consider the scenario in Fig. 5.8 when a burst of frames arrives which cause the counter increase from zero to overflow. Let t_{frame} be the bus time taken by the first frame (frame1). We have

$$(u - d)t_b + dt_{frame} = T \tag{5.2}$$

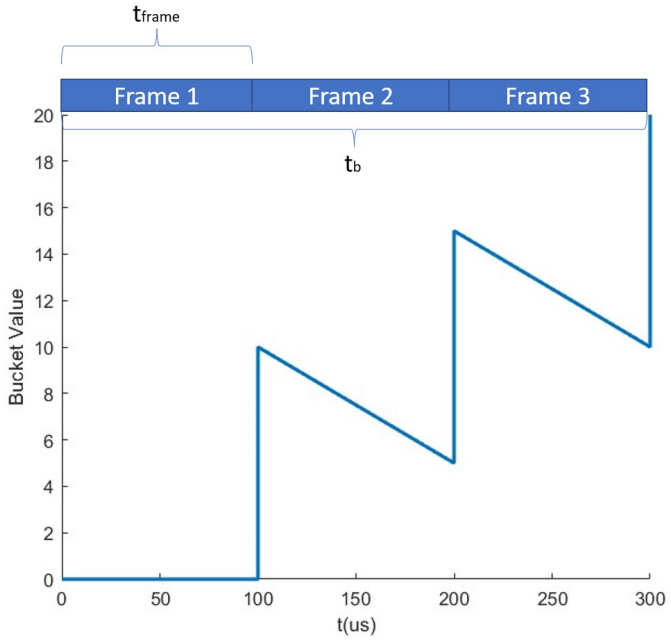

Figure 5.8: **Maximum burstiness until the leaky bucket overflows**

Assume t_b to be long enough to contain a large number of frames. Then ignore the effect of t_{frame} and we have

$$(u - d)t_b = T \tag{5.3}$$

Consider the scenario in Fig. 5.9 where frames arrive at a constant rate which causes no change in the counter value in the process. We have

$$at_w u - t_w d = 0 \tag{5.4}$$

Solve equations (5.1), (5.3), and (5.4), we obtain

$$u = \frac{T}{at_w(1-a)} \tag{5.5}$$

$$d = \frac{T}{t_w(1-a)} \tag{5.6}$$

In conclusion, we can determine d (decrease rate), u (increase rate) after specifying t_w (time window), a (average traffic) and T (overflow threshold).

5.4.3 Error Analysis

In the automotive environment, customers don't want to see the malfunction indicator light on when nothing is wrong. This may be caused by measurement errors which increase the bucket value above the threshold by accident. Therefore, error analysis should be carried out to guide engineers to design proper threshold value without the risk of overflowing the bucket.

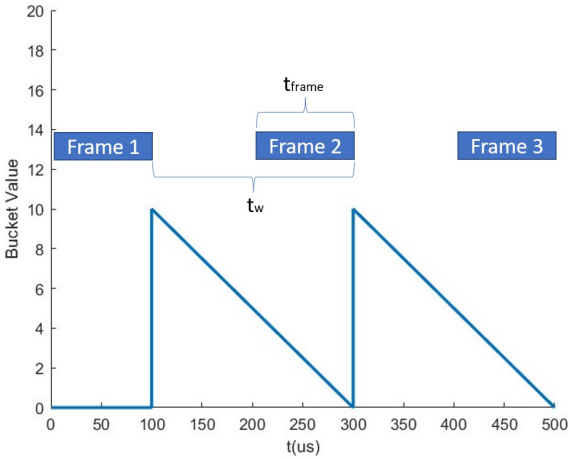

Figure 5.9: **Maximum occupation without changing the counter value**

To avoid being identified as flooding by the secure transceiver, the host needs to make sure it occupies the bus less than at_w in any time window t_w. As mentioned in Section 5.2, the secure transceiver measures the duration of a frame with the transmission counter which increment every time the 1st level counter overflows. This can save memory as the length of bucket value registers can be reduced. However, it also induces sampling error due to the long sample period of the transmission counter. Fig. 5.10 shows the measurement error of the transmission counter. Note that measurement results vary depending on the sampling start time and step used. This effect is further depicted in Fig. 5.11 where sampling results of the same time period are 6 and 5 for blue sampling point and red sampling point, respectively. Define the measurement step as Δt. The duration of the frame is $k\Delta t + t_{\text{last}}$, where k is an integer and t_{last} is the remaining time not able to fit in Δt that follows a uniform distribution $t_{\text{last}} \sim U(0, \Delta t)$. The probability distribution of measurement result, denoted as r, is expressed in equation 5.7.

$$\begin{cases} r = k\Delta t, & p = 1 - \dfrac{t_{\text{last}}}{\Delta t} \\ r = (k+1)\Delta t, & p = \dfrac{t_{\text{last}}}{\Delta t} \end{cases} \tag{5.7}$$

Therefore, the distribution of measurement errors, denoted as e, is shown in equation

$$\begin{cases} e = -t_{\text{last}}, & p = 1 - \dfrac{t_{\text{last}}}{\Delta t} \\ e = \Delta t - t_{\text{last}}, & p = \dfrac{t_{\text{last}}}{\Delta t} \end{cases} \tag{5.8}$$

Probability Analysis

To derive the distribution of e that depends on another random variable t_{last}, the joint probability density of e and t_{last} has to be calculated at first. Let

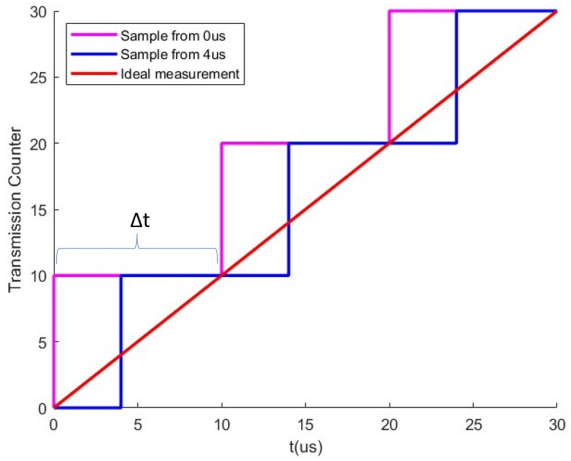

Figure 5.10: **Measurement error of the transmission counter**

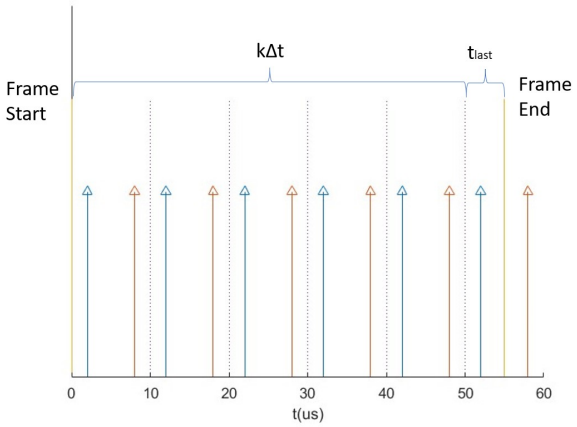

Figure 5.11: **Effect of different sample phases on the error**

random variable X denote t_{last} and Y denotes e. Equation 5.8 can be rewritten in the form of conditional probability density function as

$$f_{Y|X}(y|x) = \frac{f_{X,Y}(x,y)}{f_X(x)} = (1 - \frac{x}{\Delta t})\delta(y+x) + \frac{x}{\Delta t}\delta(y - \Delta t + x) \tag{5.9}$$

The marginal probability density function of X is

$$f_X(x) = \begin{cases} \frac{1}{\Delta t} & , 0 < x < \Delta t \\ 0 & , \text{Others} \end{cases} \tag{5.10}$$

From equation 5.9 and 5.10, we can obtain the joint probability density function of X and Y as

$$f_{X,Y}(x,y) = \begin{cases} \frac{\Delta t - x}{\Delta t^2}\delta(y+x) + \frac{x}{\Delta t^2}\delta(y - \Delta t + x) & , 0 < x < \Delta t \\ 0 & , \text{Others} \end{cases} \tag{5.11}$$

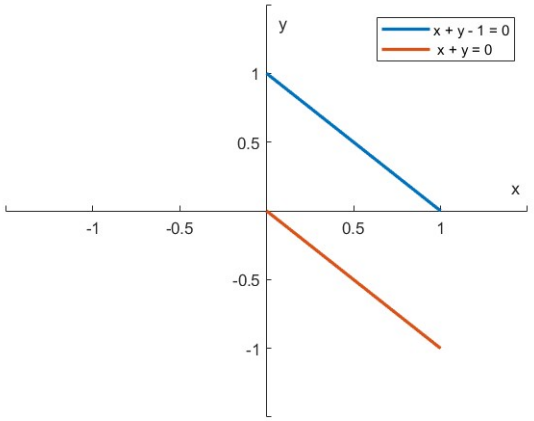

Figure 5.12: **Joint probability density function $f_{X,Y}(x,y)$ (Assume $\Delta t = 1$)**

Fig. 5.12 depicts the joint probability density function $f_{X,Y}(x,y)$ where points on two lines have probability density not equal to zero. Depending on the range of y, the marginal probability density function of X is calculated as

$$f_Y(y) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)dx = \begin{cases} \frac{\Delta t - y}{\Delta t^2} & , 0 \le y < \Delta t \\ \frac{\Delta t + y}{\Delta t^2} & , -\Delta t < y < 0 \\ 0 & , \text{Others} \end{cases} \tag{5.12}$$

The curve of $f_Y(y)$ is shown in Fig. 5.13. Its expectation and variance are

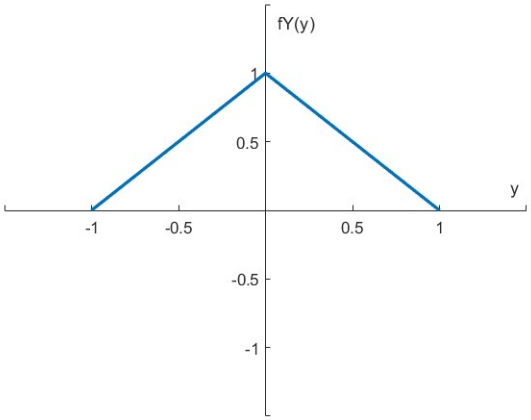

Figure 5.13: **Marginal probability density function $f_Y(y)$ (Assume $\Delta t = 1$)**

$$E(Y) = 0 \tag{5.13}$$

$$D(Y) = \frac{1}{6}\Delta t^2 \tag{5.14}$$

The definition of average traffic and time window states that the host is able to occupy the bus time at most at_w in a time window t_w. Assume there are N frames within t_w. The total error of the transmission counter is the sum of N independent errors. Note that the value of N is in the several hundred range. According to the central limit theorem, the total error follows a normal distribution as shown in equation 5.15.

$$\sum_{i=1}^{N} e_i \sim \mathcal{N}(0, \frac{1}{6}N\Delta t^2) \tag{5.15}$$

If we use $\pm 3\sigma$ as the range of the normal distributed random variable, the range of the total error can be represented as

$$|\sum_{i=1}^{N} e_i| < \frac{\sqrt{6}}{2}\sqrt{N}\Delta t \tag{5.16}$$

Note that u means the increase of transmission counter per second. Since the bucket value can only be an integer, the measurement step is

$$\Delta t = \frac{1}{u} \tag{5.17}$$

Let t_{frame} denote the duration of a frames. We have

$$at_w = Nt_{\text{frame}} \tag{5.18}$$

If we consider the worst case, N should be as large as possible, and t_{frame} should be as small as possible, according to equations 5.16 and 5.18. Let t_{fmin} denote the duration of the shortest frame. Combining equations 5.5, 5.17 and 5.18, we can further calculate equation 5.16 as

$$|\sum_{i=1}^{N} e_i| < \frac{\sqrt{6}}{2}\sqrt{\frac{at_w}{t_{\text{fmin}}}}\frac{at_w(1-a)}{T} \tag{5.19}$$

In most cases, designers are primarily concerned with the relative error, e.g., the measurement error of bus time can't be more than 5% of the maximum bus time allowed in a time window, i.e. burstiness. Let the percentage be p, we have

$$pat_w = \frac{\sqrt{6}}{2}\sqrt{\frac{at_w}{t_{\text{fmin}}}}\frac{at_w(1-a)}{T} \tag{5.20}$$

Solve equation 5.20, the minimum threshold is calculated as

$$T = \frac{\sqrt{6}}{2}\sqrt{\frac{at_w}{t_{\text{fmin}}}}\frac{1-a}{p} \tag{5.21}$$

Worst Case Analysis

In some safety critical scenarios, high accuracy is required, and memory resource can be sacrificed to some extent for better accuracy. In this case, we consider only the worst-case scenario where all errors are the maximum and in the same direction. Note that it is a conservative estimate and very unlikely to happen.

From equation 5.8, we know the biggest possible error of one measurement is

$$-\Delta t < e < \Delta t \tag{5.22}$$

In the worst case, the range of total error of the transmission counter is

$$|\sum_{i=1}^{N} e_i| < N\Delta t \tag{5.23}$$

In the same way as equation 5.21, a more conservative threshold is derived as

$$T = \frac{at_w(1-a)}{t_{\text{fmin}}p} \tag{5.24}$$

5.4.4 Example of Parameters Calculation

Here, we assume arbitration-phase $f_a = 500$ kbit/s, XL data-phase $f_d = 10$ Mbit/s. According to Fig. 3.3, we have the relation between t_{frame} and DLC (denoted as L) is:

$$t_{\text{frame}} = 1.1(\frac{32}{f_a} + \frac{123+8L}{f_d}) \tag{5.25}$$

Equation 5.25 considers extra 10% bits in a CAN XL frame that should be stuffed for synchronization. When $L = 0$, we obtain the frame shortest $t_{\text{fmin}} = 67.7$us.

In the very beginning, designers should specify initialization parameters: average traffic a and time window t_w. t_w determines the maximum delay allowed in the CAN bus, which should be less than 1 second. Also, the degree of permissible error p should be clearly defined. Here we specify them in Tab 5.1. Then use equation 5.21 to calculate the minimum threshold as $T = 538.07$. Since T should be an integer, we set it as $T = 539$. According to equations 5.5 and 5.6, derived parameters can be calculated as $u = 11977.7$ and $d = 1197.7$.

a	t_w	p
10%	0.5s	5%

Table 5.1: **Initialization parameters**

Considering the hardware implementation, u and d should be converted as the number of clocks that one tick of the counter takes, which are denoted by n_u and n_d, respectively. To synchronize with CAN frames properly, the sample rate of the protocol engine should be over 15. Therefore, the system clock is set to be $f_c = 15 * f_d = 150$MHz. Following equations 5.26 and 5.27, the number of clocks for one increase and decrease tick are $n_u = 12523$ and $n_d = 125232$.

$$n_u = \frac{f_c}{u} \tag{5.26}$$

$$n_d = \frac{f_c}{d} \tag{5.27}$$

Similarly, in the worst case scenario we obtain from equation 5.24 that $T = 10724$, and from equations 5.26 and 5.27 that $n_u = 629$ and $n_d = 6294$.

