HOWARD UNIVERSITY

Fail-Operational Intrusion Detection System (FO-IDS): A Mechanism for Securing Automotive In-Vehicle Networks

A Dissertation
Submitted to the Faculty of the
Graduate School

of

HOWARD UNIVERSITY

in partial fulfillment
of the requirements for the
degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering and Computer Science

by

Habeeb Dipo Olufowobi

Washington, DC
May 2019

HOWARD UNIVERSITY

GRADUATE SCHOOL

Department of Electrical Engineering and Computer Science

DISSERTATION COMMITTEE

Danda Rawat, Ph.D.
Chairperson

Gedare Bloom, Ph.D.

Legand Burge III, Ph.D.

Hassan Salmani, Ph.D.

Indrakshi Ray, Ph.D.
Professor, Dept. of Computer Science
Colorado State University, CO

Gedare Bloom, Ph.D.
Dissertation Advisor

Candidate: Habeeb Olufowobi
Date of Defense: April. 10, 2019

to the most important woman in my life

K.K.B. Olufowobi

for your unwavering love, I am absolutely overwhelmed with gratefulness

Acknowledgements

In the name of Allah, the Most Benevolent, the Most Compassionate.

First and foremost, I would like to thank Allah (SWT) for making me see the light of this day and making this journey possible. Which of the favors of my Lord would I deny? **None!** Alhamdulillah always.

I would like to express my gratitude to my advisor Dr. Gedare Bloom for believing in me and for his countless hours of guidance, painstaking attention in reading my dissertation, encouragement, and most of all, tolerance throughout the entire process of my degree.

Worthy of thanks and appreciation are my committee members Dr. Legand Burge III, Dr. Danda Rawat, Dr. Hassan Salmani, and Dr. Indrakshi Ray who are more than generous with their feedback and criticisms of my research idea. I would like to thank David Hassan, Adeola Odunlami, Gloria Peterson and Ibrahim Onafeko for supporting and tolerating me throughout my graduate studies. I would like to also thank my colleagues and the members of the Embedded System Security Laboratory for supporting my research activities. Special thanks to the faculty of the Department of Electrical Engineering and Computer Science, Howard University and the National Science Foundation (Grant No. CNS 1646317 and CNS 1645987) for their generosity in funding my research. I would like to thank friends and everyone whose name I might not have mentioned that have contributed in part to the successful completion of my doctorate degree.

Ultimately, my sincere gratitude goes to my immediate family for their love, tutelage, constant motivation and support. I am happy to have all of you in my life. I say Jazak Allahu Khayran.

Abstract

The security and privacy of automotive vehicles is a significant problem to address. By adding functionality to enhance safety and comfort, vehicles increasingly depend on electronic control units (ECUs) that communicate through the in-vehicle networks such as the controller area network (CAN). The proliferation of these ECUs in modern vehicles has opened up the vehicular system to cybersecurity risks and attacks. This is because the attack surface of the vehicle increases proportionally to added functions and can be infiltrated through physical or remote access to the vehicular networks. Protecting these networks against attacks has been challenging as the network does not implement any security protocol that can protect the vehicles.

Security solutions such as digital signatures, message encryption, and authentication have been proposed. However, their adoption has been taxing because of the algorithms conflict with the size, weight, power, and cost constraints of the embedded devices. Also, traditional algorithms proposed for the vehicular network do not consider strategies for recovering from detected anomalous or malicious events on the network bus of the vehicle. Therefore, this dissertation introduces a novel fail-operational intrusion detection algorithm that can detect when an attack is imminent in the vehicular network, block the attack from propagating through the network and recover the compromised component.

We developed an algorithm to extract the real-time model parameters of the CAN bus by monitoring the release time of message frames on the bus and develop a specification-based intrusion detection system (IDS). Using an anomaly-based supervised learning approach with the real-time model as input these mappings are then used to detect message frames that do not conform with the extracted timing specification. Furthermore, when a malicious message is detected, an error frame is transmitted to invalidate the frame from being acted on by the other nodes transmitting on the bus and a reboot based recovery approach is initiated to regain the initial state of the compromised node.

We evaluate the effectiveness of the timing model specification with real CAN logs collected from different passenger cars from real-world scenarios and with simulated and real attack datasets. Experimental results show that the algorithm can effectively detect data injection attacks with low false positive rates. Compared with other detection approaches using the timing features of CAN bus messages, our algorithm shows a better performance in detecting malicious events in the CAN bus of the evaluated vehicles. Also, we developed a proof-of-concept implementation of our approach to show that FO-IDS can be implemented on the CAN bus to demonstrate its applicability and evaluate the effectiveness of our recovery strategy with minimal overhead on the bus operation.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AUTOSAR	AUTomotive Open System ARchitecture
CAN	Controller Area Network
CPS	Cyber Physical System
CRC	Cyclic Redundancy Check
CUSUM	Cumulative Sum
DLC	Data Length Code
DOS	Denial of Service
ECU	Electronic Control Unit
EMI	Electromagnetic Interference
FO-IDS	Fail-Operational Intrusion Detection System
IoT	Internet of Things
ISO	International Organization for Standardization
LIN	Local Interconnect Network
MOST	Media Oriented Systems Transport
OBD-II	On-Board Diagnostic Generation Two
OSI	Open Systems Interconnection (model)

REC	Receive Error Counter
RTR	Remote Transmission Request
SOF	Start of Frame
TEC	Transmit Error Counter
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

Chapter 1

Introduction

The connected car industry is quickly growing and by some estimates will account for almost $40 billion in annual revenue by 2020 [1]. This growth is led by Cyber Physical System (CPS) advancements in enhancing safety and automation, and by expanding use of Internet connectivity for in-vehicle infotainment, which brings connected cars into the Internet of Things (IoT). These applications have increased the cyber connectivity, complexity, and contents of vehicles, as demonstrated by the rising number of Electronic Control Units (ECUs), wireless communication interfaces, and software lines of code in the modern vehicle.

Vehicles may contain over 100 embedded control systems, or ECUs, interconnected through the in-vehicle network that facilitates their communications. ECUs perform distinct operations and function individually as a node on the vehicle network. Common networks include the Controller Area Network (CAN), Local Interconnect Network (LIN), and FlexRay. Specifically, our focus is on CAN which is the most commonly used bus system as the automotive network. In CAN topology, each ECU is connected to the same channel for communication through protocols specific to CAN bus, and messages are broadcast to the entire network.

The connectivity and complexity of CPS and IoT have led to a dramatic increase in vehicle functionality, but have also left the vehicular systems, including the safety-critical systems, vulnerable to cybersecurity risks and attack [2]. Such vulnerabilities across the autonomous vehicle, vehicular ad-hoc networks, vehicle-to-vehicle, vehicle-to-infrastructure, connected car, intelligent transportation system, and even traditional (non-connected) automobiles motivate adversaries to launch cyber attacks against vehicles [3]. Today's car

lacks the necessary security mechanisms to protect the vehicular system from attack. The security of connected cars has become a significant concern for the automotive industry.

A critical asset to secure is the automotive in-vehicle network, which facilitates communication between ECUs over multiple physical networks and protocols with the most prevalent being the CAN bus. An adversary may subvert the in-vehicle network through attack surfaces that increase proportionally to new vehicle features. Physical access to the On-Board Diagnostic Generation Two (OBD-II) port can be used to easily compromise the network, while remote access through a wireless or cellular connection can significantly increase an attack's scalability and reduce exposure of the attacker. Checkoway et al. have demonstrated Bluetooth attacks [4], while Miller and Valasek [5] accessed a Jeep Cherokee through its WiFi network by exploiting a weakness in its password generation protocol. Once access to the in-vehicle networks is achieved, the attacker can manipulate and delete data, degrade vehicle functions, and even take over control of the vehicle. The limited computational, memory, and power resources of ECUs hinder the implementation of complex security mechanisms. Hence, lightweight and computationally efficient algorithms are an essential requirement in implementing security mechanisms for the in-vehicle network.

Current research on the security of automotive CAN bus has developed varied approaches to detect malicious and anomalous events but are limited in their detection approach either in the algorithm or the kinds of attacks that can be detected. Also, none of these approaches provide a recovery strategy for the vehicle after an attack is detected. Therefore, this dissertation introduces a Fail-Operational Intrusion Detection System (FO-IDS) for vehicle networks focusing on the CAN bus. CAN bus is the most common communication interface in modern vehicles. It connects the most critical vehicle components and a primary target for cyber attackers. FO-IDS addresses the security issues arising out of the growing use of the access points to vehicular distributed systems that a potential attacker can exploit. When an intrusion is detected, FO-IDS forces the system under attack to undergo a mode switch to a predefined fail operational state. The construction of FO-IDS is divided into two distinct phases: the detection phase that uses the real-time model

of the CAN bus to specify expected behavior and then detects violations of the model as signs of a compromised network, and the recovery phase that implements algorithm-based fault-tolerance approach using a reboot-based recovery.

1.1 Attack and Threat Model

This dissertation focus is on the use of different attack scenarios, in which the goal of the adversary is to control the vehicle. We assume an adversary is able to receive and send messages on the CAN bus. A receive operation involves eavesdropping messages, and a send operation involves transmitting injected (forged or replayed) messages in the CAN bus. We assume the adversary does not interfere with any regular transmission of messages, i.e., it does not cause any denial-of-service attack during access. This assumption fits with the known attacks that penetrate the CAN bus by first subverting a non-critical ECU, and then eavesdrop and inject messages targeting the critical ECUs, but do so while behaving according to the bus protocol.

1.1.1 Access Types

The evolution of ECUs has led to new threats and attack vectors that an adversary can exploit. The landscape of these attacks continues to grow as attackers develop cyber attacks and techniques. Each connected functionality creates new attack vectors, and the diverse, heterogeneous supply chain of automotive ECUs frustrates precise definition of the attack surface. An adversary can gain access to the CAN bus through physical or remote attack surfaces of the vehicle as depicted in Figure 1.1 to target a particular node (ECU) and use it as a foothold from which to compromise the entire network.

Physical access means that the attacker has a direct connection to the OBD-II port of the vehicle connected to the CAN bus and all ECUs. The OBD-II port is a must have for all vehicles sold in North America since 1996. The main purpose of this port is for maintenance and engine failure diagnostics. This port can be accessed easily by an adversary with the

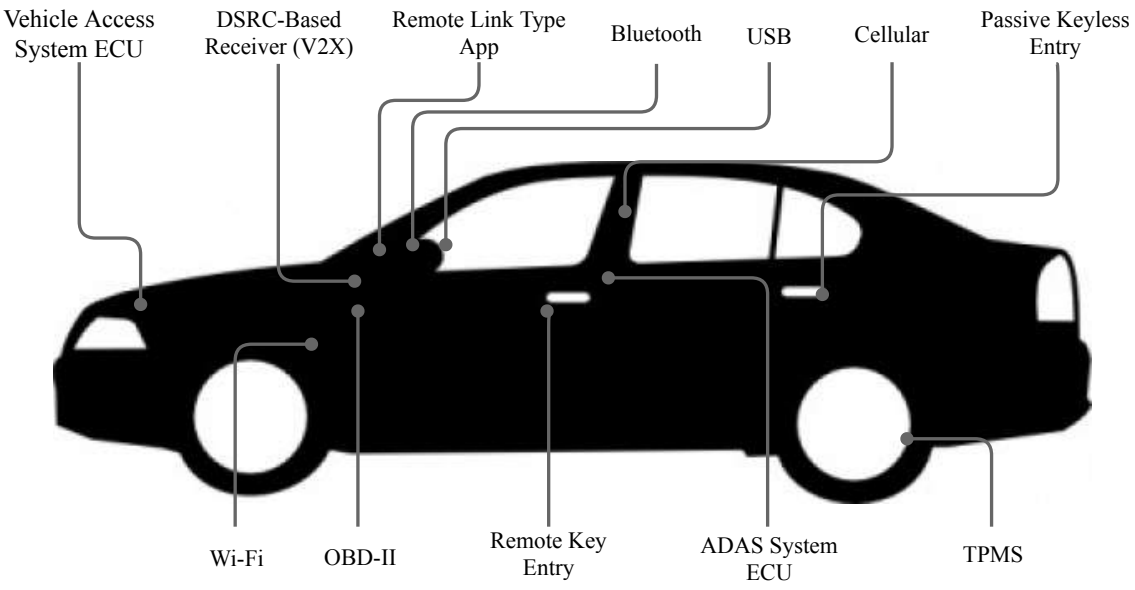


Figure 1.1: Automotive attack surfaces.

right equipment and a window of opportunity. The attacker can plug a small dongle into the OBD-II port to gather information or inject messages directly into the vehicle. Protecting vehicles under such attack is nearly impossible. Also, an attacker can plug a device into the port and access it remotely. Alternatively, an attacker may gain access to the network through the use of the Universal Serial Bus (USB) port. Using these methods, multiple teams have demonstrated overriding security controls to reflash ECUs [6, 7], providing the opportunity to inject or monitor CAN traffic without leaving any physical traces. In this dissertation we assume that the attacker does not have physical access to the vehicle.

Remote and wireless attack surfaces are more worrying because the attacker does not need to physically connect any dongle to the vehicle. These attack surfaces include in-vehicle Bluetooth and the telematics unit that are common in vehicles for wireless and cellular connectivity. Checkoway et al. [4] has demonstrated Bluetooth attacks using various methods of connecting to the communication bus through Bluetooth with malware installed on an already-paired Android phone, and with a method they developed for unauthorized pairing. Miller and Valasek [5] demonstrated unauthorized CAN bus access to 2015 Jeep

Cherokee through its WiFi network that exploits the weakness in its password generation protocol.

Once access to the in-vehicle networks is achieved, the attacker can produce various effects which include, but not limited to, data manipulation, control override, data falsification, data erasure, data replay, and vehicle function degradation. An adversary may compromise various potential attack surfaces given the scale of ECUs in automotive systems and their long lifecycles. Consequently, there exist several legacy systems with no cyber protection capabilities. Hence, a requirement is an efficient approach to security and resiliency for safety and proper operation.

1.1.2 Attack Goals and Capabilities

In this work, we consider the typical effects of attacks on the communication of the CAN bus such as message delays and losses. The attackers have various intent and these can be stated in terms of the impact on the vehicular system operation which includes the safety of its occupants. These goals include vehicle theft, remote hijack, profit and desire for infamy or twisted pleasure [3]. Our focus is on message falsifications due to the malicious activities of the attackers. Hence, we assume that the communication of the CAN bus is reliable and the constraints for the physical layer implementation are met. These constraints includes the requirement that all nodes synchronize whenever a transmission takes place, the arbitration mechanism, and the need for all nodes to agree on the encoded logical value. Also, we assume that other than the expected transient errors, any possible discrepancies in the transmitted and received messages are due to malicious activities of the adversaries.

We identify different scope of attack scenarios which include the adversary a priori knowledge of the automotive networks, their capabilities and resource revealing tools. The a priori knowledge of the network allows the adversary to construct complex attacks that are difficult to detect and with consequences that are severe. Likewise, resource revealing tools like CAN sniffers facilitates the adversary course in obtaining information sensitive

to the internal working of the network that violates data confidentiality. Note that these tools alone are not able to affect or disrupt the vehicle operation but aid the adversary.

We consider several attack scenarios where the goal of the adversary is to place the vehicular system in an unsafe state while remaining stealthy. These scenarios include Denial of Service (DOS) attack, replay attack, and false data injection attack. By accessing the network through physical or remote access an adversary can execute a replay or man-in-the-middle attack by sniffing the legitimate operation of the network. Using the sniffed messages, the attacker can perform a data injection attack by emulating a victim ECU message structure and transmitting arbitrary messages in the bus at random or periodically to disrupt the normal working of the network.

The physical and safety implications of a successful attacks on vehicle connectivity system cannot be over emphasized as national governments have taken steps and proposed bills that would make cybersecurity a necessity of any autonomous driving system.

1.1.3 Attacks Scenarios

Denial of Service Attack. DOS attacks occur when the attacker takes control of the network resources from reaching their respective destinations and resulting in unavailability of data for the vehicular systems to operate accordingly. The attack compromises the overall availability of the network, an essential requirement for the vehicle operation.

Replay Attack. In replay attacks, the adversary performs traffic analysis by intercepting and analyzing communication patterns of nodes and then begins replaying the recorded data with a different timestamps until the end of the attack.

Bias Injection Attack. This attack involves the interception and altering of the communicated message and/or the data field of a message in a coordinated way. The computation of the bias is such that the impact on the system steady is maximized. These attacks will not only achieve the malicious intent of the attacker to undermine the operation of the vehicular network or increase the error detection counter considerably but can also be designed to remain undetected by some of the current anomaly detection algorithms.

Impersonation Attack. An impersonation attack is performed by transmitting injected messages that use the same ID as a legitimate node. The malicious ECU claim false identity or imitate other legitimate ECU (stolen identities) in the network. ECUs on the CAN bus take action based on the most recently received data field of specific IDs that they are programmed to monitor. By transmitting the injected message soon after the authentic message of the same ID is transmitted, the attacker's injected message will be acted on by the ECUs on the bus instead of the authentic message.

For a successful attack on the CAN bus, the adversary needs to understand when to transmit the malicious message. For every real message, the attacker needs to send just one message if the transmission time is known in advance. Otherwise, the message is injected at a high enough rate to arrive soon after the release time of most of the authentic messages. An attacker in the CAN MAC layer will be able to sense when the authentic message is released and fire off a malicious message until it wins arbitration. This attack model will distort the transmission pattern of the injected IDs.

1.2 Research Questions

Different communication protocols have been developed to support in-vehicle networks. The CAN is the most popular and the de facto standard vehicle network communication. Relevant information such as the diagnostics, informative and control data are delivered through the CAN bus for automotive operations. This information must be secured for the driver and passenger safety. However, the in-vehicle network includes several security flaws that have not been addressed.

This dissertation addresses the problem of cybersecurity and resilience in CAN bus. CAN bus do not naturally support any security feature, and traditional security approaches are limited in the scope of the detection as they either profile message properties, rely on the physical features of the ECUs or are not particularly practical during implementation. Therefore theory and tools to examine and model the CAN bus against cyber and physical

threats are, thus, lacking and in need to be developed. The following set of questions are the focus of this dissertation:

1. How can an attack be detected practically and how does the automotive system respond to such an attack?

2. How can the complex challenges be solved to ensure the safety of user and vehicle from cyber attackers? What core components or nodes should be considered in the vehicular network for cyber-secure and resilient vehicular systems?

3. What metrics should be used to evaluate and compare the effectiveness of attack detection algorithms? How can these metrics be applied to estimate the effectiveness of defensive actions?

4. What methods can enhance the cybersecurity and resilience of automotive vehicular networks? Can such methods be implemented and used in real-time?

1.3 Research Contribution

Various research has highlighted the importance of integrating intrusions detection systems in in-vehicle networks. However, due to the complex algorithms and considerable calculations involved in their use, the technology cannot be easily implemented.

The significant contribution of this dissertation is addressing the ability of in-vehicle networks to not only detect intrusions but to correlate the impact on the ability to achieve minimum normalcy during an attack. The technical contribution of this research is in understanding how security and fail-operational can be combined to enhance network resilience. The contributions of this dissertation are as follows:

1. **Specification-based Intrusion Detection using Controller Area Network Timing (SAIDuCANT).** The design and implementation of a specification based IDS that can identify a potential attack in real time using the schedulability analysis

of the CAN bus to specify the intended behavior and then detect violations of the model as signs of a compromised network.

2. **Algorithm-based fault-tolerance approach.** The design and implementation of recovery strategy capable of blocking and terminating malicious injected messages that interface with the detection approach (SAIDuCANT) to perform fault recovery.

3. **Metrics for performance**. Introduced two new metrics for measuring the performance of automotive intrusion detection systems.

4. **Experimental evaluation of the approach.** Evaluation of the efficiency of the detection method on real CAN logs generated from passenger sedan vehicles and simulation of a recovery strategy based on the CAN bus model.

5. **End-to-end evaluation.** Comprehensive end-to-end evaluation of fail operational intrusion detection system (FO-IDS) that can detect fault/failure in the system and perform state recovery.

1.4 Organization of Dissertation

The remaining portion of this dissertation is organized as follows. Chapter 2 discusses the primer on in-vehicle networks, their fault tolerant capabilities, the security challenges and the security mechanism that can be adapted to counter these challenges. Chapter 3 provides a comprehensive survey of background and related work on intrusion detection systems. Also, related work on novel approaches for anomaly detection in other domain is discussed. Chapter 4 outline the details of sequential anomaly detection using adaptive cumulative sum change-point approach. Additionally, the experiment, the evaluation criteria, and performance of the detection approach are discussed. Chapter 5 discusses the response time analysis of the CAN bus and describes the specification-based detection approach using the timing model of the CAN bus. Furthermore, the chapter highlights

the proposed performance metrics and how they are used in the evaluation. Chapter 6 discusses the recovery approach coupled with the detection mechanism with an analysis of both approaches. Finally, Chapter 7 concludes the dissertation and provides a summary of the accomplishment along with an outline of future work.

Chapter 2

Background

In this chapter, we provide an insight into in-vehicle networks and intrusion detection systems as well as a preliminary concretization of the intrusion detection systems for typical in-vehicle network environments.

2.1 Primer on In-Vehicle Networks

A network is a system of interconnected devices in which information can be shared through the transportation medium. While a standard local area network consists of a group of computers, hubs, printers, etc., sharing resources and communicating with each other, an in-vehicle network consists of several complex control units. Such control units include engine management system, transmission control system, the electronic stability control program. These modules communicate via the data bus in real-time to assist in the vehicle operation. Electrical and electronic systems in motor vehicles are not always independent of one another, but influence and complement each other [8]. The demand for enhanced safety, comfort and stringent legal requirements on emission gases for automobiles rapidly increased the number of electronic systems, which in turn increased the demand in the scope of exchanged information in the data bus that led to the development of serial bus system.

An in-vehicle network consists of nodes, gateways, and buses. Figure 2.1 shows a typical in-vehicle network for an automobile. A node is an ECU, which is connected to the bus, the shared data transfer media. Together these buses and nodes form a network. There are many different types of network bus domains in an in-vehicle network, from the most

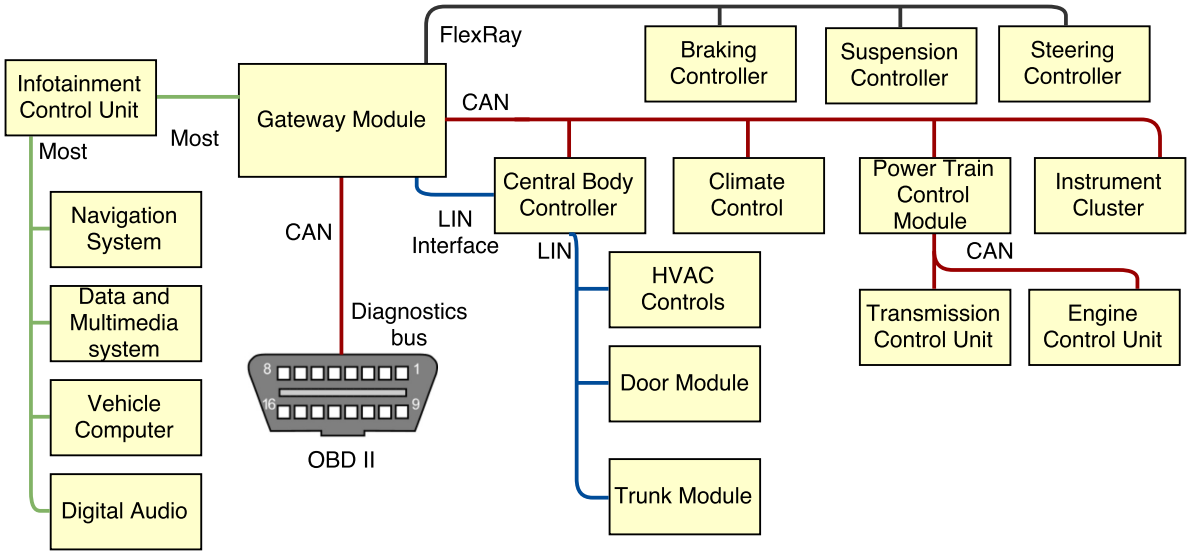


Figure 2.1: Automotive In-Vehicle Network

common CAN, to LIN, and Media Oriented Systems Transport (MOST).

Data is transferred from one network to another through the gateway module, which plays a vital role in the functionality of the ECUs. These ECUs have connectivity to different types of vehicular buses, and the gateway module is tasked with the responsibility of transmitting data between various vehicle domain bus systems. These transferred data between buses with or without the same protocol are critical to ensure the ECUs have the right information to their required function. For example, suppose an ECU on the low-speed bus sends a message to another ECU on a different CAN bus, say the high-speed bus, the gateway is responsible for baud rate and voltage conversion on the bus. Therefore, the gateway is the enabler of communication in the in-vehicle network and functions as a data router as well as a central computing unit between vehicle network domains. Additionally, the gateway module serves as a host for other applications and functions such as vehicle energy management or functional server of the vehicle. A comparison of the various buses of the in-vehicle network is shown in Table 2.1.

LIN is a cost-effective and deterministic communication system based on serial commu-

Table 2.1: In-Vehicle Network Bus Comparison

Bus	**CAN**	**LIN**	**FlexRay**	**MOST**
Application	Powertrain, Chassis, Body Control, Safety	Door/Seat, Engine/Climate, Roof, Steering Wheel	X-by-Wire, Body Control Module, Stability Control	Telematics, Multimedia
Area Used	Soft real-time	Subnets	Hard real-time	Multimedia
Message Transmission	Asynchronous	Synchronous	Synchronous, Asynchronous	Synchronous, Asynchronous
Architecture	Multi-Master	Single-master, Multiple-slave	Multi-Master up to 64 nodes	Single-Master up to 64 nodes
Access Control	CSMA/CA	Polling	TDMA	TDM CSMA/CA
Data Rate	1 Mbps	20 kbps	10 Mbps	25 Mbps
Message Identification	Identifier	Identifier	Time slot	Bits stream
Physical Layer	Dual-Wire	Single Wire	Dual-Wire (Optical Fiber)	Optical Fiber
Error Protection	CRC Parity bits	Checksum Parity bits	CRC Bus Guardian	CRC System Service
Latency Jitter	Load dependent	Constant	Constant	Data stream
Babbling Idiot	Provided	n/a	Provided	n/a
Extensibility	High	High	Low	High

nication protocol for connecting ECUs with smart sensors and actuators. LIN is a single-master, multi-slave, Universal Asynchronous Receiver-Transmitter (UART)-based network architecture that uses a single wire to transmit data. In the in-vehicle network, LIN is primarily used for short distance network communication in the area of comfort functions, such as door locks, car seat, mirrors, etc., because of their low speed requirements. FlexRay is a scalable, flexible high-speed, time-deterministic communication system that is used in safety critical areas of an automobile. FlexRay offers data transfer with high bit rates (10 Mbit/sec per channel), fault tolerance, and guaranteed compliance with the transfer properties (scalable data transmission). FlexRay is designed to support flexible network layout such as bus (multi-drop), star, or hybrid topology and it complements the CAN and LIN

bus because of its suitability for both powertrain systems and x-by-wire systems. MOST Infotainment network is a dedicated serial communication system for transmitting multimedia streaming data such as audio, video and control data via fiber-optic cables. The MOST bus is a synchronous network that offers plastic optical fiber and coaxial physical layers. The bus supports a data rate of approximately 25 Mbps in synchronous control channel and 14.4 Mbit/s in asynchronous mode of transmission. MOST supports point-to-point network topology via the ring topology.

2.1.1 Controller Area Network (CAN)

CAN bus was first introduced in 1991 to mass production of motor vehicles [8] and it is the most commonly used bus system in automotive network. CAN is an asynchronous, serial, multi-master communication network protocol that connects electronic control units (ECUs) in a transport network. CAN supports bit rates in the range of 1Kbps to 1Mbps. Low speed CAN normally has a data rate up to 125Kbps while the high speed CAN has a data rate of 125Kbps to 1Mbps. Beyond automotive applications, the CAN protocol is being used as a generic embedded communication system for micro-controllers as well as a standardized communication network for industrial control systems and avionics [9].

CAN is a message-based protocol that uses a lossless bitwise arbitration to transmit binary signals data. It uses the term dominant bits to represent the logical 0 and recessive bits for the logical 1 signals. When the voltage difference between the two wires is large, the state is dominant. The state is recessive when the voltage difference is small between the two wires. Dominant state overwrites the recessive state when two or more ECUs send messages at the same time. As shown in Figure 2.2, data is transmitted between ECUs in the Data Frame of the CAN packet, which include an Arbitration field, Control field, Data field and a Cyclic Redundancy Check (CRC). The CAN protocol includes collision detection and avoidance, error detection, signaling, and fault confinement.

The CAN bus communication follows the Open Systems Interconnection (model) (OSI) model with four different layers which include the application, object, transfer and physical

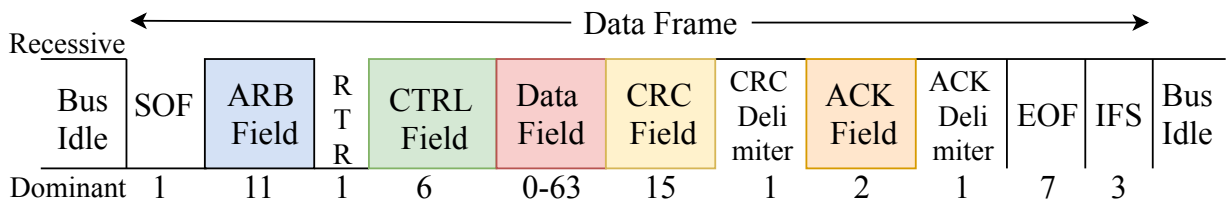


Figure 2.2: CAN Message Structure

layers. As shown in the Figure 2.3, CAN bus usually requires three of the seven OSI layers for on-board communication except for the OBD. These layers are the Physical Layer, the Data Link Layer, and the Application Layer. The process of signal transmission between different medium is defined in the physical layer while the transfer layer denotes the CAN protocol kernel which comprises message framing, arbitration, validation, acknowledgment, error signaling and detection, fault confinement, and transfer rate and timing. The object layer is involved in message status, filtering, and handling [8].

CAN efficiently implements static fixed priority non-preemptive scheduling of messages through bus arbitration. CAN messages may be periodic, sporadic, or aperiodic. Periodic message instances arrive at a regular interval with a fixed length called period. Sporadic messages recur with a minimum inter-arrival time between successive instances, while aperiodic message instances occur at arbitrary times. The period of a message is the fixed time interval after which a message releases another instance.

Each transmitting message goes through the arbitration process to determine which wins the bus. When a message wins arbitration and starts transmission, it becomes non-preemptable. Messages win arbitration according to their priority, which is determined by the message identifier (ID). A message with a lower ID has higher priority.

CAN implements the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism with no central controller for the network itself. The CAN protocol does not use an addressing scheme, but instead packets are broadcast to all nodes. The connected ECUs communicate through protocols specific to the CAN bus. When two or more ECUs transmit messages simultaneously, arbitration is done and the dominant (logic 0) wins as

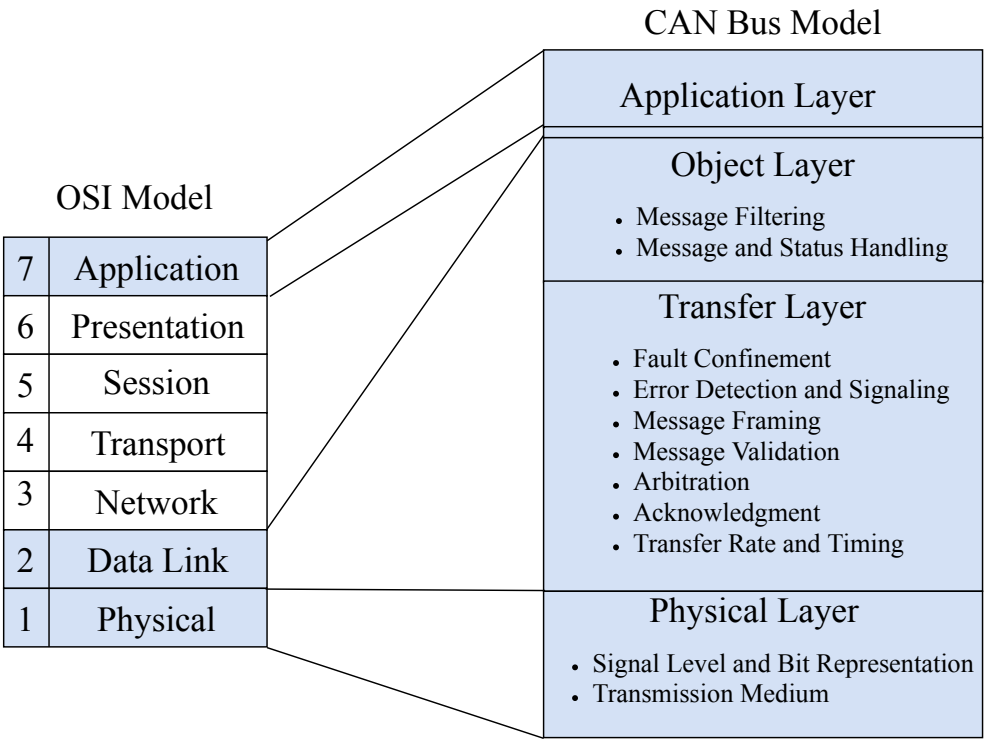


Figure 2.3: CAN Bus Layered Structure

low-valued identifiers always have higher priority to access the bus. Any node (ECU) may start a transmission if it determines that the bus is idle. When an ECU starts to transmit a Data Frame, Start of Frame (SOF) is transmitted, then the first bit of ID field is transmitted. When two or more ECUs attempt to transmit different bit values at the same time, the dominant bit overwrites the recessive bit by physical implementation of CAN. The ECU which is attempting to transmit a recessive bit cancels its transmission whenever it recognizes that another ECU is attempting to transmit a message with higher priority. This way when arbitration is realized, only the message with the highest priority is transmitted on the bus [10].

CAN bus is susceptible to faults due to Electromagnetic Interference (EMI). EMI errors can be modeled as a random single bit fault in CAN bus, which if detected by any receiver will signal an error frame and cause retransmission of the original message [11, 12]. If an error is detected either by the sending node or in the CRC field, the error is signaled

directly to all the nodes on the bus. The receiving nodes will discard the received erroneous message, and the sending node, assuming only a transient fault on the wire, then enters arbitration to retransmit the message frame. The error recovery process transmits up to 31 bits in the worst case (the error signaling and recovery time is typically between 17 to 31-bit times) in addition to the retransmission of the message.

CAN bus implements no security mechanism such as data encryption, authorization or authentication because it was designed as a closed network. CAN protocol does not use an addressing scheme, instead packets are broadcast to all nodes. Therefore, CAN messages are not immune to spoofing attacks since messages contain no information about the sender and the broadcast nature of the bus can allow an intruder to easily eavesdrop, intercept, and transmit invalid messages on the network.

2.1.1.1 CAN Frames

Message transfer is established and controlled by four different frame types in CAN protocol. They include a data frame, remote frame, error frame, and overload frame. The Data Frame transfers data between the nodes on the bus (from the transmitter to the receiver). A unit in the bus transmits the Remote Frame to request the transmission of the Data Frame with the same identifier. On detecting a bus error, an Error Frame is transmitted by any unit in the bus to signal the occurrence. The Overload Frame is used to provide for an extra delay between the leading and succeeding Data or Remote Frames.

Data transmission occurs in the Data Frame, and the data is placed in the Data Field, which can be transmitted periodically or otherwise. One Data Frame can transmit 0-8 bytes data. The Data Frame comprised seven different fields. The arbitration field contains the 11-bit identifier and the 1-bit Remote Transmission Request (RTR). The message identifier field represents the ID of each message and its priority while the RTR field distinguishes a data frame from a remote frame. A message with lower ID have higher priority, and when two or more nodes start transmitting messages at the same time, arbitration is done, and the message with the highest priority gets to transmit. Arbitration mechanism guarantees

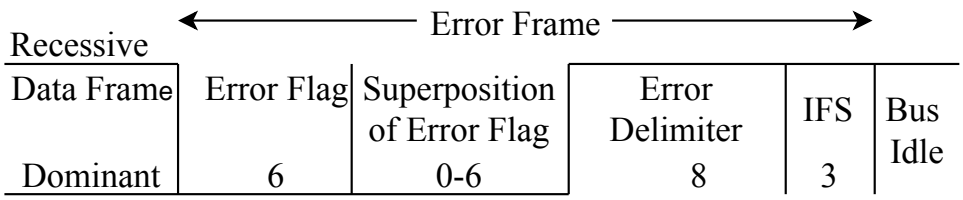


Figure 2.4: Error Frame

no information or time lost. When in arbitration, the transmitting node compares the level of the transmitted bit with the level monitored on the bus. If these levels are equal, the node may continue to send else, if a dominant level is observed, the node has lost arbitration and must withdraw without sending one more bit. This way, arbitration is realized, and messages with the highest priority get transmitted on the bus.

A message that announces an error is called an Error Frame, which is composed of a 6-bit error flag and 8-bit recessive. There are several kinds of errors defined by the CAN protocol while sending or receiving a message. These errors are; Bit Error, Staff Error, CRC Error, Form Error and ACK Error. When a node detects any of these five errors, it transmits an Error Frame to all nodes to control the error as depicted in Figure 2.4. A bit error occurs when the transmitted bit logic differs from the received bit logic. When a bit stuffing rule is violated, a stuff error occurs. When an error is detected in the CAN controller, the values of the Transmit Error Counter (TEC) and Receive Error Counter (REC) are increased. If the transmission ECU detects a bit error or a stuff error, the TEC of the ECU increases by 8.

2.1.1.2 CAN Error Management and Fault-Tolerance

The CAN specification protocol includes a control mechanism for error detection, signaling, and fault confinement. The error detection and signaling of the controller ensures that data flowing through the network is correct and consistent. Defective nodes are prevented from interrupting the operation of the network by placing them in a passive, or bus-off state. The CAN protocol provides a level of error detection such as the cyclic

redundancy check that performs error checking on the content of the data frame, acknowledgment check, monitoring, and bit stuffing. When an error or a fault is detected by the CAN controller, the transmission is interrupted by sending an error frame immediately. The message is canceled at all receiving nodes to ensure consistency, and the sender can attempt to resend the message at a later time by further bus arbitration. CAN uses a twisted pair cable to communicate at speeds up to 1 Mbps with up to 40 devices. The use of balanced differential receivers and twisted-pair cabling enhance the common-mode rejection and high noise immunity of a CAN bus.

Signaling is differential which is where CAN derives its robust noise immunity and fault tolerance. Balanced differential signaling reduces noise coupling and allows for high signaling rates over twisted-pair cable. Balanced means that the current flowing in each signal line is equal but opposite in direction, resulting in a field-canceling effect that is a key to low noise emissions. The use of balanced differential receivers and twisted-pair cabling enhance the common-mode rejection and high noise immunity of a CAN bus.

2.2 Security Challenges of Automotive In-vehicle Networks

Security of the in-vehicle network has been a major challenge for the automotive industry. In-vehicle networks are originally designed to be a closed system that does not need to communicate with an external network. Hence, the security of the network was not explicitly addressed. However, due to the increase in vehicle functionality, in-vehicle networks are connected to the external network with little or no provision for security. With no security mechanism in place, an attacker will be able to access the network and perform unauthorized actions. These actions may cause system malfunction and lead to safety-critical hazards such as accidents on the road.

2.2.1 Security Challenges of CAN

There have been several publicized attacks on the vehicular network by researchers that requires cyber protection. These attacks do affect not only the vehicles but also the infrastructure which in turn influence the traffic situations. Securing automotive in-vehicle networks is challenging because of resource constraints and the need to maintain predictable performance of the ECUs. Due to their low-cost nature and low data rates, most ECUs have limited computing power and memory to implement security mechanisms. Automotive networks have significantly lower transmission capability compared to network systems. CAN bus has a maximum and nominal data rate of 500 Kbps and the CAN base frame with 11 bits identifier has a maximum of 134 bits. With an original frame of maximum payload (64-bit), a security mechanism that adds message authentication codes to this frame might result in it splitting into multiple smaller frames. This could in turn increase communication overhead due to increased bus utilization.