To ensure the robustness of our approach, we perform multiple recalculations using various initialization settings to represent different scenarios. We summarize sets of initialization settings in Table 5.2. The resulting normal and conservative derived parameters are summarized in Tab 5.3 and Tab 5.4, respectively.

	Design specifications				Initialization parameters	
Number	fa	fd	p	fc	a	tw
1	500k	10M	5%	150M	0.1	0.5
2	500k	10M	5%	150M	0.2	0.5
3	500k	10M	5%	150M	0.1	1
4	500k	500k	5%	50M	0.1	0.5
5	500k	10M	1%	150M	0.1	0.5

Table 5.2: **Sets of configurations**

Memory Usage Estimate

Memory is a critical resource of chips. Estimating memory usage helps in planning the overall resource allocation for the chip and assessing its feasibility.

Since the secure transceiver should be configurable to work in different scenarios, the register width should be designed to fit in the maximum number

	Normal derived parameters					
Number	tfmin	T	u	d	nu	nd
1	84us	539	11978	1198	12523	125232
2	84us	677	8462	1692	17725	88626
3	84us	761	8456	846	17740	177398
4	341us	267	5933	593	8426	84269
5	84us	2691	59800	5980	2508	25083

Table 5.3: **Sets of normal derived parameters**

	Conservative derived parameters				
Number	T	u	d	nu	nd
1	10724	238311	23831	629	6294
2	19064	238300	47660	629	3147
3	21447	238300	23830	629	6294
4	2640	58667	5866	852	8523
5	53617	1191489	119149	126	1259

Table 5.4: **Sets of conservative derived parameters**

possible, including normal and conservative configurations. Consider configurations in Tab 5.2, biggest parameters and their widths are listed in Tab 5.5.

Parameter	T	nu	nd
Maximum number	53617	17740	177398
Width (bits)	16	15	18

Table 5.5: **Width of parameters**

To save the chip area as much as possible, the bit width of registers should be set to the smallest value that can hold the maximum number shown in Tab 5.5. Therefore, widths of registers in the transmission counter and leaky buckets are designed as in Tab 5.6. Note that the bucket value register should be 1 bit wider to handle further increase after overflow.

5.5 Flooding Detection Measures

In previous sections, leaky bucket is introduced as an important component of our design. To tackle flooding attacks on CAN XL bus, the secure transceiver should be able to detect:

1. Whether the CAN XL bus is under DoS attack

2. Where flooding traffics come from

In this section, the classification of CAN XL frames from the gateway host is discussed. Then, the leaky buckets tree is proposed to detect and trace flooding traffics.

	Register	Width
Transmission counter	1st level counter	15
	2nd level counter	16
Leaky bucket	Counter (Prescaler)	18
	Bucket value register	17
	Overflow threshold register	16

Table 5.6: **Size of registers**

5.5.1 CAN XL Frames Classification

According to Section 5.2.1, the protocol engine determines the type of traffic of gateway frames using control fields of CAN XL frames. This is because the SDT field defines how a receiving node interprets the VCID field, the AF field and the Logic Link Control (LLC) data field. Table 5.7 defines the values of SDT. Table 5.8 shows the interpretation of all standardized SDT values.

Value	Description
00h	Reserved
01h	Content-based addressing
02h	Source and destination address
03h	Classical CAN/CAN FD mapped tunneling
04h	IEEE 802.3 (MAC frame) tunneling
05h	IEEE 802.3 (MAC frame) mapped tunneling
Others	Reserved

Table 5.7: **SDT values**

SDT	VCID	AF	Data Field
01	x	Message ID	CAN Data
02	x	Dest. And Source Address	CAN Data
03	x	CAN Frame ID	Classical/CAN FD Frame
04	x	x	Ethernet Frame
05	VLAN ID	Truncated Dest. MAC Address	Ethernet Frame

Table 5.8: **Interpretation of standardized SDT values**

Based on the values of SDT, VCID and AF, the leaky buckets tree is created to control the bus occupation of each type of CAN XL frames individually. Next the classification of CAN XL frames in terms of the SDT value is discussed in detail.

Content-Based Addressing and Source and Destination Address

When the zonal gateway transmits CAN XL frames with SDT = 0x01 or SDT = 0x02 to the CAN XL bus, the leaky bucket tree should identify that these frames come from ECUs in another CAN XL or Flexray bus which communicates to the destination CAN XL bus via the zonal gateway, as shown in Fig. 5.14. The zonal gateway receives and interprets these frames and forwards them to the destination CAN XL bus.

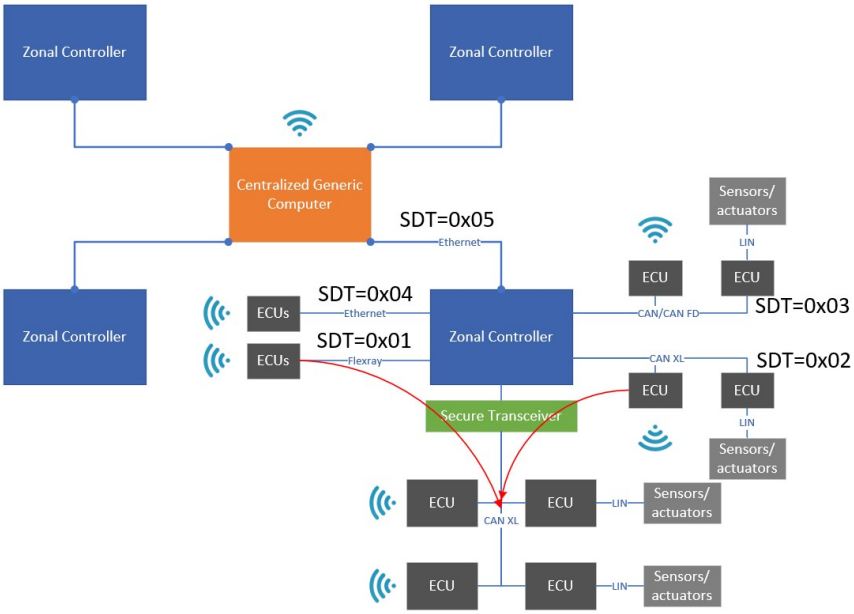

Figure 5.14: **Frames with SDT = 0x01 or 0x02**

The content-based addressing with SDT = 0x01 indicates that a content-ID is mapped into the AF field. The source and destination address with SDT = 0x02 are mapped into higher half and lower half of the AF field, respectively. Therefore, the source of CAN XL frames with SDT = 0x01 or 0x02 is indicated by the value of the AF field. For frames with SDT = 0x01, the Content ID in AF field indicates where they are from. Therefore, each group of Content ID should correspond to one leaky bucket. Likewise, the Source Address of frames with SDT = 0x02 indicates their sources. Therefore, each group of Source Address should correspond to one leaky bucket for frames with SDT = 0x02.

Classical CAN/CAN FD Mapped Tunneling

The zonal gateway sends CAN XL frames with SDT = 0x03 when a ECU from another Classical CAN/CAN FD bus sends frames to the destination CAN XL bus via the gateway, as shown in Fig. 5.15. The gateway receives these frames and tunnels them into the payload of CAN XL frames.

The Classical CAN/CAN FD 11(29)-bit identifier is mapped into the AF field of CAN XL frames with SDT = 0x03. In Classical CAN and CAN FD, the identifier field shows the identity of the transmitter. Hence, the source of this type of CAN XL frames is indicated by the value of identifier mapped into the AF field. In the leaky buckets tree, each group of Source Address should correspond to one leaky bucket.

MAC Frame Tunneling or MAC Frame Mapped Tunneling

The zonal gateway sends CAN XL frames with SDT = 0x04 or 0x05 when a ECU or the centralized generic computer that has Ethernet connection to the

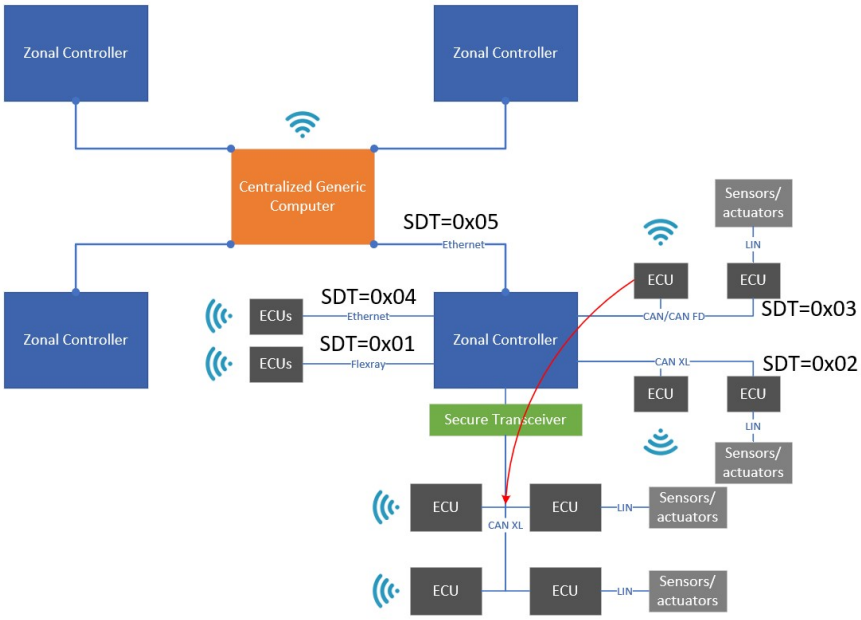

Figure 5.15: **Frames with SDT = 0x03**

zonal gateway sends message to the destination CAN XL bus via the gateway, as shown in Fig. 5.16. In this scenario, the transmitter and receiver should be in the same Virtual LAN (VLAN). The zonal gateway receives Ethernet frames and tunnels them into the payload of CAN XL frames.

Since neither VCID nor AF field provides information about the transmitter, all frames with SDT = 0x04 should share one leaky bucket. This isn't a good strategy as one compromised Ethernet ECU also prevents other uncompromised ECUs from sending Ethernet messages to the victim CAN XL bus.

The VLAN ID is mapped into the VCID field of CAN XL frames with SDT = 0x05. Therefore, the source of this type of CAN XL frames is indicated by the value of VCID field. Each VCID value should correspond to one leaky bucket. The value of AF field (contains part of destination address field) isn't helpful and should be ignored.

Flooding Traffic Generated by the Compromised Gateway

The possibility that the gateway itself is hacked and start flooding the connected CAN XL bus shouldn't be ignored. In this case, classifying the flooding message with regard to SDT, AF, etc. is meaningless since a smart hacker can easily randomize all these values. Moreover, a compromised gateway may not necessarily transmit valid frames. It is possible that flooding frames are truncated halfway or even random bits.

To make sure at least other nodes in the CAN XL bus can still communicate, a general leaky bucket should be implemented before checking specific fields e.g., SDT. If the general leaky bucket is full at this point, then the gateway is assumed to be compromised and every message sending from the gateway is unreliable. Here, the secure transceiver can safely block all traffic from the

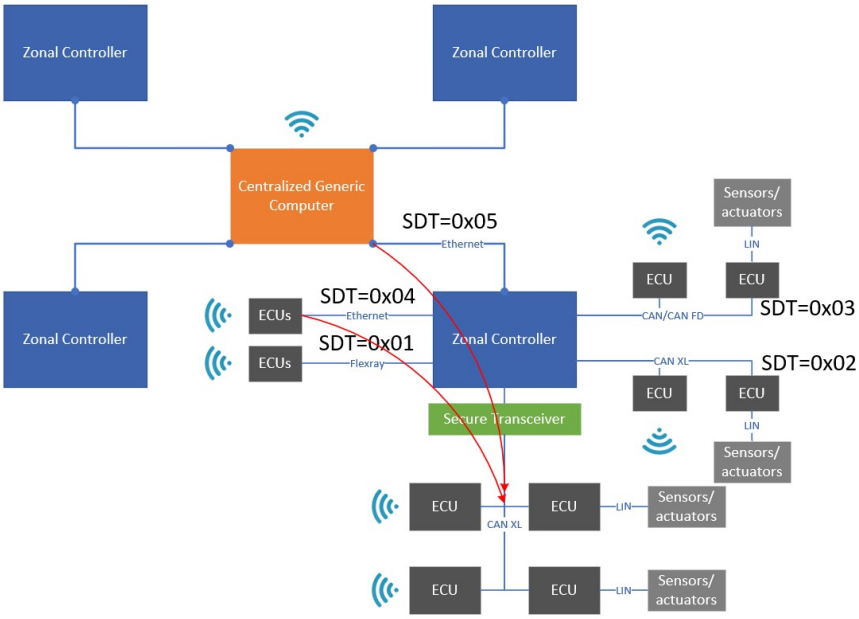

Figure 5.16: **Frames with SDT = 0x04 or 0x05**

gateway.

5.5.2 Structure of the Leaky Buckets Tree

Fig. 5.17 depicts the structure of the leaky bucket tree. The value of the transmission counter and the priority of the incoming frame will first input the main leaky bucket. If it is full at this point, then the gateway is assumed to be compromised and its message should be blocked immediately regardless of the remaining information. If it is not full, a particular leaky bucket will be selected according to the value of SDT, VCID and AF. Each leaky bucket corresponds to a group of ECUs connected to the same bus. For example, content ID group 1 when SDT = 0x01 corresponds to a group of ECUs within another CAN XL bus. Only ECUs in this bus are able to send CAN XL frames whose content ID falls into content ID group 1. Here, the content ID when SDT = 0x01 indicates the source of the CAN XL frame.

Memory Usage Estimate

As listed in Tab 5.6, each leaky bucket uses 51 bits of memory. In the protocol engine, 11+8+8+32 = 59 bits of memory are needed to buffer all relevant fields. Let the number of groups when SDT = 1, 2, 3 and 5 is N_1, N_2, N_3 and N_5, respectively. Since we don't have any information available to subdivide ECUs when SDT = 0x04, only 1 leaky bucket is for this scenario. The transmission counter takes 15+16 = 31 bits of memory. In total, memory usage of the secure transceiver is $51(N_1 + N_2 + N_3 + 1 + N_5) + 59 + 31$. We assume $N_1 = N_2 = N_3 = N_5 = 5$, the total memory usage will be 1161 bits, or 146 bytes. It is quite feasible for a small scale, low price chip.

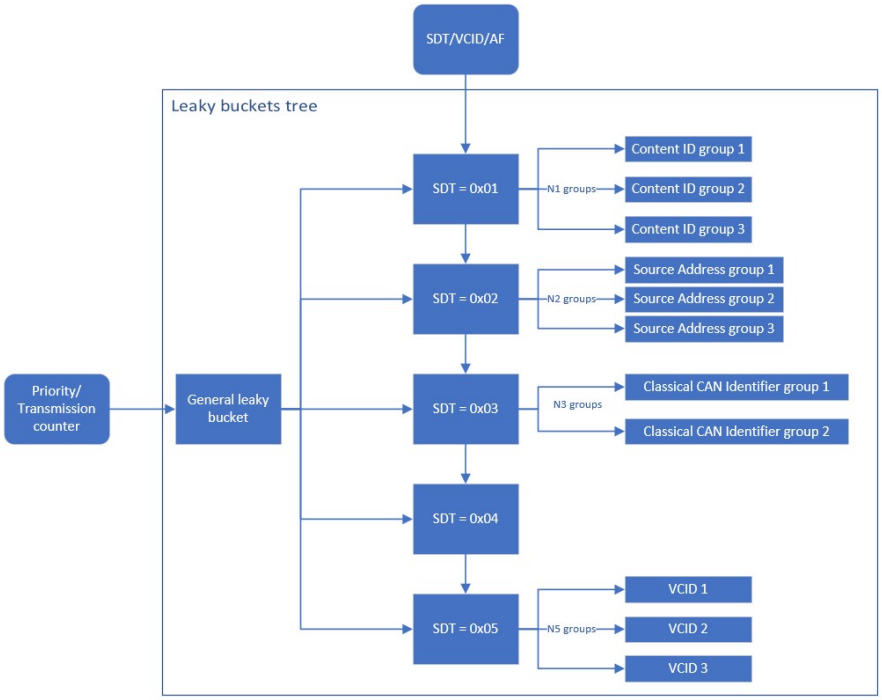

Figure 5.17: **Structure of the leaky buckets tree**

Head									
Arbitration Field		Control Field							
Priority ID	XL	ADS	SDT	SEC	DLC	SBC	PCRC	VCID	AF
Tail									
Data Field	CRC Field		ACK Field		EOF Field				
Data Bytes	FCRC	FCP	DAS	ACK	EOF				

Table 5.9: **Division of CAN XL frame**

5.6 Flooding Prevention Measures

The flooding intrusion is detected when one or more leaky bucket overflows. To ensure the availability of CAN bus, flooding prevention measures should be designed to stop the compromised node from taking the bus all the time and allow other nodes to use the bus.

To facilitate the analysis of the flooding prevention process, the CAN XL frame is divided into two parts: Head and Tail, shown in Table 5.9. Head ranges from SOF to AF and includes information characterizing the payload. Tail ranges from Data bytes to EOF and includes the data payload.

5.6.1 Division of Scenarios

Depending on the configuration of the host ECU and the CAN XL bus it connects to, the best flooding prevention strategy can vary significantly. There are two important factors we need to pay attention to: the origin of attack traffic,

Source of flooding	Gateway	Outside ECU group
Error signalling on	Disconnection	Disconnection + Loopback Error Flag
Error signalling off	Disconnection	Disconnection + Loopback

Table 5.10: **Flooding prevention measures in all possible scenarios**

and error signalling functionality, which are explored separately as follows:

Origin of Attack Traffic

The attack traffic can be traced using leaky buckets trees as discussed in Section 5.5. Generally, the overflowed bucket can be either general bucket or buckets associated with specific groups, which determines the flooding prevention mechanism to use.

1. When a specific sub-bucket overflows, a compromised node in the corresponding ECU group outside of this CAN XL bus is flooding. In this case, countermeasures can be carried out only when the Head is finished, since the right sub-bucket can't be selected until the protocol engine has decoded all control fields (SDT, VCID and AF). The aim of the countermeasure is that we only block all traffic coming from that specific ECU group while allowing other traffic to pass.

2. When the general bucket overflows, it is very likely that the gateway itself is compromised. In this case, countermeasure can be carried out as soon as the general bucket overflows before the flooding frames being sent. The aim of the countermeasure should be completely blocking all traffic from the host until the bucket value goes down to not full state.

Error Signalling Functionality

As soon as CAN XL frames from an outside ECU are identified as flooding frames at the end of the Head part, they will be blocked using flooding prevention measures in various possible ways. This process will break the standard form of CAN frames, causing errors on the bus. Since the intention of secure transceivers is preserving the normal operation of the CAN XL bus, errors on the bus must be properly dealt with without totally breaking the communication.

It is worth noting that the error signalling and fault confinement can be optionally enabled or disabled in CAN XL, as explained in Section 3.4.6 and 3.4.7. The existence of error signalling mechanism determines the flooding prevention strategy.

Considering the above two factors, four scenarios are defined and for each scenario a flooding prevention measure needs to be designed. All possible scenarios and designs of flooding prevention measures are shown in Table 5.10, which will be explored in the following sections.

5.6.2 Set of Metrics for Flooding Prevention Measures

Flooding prevention measures don't perfectly destroy the flooding attack. Despite countermeasures, flooding attacks still damage the performance of the com-

munication system. Therefore, a set of metrics should be set in order to judge and compare flooding prevention measures.

Since flooding prevention measures should aim to ensure the availability of the CAN XL bus, several important metrics associated with communication systems can be utilized to assess its effectiveness.

- Throughput: Throughput measures the rate at which data can be successfully transmitted over the system. It is usually expressed in bits per second (bps) or a similar unit.
- Latency: Latency refers to the time delay experienced by data as it travels through the communication system. Lower latency is desirable for time-critical applications, such as engine control unit and transmission control module.
- Availability: Availability measures the percentage of time the communication system remains operational and accessible to users. High availability is crucial for reliable communication.

In zonal architecture discussed in chapter 2, three types of communication should be considered to assess the CAN system, as shown in Fig. 5.18:

- From outside to inside (RED): An outside ECU sends messages to the zonal gateway which forwards it to the local CAN network.
- From inside to outside (YELLOW): An ECU on the local CAN bus sends messages to the zonal gateway which forwards it to the target outside ECU.
- From inside to inside (GREEN): Communication within the local CAN bus.

After evaluating communication metrics listed above, different flooding prevention measures can be assessed and compared.

5.6.3 Hardware Architecture for Flooding Prevention

The CAN transceiver is located between its host and the CAN bus. It has four ports: TXD from host (input), TXD to bus (output), RXD from bus (input), and RXD to host (output). A conventional CAN transceiver should simply convert the single-ended logic used by the CAN controller to the differential signal transmitted over the CAN bus. To achieve the functionality of flooding prevention, secure CAN transceivers should decode and manipulate these 4 ports. Hardware architecture means the hardware connection used to deal with these ports.

Fig. 5.19 depicts the hardware architecture for flooding prevention. The Protocol Engine detects flooding threat and controls two multiplexers. The multiplexer TXD routes TXD from host (control TXD = 0), recessive (control TXD = 1), or dominant (control TXD = 2) to TXD to bus. The multiplexer RXD routes RXD from bus (control RXD = 0), dominant (control RXD = 1), or TXD from host (control RXD = 2) to RXD to host. Note that the secure transceiver is transparent with the configuration of Control RXD = Control

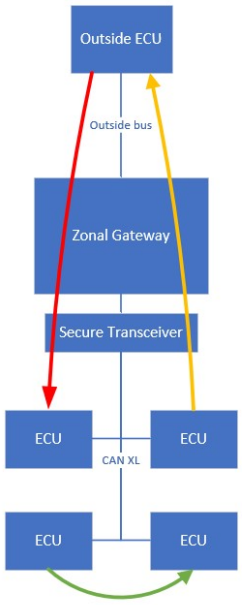

Figure 5.18: **Three types of communication**

TXD = 0, which is applied when no threat is detected. With different combination of control signals, the secure transceiver can either disconnect the host and the bus or act as a conventional transceiver as if it is transparent.

5.6.4 Configurations of Disconnection Measures

Disconnection architecture tackles flooding from the host by disconnecting TXDs and send back appropriate RXD to host. It has different configurations depending on the origin of attack threat and error signalling functionality. This section explores configurations of disconnection measures in all possible scenarios.

When the Gateway is Flooding

When the gateway is determined as the source of flooding traffic, disconnection measure should be taken. The measure consists of two states: Normal State and Block State. When no threat from the host is detected, the secure transceiver works in Normal State. As soon as the general bucket overflows, the secure transceiver can draw the conclusion that the gateway is the source of flooding traffics. At the end of the Head part, the protocol engine immediately enters Block State in which:

1. Recessive is routed into TXD to bus, which means TXD from host is blocked.

2. Dominant is routed into RXD to host, which prevents the host from accessing the bus.

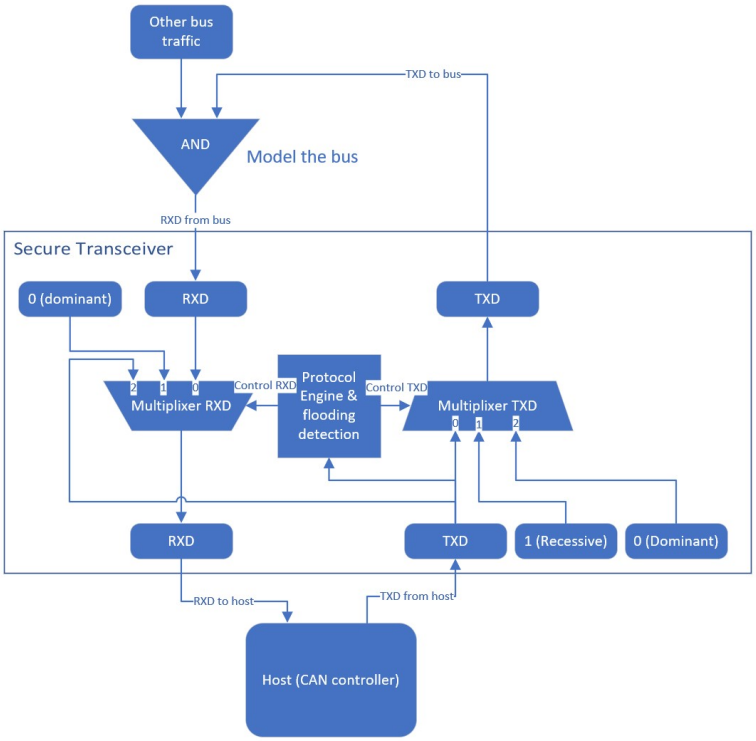

Figure 5.19: **Diagram of hardware architecture for flooding prevention**

By doing this, the secure transceiver effectively disconnects the host and the CAN XL bus. Protocol engine exits Block State and enters Normal State when the general bucket value goes down below the overflow threshold and the bus is in idle state (to protect the ongoing transmission).

In summary, the configuration rule when the gateway is flooding is: when the secure transceiver works in Block State, both Control signals are set to 1; Otherwise, they are set to 0.

Fig. 5.20 shows the waveform when the general bucket overflows. Since no error occurs in the process, it doesn't matter whether error signalling is enabled or disabled. What happens in the waveform can be broken down into 4 phases:

1. In the beginning, the host wins the arbitration because of its very high priority. Note that the general leaky bucket value is still below the overflow threshold at this moment.

2. When the frame finishes, and the bus enters the idle state, the protocol engine increases the general bucket value which makes it exceeds the overflow threshold.

3. The secure transceiver enters the Block State. The protocol engine blocks this frame by setting both Control TXD and Control RXD to 1.

4. Other nodes join the arbitration and access the bus while the gateway is blocked.

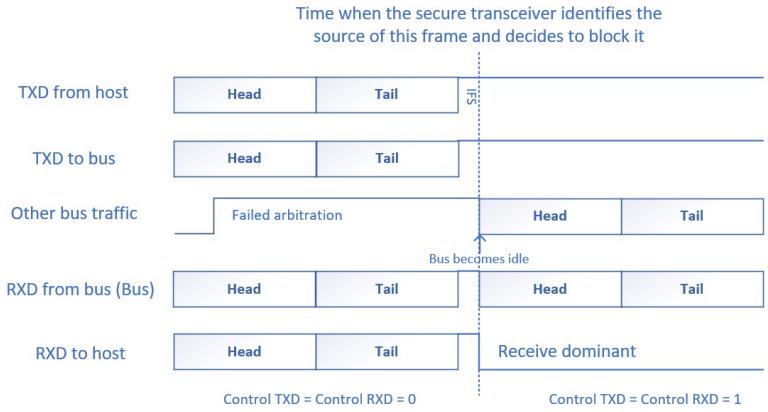

Figure 5.20: **Waveform when the general bucket overflows**

When the Flooding is From Outside

When the gateway host isn't responsible for the flooding traffic, the aim of the flooding prevention measure is to block only traffic coming from compromised ECU groups , while allowing traffic from other ECU groups to pass.

Disconnecting the host and the bus creates bit errors because TXD and RXD are different in the perspective of the host. According to Section 3.4.4, the CAN controller retransmits the same frame immediately if the previous frame fails. To stop retransmission and enable further valid transmission, the frame has to be either validated by the gateway host as a transmitter (see Section 3.4.5) or aborted. Otherwise, the repetitive retransmission of the Tx SMB will halt the arbitration process of Tx MBs until the transmission is successfully validated or aborted. In consequence, subsequent non-flooding frames end up being stuck in Tx MBs and can't get transmitted (see Section 3.6).

Loopback should be applied to meet the validation condition of the host. To pass the validation of a frame, RXD to host should be identical to TXD from host and ACK field should be received as dominant. This can be done if Control RXD in Fig. 5.19 is set to 1 during ACK field, and 2 during other fields of the frame.

When error signalling is disabled:

According to the CAN XL protocol standard [43], when error signalling is disabled, A transmitter shall ignore errors that occur in the range from SOF bit up to the AH2 bit, including the SOF bit and the AH2 bit. The frame is validated by transmitter only if the ACK slot is sampled dominant (see Section 3.4.5).