The CAN bus has inherent vulnerabilities to eavesdropping, denial-of-service, message injection, and impersonation attacks because of its broadcast nature, lack of message authentication, lack of message encryption, and message prioritization.

2.2.1.1 Broadcast Nature

Every CAN message is broadcast to all nodes in the network. The ECUs then decide whether the message is meant for them, to act on. Broadcast allows for an intruder node to easily eavesdrop and analyze the messages transmitted in the network.

2.2.1.2 Lack of Message Authentication

A CAN message does not contain any information that indicates its source. A receiving node is unable to distinguish a malicious message from a benign one since there is no built-in message authentication mechanism. Hence, an adversary can easily spoof messages and take control of the vehicle.

2.2.1.3 Lack of Message Encryption

Message encryption provides confidentiality and integrity. CAN network implements no message encryption, and an adversary can easily analyze, modify and inject malicious messages into the network.

2.2.1.4 Message ID Priority

In CAN network, the messages with smaller message ID has higher priority. Messages with higher priority always win arbitration during transmission, and the dominant bit always overwrites the recessive bit when two or more ECUs attempt to transmit different bit values at the same time. Thus, a malicious node can deny legitimate nodes by continuously transmitting higher priority messages that will block the legitimate messages, which is a denial of service attack (DOS).

Previous studies on physical and remote access on the CAN network have shown several vulnerabilities in the security of the in-vehicle network. Possible attack vectors that may be exploited by an adversary include message sniffing, malicious messages injection, and denial of service attacks to disrupt the normal vehicle operations.

Message injection attacks involve the ability to inject sniffed or dominant message bit into the message frame of the CAN bus by impersonating a legitimate ECU to control other ECUs. Similarly, continuous message injection and generating continuous dominant bits will completely prevent other nodes from transmitting messages on the bus causing a denial of service attacks to communicating ECUs. This kind of attack can be directed to a single ECU or compromise the entire bus communication.

Message injection attacks can be achieved through single, multiple or massive message ID injections to disrupt the bus operations. Injecting a single message ID continuously into the CAN bus aims to make the vehicle conform and operate in a manner corresponding to the injected message. Multiple message ID injections aim to cause malfunction of the vehicular systems. Massive injections of messages or flooding the CAN bus with thousands

of messages per second can paralyze the whole operation of the bus causing a denial of service (DoS) attack and endangering the ability to control the vehicle. An impersonation attack occurs when the injected message ID is one used by a different ECU than the (malicious) sending ECU.

2.3 Intrusion Detection Systems (IDSs) for In-Vehicle Networks

Several IDSs have been proposed for securing the in-vehicle network. Both specification based and anomaly-based detection approach have been addressed in automotive domain. Prior approaches are explained below.

2.3.1 Intrusion Detection Systems (IDSs)

Intrusion is an unauthorized or unapproved activity in a computer system or network and is defined as attempt to compromise the confidentiality, integrity, availability, or security mechanisms of a computer or network [13]. The security of a network has been an important topic since the inception of computer networks.

An IDS is a security tool that monitors network traffic to identify and report possible attacks within the network. Usually, an IDS consists of sensors, management and reporting systems. The sensors are the monitoring agents in the network. They collect real time network or host data to detect any malicious activities. IDS are designed to reveal intrusions, they are required not to introduce new weaknesses to the system. IDS are commonly classified as network-based IDS (NIDS) and host-based IDS (HIDS). HIDS monitors the operation of a computer system such as systems calls, file systems, running processes, etc, and the network traffics on its network interface. NIDS monitors and analyzes incoming network traffics on one or more network segments. Many IDS can also be classified by their detection methodologies as anomaly-based, signature-based, or specification-based. Worthy

of note is that IDSs can only detect an intrusion, and they are not able to offer comprehensive protection for network or information systems. Preventive and reactive measures against detected attacks have to be implemented in order to have a secure network.

2.3.1.1 Anomaly-Based Detection

Anomaly-based intrusion detection method detects an attack based on the deviations from the established normal activities profiles [14]. Any activities that exceed the predefined threshold are considered an intrusion (anomalous behavior), while activities below the threshold are considered normal behavior. Anomaly detection finds patterns in data that deviates from the normal behavior. Anomaly detection has the ability to detect new or unknown attacks and, likewise, has the potential of generating many false positive (false alarms). One major advantage of this detection method is the ability to customize the baseline of normal activities of a system, and new threats can be detected without needing an update. Anomaly detection based on supervised approaches has an advantage in detecting novel or modified attacks.

2.3.1.2 Signature-Based Detection

In signature based intrusion detection method, IDS detects an attack based on predefined rules of different security attacks. Signatures of earlier known attacks are generated and stored in the IDS internal database. These are used as a reference to detect future attacks. Any system or network activities that match the stored signatures are considered an intrusion. The advantage of this detection method is that it can detect known attacks efficiently with low false positive rate. However, the difficulty in defining its rule set is a major drawback. Furthermore, signatures are very ineffective in detecting new attacks and variants of known attacks, because a matching signature is still unknown for these kind of attacks [15]. A signature-based IDS works just like an antivirus software, and keeping the signature database up to date is sometimes a daunting task.

2.3.1.3 Specification-Based Detection

Specification-based intrusion detection is a form of anomaly based intrusion detection where no user, group or data profiling is used. Instead legitimate behaviors are specified, and a node's misbehavior is measured by its deviation from the specification [16]. In specification based detection method, expected behavior of critical network components are manually extracted and crafted as security specifications [17]. These specifications are used to characterize the legitimate behavior of the systems and any deviation from these behaviors are considered an intrusion. Intrusions which normally cause an incorrect object behavior can be detected without any exact knowledge about them. Specifications that are manually defined usually provide low false positive rate when compared with anomaly based detection method [18]. An advantage of this method of detection is that the system is effective immediately when the specifications are defined as there is no user or data profiling involved. However, the amount of work required in generating specifications is a major drawback of this approach.

2.3.2 Response Type

An IDS has two different response types in general, active and passive: An active response IDS provides an automatic response immediately when an attack is detected while passive response IDS are those that produce an alarm for a detected attack.

2.3.2.1 Active Response

Minimizing the effect of intrusion and thwarting further damage in a target system is the main idea of the active response IDS. Active responses can be described by three different categories: collecting additional information about the attack to resolve the attack type and effects, changing the environment by blocking or reconfiguring the system components and taking action against the intruder [19]. An IDS with an active response mechanism requires a real-time detection method since the difference in time of the start of an attack

and the detection exposes the system to exploitation. A major concern of this method is the likelihood of false response, such as blocking normal traffic, because of an incorrect implementation or configuration [20].

2.3.2.2 Passive Response

The response is intended to notify the user of the system or an administrator of potential intrusion rather than taking actions automatically to halt the attack. The alarm may come with a report that includes system logs, potential vulnerabilities, and attack types to allow the administrator to perform a further investigation. A major issue of this approach for critical systems is the delay between the intrusion and the human response [21].

Chapter 3

Related Work

Security problems of in-vehicle networks have been studied over the years by several researchers [22]. Koscher et al. [23] were the first to demonstrate and perform practical attacks on vehicles. The authors demonstrated complete control of a wide range of automotive functions by sniffing the CAN bus and reverse engineering of ECU code. These attacks include disabling the brakes, stopping the engine, and inserting malicious codes in the car's telematics unit and that will ultimately delete any proof of its presence after a crash. They were able to launch these attacks by directly interfacing with the OBD-II port and remotely using the external facing vulnerabilities of the car. The authors provide comprehensive experimental results to specific attacks by assessing the behavior of real automobiles and its components. Hoppe et al. [24] demonstrated practical attacks on the CAN bus and their vulnerabilities, and demonstrated an anomaly detection method by looking at frequency of messages transmitted on the bus while Checkoway et al. [4] presented an experimental analysis of the externally-facing attack surfaces in a modern automobile.

Existing works have applied cryptographic techniques to in-vehicle networks, such as digital signature, encryption, and message authentication codes [25, 26, 27, 28, 29, 30, 31], but the communication overhead of these techniques are very high, making them unsuitable or at least difficult in practice for the CAN bus.

A plethora of automotive in-vehicle network IDSs have been developed over the years [32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44] that explore methods of detecting anomalies as indicative of intrusions. We investigate how researchers are applying different detection techniques to securing CAN bus. In Table 3.1, we summarized some of these works, the type of intrusion detected and their evaluations. Portion of the work presented in this

chapter has also been discussed in [45].

Table 3.1: Comparison of Proposed IDSs for In-Vehicle Networks

Detection Feature	Proposed System	Intrusions Detected	Evaluation
Message Timing Feature	Miller and Valasek (2016) [7]	Message Injection	Live Road Tests
	Gmiden (2016) [46]	N/A	No Evaluation
	Song (2016) [38]	Message Injection	Live Road Tests
	Taylor (2015) [47]	Message Injection	Real Vehicle Simulation
	Moore (2017) [42]	Message Injection	Real Vehicle Simulation
	Young (2019) [48]	Message Injection	Real Vehicle Simulation
	Olufowobi (2019) [49]	Message Injection	Real Vehicle Simulation
Rule-Based	Hoppe (2011) [50]	Message Injection and Deletion	Testbench Simulation
	Müter (2010) [32]	Message Injection	No Evaluation
	Larson (2008) [51]	Known Attacks with Defined Signatures	Theoretical Simulation
	Cho and Shin (2016) [52]	Bus off (DoS)	Real Vehicle Simulation
Entropy	Müter (2011) [53]	Various Attacks that Effect System Entropy	Real Vehicle Simulation
	Marchetti (2016) [39]	Message Injection	Real Vehicle Simulation
Physical Characteristics	Cho and Shin (2016) [37]	Spoofing	Real Vehicle Simulation
	Ji (2018) [54]	Injection and Suspension Attacks	Testbench Simulation
	Choi (2018) [43]	Bus-Off Attack	Real Vehicle Simulation
CAN Data Fields	Boudguiga (2016) [41]	Message Spoofing	No Evaluation
	Markovitz (2017) [55]	N/A	Real and Simulated CAN Traffic
	Kang and Kang (2016) [36]	Attacks Based off Statistical Features	SW Simulation with OCTANE
	Jichici (2018) [56]	Message injection	SW Simulation with CANoe
	Taylor (2016) [57]	Message Injection	Real Vehicle Simulation
	Wang (2018) [58]	Replay Attack	Real Vehicle Simulation
	Martinelli (2017) [59]	Message Injection	Real Vehicle Simulation
	Kuwahara (2018) [44]	Message Injection	Real Vehicle Simulation

3.1 Message Timing-Based IDS

The concept of analyzing the rate of messages for CAN bus intrusion detection was presented by Miller and Valasek [5]. The number of messages on the CAN bus is the sum of the number of normal messages and attack messages. By analyzing the distribution rate of messages, it should be possible to detect anomalous messages. Various researchers have explored message timing features for detecting anomalies in CAN network. These works have used message intervals and frequency for detecting cyber threats and attacks on automotive networks.

Gmiden et al. [46] proposed a simple intrusion detection method for CAN bus based on the analysis of time intervals of CAN messages. An advantage of this approach is that it does not require any modification in the hardware layer and implementation in each ECU.

An algorithm that measures inter-packet timing over a sliding window was proposed by Taylor et al [47]. Adapted from industrial control systems, the authors applied a flow based method to the CAN bus to measure changes in the data content and their frequencies then compared them to the historical value for anomaly detection. Communications between endpoints of the CAN networks are contained in the flow statistics which are trained using the One Class Support Vector Machines (OCSVM) for flow classification. The result of this approach was evaluated using time test statistics of the flow detector and consistent detection improvement is observed with OCSVM as the number of packet insertions increased with time.

Song et al. [60] demonstrated a low false positive IDS that uses time as a feature for CAN bus anomaly detection. The authors proposed a lightweight intrusion detection algorithm for in-vehicle networks by examining the time interval of CAN messages. They determined that the time interval is a good feature in detecting message injection attacks in CAN bus traffic. The IDS computes the time difference in the arrival of every new messages appearing in CAN network and consider it an injected message if the time interval is shorter than the predefined normal. The IDS also increases the denial of service attack score by one per message interval arriving with less than 0.2 ms in a row. The authors claim that their approach shows an improved performance of better detection accuracy over the message rate based IDS technique.

Moore et al. [42] used the assumption that most ID signals are frequently and needlessly sent to create an hypothesis that model and detect anomalies in the inter-signal wait time of the CAN packet. They proposed that an anomaly detection system monitoring the inter-signal wait times of CAN bus traffic will provide accurate detection of a regular-frequency signal injection attacks. The authors conclude that this approach provides a near perfect true positive and false detection rates as evident in their empirical results.

Using CAN message timing features for attack detection has shown good promise with minimal change to the vehicular networks. This approach has shown the most success in identifying known attacks. However, the simplicity of these methods currently limits them

from detecting well-crafted attacks against the automotive network. While the majority of demonstrated attacks have been message injection, it is conceivable that there are other attack methods. Thus, alternative detection approaches need to be examined.

Salem et al. [40] proposed inter-arrival curves to model the behavior of a system using lower and upper bounds of inter-arrival occurrences of events within a trace. The authors analyze the behavior of events within a CAN trace then demonstrate how inter-arrival curves act as a good feature to extract the recurrent behavior of the CAN network for anomaly detection. However, the authors highlight that inter-arrival curves are not able to detect anomalies that affect event timestamps. Our approach clearly differs from this because the timing of each event is considered in detecting anomalies.

3.2 Rule-Based IDS

Early works on detecting intrusions in the vehicular networks utilized rule-based detection approaches.

Müter et al. [32] introduced an approach for anomaly detection using sensors to recognize attacks on in-vehicle networks during normal vehicle operation. The authors discussed the design and the application criteria for attack detection in the network, especially the CAN bus, without causing false positives. This detection scheme consists of eight different sensors for detecting an attack. The sensors serve as a criterion for recognizing a threat to the automobile by monitoring different aspects of the network. In their proposed approach, the applicability of these sensors is based on different criteria such as the type and number of messages, the number of buses they need to access, and if the payload of the message needs inspection. The authors showed sensor data results can be evaluated and how to integrate the approach into a holistic IDS concept, but provide no information on the experiment run that provides no false positive.

Hoppe et al. [50] demonstrated practical attacks on the CAN bus and their vulnerabilities. The demonstrated attacks include manipulating the electric window lifts, warning

lights, and the airbag systems. The authors proposed an IDS for in-vehicle networks by identifying notable attack patterns such as increasing the frequency of messages, the apparent message IDs misuse and communication characteristics of ECUs that can be used to detect attacks in a vehicle. The authors claimed that the implemented approach is appropriate in detecting exemplary attacks on the warning light system.

Larson et al. [61] investigated the application of specification-based IDS approach for the CANopen protocols. They introduce an approach to detect an attack on in-vehicle networks by creating a set of security specification for in-vehicle networks and compare the current system behavior with the predefined patterns. They show that potential attacks can be detected from the trace of extracted information through theoretical simulation. The authors concluded that the most important ECU to protect is the gateway ECU because if its compromised, investigated attacks can be performed.

Cho and Shin [52] proposed a type of denial of service attack called bus-off attack that exploits the error handling scheme of the CAN bus to disconnect or shut down an uncompromised ECUs. The proposed attack is limited to periodic messages as it uses the periodic feature of CAN messages for synchronizing its transmission time with the victims. The authors implement and demonstrate this attack on a CAN bus prototype and an actual vehicle. Furthermore, they developed and evaluated a countermeasure for detecting and neutralizing such attacks which involves resetting the ECU or its error counter after consecutive errors frames occur.

Studnia et al. [62] also proposed a language based IDS which uses language theory to elaborate a set of attack signatures from a behavioral model of the ECU. This idea involves automatic generation of forbidden sequences from patterns characterizing the expected behavior of ECU based on language theory. Formal language is used here to describe a set of attacks which vehicles may be subject to. This approach describes an algorithm that can be used to overcome the excessive use of memory which may be necessitated by an increase in states caused by the language.

3.3 Anomaly-Based IDS

This section studies the anomaly detection approaches designed to monitor the network to detect possible anomalies, i.e., deviations from nominal behavior. Anomaly detectors are assumed to be placed on the network to evaluate its behavior. One of the limiting factors in implementing complex IDS approach is the computation power of ECUs. Anomaly-based intrusion detection system has been applied to traditional network-based systems to detect anomalous behaviors in the network. As most messages in the CAN bus tend to be periodic, detection approach using message entropy, ECU physical characteristics, and machine learning based approaches have been proposed to analyze and detect anomalous behavior in the network. Some of the techniques are computationally intensive and may require a significant redesign of the underlying vehicular architecture for their implementation.

3.3.1 Message Entropy

Müter and Asaj [53] proposed an IDS that records normal traffic and detects anomalies using an entropic measure. The authors investigate their approach through a different dimension of information-theoretic detection approach. The first dimension includes the data abstraction level, considering existing classifiers of the binary, signal, and protocol level for the data selection. The second dimension involves the information-theoretic measures used for the detection which includes conditional self-information, entropy, and relative entropy. Lastly, the vehicle status, or the state in which the vehicle is in, is also considered in the evaluation. Using these dimensions, the authors evaluate three attack scenarios: increased frequency, message flooding, and plausibility of interrelated events to reveal that an information-theoretic detection approach can recognize differences in normal behavior of in-vehicle networks.

Marchetti et al [39] proposed an experimental evaluation of anomaly detectors based on the entropy of the in-vehicle network. The algorithm is based on the assumption that the

entropy values that are too distant from the average entropy are anomalous. Entropy values are said to be stable over time and distributed normally. The effectiveness of the algorithm on CAN trace data was evaluated by its ability to distinguish between anomalous (forged) messages from normal. The authors conclude that anomaly detectors based on entropy are practicable detection approach for identifying injection attacks on the CAN network. They claimed that the time granularity used by the detection model has no direct impact on the performance of the entropy-based detection approach as shown by the experimental results.

3.3.2 ECU Fingerprints

Cho and Shin [37] introduced a clock based IDS that uses clock skew (timing error) to authenticate ECUs. The IDS records communications on the CAN bus and creates fingerprints of the every ECUs on the network. Each ECU is assigned a fingerprint based on specific clock skew and this is used to distinguish them. The authors proposed that by analyzing the CPU clock's behavior, spoofing attacks can be detected in the network.

Similarly, Choi et al. [43] proposed VoltageIDS that leverages the immutable characteristics of the electrical CAN signal to fingerprint the ECUs to detect two different ECUs sending identical messages.

Lee et al. [63] proposed an IDS called OTIDS that measures the response performance of network nodes based on the offset ratio and the time interval between request and response in CAN messages. The authors claim that each node has a fixed response offset ratio and time interval in a normal operation mode which varies significantly in attack modes. This difference in the offset ratio and time intervals is used to detect attacks in the network.

Furthermore, Ji et al. [54] investigated the application of cumulative sum algorithm for detecting identification error produced from the clock drift of ECUs. The authors examine the CAN communication protocol to learn about the normal and abnormal behavior of the sending nodes on the CAN bus then used cumulative sum algorithm to detect attacks through the agency of characteristics of clock drift of in-vehicle ECUs.

3.3.3 CAN Data Field

Boudguiga et al. [41] proposed an intrusion detection method that makes each legitimate ECU monitor the data frames on the CAN bus to detect whether another ECU is sending frames on its behalf. The authors use a hardware security module—a security processor dedicated to cryptographic computation and secure key storage—in the ECU to enforce security in the CAN bus.

Markovitz et al. [55] proposed a novel domain-aware anomaly detection system for CAN bus traffic. They discovered semantically meaningful fields through the inspection of real CAN traffic. They developed a greedy algorithm to split CAN messages into fields and classify these fields into specific types they observed. Their anomaly detection system uses classifiers to characterize the fields and build a model for the messages, based on their field types in the learning phase. In the enforcement phase, the system detects deviations from the model. They evaluated their system on simulated and real CAN traffic and achieved near zero false positives. These methods require a deeper understanding of CAN messages and reverse engineering of the messages and their data fields.

Kang and Kang [64] proposed a novel intrusion detection technique for the security of in-vehicle networks using deep neural networks (DNN) to monitor exchanged packets in ECUs and provide real time response. In their approach, DNN was initialized with probability-based feature vectors to detect anomalies in vehicular networks and this significantly improves the detection rates. They used an unsupervised Deep Belief Net to capture underlying statistical feature of CAN data and used them to classify messages as benign or anomalous. Through the use of software simulation, they checked for false positives and false negatives. They reported a 99 percent detection ratio while keeping false positives under 1 to 2 percent.

Taylor et al. [57] proposed an anomaly detector using Long Short-Term Memory recurrent neural network for CAN bus anomaly detection. In this approach, the authors trained the neural network to predict the payload of the next message. The prediction

errors are then used as signals for identifying anomalies in the message sequence.

Similarly, Wang et al. [58] proposed a distributed real-time anomaly detection system based on the hierarchical temporal learning algorithm to learn and predict CAN data sequence. The detection method monitor CAN ID without reverse engineering of the bus protocols and can detect not only the known type attacks but can also learn online continuously from CAN data stream to detect unknown attacks.

Furthermore, Martinelli et al. [59] also proposed detection method to identify attacks on CAN packets using fuzzy classification algorithms. The authors developed a fuzzy-rough nearest neighbor classification technique to classify legitimate CAN messages generated by the human driver and the injected ones.

Using message frequency, Kuwahara et al. [44] proposed a statistical anomaly detection approach based on supervised and unsupervised learning of message pattern. The authors assume that there is a likelihood of a malicious message in a sequence if the number of messages is higher than normal.

3.3.4 Detection Approaches Implemented

While we have comprehensively surveyed relevant related works on securing and detecting anomalous behavior on the CAN bus, in this subsection, we discussed related works in other domain relevant to the approaches used for our detection.

3.3.4.1 Sequential Anomaly Detection

Here, we present a brief review of recent work on anomaly-based detection techniques using sequential adaptive CUSUM. Anomaly-based intrusion detection system has been applied to traditional network-based systems to detect anomalous behaviors in the network.

Change-point detection has been applied to several systems including wireless network protocols, application, and CPS [65, 66, 67, 68]. Tang et al. [66] used the non-parametric CUSUM test to find abrupt changes in a process without any *a priori* statistical knowledge

and detected the real-time backoff misbehavior problem in IEEE 802.11 based wireless networks. Huang et al. [67] proposed the use of adaptive CUSUM algorithm for defending against false data injection attacks in smart grid networks using a Markov-chain-based analytical model. Also, Kurt et al. [69] applied the generalized cumulative sum algorithm for quickest detection of false data injection and denial of service attack in the smart grid in both centralized and distributed settings.

Banerjee and Fellouris [70] studied the problem of decentralized sequential change detection in real-time sensor monitoring systems using CUSUM to observe anomalous events that change the distribution of the observations in all sensors. Similarly, Li et al. [65] applied the CUSUM test as a collaborative quickest detection model to identify changes in distributed ad-hoc networks. Yang et al. [68] introduced the multiple CUSUM-based algorithms to address the Wiener disorder problem with post-change drift uncertainty. However, to the best of our knowledge, we presents the first application of the change-point detection approach to identify anomalous behavior in CAN bus.

3.3.4.2 Specification-Based Detection Method

A specification-based detection method relies on specifications that describe the behavior of the system components. These legitimate behaviors of the system are described by its functionalities and the constraints of other interacting components. The monitoring of the system activities involves detecting deviations from the sequence of operations outside of the specifications. Expected behavior of the system components are manually extracted and crafted as security specifications [17].

Specification-based detection has been applied to several systems including network protocols, applications, and CPS [71, 72, 73, 74, 75]. Mitchell and Chen [71, 72] proposed a behavior-rule specification-based IDS for medical CPS and unmanned aircraft systems. In their approach, they use a binary failure threshold to classify a node as normal or malicious based on the node's compliance threshold. Esquivel-Vargas et al. [75] proposed an approach to automatically deploy a specification-based IDS to monitor a building automa-

tion system. They developed an approach to generate rules that represent valid device behavior in BACnet networks that are used by the IDS to detect violations in the network traffic. This approach was implemented in a passive way and with network-wide coverage. Fauri et al. [74] proposed an approach to combine formal specification with anomaly-based monitoring to overcome the semantic gap between network anomalies and actionable alerts by leveraging the lightweight logical system specification.

3.4 Summary

In this chapter, we reviewed methods proposed for securing automotive vehicular systems from various attacks and malicious activities. We highlighted the overview of the detection approach and some of their advantages and disadvantages. We attempted to clarify and unify the concept of anomalies and intrusion detection regarding automotive security. This begins with identifying threat models for automotive security and identifying threats that affect all vehicles and not just one specific model. From a technical perspective, IDSs can work well for detecting intrusions on the CAN bus. Different implementations of anomaly detection methods can detect different types of anomalies.

Current approaches focused on the detection of message injection attack as the primary attack vector for adversary trying to subvert the normal working of vehicular functions. The link to the next step after detection is to enable prevention; an effective IDS for cyber-physical systems should have an active response to cyber attacks. We have identified ways for detecting attacks, but more research is needed on mitigating those attacks after detection.

Our proposed detection approach captures the behavior and models the timing of the CAN network messages to extract the specification of the network activities to detect intrusions. More precisely, we use the worst-case response time analysis of each message to build a set of specifications for network activities to compare with observed activities to detect intrusions. Our approach differs from the prior art because we leverage real-time

schedulability analysis of messages to automate creating a specification. The novelty of our approach is in the close coupling we create between real-time theory and intrusion detection, and in the automation of parameter extraction.

Chapter 4

Anomaly Detection Approach Using Adaptive Cumulative Sum Algorithm for Controller Area Network

In this chapter, we describe a novel anomaly-based intrusion detection approach using the Cumulative Sum (CUSUM) change-point detection algorithm to detect data injection attacks on the controller area network (CAN) bus. The hypothesis for using this approach is that anomalies in the network occur at unknown points and produce abrupt changes in the statistical features of the message stream. We evaluate CAN message properties to determine different statistical parameters that can be used in the change-detection algorithm to detect anomalous events in the vehicular network. We evaluate the performance of this approach with real CAN data collected from a sedan car. Portion of this chapter were previously published in [49].

4.1 Introduction

Detecting changes in statistical properties of a network stream have been broadly studied in different domains, and change-point detection approach has been applied [76, 77]. The key idea of this detection approach is to model CAN bus messages as a sequence of measurement over time to describe the vehicle behavior while in operation and subject to false alarm constraints, the goal is to design a stopping rule to detect changes as quickly as possible in these messages that do not conform to the normal working of the vehicle. Hence,

detecting abrupt changes in the network can be formulated and solved as a change-point detection problem. Due to external and internal events such as malicious and DOS attacks on the bus, there can be a significant change in the behavior of the messages broadcast on the CAN bus. Change-point analysis can be used to determine the point or multiple points in time where the changes occurred and their degrees with a sequential approach (average delay) while controlling the false alarm rate [78]. Therefore, we implement the adaptive CUSUM change-point detection procedure for time series analysis to model CAN bus messages.

This chapter investigates the performance of the anomaly-based sequential change-point detection using CUSUM algorithm to detect data injection attacks on the CAN bus. The change-point detection monitors and compares the features of the observed message sequence against a predetermined pattern of normal behavior of the bus to detect any significant deviation. We leverage the features of the detection algorithm to reduce the number of false positive and increase the detection accuracy. Also, we examine the performance of the algorithm with different tuning parameters and the effect of attack intensity. We evaluate the effectiveness of our approach using a real-world CAN dataset. The datasets represent different attack scenarios. The main contributions of this chapter are in threefold:

1. We develop a sliding window approach to identify sequential patterns of CAN bus logs which are used to characterize the adaptive CUSUM algorithm for detecting message injection attacks in real-time.

2. We used the model to differentiate normal and anomalous messages at varied intervals based on significant changes compared to a reasonably selected threshold value.

3. We prototype and evaluate the performance of our detection algorithm using CAN logs generated from a real vehicle.

4.1.1 Threat Model

We assume that an adversary can perform read and write operation on the CAN bus. A read operation involves eavesdropping and intercepting messages while a write operation involves forging, replaying and transmitting anomalous messages on the bus. An adversary can gain access to the CAN network through physical or remote attack surfaces to target a particular node or compromise the entire network. To evaluate the effectiveness of our detection algorithm, we investigate the following attack scenarios:

1. Data injection attack: An adversary can execute a replay or man-in-the-middle attack by sniffing the legitimate operation of the network. In this attack, a victim ECU message structure is imitated and injected into the bus at random to disrupt the normal working of the network.

2. Denial of service (DoS) attack: In a DoS attack, the CAN bus is flooded with too many messages of high priority keeping the network busy and unavailable to other nodes.

3. Fuzzy attack: In this attack scenario, the adversary injects randomly spoofed messages of different ID. As a result, nodes in the network receive lots of messages that can cause malfunction of the vehicle.

These attacks vectors are connected. An adversary starts by intercepting messages on the bus and reverse engineer them to understand their properties. The decoded messages are then injected into the network to alter the vehicle behavior. With this, an adversary can launch a DoS attack on the network that could paralyze the entire operation of the vehicular networks. The severity of the impact of potential attacks depends on the vehicular component targeted by the adversary, i.e., an attack on the vehicle brake system, steering, and accelerator will have more impact than an attack on the infotainment system.

4.2 Sequential Change-Point Detection Background

Sequential anomaly detection describes a problem of detecting patterns in an observation at which one or more abrupt changes occur in a data sequence. Analyzing sequences in data is a statistical approach and theory for processing data in which the total number of observations is not fixed but depends somehow on the observed data as they become available. Therefore, the anomaly detection problem can be modeled as a change-point detection problem [79]. This work explores the performance of anomaly detection techniques based on the sequential data model using the change-point approach to characterize the pre-change parameters with unknown post-change parameters. A change-point is an instance in time where the statistical properties of the data before and after this instance are noticeably different. It represents a transition in the state of the process that generates the data. The requirement for quality control motivates the development of change-point detection [80].

In sequential change-point detection, the goal is to detect as quickly as possible the point in time a change occurs in a statistical model of data and flag an alert signifying the change while reducing the false alarm rate. When an attack is detected at time t, the time series shows a statistical change around or at a time greater than t. For a quick response, the sequential hypothesis testing is often used when an attack occurs which saves memory and computation time. Thus, we consider the CUSUM statistics which is the basis of the change-point detection procedure. With a very light computation load, CUSUM uses the features of sequential and non-parametric tests to detect attacks in a time series data and is asymptotically optimal for a wide range of change-point detection problems when the time series are independent identically distributed (i.i.d.) with a parametric model [76].

4.3 CUSUM for CAN Anomaly Detection

CUSUM algorithm was first proposed by Page [81] and is based on hypothesis testing developed for i.i.d. random variables. The CUSUM algorithm is a sequential detection technique useful for detecting irregular patterns that cause changes in an observation. There are two different hypotheses to be considered in this approach; the difference in the statistical distribution before and after the change compared with a threshold. To detect changes in the distribution, CUSUM periodically computes two sums, the upper control limit and the lower control limit, which represents the cumulative deviation between the expected value and the observed value. This detection rule is a comparison of the cumulative sum with an adaptive threshold which is not only updated online but also keeps a total memory of the useful information contained in the past observations. An essential feature of this algorithm is in determining and defining the regular pattern of the dataset. Deviations relative to this pattern are classified as anomalies when the upper or lower control limit exceeds a certain threshold. Using a sliding window approach, CUSUM can detect small shifts in statistical parameters (e.g., mean) relative to the regular pattern. The output of the algorithm is the potential list of anomalies along with the corresponding plot of the time series and its anomalies. This detection algorithm is a cost-effective and straightforward approach that can be adapted to different vehicles. A computing module (dongle) running the CUSUM algorithm can be connected to the vehicle OBD-II port and act as a monitoring node on the CAN bus.

4.3.1 Adaptive CUSUM Algorithm

Adaptive CUSUM is proposed to solve the problem of unknown parameters that vary over time. The combination of the process of detecting change and parameter estimation is a practice considered to give better performance [77]. The idea is to estimate the parameters in a continuous form with the CUSUM test starting immediately regardless of

the estimation accuracy. Since more sample estimation could lead to more accurate estimation, the estimation process continues while performing detection. Therefore, we model the messages transmitted in the CAN bus using change-point detection procedure.

A change can be modeled using two hypotheses, θ_0 and θ_1 with thresholds 0 and h. The first hypothesis represents the statistical distribution of CAN message stream before the change while the second represent the distribution after the change. The essential steps in this algorithm are on how to decide between θ_0 and θ_1 and how to estimate the time of change efficiently from the measured sample of the message instances. These steps are called the detection and estimation steps respectively. We follow an online approach to develop the CUSUM algorithm as described in [82]. The framework of the adaptive CUSUM algorithm used to model messages transmitted in the CAN bus is described in Algorithm 4.1.

Algorithm 4.1 Adaptive CUSUM algorithm for change-point detection

Initialize:
    set the detection threshold $h > 0$
    $S_0 = G_0 = 0$
    $i = 1$
while the algorithm is not stopped **do**
    measure the current sample M_i
    $S_i = \ln\left(\frac{p(M_i,\theta_1)}{p(M_i,\theta_0)}\right)$
    $S_n = S_{i-1} + S_i$
    $G_n = \{G_{n-1} + S_n\}^+$
    if $G_n > h > 0$ **then**
        $n_d \leftarrow i$
        $\tilde{t_c} = \min_{1 \le t_c \le i} S_{t_c - 1}$
        stop and reset
    end if
    $i = i + 1$
end while

Let $M = \{M_1, M_2, \ldots, M_n\}$ be a random set of messages observed sequentially, and are independent and identically distributed on the CAN bus network. Message M represents a data frame on the CAN bus and each of the messages are released sequentially. Message M

is said to be "in-control" at first and each M_i follows a probability density function (PDF), $p(M_i, \theta)$ depending on the deterministic parameter θ. These parameters are assumed to be known mean μ and variance σ^2. This messages may contain a change that occur abruptly at time $\widetilde{t_c}$ called the out-of-control that is modeled by an instant modification to the value of θ. Therefore, $\theta = \theta_0$ before $\widetilde{t_c}$, *pre-change*, and $\theta = \theta_1$ after that, *post-change*. When a change occurs, an alarm should be signaled as soon as possible for a proper action to be taken with few false positives. In the detection step, the problem is to decide between two possible hypotheses $\mathcal{H}_0$ and $\mathcal{H}_1$ from observed messages M. The instantaneous log-likelihood ratio test is used to decide between the hypothesis i.e., test for signaling a change, which is given by:

$$S_i = \ln\left(\frac{p(M_i, \theta_1)}{p(M_i, \theta_0)}\right) \tag{4.1}$$

and the cumulative sum from 0 to n is:

$$S_n = \sum_{i=0}^{n} S_i \tag{4.2}$$

The decision function G_n and the change time estimate $\widetilde{t_c}$ are:

$$G_n = S_n - \min_{1 \leq t_c \leq n} S_{t_c - 1} \tag{4.3}$$

$$\widetilde{t_c} = \min_{1 \leq t_c \leq n} S_{t_c - 1} \tag{4.4}$$

Equations 4.2, 4.3, and 4.4 gives the direct form of the CUSUM algorithm. For the real-time detection of change, the equations are rewritten in a recursive form and are given by:

$$S_n = S_{n-1} + S_n \tag{4.5}$$

The decision function G_n compared to a positive threshold is given by:

$$G_n = \{G_{n-1} + S_n\}^+, \tag{4.6}$$

where $\{a\}^+ = sup(a, 0)$. Once the abrupt change has been detected, equation 4.4 is used to estimate the change time t_c from the measured samples $M_1, M_2, \ldots, M_n$ efficiently. Thus, the sample size M_i, the reference value k which determines the level of past memory held by the CUSUM statistics and the varying decision limits h are the tuning parameters required for operating an adaptive CUSUM.

4.3.2 Detection Approach

A significant feature of the proposed detection approach is the rate at which message instances are released and transmitted in the CAN bus. In normal operation, each message instance has a regular frequency or interval. When a message injection attack occurs on the bus, this rate or interval will change significantly as the ECUs under the attack will also be transmitting their message. Thus, the rate of messages on the bus is increased more than double the average rate. To characterize the message frequency, we use a window-based technique to extract a fixed length of overlapping windows from the attack-free dataset. The frequency of each message instance M_i in the unique window ω_i is maintained and they form the training set.

The process steps we use in detecting abrupt changes in each window follows as described in Section 4.3.1. We compute the PDF $p(M_i)$ by calculating the μ and σ^2 of each sample M_i of the dataset using the maintained frequency of message instances in ω_i. We then calculate the CUSUM, S_n as described in equation 4.5 by calculating the instantaneous log-likelihood ratio S_i given by:

$$S_i = \frac{\mu_{M_1} - \mu_{M_0}}{\sigma_M^2}\left(M_i - \frac{\mu_{M_1} + \mu_{M_0}}{2}\right). \tag{4.7}$$

When an abnormal event is detected, and there is a shift in the process mean, the algorithm terminates and signals an alarm. The algorithm considers at least five average run length (ARL) before the alarm signals for the out-of-control ARL_1 that is measured in a steady state. The steady-state ARL values are based on the delayed shifts in our chosen

Table 4.1: Overview of the dataset

Type of Attack	Total	Normal Messages	Injected Messages
DoS Attack	3,665,771	3,078,250	587,521
Fuzzy Attack	3,838,860	3,347,013	491,847
Spoofing the drive gear	4,443,142	3,845,890	597,252
Spoofing the RPM gauge	4,621,702	3,966,805	654,897

parameters.

4.4 Experimental Validation

An evaluation was conducted using real CAN dataset available for research purposes[1]. This dataset contains the normal vehicle operation and four different types of message injection attacks to disrupt the operation of the car. These attacks include DOS, fuzzy, and spoofing of the gear and vehicle RPM. The recorded datasets are logged through the OBD-II port of a real vehicle with complete knowledge of the ground truth of the normal and injected messages.

The DOS attack dataset contains attacks where the most dominant message with ID 0000 is injected every 0.3 milliseconds while the fuzzy dataset contains attacks where random message IDs are injected every 0.5 milliseconds to meddle with the vehicle operations. Other datasets are spoofing the drive gear and the rpm where their respective IDs are injected every 1 millisecond. Table 4.1 shows an overview of the overall number of messages in the dataset.