If the receiver detects an error during the following parts of the XLFF, it shall treat this as a protocol exception event: Arbitration Field, Control Field, Data Field, CRC Field, ACK Field, EOF. When a node detects such a protocol exception event inside the XL Data Phase, the node shall finish the XL Data Phase at the end of the bit, in which the event is detected. As reaction to the protocol exception event, the error counters shall not be changed, the hard synchronization shall be enabled, the node shall send recessive bits and shall enter the bus re-integration state.

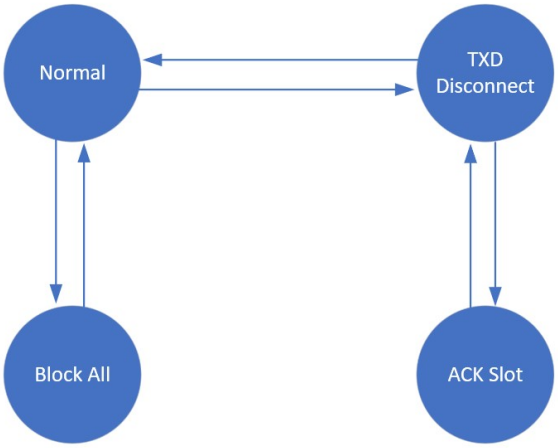

Figure 5.21: **State transition diagram of flooding prevention when error signalling is disabled**

The flooding prevention measure in this scenario consists of 4 states: Normal state, TXD disconnect state, ACK state, and Block All state. Fig. 5.21 is the state transition diagram. Configurations of protocol engine in these states are:

- Normal state: Control TXD = Control RXD = 0; TXD from host and to bus are connected as if the secure transceiver is transparent.
- TXD disconnect state: Control TXD = 1, Control RXD = 0; Enter this state whenever the flooding traffic is detected by protocol engine. Recessive is multiplexed into TXD to bus, while RXDs are still connected. Enter Normal state when both TXD from host and the bus become idle again.
- ACK state: Control TXD = Control RXD = 1; ACK state only occurs during the ACK slot of TXD from host. It overwrites the recessive ACK slot which validates the frame on the side of the transmitter.
- Block All state: Control TXD = Control RXD = 1; Enter this state when the general bucket overflows. Block all traffic from the host and suppress further transmission.

Fig. 5.22 depicts the waveform of this countermeasure in which the compromised host is competing for the bus access with other nodes. Here, bold font means successful transmission. Important points in time are:

- T0: Time when the secure transceiver identifies the source of this frame and decides to block it at the end of Head.
- T1: Time when both TXD from host and the bus become idle again.

Next we break the countermeasure down into 7 steps as follows:

1. In the beginning, the host wins the arbitration because of its very high priority.

2. When the Head finishes, the protocol engine identifies the source of this frame and enables the corresponding sub-leaky bucket.
3. The selected sub-leaky bucket gives the output to protocol engine that it has overflowed.
4. The protocol engine enters the TXD disconnect state to block this frame and free the bus.
5. Other nodes on the bus detects stuff error in the current frame sent by the host. Since error signalling is disabled, they treat this as a protocol exception event and enter the bus re-integration state.
6. Other nodes detect idle on the bus while the host's frame is blocked.
7. One node joins the bus and successfully transmits a frame. In the meantime, when TXD from host enters the ACK slot, the protocol engine enters the ACK state to validate this frame.

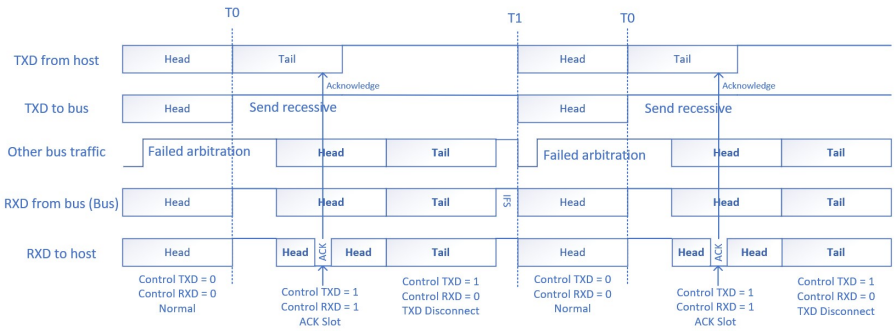

Figure 5.22: **Waveform of flooding prevention when error signalling is disabled**

When error signalling is enabled:

As explained in Section 3.4.5, the condition of frame validation is stricter than when error signalling is disabled. The frame shall be valid for a transmitter if there is no error until the end of EOF. In addition, nodes will send error flags whenever errors are detected. The presence of error signalling functionality further complicates the flooding prevention measure.

The flooding prevention measure in this scenario consists of 5 states: Normal state, Loopback state, TXD disconnect state, ACK state, and Block All state. Fig. 5.23 shows the state transition diagram. Configurations of protocol engine in these states are:

- Normal state: Control TXD = Control RXD = 0; TXD from host and to bus are connected as if the secure transceiver is transparent.
- Loopback state: Control TXD = 1, Control RXD = 2; Bit levels transmitted on TXD from host are loopbacked into RXD to host, which is a part of frame validation. Recessive is multiplexed into TXD to bus. Enter this state whenever the flooding traffic is detected by protocol engine at the end of Head.

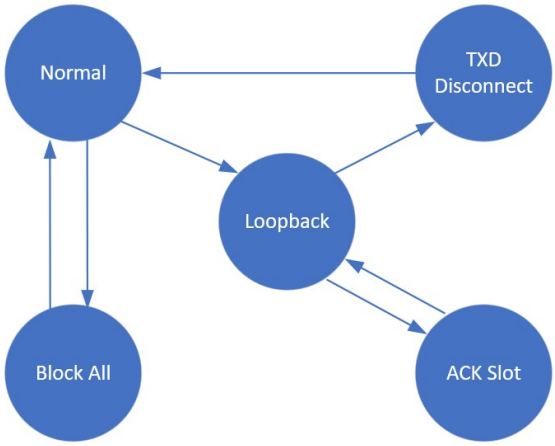

Figure 5.23: **State transition diagram of flooding prevention when error signalling is enabled**

- TXD disconnect state: Control TXD = 1, Control RXD = 0; Recessive is multiplexed into TXD to bus, while RXDs are still connected. Enter this state whenever the host finishes its frame at the end of the EOF. Enter Normal state when both TXD from host and the bus become idle again.
- ACK state: Control TXD = Control RXD = 1; ACK state only occurs during the ACK slot of TXD from host. It overwrites the recessive ACK slot which validates the frame on the side of the transmitter.
- Block All state: Control TXD = Control RXD = 1; Enter this state when the general bucket overflows. Block all traffic from the host and suppress further transmission.

Fig. 5.24 depicts the waveform of this countermeasure in which no other node is competing for the bus access with the compromised host. Here, bold font means successful transmission. Important points in time are:

- T0: Time when the secure transceiver identifies the source of this frame and decides to block it at the end of Head.
- T1: Time when the host finishes its frame at the end of the EOF.
- T2: Time when both TXD from host and the bus become idle again.

Fig. 5.25 shows the waveform of this countermeasure in which other nodes are competing for the bus access with the compromised host. The definitions of T0, T1 and T2 are identical to Fig. 5.24. The waveform in Fig. 5.25 can be broken down into steps as follows:

1. In the beginning, the host wins the arbitration because of its very high priority.
2. When the Head finishes at T0, the protocol engine identifies the source of this frame and enables the corresponding sub-leaky bucket.

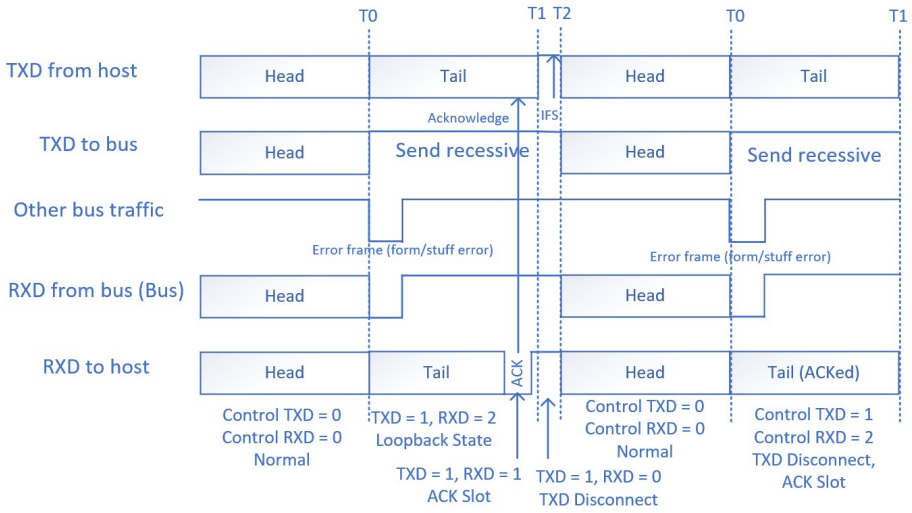

Figure 5.24: **Waveform of flooding prevention when error signalling is enabled and no other node competes for the bus**

3. The selected sub-leaky bucket gives the output to protocol engine that it has overflowed.

4. The protocol engine enters the Loopback state to block this frame and free the bus. It also validates the frame on the side of the host. In the meantime, when TXD from host enters the ACK slot, the protocol engine enters the ACK state to validate this frame.

5. Other nodes on the bus detects stuff error in the current frame sent by the host. As a reaction, they send error frames and wait for the bus to be idle.

6. Other nodes detect idle on the bus while the host's frame is blocked.

7. One node joins the bus and successfully transmits a frame.

8. The host finishes its frame at the end of the EOF at T1. The protocol engine enters the TXD disconnect state.

9. Since the bus should be in IFS phase from the perspective of the host, the traffic on the bus from another node is regarded as an error (can take many forms, such as bit error, form error). As a reaction, the host sends an error frame which doesn't appear on the bus.

10. After the error frame, the host sends recessive and suspends all activities until the bus becomes idle.

11. At T2, the transmitter finishes the frame. Both TXD from host and the bus become idle again. Then, the protocol engine goes back to the Normal state in which all nodes on the bus can join a new cycle of arbitration.

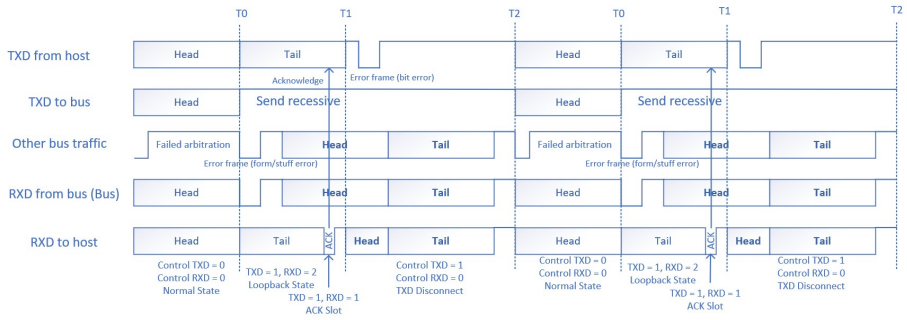

Figure 5.25: **Waveform of flooding prevention when error signalling is enabled and other nodes are competing for the bus**

5.6.5 Configuration of Error Flag Measure

Error flag measure is another option to prevent flooding attacks. It can only be adopted when error signalling is enabled, and the flooding is from an outside bus which means the gateway isn't compromised. The secure transceiver makes use of the fault confinement mechanism of CAN protocol to block the compromised node and increase the availability of the bus (see Section 3.4.6). In this measure, Control RXD is always configured to 0, which means RXDs are always connected.

In normal operation, the protocol engine sets Control TXD = Control RXD = 0. TXD from host and to bus are connected as if the secure transceiver is transparent. Whenever the flooding traffic is detected by protocol engine, an error frame is multiplexed into TXD to bus and TXD from host is ignored. In summary, the error flag measure has 2 states: Normal state and Error flag Configurations of protocol engine in these states are:

- Normal state: Control TXD = Control RXD = 0; TXD from host and to bus are connected as if the secure transceiver is transparent.
- Error flag state: Control TXD = 2, Control RXD = 0; An active error flag (6 dominant bits) is multiplexed into TXD to bus, while RXDs are still connected. Enter this state whenever the flooding traffic is detected by protocol engine at the end of Head. This state lasts for 6 bit-time. Protocol engine enters the Normal state at the end of the error flag.

The whole flooding prevention process is broken down into 2 phases. In phase 1 the host is in error active state, whereas in phase 2 the host is in error passive state.

Phase 1

Fig. 5.26 shows the waveform of countermeasure phase 1 in which the compromised host is competing for the bus access with other nodes. Here, bold font means successful transmission. Important points in time are:

- T0: Time when the secure transceiver identifies the source of this frame and decides to block it at the end of Head.

- T1: Time when the 6-bit error flag finishes.

In phase 1, no frame transmission is successful. Next, we break the countermeasure down into 7 steps as follows:

1. In the beginning, the protocol engine is in the Normal state. The host wins the arbitration because of its very high priority.

2. When the Head finishes at T0, the protocol engine identifies the source of this frame and enables the corresponding sub-leaky bucket.

3. The selected sub-leaky bucket gives the output to protocol engine that it has overflowed.

4. The protocol engine enters the Error flag state and sends an error flag on TXD to bus.

5. All nodes on the bus, including the host, detect stuff error caused by the error frame. As a result, the host stops the transmission, and everyone sends their own error frames. It will cause TEC (transmit error counter) of the host to increase by 8 and REC (receive error counter) of all other nodes to increase by 1.

6. When the error flag finishes at T1, the protocol engine goes back to the Normal state.

7. In the next frame, the host and other nodes join the arbitration at the same time. The host will win the arbitration again and take the bus.

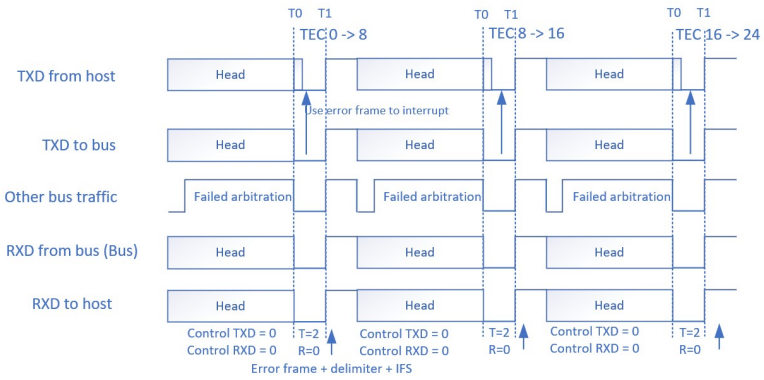

Figure 5.26: **Waveform of error flag measure phase 1**

This process repeats for at most 16 times until the host becomes error passive when its TEC = 128.