To measure the performance of our algorithm, we used the ARL function. The ARL function is the expected number of samples before alarm signals. The signal can be an actual shift in the process mean or false alarm. The ARL function takes two values with respect to θ and is given by: $ARL = E_\theta[N_d]$, where N_d is the detection time of the adaptive

[1]https://sites.google.com/a/hksecurity.net/ocslab/Datasets/CAN-intrusion-dataset

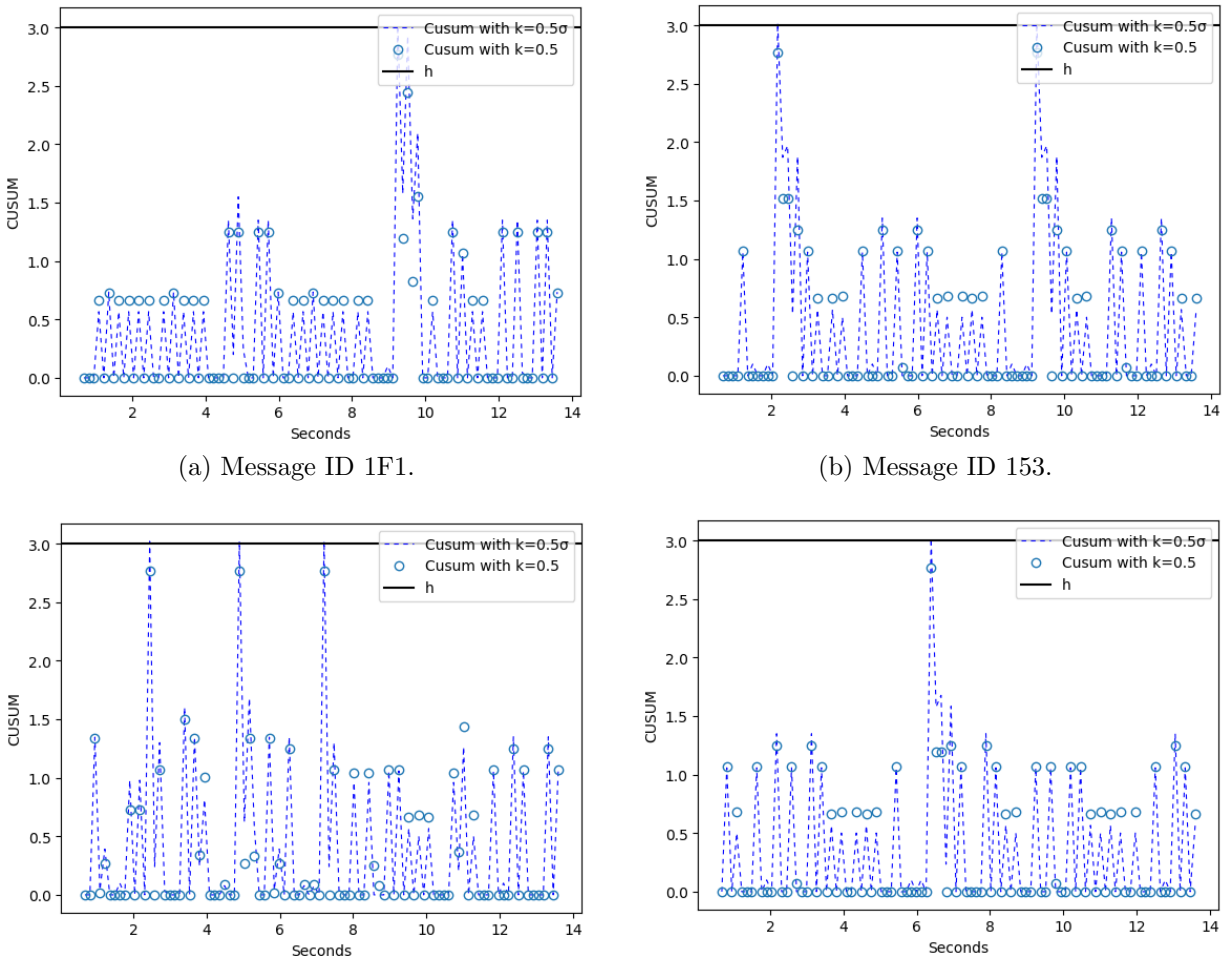
CUSUM algorithm, and the parameter θ is the assumed constant for all message instances. With respect to θ, the ARL function takes two values: $\theta = \theta_0$, the in-control ARL_0 is the expected number of samples before a false alarm, and $\theta = \theta_1$, the out-of-control ARL_1 is the expected number of samples before a shift in the mean is detected. A specific value is required for ARL_0 while we aim to minimize ARL_1 value over a range of process shifts. We also evaluate the performance by calculating the true positive rate (TPR) and the false positive rate (FPR) after measuring the true negative, true positive, false positive, and false negative.

4.4.1 Experimental Setup

The behavior of messages in the CAN bus can be learned by examining the average number of message instances and intervals between the subsequent message of the same ID. Our goal is to obtain the optimal parameters by learning these features.

We run our detection algorithm on the attack-free dataset to achieve the lowest possible false positives based on the selected interval, threshold, and window size. We learned the number of message instances in 0.335 seconds window with a usual choice of $k = 0.5$ and $h = 3$ as the CUSUM value is never greater than 3 as shown in Figure 4.1. These parameter values are chosen based on the performance of the algorithm on the attack-free dataset such that the algorithm reaches desired performance in respect to the mean time between false alarms ARL_0 and mean detection delay ARL_1.

As observed in the Figure 4.1, the CUSUM algorithm was run for 14 seconds for the attack-free dataset for different IDs with multiple instances. The graph stays in-control, and there is no presence of a change in the mean or false alarm as G_n is calculated. It is also common to set the value of k at (0.5 to 1) of the standard deviation σ. Therefore, we varied the value of k using the standard deviation while keeping other variables constant as depicted in the Figure 4.1. We realized that k at $(0.5 * \sigma)$ has a better chance of detecting small shifts early. Cumulative results of the CUSUM are presented using the receiver operating characteristics (ROC) showing how performance of the algorithm changes with



(a) Message ID 1F1.

(b) Message ID 153.

(c) Message ID 164.

(d) Message ID 220.

Figure 4.1: Plots of CUSUM algorithm with reference parameter k fixed to 0.5 and varied at $(0.5 * \sigma)$ for attack free dataset.

varying threshold for different window sizes as shown in Figure 4.2. As the window size increases, the relative variability of messages increases, thus resulting in higher TPR and FPR.

4.4.2 Experimental Results

We conducted three different experiments with the parameters obtained from the attack-free dataset to evaluate the adaptive CUSUM algorithm. As in the case of the attack-free

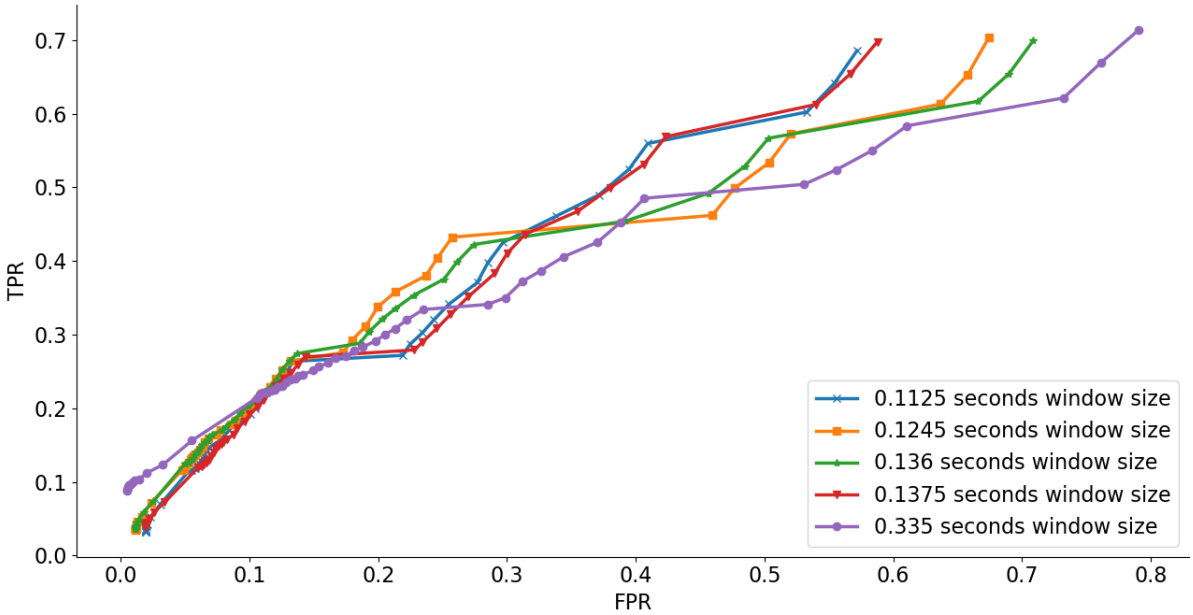
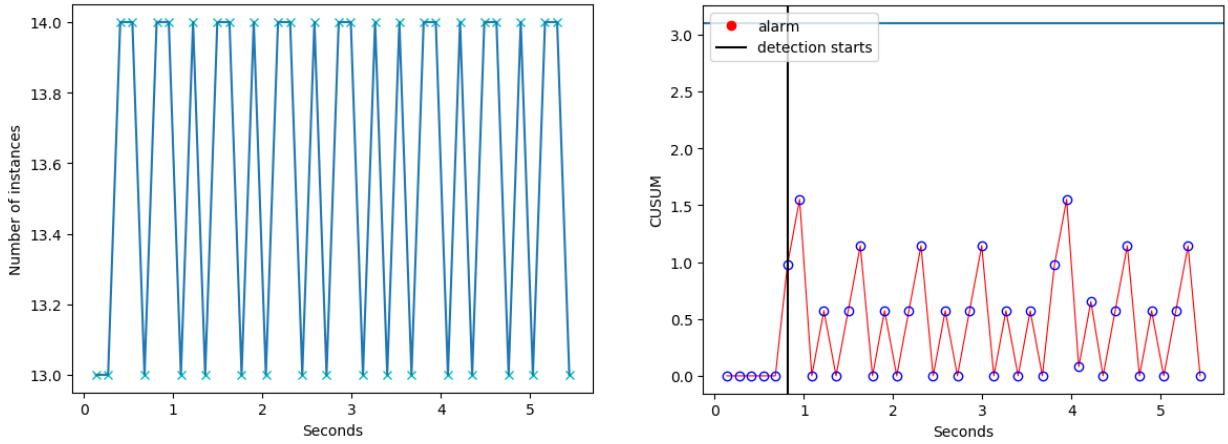


Figure 4.2: Adaptive CUSUM Algorithm performance on varying thresholds for different window sizes using RPM dataset

dataset, we assume that the first five windows of the attack datasets do not contain anomalous messages instances and they form the training set used in estimating the parameters. In our datasets, these windows do not contain any attack data. We expect the mean of the message instances to change at an unknown time. We set the detection threshold h = 3 to detect attacks very quickly with low false alarm rate. When an attack is detected, the decision function grows continuously after the change, and an alarm is signaled when it is greater than the threshold.

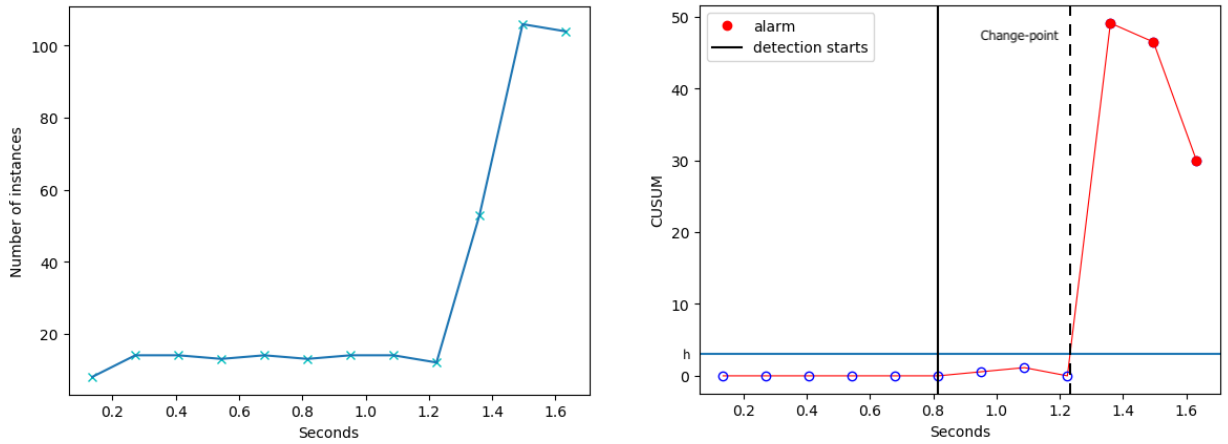
For spoofing the gear and the RPM dataset, we identified the injected IDs and plotted the message instances against time in seconds in the samples and CUSUM graphs corresponding to both IDs. Figure 4.3 shows the plots for the attack-free dataset instances and the one-sided CUSUM chart for gear and rpm IDs. Visual inspection reveals that there is no alarm signal as $G_n < h$.

The corresponding Figure 4.4 shows the same plots with the CUSUM signaling an alarm when the values of $G_n > h$. As shown in the figure, the algorithm analyzed the incoming messages to calculate the CUSUM parameters and started detection when it reached a



(a) Message distribution (b) CUSUM statistic.

Figure 4.3: Plot of message instances against the time (secs) and CUSUM algorithm for gear ID with a threshold $h = 3$ for attack free dataset.



(a) Message distribution (b) CUSUM statistic.

Figure 4.4: Plot of message instances against the time (secs) and CUSUM algorithm for gear ID with a threshold $h = 3$ for spoofing the drive gear dataset.

steady state. If $G_n \approx 1$ is greater than $h = 3$, the CUSUM algorithm alert that change has occurred and an alarm is signaled before the algorithm terminates. Similar plots were obtained with spoofing the RPM gauge dataset and the DOS attack dataset. The figures show the successful detection of the injected IDs with a very short delay. By manual analysis of both datasets, we observe that the first set of injections for the spoofing gear

dataset was around time $t = 1.2682$ while our detection algorithm signals the alarm at $t \approx 1.30$. This implies that the detection delay for the gear data injection is $n_d \approx 0.032$. Similarly, manual inspection of the RPM gauge dataset reveals the set of injections occurred at $t = 0.9667$ seconds, and our detection algorithm signaled the alarm at $t \approx 1.08$ seconds which imply that the detection delay is on the average of $n_d = 0.113$. Furthermore, we conduct similar analysis on the fuzzy and the DOS attack dataset, and their detection delays are $n_d = 0.092$ and $n_d = 0.165$ seconds respectively. The corresponding ROC curve for fuzzy attack dataset is shown in Figure 4.5.

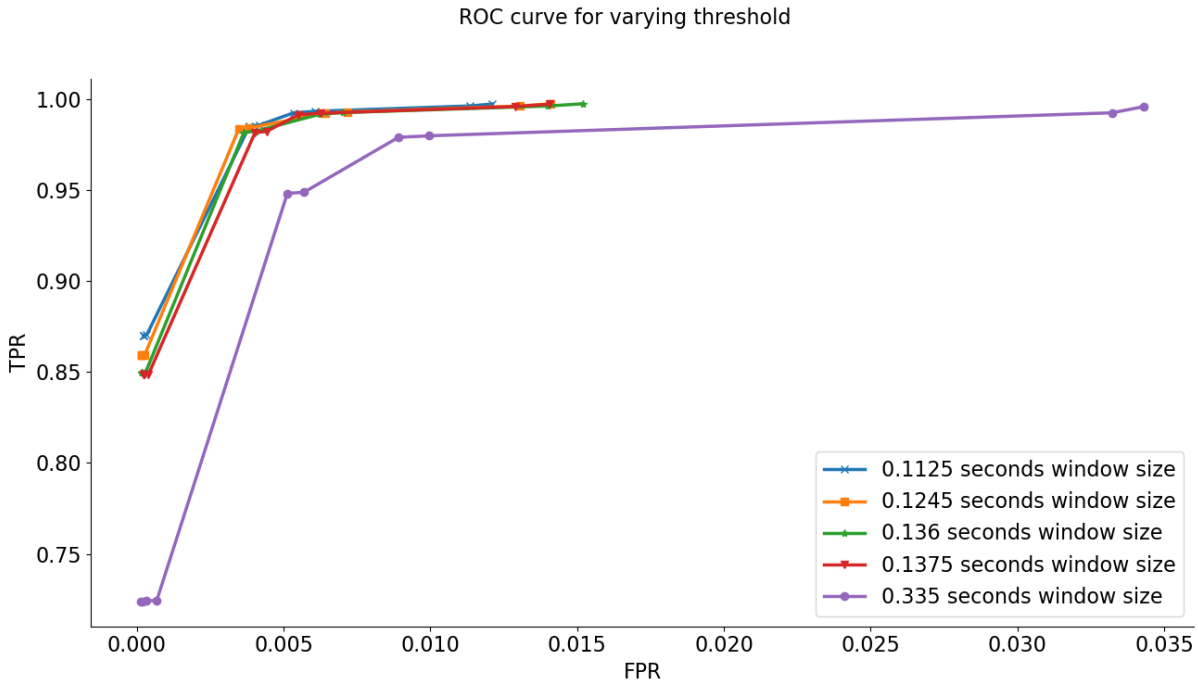


Figure 4.5: ROC curve of varying thresholds for different window sizes using fuzzy attack dataset.

To enhance the performance of our detection algorithm, we remark that varying the required parameters k, h, and large enough window size ω improves the detection accuracy. While executing the algorithm at different time intervals, we obtained different results, and the number of false alarms ranged between different interval counts. Subsequently, when the threshold value is lowered with the same window size, a degraded performance

is noticed, i.e., the false positive rate increased significantly. Similarly, when the window size is decreased using the same interval and threshold values, we get a high rate of false alarms.

4.5 Summary

In this chapter, we present an anomaly intrusion detection system to identify message injection attacks on CAN bus. The proposed approach is based on change-point detection techniques using adaptive CUSUM algorithm to detect statistical changes and intrusions in CAN bus message stream. We utilized the instance of messages in a sample window and carefully chosen tuning parameters to identify differences in the statistical properties and detect irregular patterns of the messages. Analytical results have shown that the proposed detection algorithm can efficiently detect data injection attacks with low detection delay. Through our experiment, we showed that when the required parameters are carefully selected, there is high detection accuracy with low false alarm rate.

Chapter 5

SAIDuCANT: Specification-based Automotive Intrusion Detection using Controller Area Network (CAN) Timing

This chapter introduce a novel algorithm to extract the real-time model of the controller area network (CAN) and develop a specification-based intrusion detection system (IDS) using anomaly-based supervised learning with the real-time model as input. Experimental results show that the algorithm can effectively detect data injection attacks with low false positive rates. Compared with other detection approaches using the timing features of CAN bus messages, SAIDuCANT shows a better performance in detecting malicious events in the CAN bus of the evaluated vehicles. Portions of this chapter have been publish in [83] and submitted to IEEE Transactions on Vehicle Technology.

In this chapter, a specification-based detection approach is proposed to identify exploitation of vehicle network vulnerabilities. We focus on the application of schedulability analysis of the CAN network message trace to build a specification of the valid operation sequences of the network activities. A formal specification based on the timing model of the network operations is used to express the set of valid operation sequences of the CAN network. A deviation from these sequences is tagged anomalous.

We introduce a specification-based intrusion detection system (IDS) that uses the real-time model of the CAN bus called Specification-based Automotive Intrusion Detection

using Controller Area Network Timing (SAIDuCANT) to specify intended behavior, and then detects violations of the model as signs of a compromised network. Given an instance of a message, we aim to determine if the completion time aligns with the timing model specification of the message. Our approach to this problem is to infer the parameters of the real-time model of the CAN bus during normal operation using the schedulability analysis of the network. The schedulability analysis guarantees that message deadlines will be met in the worst-case. We derive the timing model specification of a message trace and hypothesize that messages that do not fit into this timing model are anomalous. Combined with a simple protocol state machine, the timing model expresses the behavior of CAN bus from which anomalous deviations indicate an attack is in progress. Although our focus is on the CAN bus as the in-vehicle network, we expect our results would apply well to any network amenable to real-time analysis.

The contributions of this chapter are:

1. A method for extracting real-time model parameters from observations of CAN bus message behavior without prior knowledge.

2. A specification-based IDS based on real-time schedulability response time analysis of the CAN bus.

3. Two new metrics for measuring the performance of automotive intrusion detection systems.

4. Prototype and evaluation of real-time model specification-based IDS using real CAN logs generated from passenger sedan vehicles.

5.1 Response Time Analysis of CAN

Tindell et al. [12, 84] and Davis et al. [85] present a real-time model and worst case response time analysis of the CAN bus derived from fixed priority response time analysis

of CPU scheduling. We adopt their terminology and rely on some of their key results in developing our specification-based approach. For readers familiar with real-time schedulability, the key difference between task scheduling and CAN message scheduling is the use of messages in place of tasks, and each release of the message is a message instance rather than a job. A message is parameterized by its period and ID, which is a unique identifier and also the message's priority, with a lower ID having a higher priority. Every period units of time, a message releases another message instance. Each message instance has its own transmission time and queuing jitter with a data payload of up to 8 bytes specified in the header field of the frame called Data Length Code (DLC). A message is said to be schedulable if its response time is less than or equal to its deadline.