Phase 2

: Phase 2 begins when the host becomes error passive. CAN standard stipulates that an error-passive node, which has been transmitter of the previous

frame, shall suspend the start of further frame transmissions for 8 bit times following intermission. If another node starts a transmission during that suspend transmission time, the node shall become a receiver of this frame.

As shown in Fig. 5.27, other error-active nodes can join the arbitration earlier than the host after an error frame at step 7, thus giving them bus access if they start a new transmission during suspend transmission of the host.

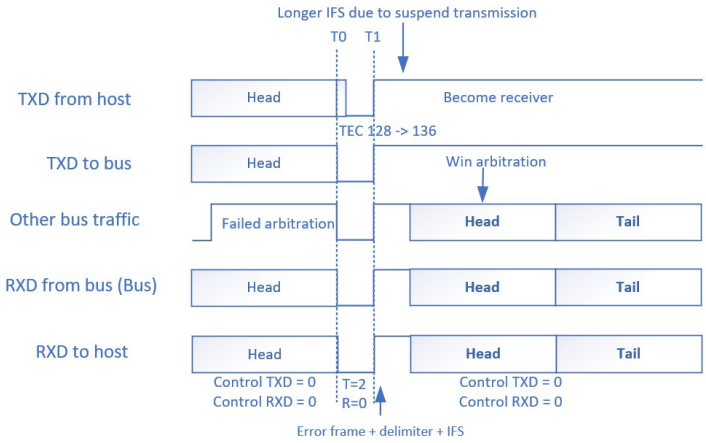

Figure 5.27: **Waveform of error flag measure phase 2**

5.7 Comparison Between Proposed Measures

Based on metrics proposed in Section 5.6.2, this section draws a comparison between proposed disconnection measure and error flag measure. Since error flag measure is applicable when error signalling is enabled and the flooding is from outside, the comparison will be made in this scenario.

5.7.1 Throughput Comparison

In networking, throughput refers to the amount of data that can be transmitted over a network connection in a given time. For simplicity, in this thesis it is defined as the proportion of time taken for valid frame transmission compared to the total communication time.

Disconnection Measure

Conforming to this definition, the throughput of disconnection measure T, as depicted in Fig. 5.25, can be calculated as

$$\mathrm{T} = \frac{\mathrm{Head} + \mathrm{Tail}}{2\mathrm{Head} + \mathrm{Tail} + \mathrm{ErrorFlag} + \mathrm{ErrorDelimiter} + 2\mathrm{IFS}} \tag{5.28}$$

Error Flag Measure

This measure consists of 2 phases. All frames are invalidated by error frames in phase 1 when the host is error active. Therefore, throughput of error flag measure in phase 1 is 0.

One significant drawback of the Error flag measure is the retransmission of flooding frames. Even though flooding frames are interrupted halfway by Error flags sent by the secure transceiver, they will be automatically retransmitted by the host CAN controller. Thus, the host has no chance for further valid transmission. Loopback tackles this problem with frame validation, while the error flag measure must rely on frame abortion mechanism.

According to Section 3.4.4, the current frame is aborted after a configured number of automatic retransmissions, provided that allowed retransmission attempts is finite. In this case, normal frames prepared in Tx MBs are able to be transmitted after a few retransmissions of flooding frames.

Let the maximum number of automatic retransmission is configured to N, the throughput of error flag measure in phase 2, as shown in Fig. 5.27, can be calculated as

$$\mathrm{T} = \frac{\mathrm{Head} + \mathrm{Tail}}{(\mathrm{N}+1)(2\mathrm{Head} + \mathrm{Tail} + \mathrm{ErrorFlag} + \mathrm{ErrorDelimiter} + 2\mathrm{IFS})} \tag{5.29}$$

Comparison

From equation 5.28 and 5.29, we have

$$\mathrm{T}_{\mathrm{Disconnection}} = (\mathrm{N}+1)\mathrm{T}_{\mathrm{ErrorFlag}} \tag{5.30}$$

Here, N denotes the maximum number of automatic retransmissions before the frame is aborted. Comparing with the error flag measure, network adopting the disconnection measure has larger throughput, especially when N is configured to a large value.

5.7.2 Latency Comparison

Latency in a communication system refers to the time delay or the amount of time it takes for data to travel from the source to the destination through the communication medium. Here, we define latency as the time interval between the initiation of a transmission and the moment the first bit of data reaches the destination.

Disconnection Measure

The longest latency caused by disconnection measure is derived in Fig. 5.25. A valid frame transmission initiates after a failed arbitration, an error frame and IFS. Hence, the latency L can be expressed as

$$\mathrm{L} = \mathrm{Head} + \mathrm{ErrorFlag} + \mathrm{ErrorDelimiter} + \mathrm{IFS} \tag{5.31}$$

Error Flag Measure

The longest latency caused by Error Flag measure is the time taken for the host to become error passive and another node wins the bus arbitration. In phase 1, there are 16 repetitions of Head transmission and error frames. In phase 2, the valid frame has to wait for a Head and an error frame. Therefore, the total time would be

$$\mathrm{L} = 17(\mathrm{Head} + \mathrm{ErrorFlag} + \mathrm{ErrorDelimiter} + \mathrm{IFS}) \tag{5.32}$$

Comparison

Connecting equation 5.31 and 5.32, we have

$$\mathrm{L}_{\mathrm{Disconnection}} = \frac{1}{17}\mathrm{L}_{\mathrm{ErrorFlag}} \tag{5.33}$$

Comparing with the disconnection measure, messages experience larger latency with the error flag measure in worst case.

5.7.3 Availability Comparison

Availability in a communication system refers to the ability of the system to be operational and accessible to users. In the context of this thesis, availability is measured by conditions of three types of communications in Fig. 5.18.

Disconnection Measure

During the flooding attack from outside, disconnection measure guarantees RED communications and GREEN communications. However, when Control RXD in Fig. 5.19 is not 0 (true in Loopback state and ACK state), the gateway ignores YELLOW communications since RXDs are disconnected.

Error Flag Measure

During the flooding from other branches via the the gateway, error flag measure guarantees the availability of GREEN and YELLOW communications, if the measure enters phase 2. However, the availability of RED communications depends of the automatic transmission configuration of the CAN controller: infinite times of retransmission will completely block the TX MB, making RED communications unavailable.

Comparison

Comparing between availability of the disconnection measure and the error flag measure, both measures guarantee communication among nodes during flooding attack. The former focuses on the protection of RED communication, whereas the latter focuses on the preservation of YELLOW communication.

5.7.4 Summary of Results

From the above analysis of flooding prevention measures, disconnection measure is significantly better than error flag measure in terms of throughput and latency among flooding attack. When it comes to availability, both schemes perform similarly. In conclusion, the disconnection measure is clearly superior to the error flag measure. In the rest of this thesis, the disconnection measure will be adopted and implemented.

Chapter 6

Validation and Results Analysis

In this chapter, the proposed CAN XL secure transceiver is validated in two aspects: software simulation (Section 6.1) and hardware validation (Section 6.2). For each aspect, we introduce the test platform and implementation details. Finally, the validation result is discussed.

6.1 Software Simulation

This section performs software simulation on the secure transceiver for CAN XL designed in chapter 5.

6.1.1 Simulation Platform

The secure CAN XL transceiver is implemented in VHDL[44], a hardware description language widely used in digital system design. Fig. 6.1 depicts the simulation environment of the Design Under Test (DUT). The simulation input is generated by Tool Command Language (Tcl) [45] scripts. Tcl is a scripting language commonly used for automating tasks, application control, and rapid prototyping. Tcl plays a significant role as a scripting language in Electronic Design Automation (EDA) tools. Many EDA tools, such as circuit simulation, synthesis tools, and place-and-route tools, support Tcl for customization and automation. One of Tcl scripts in Fig. 6.1 simulates the behavior of the gate CAN controller and generates both normal frames and flooding frames. The other Tcl script simulates the behavior of other nodes on the CAN XL bus that arbitrate with the gateway host. The AND gate models the electrical characteristic of CAN bus: the dominant state (bit 1) overrides the recessive state (bit 0). The clock generator serves as the clock source for DUT.

The bit comparison module verifies if any bit error or ack error occurs (see Section 3.4.4). If TXD from host and RXD to host doesn't agree (except for ACK slot), the host experiences a bit error.

The simulation is conducted in Xcelium and viewed and debugged in SimVision. Xcelium is a high-performance digital simulator developed by Cadence. It is widely used in the semiconductor industry for digital design verification

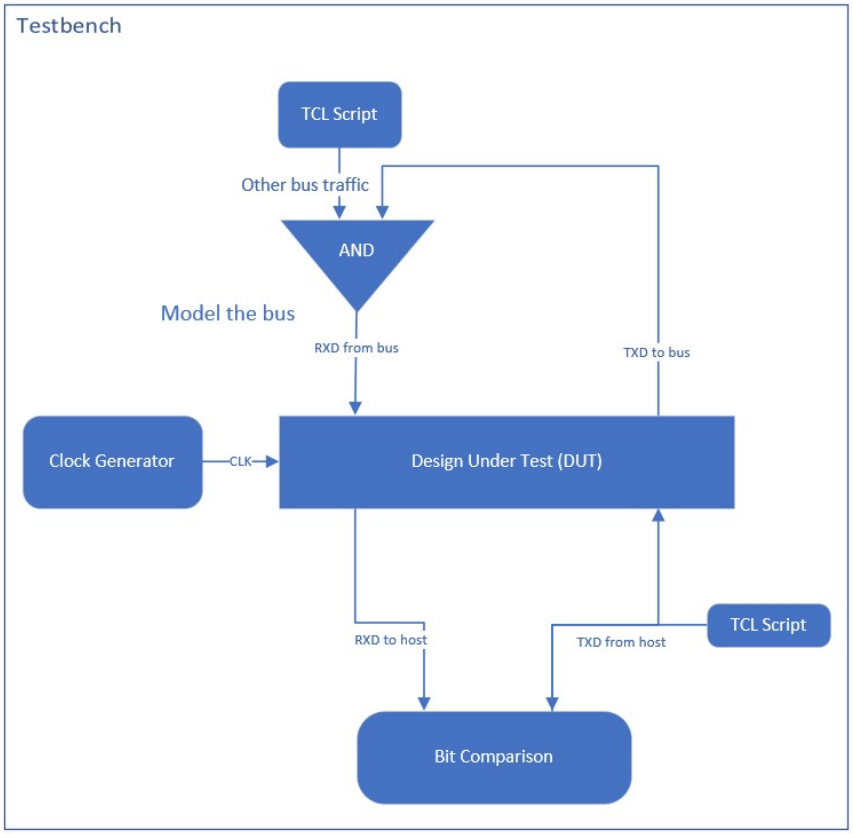

Figure 6.1: **Diagram of simulation testbench for secure transceiver**

and simulation. SimVision is a waveform viewer and debugging tool that is integrated with Xcelium.

6.1.2 Implementation Details

At the time when this thesis is written, despite being standardized and researched by many companies, there is no CAN controller compatible to CAN XL protocol on the market. To be able to validate it in the lab, encapsulation is adopted as a means to bridge the gap in techniques.

Encapsulation, as the name suggests, refers to the process of encapsulating one network protocol within another network protocol[46]. In encapsulated protocol, the original data packets, which are based on one protocol, are encapsulated within the payload of another protocol. Encapsulation enables the transmission of data frames of CAN XL through a network that only supports Classical CAN and CAN FD.

Since this project is only interested in a few fields of CAN XL, including priority, SDT, VCID, AF, and Data, other fields of encapsulated CAN XL frame are omitted. In addition, the priority field of CAN XL works the same way as that of Classical CAN and CAN FD. Therefore, the payload of encapsulating CAN FD frame needs to include only SDT, VCID, AF, and Data.

FD frame	XL frame
Priority	ID
Data Byte 0	SDT
Data Byte 1	VCID
Data Byte 2 to 5	AF
From Data Byte 6 onward	Data

Table 6.1: **Frame format of CAN XL encapsulation in CAN FD**

To allow long frame payload, CAN FD which supports up to 64 data bytes is selected to encapsulate the CAN XL frame. The frame format of encapsulation is shown in Table 6.1. In this way, CAN FD frames can be used to simulate the behavior of CAN XL frames with an extra coding and decoding step.

As we move to encapsulated CAN XL protocol, equation 5.25 should be revisited as well. According to Fig.3.3, we have the relation between t_{frame} and DLC (denoted as L) is:

$$t_{\text{frame}} = 1.15(\frac{27}{f_a} + \frac{30 + 8L}{f_d}) \tag{6.1}$$

Equation 6.1 adds extra 15% stuff bit as the stuffing rate of CAN FD is higher than that of CAN XL.

Section 5.5 proposes the leaky buckets tree classifying CAN XL frames into 5 categories with regard to SDT values. To simulate the real in-vehicle network zonal architecture, CAN XL frames coming from various zones are generated. For frames with identical SDT values, frame fields further differentiate the source of them. Next all SDT values stipulated in the standard are created in the simulation:

1. SDT = 0x01. Assume 2 groups of ECUs on an in-vehicle network send this type of frames and content ID in AF field indicates the source. The first group has content ID $AF(0) = 0$, and the other group has $AF(0) = 1$.

2. SDT = 0x02. Assume 2 groups of ECUs on an in-vehicle network send this type of frames and Source Address in AF field indicates the source. The first group has Source Address $AF(0) = 0$, and the other group has $AF(0) = 1$.

3. SDT = 0x03. Assume 2 groups of ECUs on an in-vehicle network send this type of frames and the priority identifier in AF field indicates the source. The first group has priority identifier $AF(0) = 0$, and the other group has $AF(0) = 1$.

4. SDT = 0x04. Since no field in the frame gives information about the source of that frame, all frames with SDT = 0x04 are regarded as a whole. There is no further classification in this group.

5. SDT = 0x05. Assume 2 groups of ECUs on an automobile send this type of frames and the VID identifier in VCID field indicates the source. The first group has VID identifier $AF(0) = 0$, and the other group has $AF(0) = 1$.

SDT	VCID	AF	Number of groups
01	x	Group1: AF(0)=0 Group2: AF(0)=1	2
02	x	Group1: AF(0)=0 Group2: AF(0)=1	2
03	x	Group1: AF(0)=0 Group2: AF(0)=1	2
04	x	x	1
05	Group1: Vcid(0)=1 Group2: Vcid(0)=0	x	2

Table 6.2: **Classification of sources of frames in the simulated in-vehicle network**

Parameters	a	tw	p
General bucket	0.5	10ms	0.05
Sub buckets	0.3	10ms	0.05

Table 6.3: **Initialization parameters of leaky buckets for simulation**

All generated frame formats in the simulation are listed in Table 6.2.