Each message is characterized by (c_i, t_i) and $DLC_i, data, e_i, d_i$ are all instance specific of the message. DLC is the data length code, $data$ is the associated data, c_i is the transmission time calculated from the data length code, e_i is the timestamp which represents the time the message finished its transmission, t_i is the message period and d_i is the relative deadline. We denote the utilization of the bus with U, i.e., $U = \sum_{i=0}^{n} \frac{c_i}{t_i}$. The priority of a message is based on its arbitration ID. The lower the arbitration ID, the higher priority of that message. We consider a set of messages $M = \{M_1, M_2, \ldots, M_n\}$ on the CAN bus network. A message M represents a data frame on the CAN bus and each release of a message is denoted by a message instance. A message instance is released at time r_i and is released periodically. Each message instance is characterized by an invocation period t_i, transmission time c_i and a relative deadline d_i. In this analysis we assume implicit deadline of periodic messages, i.e., $d_i = t_i$. A message instance is released at the beginning of its period and should complete execution by the end of its period. The first instance of a message M_i is released at time ρ_i (a phase) which we assume that $\rho_i = 0$ in our analysis. Concretely, the kth instance of M_i, denoted as $M_{i,k}$, is released at time $\rho_i + (k-1)t_i$ and should complete its transmission by time $\rho_i + k(t_i)$, where $(k = 1, 2, \ldots)$.

Given a set of messages, $\{M_i \mid i = 1, 2, \ldots, n\}$, where $M_i = (t_i, c_i)$, the hyperperiod is the least common multiple of t_i, where $(i = 1, 2, \ldots, n)$. Our notation is summarized in

Table 5.1. M denotes an ordered set of messages, and $M_i \in M$ is a message with ID i in the set. $M_{i,k}$ denotes the k^{th} instance of M_i, which has completion time $e_{i,k}$. If M_i is periodic, the time from 0 until the occurrence of the first instance i.e., $M_{i,1}$, is the message phase, denoted by ϕ_i. Concretely, the kth instance of M_i, denoted as $M_{i,k}$, is released at time $\phi_i + (k-1)t_i$ and should complete its transmission by time $\phi_i + k(t_i)$, where $(k = 1, 2, \dots)$. A message may also have a deadline, however we assume a constrained, implicit deadline (equal to the period). Thus, M_i can be characterized by a 3 tuple (ϕ_i, c_i, t_i), representing the message phase, the message worst-case execution time, and the period respectively.

Table 5.1: Table of Notations for Response Time Analysis

Variable	Definition
M	set of messages $M = (M_1, M_2, \dots, M_n)$
$M_i \in M$	the ith message
c_i	transmission time
t_i	message period
$\widetilde{t_i}$	estimated period
R_i	worst case response time
J_i	the queuing jitter
w_i	the queuing delay
B_i	the blocking time
$f_{i,min}$	lower bound on completion time relative to release
$f_{i,max}$	upper bound on completion time relative to release
$M_{i,k}$	the kth instance of message m_i
ϕ_i	phase of M_i
$e_{i,k}$	completion time of $M_{i,k}$ (CAN message time stamp)
τ_{bit}	the transmission time of a single bit
E_i	the error overhead

Davis et al. [85] determine a message worst-case response time (WCRT) by taking the maximum response time over the instances of the message in a busy period,

$$R_i = \max_{q \in [0, Q_i - 1]} (R_i(q)) \tag{5.1}$$

where Q_i is the number of instances of message M_i that become ready for transmission

before the end of the busy period, and $R_i(q)$ is the WCRT of instance q. $R_i(q)$ and Q_i are given by

$$R_i(q) = J_i + w_i(q) - ql_i + c_i \tag{5.2}$$

$$Q_i = \left\lceil \frac{l_i + J_i}{t_i} \right\rceil \tag{5.3}$$

where J_i, the queuing jitter of the frame, corresponds to the maximum time variation between the release of a message instance and queuing the message for transmission; w_i, the queuing delay under faults, corresponds to the maximum time a message can remain queued before successfully transmitting. This delay may be due to other higher and lower priority messages using the bus. c_i, is the transmission time, which corresponds to the maximum time a message can take to be transmitted. l_i is the length of the *priority level-i busy period* during which only messages with higher priority to i get transmitted. The busy period of the message ends at the earliest time that the bus becomes idle or when messages of lower priority get transmitted. l_i is found by solving the following recurrence relation with a starting value of $l_i^0 = c_i$ and ending when $l_i^{n+1} = l_i^n$:

$$l_i^{n+1} = B_i + E_i(l_i^n) + \sum_{k \leq i} \left\lceil \frac{l_i^n + J_k}{t_k} \right\rceil c_k \tag{5.4}$$

where B_i is the blocking time, which is the longest time that any lower priority message can occupy the bus while message M_i is queued, is given by

$$B_i = \max_{k>i}(c_k). \tag{5.5}$$

The worst case overhead caused by the error recovery mechanism that can occur for a given time interval is,

$$E_i(l_i) = \Big(31\tau_{bit} + \max_{k \geq i}(c_k)\Big) F(l_i) \tag{5.6}$$

where there can be 31 overhead bits for error signaling, and τ_{bit} is the transmission time of a

single bit (determined by the bus speed). $F(l_i)$ is a step function that yields the maximum number of errors on the bus for a time interval and must be a monotonic non-decreasing function. According to Broster et al. [86], the expected number of errors for the fault model in an aggressive environment is 30 faults per seconds.

The queuing delay w_i is composed of two elements: B_i, the blocking time as given in Equation 5.5, and I_i, the interference time, which is the longest time that all higher priority messages can occupy the bus before the message i is finally transmitted, given by

$$I_i = \sum_{k<i} \left\lceil \frac{w_i + J_k + \tau_{bit}}{t_k} \right\rceil c_k. \tag{5.7}$$

Therefore, the queuing delay w_i is given by:

$$w_i = B_i + I_i \tag{5.8}$$

The worst case queuing delay w_i given an error model to account for random errors on the bus is determined by calculating the delay for each of the Q_i instances and is given by the following recurrence relation:

$$w_i^{n+1}(q) = B_i + E(w_i^n + c_i) + qc_i + I_i \tag{5.9}$$

with starting value $w_i^0(q) = B_i + qc_i$ and terminating when $w_i^{n+1}(q) = w_i^n(q)$. This analysis adds a degree of pessimism as it includes the 3-bit inter-frame space in the computed queuing delay, which can be removed by subtracting $3\tau_{bit}$ from the calculated response time values.

5.2 Real-Time Specification-Based IDS Design

Messages within a CAN bus are expected to be schedulable according to some real-time model. We do not expect to know the actual model or its parameters for a given bus, but

instead we estimate parameters for the RTA-based model described earlier and use it as the specification. Bounded parameter estimates are derived from observations of the CAN bus activity by calculating upper and lower bounds for each message's period (inter-arrival time). These parameters are fed into the IDS, which monitors the network patterns to detect any deviation from the expected specification of the normal behavior. When an instance of a message is transmitted on the bus, the IDS validates message completion time by calculating an interval of possible values that bounds the completion time of a valid message. This calculation relies on the learned parameters and the RTA model as a specification, and on the history of observations of messages that have been transmitted on the bus. The bus message history contains each message's priority, transmission time, and the data payload since the last bus idle time. History is necessary to account for blocking and interference factors that delay the time between a message instance release and its transmission.

Expected regularity of messages in the CAN bus motivates a supervised learning approach to create the specification-based IDS. Supervised learning uses training and detection phases. In the training phase, the IDS collects CAN traces that represent the normal behavior of the network and extracts real-time parameters as the features that compose the specification. In the detection phase, the behavior of each message observed on the bus is checked whether or not it conforms with the specification. In our current analysis we restrict to checking only the periodic and sporadic messages, since most of the messages in the CAN bus are repeated regularly.

Before transmitting on the bus, messages go through the steps of message release, queuing for transmission, arbitration, and finally transmission. The process involving a message release includes the preparation and storage in the software queue, which is considered part of the computation time of the node sending the message. A message release time is the time instant the message is ready to be written into the priority-based transmission buffer queue. When a message is released it is written to an available transmission buffer, or if there is no available transmission buffer it is stored in the host controller (CPU)

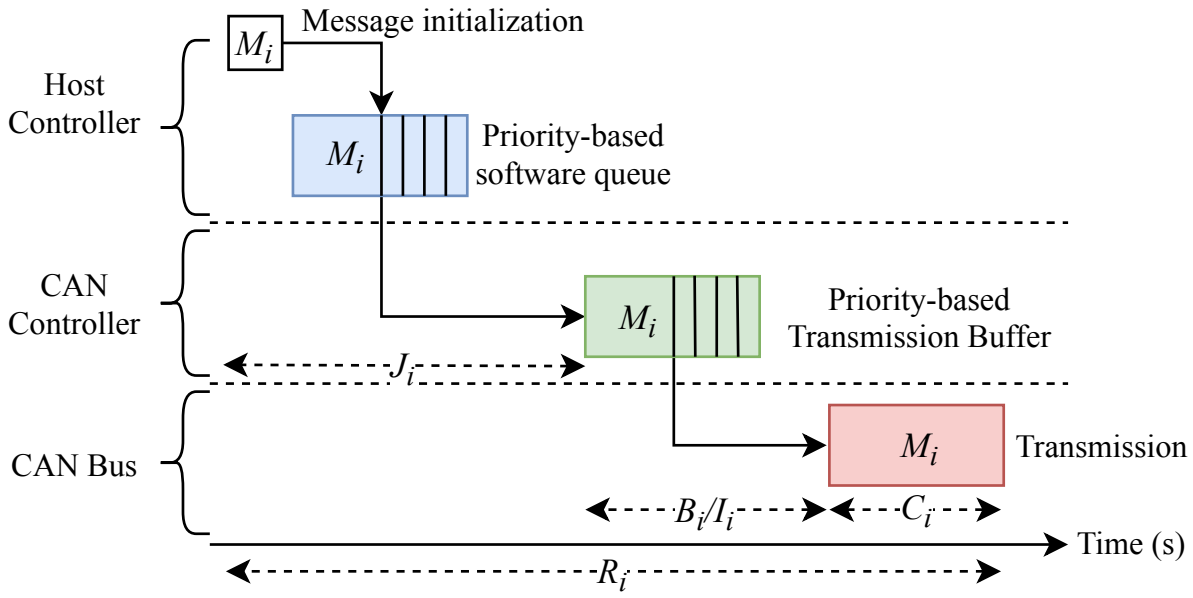


Figure 5.1: CAN Message Transmission States.

priority-based software queue until a buffer is available for writing it. Once written to the transmission buffer, the message is ready for transmission. In the transmission buffer, messages go through an arbitration process, and the message with the highest priority gets to transmit in the bus.

Using the available information of the message timestamp, we reconstruct the message release time as illustrated in Figure 5.1 by tracing back the process of release to transmission. We use this reconstruction for conjecturing the real-time parameters used to detect abnormal behavior.

The key to SAIDuCANT is that the release time is governed by only a few parameters of the real-time model, namely the period (inter-arrival) and release jitter, if any. The release time is not however observable, but the time at which the message transmits is seen on the bus and is bounded. The lower bound on the timestamp of a completed message is based on the release time plus the message's transmission time. The upper bound is based on the release time plus the message's worst case response time. The main challenge we face is extracting the parameters of the timing model to determine these bounds.

In the remainder of this section, we present the design of SAIDuCANT starting with the description of the method used to extract real-time model parameters from observations of CAN bus messages before explaining the anomaly detection algorithm using those parameters that underlies the IDS.

5.2.1 Timing Model Extraction

The exact timing model and its parameters, especially precise message periods, are difficult to obtain—they are not normally disclosed by manufacturers. Thus, we assume an RTA-based model and derive the real-time parameters for it from observations of the CAN bus messages. Algorithm 5.1 infers bounds at which the period of each message could occur by reconstructing the steps the message will go through before transmission. These bounds are derived using the analysis described in Chapter 2 applied to information available globally on the CAN. The algorithm extracts for each distinct message M_i a bounded period estimate, $f_{i,min}, f_{i,max}$, and the transmission time c_i.

Algorithm 5.1 takes as input a CAN log and message ID i. It returns the estimate $\widetilde{t_i}$ of the period by iteratively calculating upper and lower bounds on the release and inter-arrival times of successive message instances. The release time of the first message instance of a given message cannot be inferred directly, because the system state prior to the start of the log is unknown; indeed, the release of the first instance may occur prior to the start of the log. Thus, the first instance of each message is ignored. In line 4, the algorithm scans backward to find the timestamp of the previous message with lower priority or the time the bus is in an idle state. We are uncertain of the release time of $M_{i,k}$, which may have occurred at any point between the first message with lower priority that could have blocked it or an idle bus, and until the end of the intervening messages of higher priority that may have interfered with transmission. Thus, the algorithm pessimistically selects the earliest and latest possible release times of the current message, denoted L_{cur} and H_{cur}.

To construct a bounds on the period, the algorithm subtracts the latest and earliest release of the previous instance of the same message from the earliest and latest release

Algorithm 5.1 Estimate the period and release jitter of a message M_i given a partial *Log* and ID i.

```
1: function DerivePeriodicParameters(Log, i)
2:     f_{i,min}, f_{i,max} ← 0, ∞
3:     for M_{i,k} ∈ Log, k ≥ 1 do
4:         e_{l,m} ← FindPreviousTimestamp()
5:         L_cur ← e_{l,m} − c_{l,m}
6:         H_cur ← e_{i,k} − c_{i,k}
7:         if k > 2 then
8:             Δ_L ← L_cur − H_past
9:             Δ_H ← H_cur − L_past
10:            if Δ_L > f_{i,min}  and Δ_H < f_{i,max} then
11:                f_{i,min}, f_{i,max} ← Δ_L, Δ_H
12:            end if
13:        end if
14:        L_past, H_past ← L_cur, H_cur
15:    end for
16:    t̃_i = f_{i,min}
17:    J_i = f_{i,max} − f_{i,min}
18:    return (t̃_i, J_i)
19: end function
```

of the current instance, respectively, to obtain Δ_L and Δ_H. These Δ values represent the smallest and largest possible inter-arrival time between the previous and current instance of the message. The $f_{i,min}$ and $f_{i,max}$ are, eventually, the Δ_L and Δ_H that are closest to each other.

The final value of $f_{i,min}$ is taken as the estimated period $\widetilde{t}_i$, which, assuming a constant actual period and non-negative release jitter, is no greater than the actual period. The release jitter is the difference between $f_{i,max}$ and $f_{i,min}$, which describes the maximum error in the estimated $\widetilde{P}_i$ because the actual period is no greater than $f_{i,max}$.

5.2.2 Anomaly Detection

A message is considered anomalous if its completion time violates the acceptable interval defined by the specification of its real-time parameters. We obtain the response time of

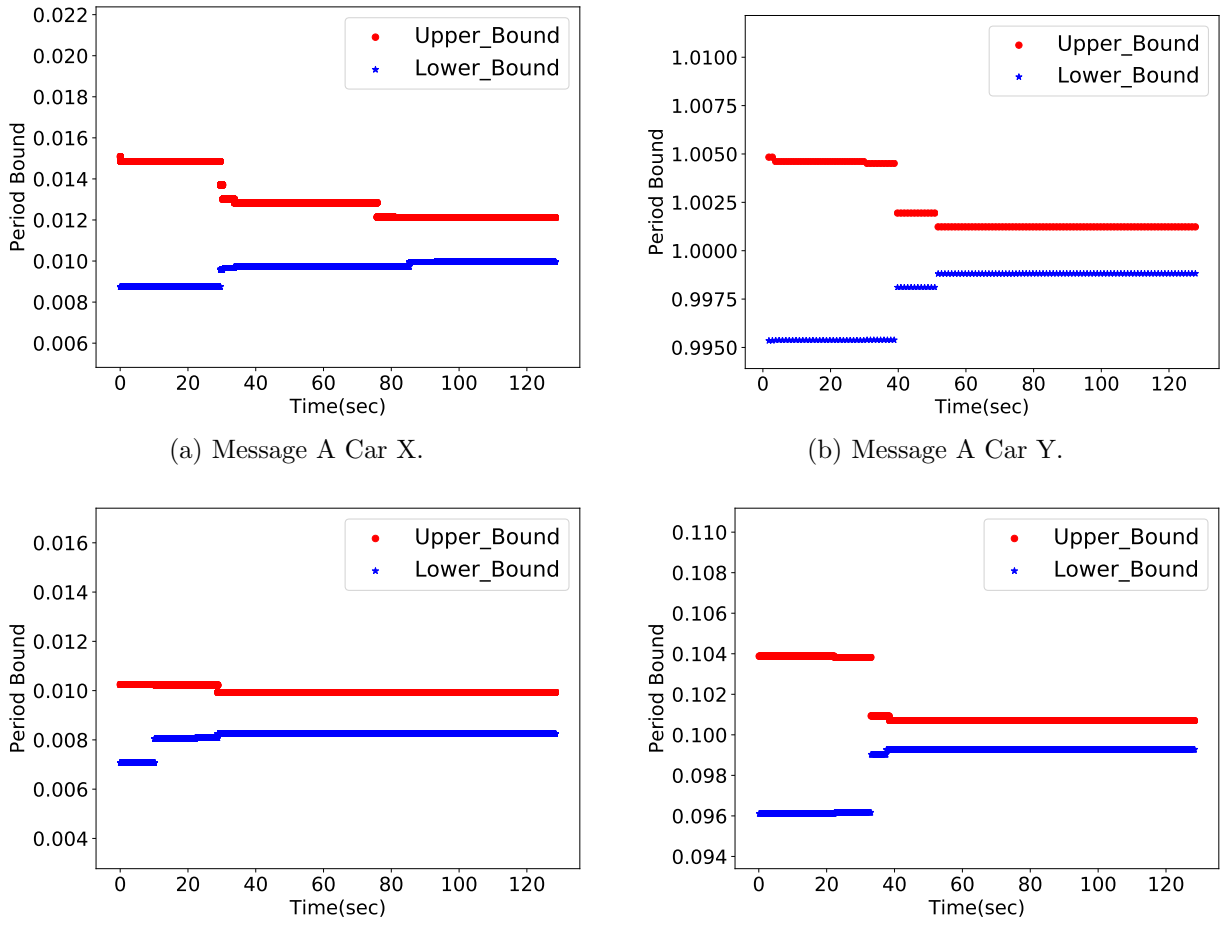
Algorithm 5.2 Anomaly detection from timing specification.

1: **function** DETECT($e_{i,k}, \widetilde{t_i}, R_i, \phi_i, k$)
2: $min_{ts} \leftarrow \phi_i + (\widetilde{t_i} * k)$
3: $max_{ts} \leftarrow min_{ts} + R_i$
4: $next_{max_{ts}} \leftarrow max_{ts} + \widetilde{t_i}$
5: **if** $e_{i,k} > next_{max_{ts}}$ **then**
6: $k \leftarrow k + 1$
7: **goto 2**
8: **end if**
9: **if** $e_{i,k} < next_{min_{ts}}$ **then**
10: **return 0** $\leftarrow$ *normal*
11: **end if**
12: **if** $min_{ts} \leq e_{i,k} \leq max_{ts}$ **then**
13: **return 0** $\leftarrow$ *normal*
14: **else**
15: **return 1** $\leftarrow$ *anomalous*
16: **end if**
17: **end function**

each message using Equation 5.1 with the estimated $\widetilde{t_i}$ and J_i determined by Algorithm 5.1. We use this response time in a supervised learning algorithm to classify messages as normal or anomalous. Algorithm 5.2 takes as input a message instance's completion time, the estimated period, response time, phase, and the instance count. Note that we estimate the phase ϕ_i as e_i minus c_i of the first instance. Algorithm 5.2 calculates the minimum timestamp that a message instance can assume by adding the phase to the instance multiplied by the period. The maximum timestamp represents the minimum timestamp plus the WCRT. The algorithm classifies the message instance as normal if its actual timestamp falls between the calculated minimum and maximum timestamps.

Figure 5.2 shows two distinct messages for Car X and Car Y with the inferred period bounds from Algorithm 5.1. The difference between the lower and upper bound represents the tightness in the minimum and maximum timestamp that a message can assume. There is a variation in this tightness as seen in Figure 5.2 which is indicative of the performance of the algorithm on the messages in a CAN bus. Car Y shows a slightly loose bound compared to Car X. We obtain the periods upper and lower bounds for each ID that

are within approximately 5ms of each other; thus an attacker cannot successfully inject a message without violating the expected period of the next authentic message because the inferred periods converges within $\pm 2ms$ of the real period. Moreover, since the real period is on the order of 10, 100, or 1000ms, an adversary with a data injection attack is constrained to inject messages within the difference between the derived lower and upper bound of the message which cannot be more than one message every 100 or 1000 messages without being detected.



(a) Message A Car X.

(b) Message A Car Y.

(c) Message B Car X.

(d) Message B Car Y.

Figure 5.2: Variation of inferred lower and upper bound of the period for consecutive instances of different message IDs.

Furthermore, because CAN bus follows a specification and calculating the bounds of

the messages in real time guarantees the schedulability of messages in the network, the primary pattern of messages transmitted in the bus is disrupted when there is an injection attack. With our approach, the tightness on the estimate of the period makes a message injection impractical.

For a periodic or sporadic message, if a message does not arrive by the maximum time of its periodicity or inter-arrival time, the message is said to be a dropped or lost message. When we have a lost message, our algorithm considers the maximum timestamp of the next instance, that is, the maximum timestamp of the current message plus the message period or inter-arrival time, to determine the instance of the following message for proper classification. A message is said to be delayed if it is received later than its expected period or inter-arrival time. Our detection algorithm does not label delayed messages anomalous if the message finishes its transmission before the minimum timestamp of its next instance.

5.2.3 Example

Consider the schedule in Figure 5.3, composed of messages M_1(0, 0.27, 0.675), M_2(0, 0.27, 0.945), and M_3(0, 0.27, 1.89) with M_1 having the highest priority (of 1) and M_3 having the least priority (of 3), and with time in milliseconds. The busy period starts at time $t = 0$ with the release of all the first message instances, $M_{1,1}, M_{2,1}, M_{3,1}$, and $M_{1,1}$ wins arbitration. Thus, $M_{1,1}$ causes interference for both $M_{2,1}$ and $M_{3,1}$. At $t = 0.675$, M_1 releases instance $M_{1,2}$ while $M_{3,1}$ is in transmission, which therefore blocks $M_{1,2}$ until $M_{3,1}$ finishes transmission. The bus is idle from $t = 1.62$ to 1.89. The embedded table shows the corresponding log for these messages with sample data, DLC, and completion time $e_{i,k}$.

To better understand how the $f_{i,min}$ and $f_{i,max}$ are calculated, consider M_1. The first instance $M_{1,1}$ is ignored. For $M_{1,2}$, scanning backward finds that the preceding message is of lower priority, which implies that the release of this message occurs during or immediately after the transmission of $M_{3,1}$. Therefore, a lower bound on the release time is given by subtracting the transmission time from the timestamp of the preceding message, i.e., $L_{cur} = e_{3,1} - c_{3,1} = 0.81 - 0.27 = 0.54$. The upper bound is always calculated directly

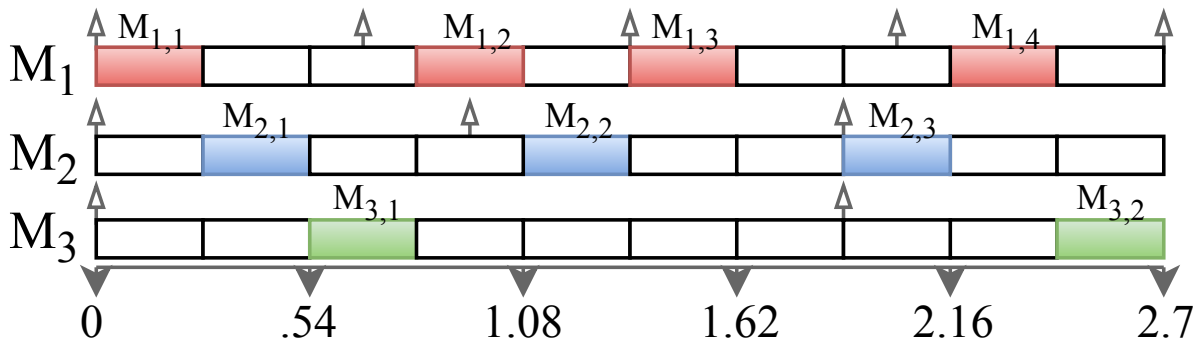


Figure 5.3: Example of periodic message behavior. (Time in ms.)

from the message instance, e.g., $H_{cur} = e_{1,2} - c_{1,2} = 1.08 - 0.27 = 0.81$. The range from $[(e_{3,1} - c_{3,1}), (e_{1,2} - c_{1,2})] = [0.54, 0.81]$ describes the maximal time interval that $M_{1,2}$ could have spent waiting for transmission. As expected, $M_{1,2}$'s actual release time $0.675 \in [0.54, 0.81]$. Because the first instance does not calculate an upper and lower bound, the second instance is not able to calculate a valid Δ_L or Δ_H, so the algorithm stops processing this instance, stores the calculated L_{cur} and H_{cur} as L_{past} and H_{past}, and moves on to $M_{1,3}$. Scanning backward from $M_{1,3}$ find that the previous message $M_{2,2}$ has lower priority, so $L_{cur} = e_{2,2} - c_{2,2} = 1.35 - 0.27 = 1.08$. Again, the upper bound is calculated as $H_{cur} = e_{1,3} - c_{1,3} = 1.62 - 0.27 = 1.35$. Now $\Delta_L = L_{cur} - H_{past} = 1.08 - 0.81 = 0.27$ and $\Delta_H = H_{cur} - L_{past} = 1.35 - 0.54 = 0.81$. These calculated bounds are used as the first estimates for the period, so $f_{1,min} = 0.27$ and $f_{1,max} = 0.81$ after processing $M_{1,3}$. The actual period of $M_1 = 0.675 \in [0.27, 0.81]$. For $M_{1,4}$, the algorithm calculates $\Delta_L = 1.89 - 1.35 = 0.54$ and $\Delta_H = 2.16 - 1.08 = 1.08$. Although the new Δ_L improves on $f_{1,min}$, the new Δ_H is worse than the $f_{1,max}$ so the bounds are not updated. As the log ends with no more instance of M_1, its estimated period and jitter are $\widetilde{t_1} = 0.27$ and $J_1 = 0.81$. Table 5.2 shows the transmission time of the example message schedule.

Table 5.2: Sample message transmission log

$M_{i,k}$	**DLC**	**Data**	$e_{i,k}$
$M_{1,1}$	8	FF FE 7E F0 86 0B 30 00	0.27
$M_{2,1}$	8	6F 9F 6F 94 0F A0 EE 0B	0.54
$M_{3,1}$	8	01 F4 02 4D 04 18 82 B6	0.81
$M_{1,2}$	8	FF FE 7E F0 86 0B 30 00	1.08
$M_{2,2}$	8	6F 9F 6F 94 0F A0 EE 0B	1.35
$M_{1,3}$	8	FF FE 7E F0 86 0B 30 00	1.62
$M_{2,3}$	8	6F 9F 6F 94 0F A0 EE 0B	2.16
$M_{1,4}$	8	FF FE 7E F0 86 0B 30 00	2.43
$M_{3,2}$	8	01 F4 02 4D 04 18 82 B6	2.70

5.3 Experimental Setup

We illustrate and evaluate SAIDuCANT using data we collected and with published datasets. We collected data from two different sedan vehicles, Car X and Car Y, which are the same make but different model and year. The vehicles are operated in a controlled setting on a dynamometer in the Cyber Security Laboratory of the National Transportation Research Center managed by Oak Ridge National Lab and CAN log data are collected through the OBD-II ports. The vehicles have a medium speed CAN bus and high speed CAN bus. Initial test data was recorded for the vehicle state comprising ignition key turn (handbrake on), acceleration, maintaining a constant speed, braking, and reverse. We performed attacks by injecting malicious messages at high frequency to override normal vehicle operations. These malicious messages were constructed by spoofing legitimate messages transmitting on the bus. We identified message IDs such as wheel speed and backup light while observing the recorded normal data to construct the attack. Messages are injected at different intervals through the OBD-II port for about 60 seconds at a frequency higher than the observed to cause a malfunction in the vehicle.

Furthermore, we evaluated the performance of SAIDuCANT using CAN data made

available for research purposes[1]. The dataset contains a standard vehicle operation and attack datasets comprising fuzzy, RPM spoofing, gear spoofing, and DoS attacks. These datasets were recorded from a real vehicle through the OBD-II port. The ground truth about the dataset is known as it contains information about regular and injected messages. For the gear and RPM spoofing attacks, the respective IDs are injected every 1 millisecond. The fuzzy attack dataset contains randomly injected messages IDs performed every 0.5 milliseconds while DoS attack dataset contains attacks where the dominant message ID 0000 is injected every 0.3 milliseconds to disrupt the vehicle functions.

We observe messages that appear just once throughout a log. These messages cannot be strictly classified as periodic, sporadic, or aperiodic because there is no other message to compare them or to extract features from them. Moreover, they appeared mostly at the beginning of the log, in which case we have classified them as messages corresponding to the initial startup of the bus. We have ignored these one-time messages in our results.

To evaluate IDS performance we use classifier accuracy. The performance is measured by collecting the number of true negatives (TN), true positives (TP), false negatives (FN), and false positives (FP) and calculating the accuracy, recall, precision, and the F1 Score. TP and FP rates usually sum to 1, while TN and TP also sum to 1.

1. True positive (TP): instances the IDS correctly labeled attack operations as anomalous.

2. True negative (TN): instances the IDS correctly labeled normal operations as non-anomalous.

3. False positive (FP): instances the IDS incorrectly labeled normal operations as anomalous.

4. False negative (FN): instances the IDS incorrectly labeled attack operations as non-anomalous.

[1]https://sites.google.com/a/hksecurity.net/ocslab/Datasets/CAN-intrusion-dataset

5. **Precision** measures the degree to which the IDS correctly labeled the attack operations as anomalous.

$$Precision = \frac{TP}{TP + FP} \tag{5.10}$$

6. **Recall** measures the rate of attack operations which the IDS labeled correctly as anomalous.

$$Recall = \frac{TP}{TP + FN} \tag{5.11}$$

7. **F1 Score** is the weighted harmonic mean of precision and recall. We use the F1 Score as an index for testing accuracy.

$$F1Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5.12}$$

8. **Accuracy** is the percentage of the correctly labeled instances divided by all the data set.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{5.13}$$

True positives represent anomalous data that was correctly classified while false negatives are cases our model incorrectly labeled as normal. Precision and recall describe how well our classification algorithm distinguishes between injected anomalous data and the standard bus operation.

We also introduce two new metrics for characterizing performance of an automotive IDS, the time to detection (TTD) and false positives before attack (FPBA), that we define as

$$TTD = e_D - e_A \tag{5.14}$$

$$FPBA = \sum_{m \in Log[0:e_A]} isFP(m) \tag{5.15}$$

where e_D and e_A denote the detection time and completion time of the first instance of an injected attack message, respectively, $Log[x : y]$ is a subsequence of messages observed on the network from time x until y, and $isFP(m)$ is a binary valued function that returns 1 if message m is a false positive, and 0 otherwise. The TTD measures the time after an attack happens before it is detected, hence it is a latency indicator of IDS performance. FPBA captures the classifier performance prior to the existence of an attack.

These metrics are introduced to measure the performance of our detection algorithm. Since we are dealing with a safety-critical system, the false alarm rate before a real attack needs to be measured since such an alarm can negatively impact system performance. The significance of early and timely detection of attacks in the vehicular network can better position proper response before safety is compromised.

5.4 Experiments

For the experiments we use message traces recorded from the high-speed bus of the test vehicles and the open-source data. We conducted four different experiments.

5.4.1 Experiment 1: All normal data

This experiment evaluates SAIDuCANT in absence of attacks. First, we recorded data for six representative datasets on Car X and five on Car Y. Each dataset is composed of data recorded for about 120 seconds of standard vehicle operations, i.e., *normal* data. One of the datasets *(training dataset)* is used to extract the timing model specifications of each message on the bus by applying Algorithm 5.1. The other datasets *(test datasets)* are used to validate the model by invoking Algorithm 5.2 for every message instance. A message instance is classified as anomalous if 1.) The message ID was not recorded during training, or 2.) Algorithm 5.2 returns *anomalous*.

For this experiment, which does not have attack data, any anomalous labels are false positives and normal labels are true negatives. Thus, the accuracy is simply the ratio of

Table 5.3: Outcome of SAIDuCANT on normal data

Cars	Messages	TN	FP	Accuracy
Car X	486091	485476	615	0.9987
	323942	323673	269	0.9992
	241157	241061	96	0.9996
	246741	246650	91	0.9996
	239107	239047	60	0.9997
Car Y	345781	345451	330	0.9990
	327604	327310	294	0.9991
	381907	381383	524	0.9986
	337575	337086	489	0.9986

normal labels to total messages. Table 5.3 shows the results of SAIDuCANT performance measured by calculating the classifier accuracy of Algorithm 5.2 over each *test dataset*. The message column indicates the total number of message instances present in each dataset. We observe false positives due to the limitations of Algorithm 5.2 that are discussed in Section 5.4.6.

We observe periodic messages with the same ID but different phases. These messages exhibit the same behavior as a regular message but share the same ID and have the same periodic behavior. This may indicate that some messages with the same ID originate at different ECUs. Our current detection algorithm is unable to classify these messages because only one phase is defined for the ID, i.e., we have assumed one message per ID. Messages from other sources are categorized as anomalous in the analysis, which constitutes about 52 percent ($\approx 52\%$) of the number of false positives. Precision and recall are not calculated for this experiment because the dataset does not contain any attack messages which implies that there is only one relevant instance or data point of interest in each dataset.

5.4.2 Experiment 2: Real Attack

This experiment considers the algorithm performance on a real attack dataset involving the vehicle backup light for Car X. We performed a message injection attack that activates the backup light every 700 microseconds. The injections are made in intervals of length 15 seconds, with 15 seconds of non-injected messages in between. Thus, the attack data contains a mix of normal and attack message instances during injection intervals $[15, 30]$ and $[45, 60]$ seconds, and normal message instances outside those intervals.

In this experiment, due to infrastructure limitations, we are not certain which logged messages are from our injection and which are from the vehicle's normal operations. Thus, we cannot calculate metrics of classifier performance for this experiment. In this experiment we injected 2,845 messages to Car X as it was being driven on the dynamometer. The attack log contains 154,564 message instances, with 3,767 of them labeled anomalous by Algorithm 5.2. Although we cannot distinguish our injected messages from authentic ones in the log, we can say that we did not observe any anomalous labels for message instances of the injected message ID outside of the injection intervals, so we have confidence that the injected messages are, mostly, correctly labeled anomalous.

5.4.3 Experiment 3: Synthetic Attacks

Here, we validate the performance of SAIDuCANT with synthetic attacks. We simulate message injection attacks on the *test datasets* by injecting a particular ID 2 to 3 times faster when an idle bus time is observed. This attack is achieved by recreating the expected message trace and injecting message IDs during the idle time. The idle time is used to ensure that the simulated attacks are accurately spaced to avoid any overlap in the message timestamp. The injected message is not altered, thus maintaining the same field properties as a normal message but with a different timestamp. The timestamps of the injected messages are set to fit within the limit of the idle time.

In this experiment, we have both attack and benign messages, and we know the ground

truth because we know which messages we injected. Thus, we present the classification FP, TP, FN, and TN. Table 5.4 shows the classifier performance of Algorithm 5.2 for the synthetically generated attack data. The message column shows the total number of messages in each dataset. The predictive value of our positive test indicates an approximation of 90 to 99 percent accuracy. An average 91 percent recall indicates that the algorithm correctly labels the injected anomalous data.

Table 5.4: Outcome of SAIDuCANT with synthetic data injection algorithm

Cars	Messages	TN	FP	FN	TP	Accuracy	Precision	Recall	F1 Score	TTD	FPBA
Car X	493042	485467	624	620	6331	0.9975	0.9103	0.9108	0.9105	0	0
	325766	323666	276	152	1672	0.9987	0.8583	0.9167	0.8865	0	0
	243698	241056	101	95	2446	0.9992	0.9603	0.9626	0.9615	0	5
	248428	246650	91	308	1379	0.9984	0.9381	0.8174	0.8736	0	7
	241739	239047	60	208	2424	0.9989	0.9758	0.9210	0.9476	0	3
Car Y	346930	345451	330	130	1019	0.9987	0.7554	0.8869	0.8159	0	0
	327604	327310	294	122	2433	0.9987	0.8922	0.9523	0.9212	0	0
	381907	381383	524	230	1933	0.9980	0.7867	0.8937	0.8368	0	2
	338869	337086	489	1	1293	0.9986	0.7256	0.9992	0.8407	0	51

5.4.4 Experiment 4: Real Attacks (open-source data)

In this experiment, we consider the algorithm performance on real open-source attack data from Hacking and Countermeasure Research Lab[2]. Table 5.5 shows the classifier performance on the datasets. For the number of false positives in spoofing the gear dataset, 99.72% is the injected ID, and 99.91% in the case of RPM dataset. We found that before the start of the message injection attack, SAIDuCANT detects no FP in both datasets. This implies that when the IDs are being injected, their periodicity changed and their transmission times become irregular. These irregularities in the periods contribute to the regular IDs missing their expected deadlines, which resulted in many false negatives. As with the false positives, the injected IDs for the RPM and gear constitute 100 percent of the false negatives. For the fuzzy attack dataset, the false positives and false negatives are

[2]https://sites.google.com/a/hksecurity.net/ocslab/Datasets/CAN-intrusion-dataset

Table 5.5: Outcome of SAIDuCANT with real attack dataset (open source)

Attacks	Messages	TN	FP	FN	TP	Accuracy	Precision	Recall	F1 Score	TTD	FPBA
Gear Spoofing	4,443,142	499,934	674,784	97,318	3,171,105	0.8262	0.8245	0.9702	0.8915	10	0
RPM Spoofing	4,621,702	534,974	798,213	119,923	3,177,591	0.8033	0.8010	0.9636	0.8748	9	0
Fuzzy	3,838,860	479,781	455,447	12,066	2,891,565	0.8782	0.8639	0.9958	0.9252	0	1
DoS Attack	3,665,771	587,521	70,475	0	3,007,774	0.9808	0.9771	1.0	0.9884	0	0

distributed across the injected IDs while for the DoS attack dataset there were zero false negatives with a minimum number of false positives ($< 0.003\%$) in the whole dataset.

5.4.5 Comparison with other detection approaches

Using the same dataset from 5.4.4, we compare SAIDuCANT with interval and frequency detection approaches. In the interval detection approach, the IDS reads the normal can frames to build a timing model for each message ID interval. The IDS checks each message ID and calculates the average time interval between subsequent messages in the attack-free dataset. The generated intervals are then used against the attack datasets. If an interval in the attack datasets is less than half of the calculated average interval for the message ID, the IDS alerts for anomalous behavior. In the frequency detection approach, the frequency of each message ID is calculated from the attack-free dataset. Frequency is the rate of messages observed in a set time interval. When the frequency of messages increases by more than twice the normal value, an anomaly is indicated. The frequency-based IDS scans datasets to calculate the frequency of message for each ID. If the frequency of messages deviates at a rate greater than two times normal, the IDS indicate an anomaly for said message ID. For this work, we used a time interval of one second.

Table 5.6 shows the performance of SAIDuCANT algorithm compared to detection algorithms using message features such as interval and frequency. The table clearly shows that our algorithm performs better in terms of the time it takes to detect attacks, and the number of false positive before attack injections started compared to other approaches. SAIDuCANT provides a significantly higher detection ratio for well-crafted attacks like DoS and fuzzy attacks compared to the other methods, and the test of accuracy (F1 Score)

Table 5.6: Comparison of the SAIDuCANT with interval and frequency detection approach

Attacks	**Detection Approach**											
	Interval				**Frequency**				**SAIDuCANT**			
	Recall	TTD	FPBA	F1 Score	Recall	TTD	FPBA	F1 Score	Recall	TTD	FPBA	F1 Score
Gear Spoofing	0.9367	2	190	0.7185	0.8739	1585	793	0.8739	0.9702	10	0	0.8915
RPM Spoofing	0.9528	0	144	0.7332	0.9618	79	160	0.9231	0.9636	9	0	0.8748
Fuzzy	0.9787	0	133	0.7708	0.8845	133	65	0.8847	0.9958	0	1	0.9252
DoS Attack	0.9998	0	139	0.8176	0.9032	204	356	0.9032	1	0	0	0.9884

for such attacks with the SAIDuCANT algorithm is over 90 percent compared to 80 percent for interval and approximately 90 percent for frequency detection approaches respectively.