For simulation, testbench program was designed that configured arbitration-phase $f_a = 500$ kbit/s, data-phase $f_d = 500$ kbit/s, and clock frequency $f_c = 20$ MHz. The error signalling functionality is disabled. Initialization parameters of both general leaky bucket and sub leaky buckets are listed in Table 6.3. Parameters for general bucket apply to all frames, whereas parameters for sub buckets are applicable to frames belong to a specific ECU group. The percentage of total bus time allowed is 50%, which is higher than that of a particular type of frame (30%).

Parameters	T	u	d	nu	nd
General buckets	76	33200	16600	602	1205
Sub bucket	83	39524	11857	506	1687

Table 6.4: **Derived parameters of leaky buckets for simulation**

Utilizing equations 5.21, 5.5, 5.6, 5.26, and 5.27, all configuration parameters of the simulated secure transceiver can be derived as listed in Table 6.4.

6.1.3 Results Analysis

In this section, the software simulation is performed with these configurations. The simulation result is analyzed to validate the flooding detection and flooding prevention functionalities of the secure transceiver.

Flooding Detection

The definition in Section 5.4.1 stipulates that the traffic from host is labeled as flooding attack if the percentage of bus time it occupies is greater than a within

the time window t_w. To validate this, a special CAN frame traffic is designed. Fig. 6.2 displays the frame transmission pattern which is the repetition of transmission cycles. One cycle of transmission consists of a burst of frames that lasts for t_{frames} and the following idle time. The period of the pattern is t_{w}. The occupation of the host, denoted as o is

$$o = \frac{t_{\text{frames}}}{t_{\text{w}}} \tag{6.2}$$

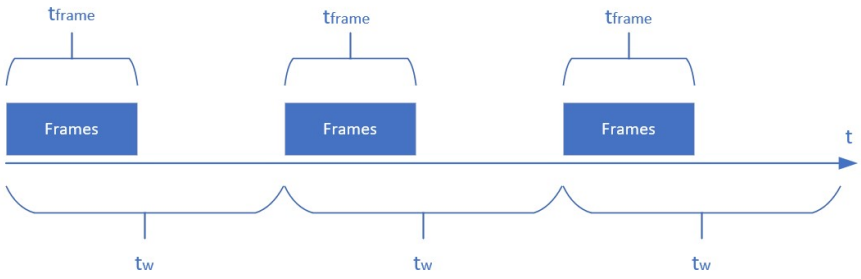

Figure 6.2: **Waveform of CAN frame traffic stimuli for simulation**

According to the definition, if $o \leq a$ then the bucket value will not overflow during the time window, because the increase of the bucket value caused by frames is completely offset by the decrease of the bucket value. As a result, the leaky bucket will never overflow no matter how long the transmission pattern repeats. If $o > a$ then the bucket value will increase during the time window, and the leaky bucket will eventually overflow after a few repetitions. The threshold of frame time in a time window is

$$t_{\text{frames}} = at_{\text{w}} \tag{6.3}$$

In the case of parameters in table 6.3, we have $t_{\text{frames}} = 5\text{ms}$ for general bucket, and $t_{\text{frames}} = 3\text{ms}$ for sub buckets. All incoming frames will increase the general bucket value, whereas a sub bucket value will increase only when the frame comes from the associated ECU group. Therefore,

Firstly, we configure the stimuli to $t_{\text{frames}} = 5\text{ms}$ and disable all sub buckets. Other nodes on the bus don't transmit anything in the meantime. The simulation waveform of 3 periods is shown in Fig. 6.3. To make the change of bucket value visible and clear, its waveform is shown in analog format. In the simulation, the flooding prevention state is always normal which indicates no flooding attack is detected because the bus occupation is within the allowed amount. Moreover, the general bucket value doesn't change after a period of test stimuli because the increase caused by the burst of traffic is completely compensated by the decrease. Therefore, the transmission pattern is always allowed no matter how many times it repeats.

Secondly, we extend the stimuli to $t_{\text{frames}} = 5.25\text{ms}$ where the occupation exceeds the $a + p$. The simulation waveform of 3 period is shown in Fig. 6.4. The flooding prevention state turns into Block All at the end of each burst of frame transmissions, which means the protocol engine regards the traffic as flooding attack and disconnects the host. Moreover, the general bucket value

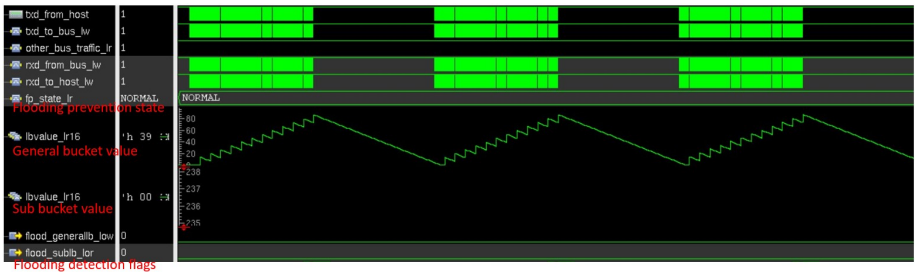

Figure 6.3: **Simulation waveform of general bucket when $t_{\text{frames}} = 5$ms**

increase after each transmission cycle. Hence, this transmission breaks the limit of maximum bus time and is not allowed by the secure transceiver.

Figure 6.4: **Simulation waveform of general bucket when $t_{\text{frames}} = 5.25$ms**

For the test of sub buckets, we at first configure the stimuli to $t_{\text{frames}} = 3$ms and disable the general bucket. Fig. 6.5 depicts the waveform of simulation in which the transmission pattern is always allowed. Then we set $t_{\text{frames}} = 3.15$ms where the occupation exceeds the $a + p$ and restart the simulation. Fig. 6.6 shows the waveform in which transmission traffic exceeds the limit of maximum bus time and blocked by the secure transceiver.

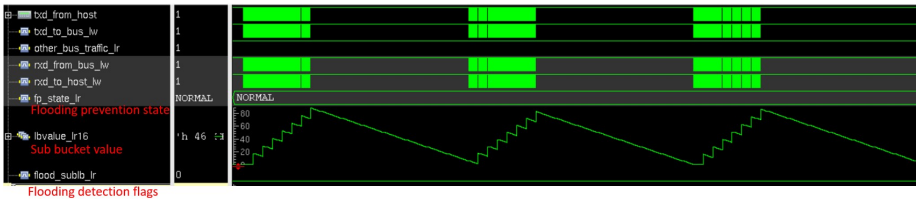

Figure 6.5: **Simulation waveform of sub bucket when $t_{\text{frames}} = 3$ms**

To further verify the detection of flooding attacks, traffic in Fig. 6.2 is modified that each burst of transmission only contains one frame. The idle time between frames is also shorter to maintain the percentage of bus occupation. Using it as the new stimuli, simulation are performed to test if different types of flooding traffic can be detected.

Firstly, we configure the stimuli so that the occupation $o = 50\%$. All sub buckets are disabled to test the general bucket. Fig. 6.7 shows the simulation waveform where the general bucket value doesn't change after a period of test stimuli because the bus occupation is allowed by the secure transceiver. Therefore, the transmission pattern is always allowed no matter how many times it repeats.

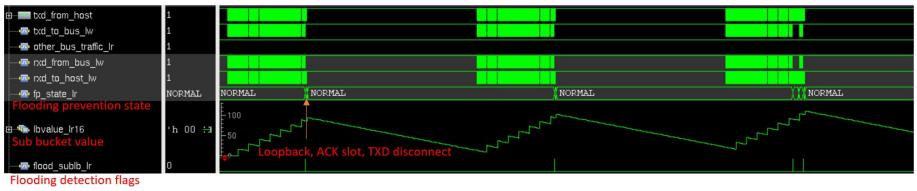

Figure 6.6: **Simulation waveform of sub bucket when $t_{\text{frames}} = 3.15$ms**

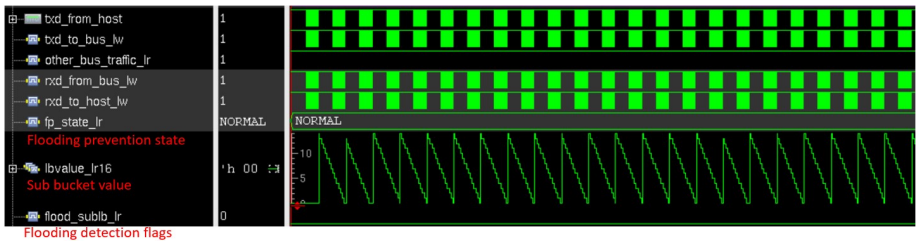

Figure 6.7: **Simulation waveform of the new transmission pattern when $o = 50\%$**

Secondly, we reconfigure the stimuli to $o = 52.5\%$ where the occupation exceeds the $a + p$. The simulation waveform of bucket overflow is shown in Fig. 6.8. The bucket value rises slightly each cycle of test stimuli as the bus occupation is higher than allowed. Eventually, the flooding prevention state turns into Block All when the general bucket overflows after a few frame transmissions.

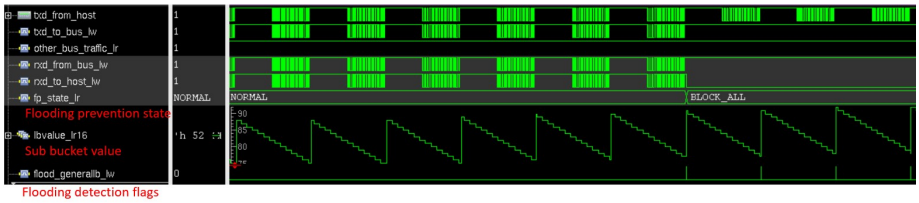

Figure 6.8: **Simulation waveform of the new transmission pattern when $o = 52.5\%$**

In conclusion, simulation results of different transmission patterns agree with the configuration of initialization parameters in Table 6.3. Hence, parameters calculated in Table 6.4 and formulas 5.21, 5.5, 5.6, 5.26, and 5.27 can be verified.

Leaky Buckets Tree

To simulate the real traffic of a flooded gateway, a transmission pattern is created that consists of 9 flooding frames with SDT = 0x01 and 1 normal frame with SDT = 0x02. The waveform of the pattern is shown in Fig. 6.9. Initialization parameters and derived parameters are still the same as Table 6.3 and Table 6.4. The occupation of flooding frames and normal frames on the bus is $o = 90\%$ and $o = 10\%$, respectively. Therefore, the sub bucket of flooding frames will overflow but normal frames will not.

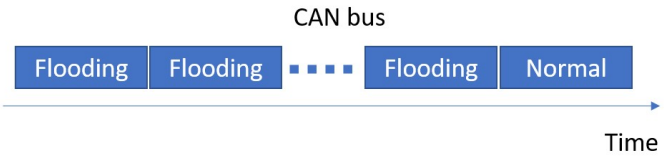

Figure 6.9: **Waveform of CAN frame traffic stimuli for simulation of the leaky buckets tree**

The simulation waveform of the leaky buckets tree is shown in Fig. 6.10. The sub bucket value of flooding frames increases quickly until it reaches the maximum value and gets clipped. By contrast, the sub bucket value of normal frames always returns to zero due to the low occupation of the bus. From the zoom in details as shown above, we can verify that the secure transceiver blocks all flooding traffic, but always let normal frames to go through.

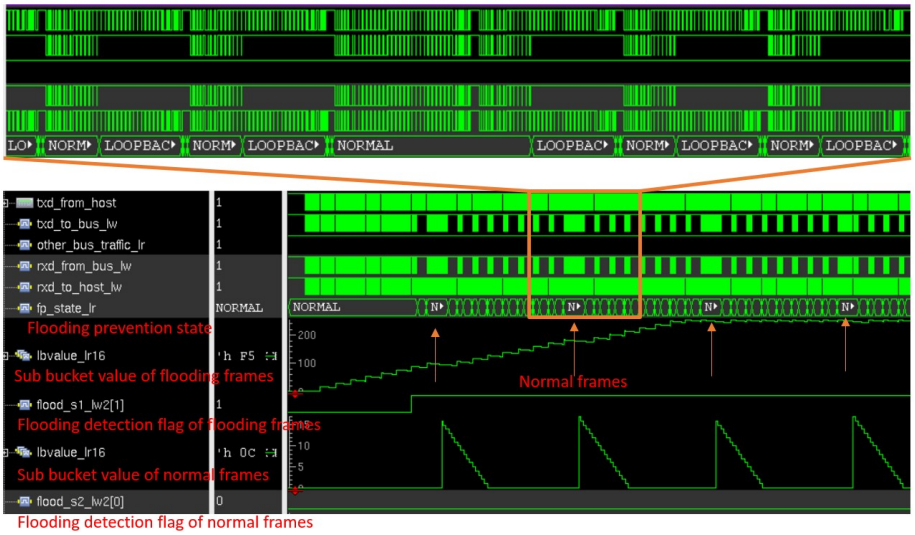

Figure 6.10: **Simulation waveform of the leaky buckets tree and zoom in details**

Flooding Prevention

The flooding prevention functionality is verified if normal messages from other nodes can access the bus while the adversary is flooding. In the simulation of this functionality, the host is transmitting flooding frames while other nodes on the bus try to join the transmission. To make the difference between normal frames and flooding frames more clear, normal frames are bit streams alternating between 0 and 1 which look denser than flooding frames. Fig. 6.11 shows the simulation in which the flooding from other branches via the the gateway and the sub bucket overflows. The waveform shows that normal nodes on the bus are able to join the transmission while the flooding transmission is blocked after the leaky bucket overflows after a few transmissions. Since RXD to host is

identical to TXD from host and the ACK slot is dominant in RXD to host, Tx MBs can be flushed without retransmission. The simulation result agrees with the waveform shown in Fig. 5.25.

Figure 6.11: **Simulation waveform in which the flooding is from other branches via the the gateway**

Fig. 6.12 shows the simulation waveform in which the gateway is flooding and the general bucket overflows. The general bucket value increase when the gateway is flooding and decrease when it is blocked by the flooding prevention measure. The waveform demonstrates that normal nodes on the bus are able to join the transmission while the flooding transmission is blocked in the Block All state.

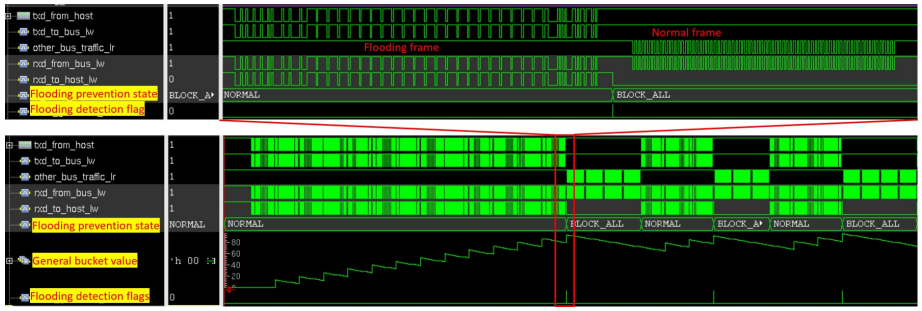

Figure 6.12: **Simulation waveform in which the gateway is flooding**

6.2 Hardware Validation

This section carries out hardware implementation and validation of the secure transceiver for CAN XL designed in chapter 5.

6.2.1 Validation Platform

Fig. 6.13 shows the validation platform of the proposed secure transceiver. This platform consists of 3 nodes (including Node 1, 2, and 3), and 2 networks

(including CAN network and PC configuration network).

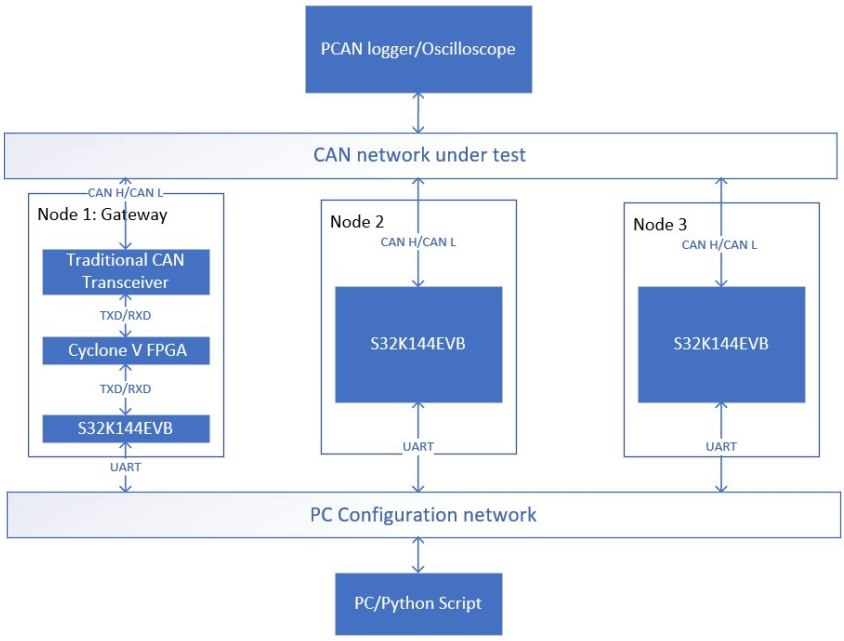

Figure 6.13: **Validation platform of secure transceiver**

Explanations of components shown in Fig. 6.13 are as follows:

1. S32K144 Evaluation Board (EVB) [47]: S32K144 [48] is 32-bit general purpose automotive micro-controllers (MCU) launched by NXP. It supports plenty of features, such as CAN, UART, Timer, etc. The S32K144 EVB integrates a CAN transceiver on the circuit board and a UART to USB converter, which enable the S32K144 MCU to communicate on the CAN bus and with PC.

2. Cyclone V Field Programmable Gate Array (FPGA) [49]: Cyclone V is an FPGA series launched by Altera (now acquired by Intel).

3. PCAN-USB FD adapter [50]: The CAN FD adapter PCAN-USB FD is a device allowing the connection of CAN FD and CAN networks to a computer via USB. It comes with a GUI interface on PC side that monitors and transmits CAN frames.

In the validation platform, Node 1 plays the role of a gateway, whereas Node 2 and Node 3 are ordinary nodes on the same bus. Node 1 consists of a S32K144 Evaluation Board (EVB), an Altera Cyclone V FPGA [49] and a traditional CAN transceiver. Node 2 and Node 3 only consist of a EVB. 3 nodes are connected to the same CAN bus, which is monitored by an oscilloscope and a PCAN-USB FD adapter. They are connected to the PC via PC configuration network which comprises 3 UART cables. The PC runs python script to send UART signals to control these nodes, as well as receive feedback signals from them.

6.2.2 Implementation Details

Fig. 6.14 is the picture of the validation platform. The bread board provides power supply and CAN H/CAN L connection, and serves as test points of the oscilloscope.

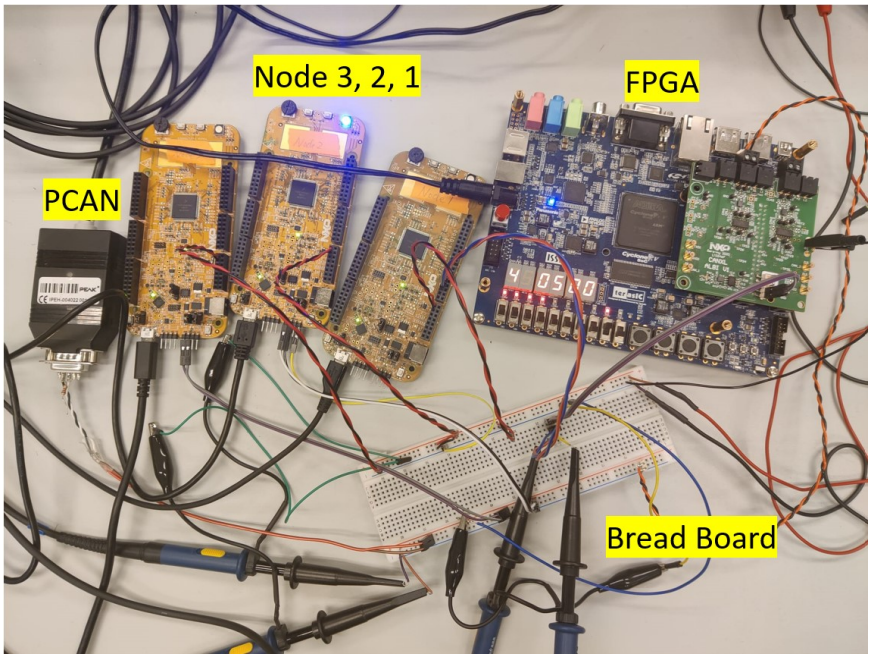

Figure 6.14: **Picture of the validation platform**

Initialization parameters and derived parameters of the CAN network under test are configured as in Table 6.3 and Table 6.4. Error signalling of CAN network is turned on. Fig. 6.15 shows the information flow of the validation platform, including normal CAN frames, flooding CAN frames, and UART acknowledge signals. The CAN traffic transmitted on the bus corresponding to 3 types of transmission in Fig. 5.18: Node 1 is the zonal gateway that transmits RED frames and receive YELLOW frames. Node 2 is the transmitter node on the local CAN bus that transmits YELLOW and GREEN messages to Node 1 and Node 3, respectively. Node 3 is the receiver node that receives RED and GREEN messages. Node 1 also generates two types of flooding frames: one is from the gateway and the other is from outside ECUs.

Table 6.5 summarizes all frames transmitted in the CAN network under test. Red messages are from an outside ECU that belongs to group of SDT = 0x04. Yellow and Green messages are transmitted by Node 2. Gateway flooding frames are generated by the compromised gateway. To simulate the scenario where the gateway is hacked, their SDT value (0x06) doesn't belong to any sub buckets. Outside flooding frames are from outside ECUs that belongs to the ECU group with SDT = 0x01 and AF(0) = 1. To guarantee flooding frames always win bus arbitration, they have the highest PID (0x01).

Upon each transmission or reception of a frame, the node sends a UART message to PC that acknowledges the transmission or reception. Therefore, if a

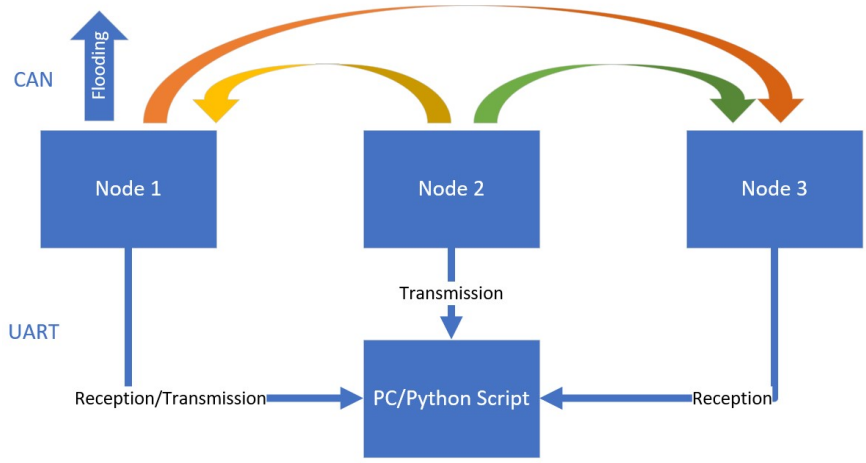

Figure 6.15: **Information flow of the validation platform**

Message type	Red	Yellow	Green	Flooding (outside)	Flooding (gateway)
Period	1000 ms	1000 ms	1000 ms		
Transmitter	Node 1	Node 2	Node 2	Node 1	Node 1
Receiver	Node 3	Node 1	Node 3		
PID	0x0F	0x0E	0x10	0x01	0x01
SDT	0x04	0x04	0x00	0x01	0x06
VCID	0x00	0x00	0x00	0x00	0x00
AF	4*0xFF	4*0xFF	4*0xFF	4*0xFF	4*0xFF
Data	10*0x64	10*0x89	10*0xAB	10*0xFF	10*0xFF

Table 6.5: **Frames transmitted in the CAN network under test**

pair of transmission ACK and reception ACK are monitored by PCAN, we can conclude that this communication is successful.

6.2.3 Results Analysis

In this section, the hardware validation is carried out with these configurations. Next waveform is captured from the oscilloscope to validate the functionalities of the secure transceiver. In addition, acknowledgement data are collected by the Python script to trace the transmission and reception of normal frames.

Waveform Analysis

Initially, flooding from the gateway is tested. We select the format Flooding (gateway) shown in Table 6.5 to be flooding frames from Node 1. At the same time, Node 2 also tries to send out normal messages. Note that since the PID of normal frames is bigger than flooding frames, transmissions from Node 2 will lose the arbitration with Node 1. The waveform of flooded CAN bus is shown in Fig.6.16. The waveform on the right side is the details of the red

box. The purple waveform is mathematical subtraction of CAN H and CAN L. The line under the purple waveform is the decoder information generated by the oscilloscope. From the waveform we can see no frames except for flooding frames can be transmitted on the bus while it is flooded.

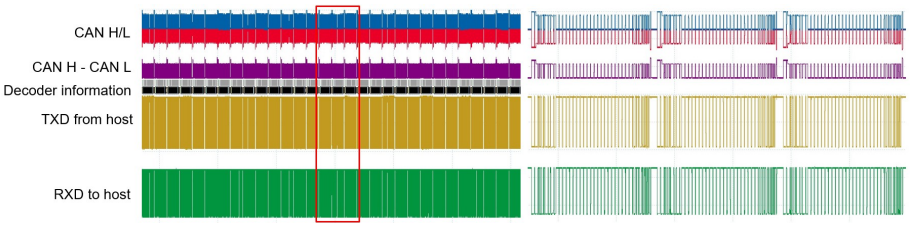

Figure 6.16: **Waveform of flooded CAN bus**

Next, the secure transceiver is enabled to test the flooding prevention result. Fig. 6.17 depicts the waveform when the flooding prevention measure takes effect after the general bucket overflows and normal frames from Node 2 have a chance to be transmitted while Node 1 is disconnected with the bus. From the waveform of CAN bus, we can also notice that the secure transceiver controls the Node 1's occupation of the bus below 50%.

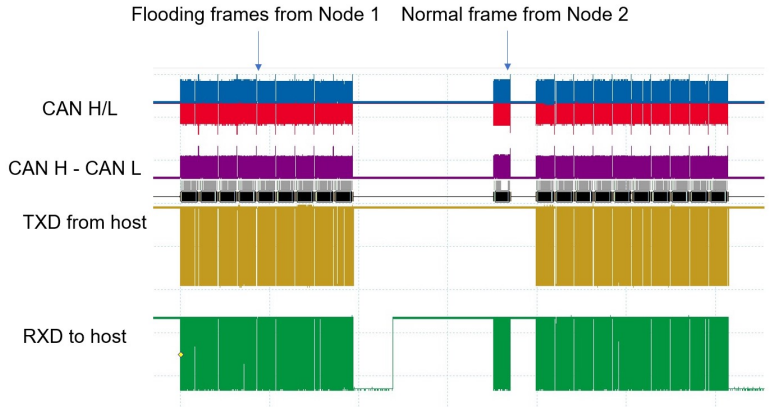

Figure 6.17: **Waveform of flooding prevention when the gateway is compromised**

Secondly, the other scenario where flooding is from other branches via the the gateway is tested. The format of flooding frames is changed to Flooding (outside) shown in Table 6.5. After enabling the secure transceiver, the sub leaky bucket overflows and the waveform is shown in Fig. 6.18. The red box on the left encloses a blocked flooding frame. Note that only the Head part appears on the CAN bus, while the Tail part is replaced by recessive to free the bus. Despite disconnection with the bus, the host still detects valid RXD without bit error. The small red box on the right expands to the picture on its right side, where we can see the ACK slot is overwritten with dominant on RXD.

Fig. 6.19 captures a normal frame from Node 2 while Node 1 is flooding the bus. In this figure, red box contains a blocked flooding frame from Node 1,

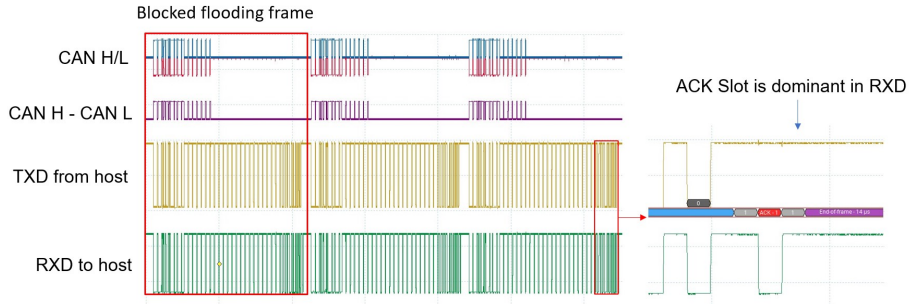

Figure 6.18: **Waveform of flooding prevention and the ACK slot when the outside ECU is compromised**

and blue box contains a normal frame from node 2 which joins the bus while a flooding frame is blocked halfway.

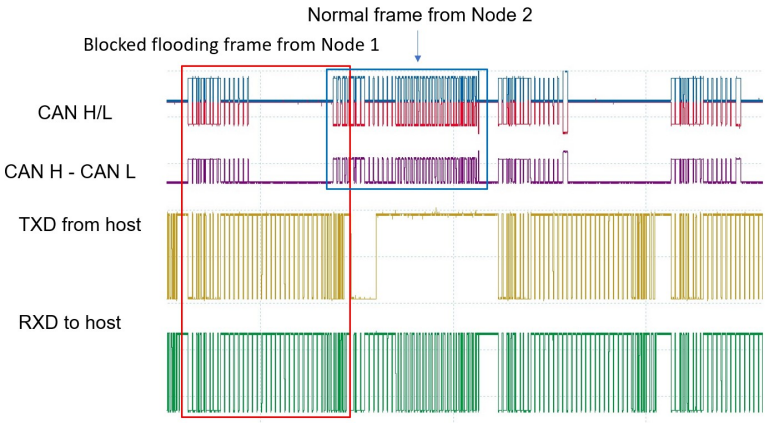

Figure 6.19: **Waveform of a normal frame while the outside ECU is compromised**

Message Reception Experiment

As shown in Fig. 6.15, every transmission and reception of a CAN frame in the experiment is feedback to the PC. The summary of experiment result is shown in Table 6.6. It proves that the secure transceiver protects the availability of the CAN (GREEN messages) bus without interrupting transmission from outside ECUs (RED messages). However, messages sending outwards can't be received by the gateway while its RXD is blocked by the secure transceiver. The result agrees with our analysis in Section 5.7.3.

6.3 Discussion

From the experiment above, the secure transceiver performs according to expectations at blocking flooding attacks and protecting legal communications.

Secure transceiver	Message	Transmit	Receive
Enable	RED	Yes	Yes
	GREEN	Yes	Yes
	YELLOW	Yes	No
Disable	RED	Yes	Yes
	GREEN	Yes	No
	YELLOW	Yes	No

Table 6.6: **Record of message transmission and reception**

Despite the attempt of the host to stop all communication by flooding the bus, the secure transceiver successfully distinguishes between normal messages and flooding messages and only blocks the latter. Additionally, flooding attack isn't the only target of CAN XL security. Other threats such as spoofing and tempering also deserves researcher's attention. For example, spoofing messages from the compromised node can be invalidated by the secure transceiver using error frames, similar to the strategy proposed in Section 5.6.5.

Chapter 7

Conclusions

This chapter summarizes all the achievements of this thesis and highlights a few future research directions. Section 7.1 presents a summary of the main conclusions of this thesis. Section 7.2 suggests a few directions for future research.

7.1 Summary

Chapter 1 of this thesis introduced the threat faced by CAN networks. The current trend is that the automotive industry is extending in connectivity. Hence, attacks on the CAN network have the potential to inflict significant and detrimental damage on passengers. The chapter discussed recent research on these attacks and defenses, and pointed out that this thesis focuses on extending the design of the NXP TJA115x secure transceiver to CAN XL.

Chapter 2 defined the concept of in-vehicle network and highlights its advantages in vehicle systems. In addition, it described the transition from the domain controller architecture into the zonal architecture. The transition reduced wiring and physical hardware in a vehicle.

Chapter 3 explained the CAN protocol in detail. It decomposed the CAN protocol into 4 abstraction layers: application layer, object layer, transfer layer, and physical layer. The introduction included bus arbitration, message frame format, etc. New functionalities specific to CAN XL were highlighted.

Chapter 4 of this thesis gave a brief introduction to the NXP TJA115x secure CAN transceiver. System impacts with state-of-the-art solutions included secure communication startup delay, increase of busload and bandwidth, etc. It highlighted that security features of TJA115x secures the CAN bus without causing these system impacts. This chapter laid a foundation for the new design proposed in this thesis.

Chapter 5 of this thesis proposed the design of the secure transceiver for CAN XL. It pointed out the setback of the strategy used in TJA115x when applied to zonal gateways and proposed flooding detection measure based on leaky buckets tree and two flooding prevention measures, including disconnection measure and error flag measure, to address the challenge. To prevent auto-retransmission of flooding frames, Loopback state was devised to validate the transmissions. It also derived all parameters of the leaky bucket in formulaic form. This chapter finally compared two flooding prevention measures with regard to throughput,

latency, and availability and selected the disconnection measure to be implemented in the secure transceiver.

Chapter 6 validated the design of this thesis in two aspects: software simulation and hardware validation. The chapter first explained the experimental platform for simulation and validation, and then described implementation details for experiment. Finally, the result of both simulation and validation was presented. The result of simulation verified the correctness of parameters derived in Chapter 5 and working of flooding detection and prevention. The result of validation demonstrated that the proposed secure CAN XL transceiver is capable of blocking all flooding attacks while protecting legitimate communication.

7.2 Future Work

In this section, we list a few recommendations to further improve the work of this thesis.

1. Port the implementation of secure transceiver to real CAN XL protocol. As discussed in Section 6.1.2, the current implementation is actually based on CAN FD encapsulation, because lack of hardware support of CAN XL at present. The next step can be the implementation with CAN XL protocol engine.

2. Add more security functionalities for CAN XL, such as spoofing prevention and tampering prevention. The design in this thesis support only defense of flooding attacks on CAN XL bus. Further improvement direction can be novel spoofing and tampering prevention schemes for CAN XL. For example, message filtering strategy with passlist and blocklist is adopted by NXP TJA115x for spoofing prevention. However, it doesn't work on CAN XL since the identifier of classical CAN and CAN FD is now split in PID and AF. In addition, sending error frames to invalidate an malicious frame is not possible if error signalling is disabled in the current CAN XL bus. Research on this problem could complete the security features of the secure transceiver.

Bibliography

[1] Robert N Charette. This car runs on code. *IEEE spectrum*, 46(3):3, 2009.

[2] Hailong Zhu, Wei Zhou, Zhiheng Li, Li Li, and Tao Huang. Requirements-driven automotive electrical/electronic architecture: A survey and prospective trends. *IEEE Access*, 9:100096–100112, 2021.

[3] Tamás Tettamanti, István Varga, and Zsolt Szalay. Impacts of autonomous cars from a traffic engineering perspective. *Periodica Polytechnica Transportation Engineering*, 44(4):244–250, 2016.

[4] Yoshiyasu Takefuji. Connected vehicle security vulnerabilities [commentary]. *IEEE Technology and Society Magazine*, 37(1):15–18, 2018.

[5] Qian Wang, Zhaojun Lu, and Gang Qu. An entropy analysis based intrusion detection system for controller area network in vehicles. In *2018 31st IEEE International System-on-Chip Conference (SOCC)*, pages 90–95, 2018.

[6] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. *black hat USA*, 2014:94, 2014.

[7] Robert Bosch. Can specification. *version 2.0*, Stuttgart, 1991.

[8] Jiajia Liu, Shubin Zhang, Wen Sun, and Yongpeng Shi. In-vehicle network attacks and countermeasures: Challenges and future directions. *IEEE Network*, 31(5):50–58, 2017.

[9] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91):1–91, 2015.

[10] Andy Greenberg. Hackers remotely kill a jeep on the highway—with me in it. `https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/`, 2015. Last accessed: Jun. 14th, 2023.

[11] Andy Greenberg. After jack hack, chrysler recalls 1.4 m vehicles for bug fix. `https://www.wired.com/2015/07/jeep-hack-chrysler-recalls-1-4m-vehicles-bug-fix/`, 2015. Last accessed: Jun. 14th, 2023.

[12] NXP semiconductors. Secure tja115x can transceiver family. `https://www.nxp.com/products/interfaces/can-transceivers/secure-can-transceivers:SECURE-CAN`, 2023. Last accessed: Jun. 14th, 2023.

[13] CAN in Automation (CiA). Can in automation (cia): Controller area network extra long (can xl). https://www.can-cia.org/can-knowledge/can/can-xl/, 2023. Last accessed: Jul. 14th, 2023.

[14] CAN Newsletter. The automotive industry is waiting for can xl. https://www.vector.com/int/en/download/the-automotive-industry-is-waiting-for-can-xl/, 2023. Last accessed: Jun. 14th, 2023.

[15] Tobias Hoppe and Jana Dittman. Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy. In *Proceedings of the 2nd workshop on embedded systems security (WESS)*, pages 1–6, 2007.

[16] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks–practical examples and selected short-term countermeasures. In *Computer Safety, Reliability, and Security: 27th International Conference, SAFECOMP 2008 Newcastle upon Tyne, UK, September 22-25, 2008 Proceedings 27*, pages 235–248. Springer, 2008.

[17] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX security symposium*, volume 4, page 2021. San Francisco, 2011.

[18] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, 2010.

[19] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. *Def Con*, 21(260-264):15–31, 2013.

[20] Craig Smith. *The car hacker's handbook: a guide for the penetration tester*. no starch press, 2016.

[21] Kyong-Tak Cho and Kang G Shin. Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1044–1055, 2016.

[22] Andrea Palanca, Eric Evenchick, Federico Maggi, and Stefano Zanero. A stealth, selective, link-layer denial-of-service attack against automotive networks. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*, pages 185–206. Springer, 2017.

[23] Gedare Bloom. Weepingcan: A stealthy can bus-off attack. In *Workshop on Automotive and Autonomous Vehicle Security*, 2021.

[24] Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede. Canauth-a simple, backward compatible broadcast authentication protocol for can bus. In *ECRYPT workshop on Lightweight Cryptography*, volume 2011, page 20. ECRYPT, 2011.

[25] Chung-Wei Lin and Alberto Sangiovanni-Vincentelli. Cyber-security for the controller area network (can) communication protocol. In *2012 International Conference on Cyber Security*, pages 1–7. IEEE, 2012.

[26] Ahmed Hazem and HA Fahmy. Lcap-a lightweight can authentication protocol for securing in-vehicle networks. In *10th escar Embedded Security in Cars Conference, Berlin, Germany*, volume 6, page 172, 2012.

[27] Bogdan Groza, Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. Libra-can: A lightweight broadcast authentication protocol for controller area networks. In *Cryptology and Network Security: 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings 11*, pages 185–200. Springer, 2012.

[28] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Applying intrusion detection to automotive it-early insights and remaining challenges. *Journal of Information Assurance and Security (JIAS)*, 4(6):226–235, 2009.

[29] Michael Müter and Naim Asaj. Entropy-based anomaly detection for in-vehicle networks. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 1110–1115. IEEE, 2011.

[30] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network. In *2016 international conference on information networking (ICOIN)*, pages 63–68. IEEE, 2016.

[31] Weiying Zeng, Mohammed A. S. Khalid, and Sazzadur Chowdhury. In-vehicle networks outlook: Achievements and challenges. *IEEE Communications Surveys & Tutorials*, 18(3):1552–1571, 2016.

[32] Kathy Pretz. Fewer wires, lighter cars. *The Institute*, 2013.

[33] Leandro D'Orazio, Filippo Visintainer, and Marco Darin. Sensor networks on the car: State of the art and future challenges. In *2011 Design, Automation & Test in Europe*, pages 1–6, 2011.

[34] Ning Lu, Nan Cheng, Ning Zhang, Xuemin Shen, and Jon W Mark. Connected vehicles: Solutions and challenges. *IEEE internet of things journal*, 1(4):289–299, 2014.

[35] NXP semiconductors. Automotive zone controller. `https://www.nxp.com/applications/automotive/vehicle-networking/automotive-zone-controller-:AUTOMOTIVE-ZONE-CONTROLLER`, 2023. Last accessed: May. 30, 2023.

[36] NXP semiconductors. How zonal e/e architectures with ethernet are enabling software-defined vehicles. `https://www.nxp.com/company/blog/how-zonal-e-e-architectures-with-ethernet-are-enabling-software-defined-vehicles:BL-HOW-ZONAL-EE-ARCHITECTURES`, 2023. Last accessed: May. 30, 2023.

[37] guardknox. Zonal architecture automotive e/e. `https://www.guardknox.com/automotive-zonal-architecture/`, 2022. Last accessed: May. 30, 2023.

[38] Dietmar P. F. Möller and Roland E. Haas. *Automotive E/E and Automotive Software Technology*, pages 83–169. Springer International Publishing, Cham, 2019.

[39] Road vehicles – controller area network (can) – part 1: Data link layer and physical signaling, 2021. ISO 11898-1:2021.

[40] Magnus-Maria Hell. The physical layer in the can xl world. *17th iCC proceedings. CiA, Nuremberg*, 2021.

[41] NXP Semiconductors . Can signal improvement capability (sic). `https://www.nxp.com/products/interfaces/can-transceivers/can-signal-improvement:CAN-FD`, 2023. Last accessed: July. 9th, 2023.

[42] Bosch semiconductors for Automotive. Can xl next step in can evolution. `https://www.bosch-semiconductors.com/ip-modules/can-protocols/can-xl/`, 2023. Last accessed: Jun. 14th, 2023.

[43] CAN in Automation. Controller area network extra long (can xl). `https://www.can-cia.org/can-knowledge/can/can-xl/`, 2023. Last accessed: Jun. 14th, 2023.

[44] Ieee standard for vhdl language reference manual. *IEEE Std 1076-2019*, pages 1–673, 2019.

[45] Tcl Core Team (TCT) . Tcl developer site. `https://www.tcl.tk/`, 2023. Last accessed: July. 9th, 2023.

[46] Eric Conrad, Seth Misenar, and Joshua Feldman. *Domain 2: Telecommunications and Network Security*, pages 23–43. 12 2014.

[47] S32k144 evaluation board website. `https://www.nxp.com/design/development-boards/automotive-development-platforms/s32k-mcu-platforms/s32k144-q100-general-purpose-evaluation-board:S32K144EVB`, 2022.

[48] S32k general-purpose microcontrollers website. `https://www.nxp.com/products/processors-and-microcontrollers/s32-automotive-platform/s32k-general-purpose-mcus:S32K-MCUS`, 2022.

[49] Cyclone v fpga and soc fpga. `https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/v.html`, 2023.

[50] Peak can driver website. `https://www.peak-system.com/`, 2022.