5.4.6 Discussion

Due to the stochastic nature of driving, we obtained different results for each test dataset. The variability in different driving modes is one of the causes of the disparities in the results. Some of the data are recorded while the vehicles are in an accessory mode, drive to accelerate, drive to decelerate, accelerate in reverse, decelerate in reverse, maintaining a constant speed and braking operations. Also, the driver's actions and the underlying driving operations are contributing factors to the difference in results.

Most of the false positives in the result of the normal data can be reduced by manually tuning the upper bound of some of the IDs with arbitrary large periods (IDs with periodicity in the order of seconds) by 0.05ms without increasing the attacker's chance of successful data injection. Applying this tuning number to the entire set of IDs will cause overfitting which in turn increases the chance of some injected IDs being not detected.

Presently, our detection algorithm can only detect periodic and sporadic messages but not aperiodic messages and message IDs with several message instances per period. However, most of the significant information relating to the control of the safety systems in vehicles are transmitted periodically and sporadically with a single message instance from a single source ECU.

5.5 Summary

In summary, we present SAIDuCANT an approach for detecting intrusions in in-vehicle networks using a specification-based IDS. The specification is developed through observations of message timing and worst case response time analysis of the CAN bus. We developed an efficient and straightforward algorithm to estimate the real-time parameters of the RTA-based model online in a black box approach. A key strength of this approach is that the response time analysis includes the failure and error model of the CAN bus. We have evaluated SAIDuCANT experimentally on datasets generated from the high-speed CAN busses of two different car models and open-source vehicle data. The IDS can detect message injection attacks on the CAN bus with high accuracy and low false positive rates in real time. Compared to other detection approaches, SAIDuCANT shows improved performance despite its simplicity. The weighted average of precision and recall (F1 Score) shows an improvement over other approaches while reducing detection delay. Furthermore, we introduced two new metrics, TTD and FPBA to measure the performance of an IDS. Both TTD and FPBA for SAIDuCANT yields better and consistent performance across various attack scenarios as compared to other detection algorithms. SAIDuCANT can be easily implemented on most vehicle's gateway ECU with limited computing power.

Chapter 6

Reboot-Based Intrusion Prevention Approach

In this chapter, we describe the framework for reboot-based recovery approach which denotes the reactive feature of our fail-operational algorithm. The chapter highlights the steps that a compromised ECU will have to undergo to recover from anomalous behavior. The chapter introduces the fusion of the developed IDS and the recovery approach aimed at detecting and thwarting advanced threats and attacks such as multistep attacks and those using sophisticated toolset against the automotive network. We demonstrate the effectiveness of this approach on a testbed and measured the performance of the designed node in detecting injected message frames, the total recovery time to drive the compromised node to a bus-off state (more on this in section 6.1.3) and reboot.

The network systems are subjected to problems which range from signal distortion to component failures. Performance, reliability, and safety are crucial features of safety-critical applications such as the automotive systems [87]. A significant challenge for designers is to balance the requirements of safety, security, and functionality. The priority of safety is the passenger's well-being, and to ensure safety, even in the worst of conditions, certain features of the vehicle must remain operational such as airbags and collision avoidance systems. This implies that the safety of critical features and operation have priority over security. However, what follows in situations where one of these safety features is compromised or is the target of an attack? The challenge lies in keeping safety features operational while limiting them from compromising the rest of the vehicular features. To ensure the security of the vehicle some features and operations may need to be limited or be brought down

into a degraded state, while the vehicular systems maintain operational safety. This implies that the safety of critical features and operation have priority over security.

In this chapter, we present a reboot-based intrusion-tolerance approach for security covering arbitrary faults of network nodes that are under attack and also considering the effect of attack propagation in the CAN bus. Presently, our focus is on message spoofing attacks that impersonate or masquerade as a functional ECU on the bus. The reboot-based recovery is a practical recovery method for ECUs that have been compromised by a remote adversary. The goal of this approach is to prevent adversaries from propagating attack messages further into the network and control the safety-critical CAN bus operations. The high-level overview of the proposed recovery method is illustrated in Figure 6.1.

The contributions of this chapter are:

1. A practical approach for blocking and terminating malicious data injection from transmitting and propagating on the CAN bus.

2. A reliability analysis that evaluates the proposed approach and quantifies its resiliency to malicious attack.

3. An approach demonstrating how bus-off state is beneficial to recovery for nodes with remotely accessible interfaces transmitting on the bus.

6.1 Background

This section introduces the high-level description of our recovery approach, fault-tolerant technique and how faults are handled in automotive in-vehicle networks. We propose a method to detect and counter unauthorized message from transmitting on the CAN bus. In this approach, we deploy a CAN node called the *detector* that check the transmitted data and operation of the bus. The detector consists of the IDS (SAIDuCANT), CAN controller and transceiver that can invalidate or destroy unauthorized data frames through overwriting of the error frame in real-time. SAIDuCANT monitors the release of messages

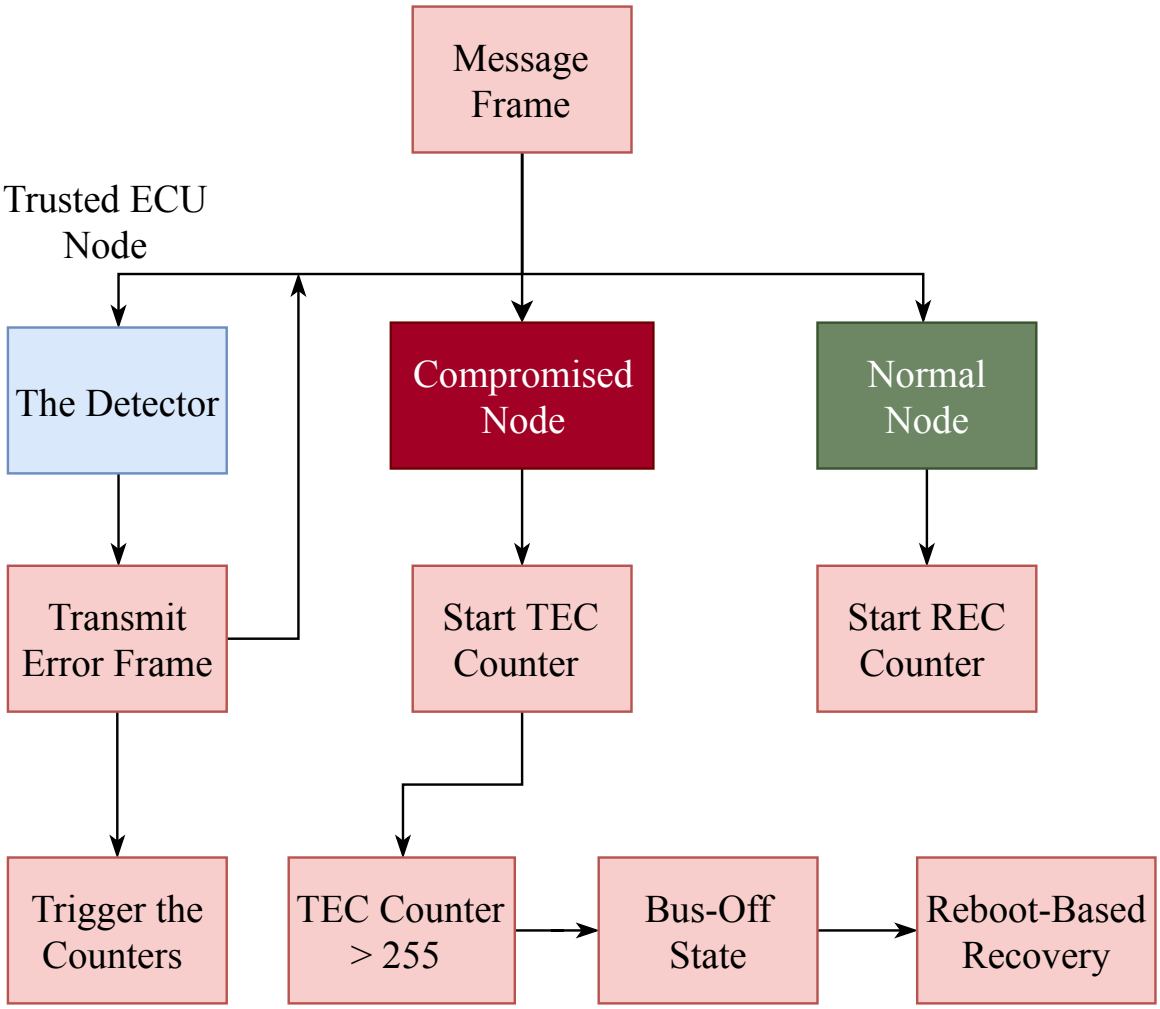


Figure 6.1: Process steps of the proposed recovery approach

as received by the CAN controller and then detects violations of the timing properties as stated in the message lookup table. When a message violates its timing specifications, and it is flagged anomalous by the detector, an error frame is transmitted to override and immediately invalidate the message before it can cause damages on the bus.

6.1.1 Fault-Tolerance

Fault-tolerance is a feature that enables a system to continue execution in the presence of a defect or fault. Systems that exhibit this feature is classified as either highly available or highly reliable system. Availability is the capability of a system to achieve its expected functions when it is required while reliability is the ability of a system to function under stated conditions for a specified period. Fault-tolerance can be implemented with software embedded in hardware or a combination of both, and it is a property that is important to many applications such as automotive, aerospace and power grid [88]. Fault tolerance is required in automotive systems because of the safety critical nature of their operations. Fault-tolerance may require more systems resources to achieve but protects against a more extensive array of faults than fault avoidance.

Typically, fault-tolerance is achieved by adding redundancy to the system, and the system is redundantly protected via replication [89]. Replication is a type of redundant fault tolerance that involves placing identical copies of a component and a switch to ensure that only one is active at a time. Fault-tolerance that is based on redundancy to detect and mask errors can reveal significant problems in the context of real-time dynamically reconfigurable systems [90].

Software fault tolerance is the ability to use software to detect and recover from faults arising or that has occurred in a software or hardware of a system. It requires dynamic configuration of tasks at runtime which can compromise the execution predictability [91]. To design a system that is fault-tolerant, the system processes and how failure occurs needs to be fully understood. A fault is a defect that causes a noticeable error in the system, and a failure occurs when the error makes the system behave in a way that is inconsistent with

the specification [88].

With the increase in electronic components and software in today's automobile, fault tolerance has become a design requirement and it is important to maintain the functionality of system component despite failures and errors. This need arises from the fact that failure of electronic components generates visible changes in the behavior of the system, unlike mechanical components that degrade gracefully in the event of failure. This failure can also result in a secondary failure leaving the system at risk of cascading failures.

A failure is said to have a cascading effect when it spreads across a series of interconnected systems. Failure of a component can result in a domino effect by propagating itself to the point of an overall systems failure. The identification and recovery from such failure are paramount in automotive safety-critical systems and implementing a fault detection, isolation, and recovery (FDIR) approach can improve the resiliency of the automotive bus system.

When an error is detected in the system, the damage must be assessed on the system state while actions are taken in keeping the error from propagating to other sections of the system to prevent further damage. Once the error is under control, error recovery techniques are applied.

6.1.2 CAN Error Recovery

CAN protocol implements error handling feature for nodes transmitting on the bus in order to monitor the health of the bus. This error handling feature is essential for fault-tolerance, which is vital for maintaining the functionality of system components despite failures and errors. This need arises from the fact that the failure of electronic components generates visible changes in the behavior of the system, unlike mechanical components that degrade gracefully in the event of failure. This failure can also result in a secondary failure leaving the system at risk of a cascading failure. In the CAN protocol, this feature allows nodes on the network to exercise actions like raising error flags, and retransmitting or discarding frames when an error is detected. The CAN protocol defines the following

error types for error handling [92]:

1. **Bit error:** Nodes transmitting frames on the bus also monitor the bus and compare the bit level to be transmitted with that monitored on the bus. A bit error occurs if these bits are different except during message arbitration.

2. **Stuff error:** A stuff error occurs when six consecutive equal bits is observed in the data field which should have been coded by the method of bit stuffing.

3. **Cyclic Redundancy Check (CRC) error:** A CRC error is raised when the CRC received for a message frame is different from the one calculated by the receiver for the frame.

4. **Form error:** A form error occurs when a fixed-form bit field of a message frame contains illegal bits.

5. **Acknowledgment (ACK) error:** ACK error is raised when a transmitting node receives no dominant bit issued by a receiver in the ACK slot.

When a node detects an error on the bus, it flags the corrupted message and transmits an error flag. If the node is in error active state, an active error flag is transmitted, otherwise a passive error flag is transmitted. The transmission of such a message will abort and attempt to retransmit at the next bus idle time.

Each node implements the two error counters transmit error counter (TEC) and receive error counter (REC). These counters start at zero, and they increment when an error is observed or decrement whenever the controller successfully transmits or receives a message according to the predefined rules specified by the CAN protocol. When an error is detected at the sending node, a sending node TEC is increased by 8, and the other nodes' RECs are increased by 1. When an error is detected at a receiving node, the receiver node REC is increased by 8. For a successfully transmitted message, the TEC and RECs of both the sending and the receiving nodes are decreased by 1, respectively.

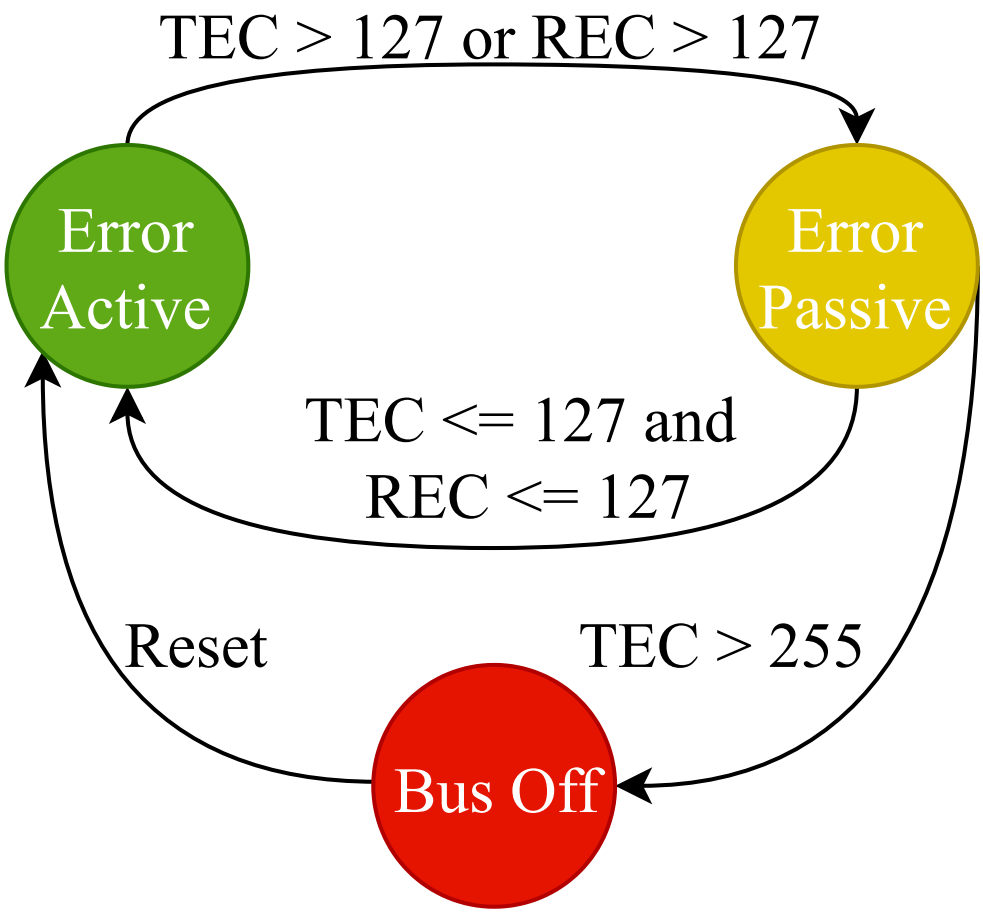


Figure 6.2: Flowchart of CAN Bus error Counter

The values of the TEC and the REC affect the error handling of the CAN bus as nodes change their error status. The transitions between the error states are shown in Figure 6.2. In error active state, the node is said to be in a healthy state when the TEC $\leq$ 127 and REC $\leq$ 127. A node will transition into the error passive state when the $TEC > 127$ or the $REC > 127$. A node in this state can partake in the bus communication but can only transmit a passive error flag when an error is detected. The error flag in this state is changed to 6 consecutive recessive bits to avoid any impacts on the bus operation and must wait before initiating further transmission. In general, a value of an error count that is higher than 96 indicates an extremely disturbed bus.

6.1.3 CAN Bus-Off State

The bus-off state is an error state of the CAN controller set by the transmitting node when the TEC exceeds 255. In this state, the node is switched off from the bus and can not transmit or acknowledge frames compulsorily. This error state is usually a result of critical hardware or software problems. A node in a bus-off state is not allowed to influence the bus operation and can only rejoin the network by transitioning to error active when its error counters are set to zero after monitoring 128 occurrences of 11 consecutive recessive bits on the bus.

6.2 Related Work

Kurachi et al. [93] proposed a centralized authentication system for preventing malicious message transmission in CAN using the error frame. The authors used a single centralized ECU to verify the MAC of all CAN messages and send out an error frame if the MAC attached is invalid. Their method requires a modification to all ECUs to be able to share keys and generate MACs when messages are transmitted. This approach will be very costly to implement and deploy for modern vehicles that may contain over 100 ECUs. Since ECUs need software modification to execute the node authentication and key exchange, the ECUs might not have enough computational power or memory to execute such an algorithm.

Matsumoto et al. [94] proposed an approach for preventing unauthorized message transmission in CAN bus using the error frame. In their approach, each ECU detects unauthorized data transmission using its message ID by monitoring the data on the bus. The ECU transmits an error frame to override the message if it detects that the message is unauthorized before it finishes transmission.

Dagan and Wool [95] proposed the Parrot system to mitigate spoofing attacks in CAN bus. In their approach, the Parrot defense launches a counter-attack of carefully crafted collisions to damage the spoof message and drive the compromised ECU into a bus-off

state. This solution can be implemented as a software patch to each ECU.

Abbott-McCune and Shay [96] proposed an intrusion prevention system that monitors the CAN bus to detect invalid messages by matching the message start-of-frame field with the one preprogrammed in the ECUs connected to the CAN bus. When a match is detected while the connected ECU is not transmitting, the ECU identifies a replay attack and send an alert to the detector to signal a replay attack. In this approach, each segment of the network requires a device that can be implemented in the gateway to monitor the network activities and compare the message IDs transmitted to the valid ID then flag non-matching ones as anomalous.

Souma et al. [97] proposed a countermeasure to bus-off attacks in the CAN bus. The counter-attack method transmits a single burst of consecutive dominant bits to disable the adversary at once, which may cause unintended negative side effects on the CAN bus as it does not conform to CAN specifications.

The prior approaches require modification to the software stack interfacing with CAN controllers, which implies modified software and hardware for each ECU. Our approach is similar in nature, but we do not require the authentic ECU to act as part of the defense scheme. Although an authentic ECU is potentially driven to a bus-off state with the compromised node, because of the remote attack surface considered in this work and our assumption that the malicious and the victim nodes are not communicating on the same bus, the error frames that drive the attacker to bus-off will not be visible to the victim (spoofed) node.

6.3 Intrusion Prevention System Design

In this section, we present the general system model of the proposed approach as well as the assumptions that have been made in designing the system.

6.3.1 Assumptions

We present a reboot-based recovery approach covering arbitrary faults of network nodes that are under attacks and also considering the effect of fault propagation in the network. We make the following assumptions in designing our approach:

1. all CAN controllers are trustworthy, i.e., the hardware controller behaves correctly with respect to the CAN protocol.

2. safety-critical ECUs are connected to each other through the CAN bus.

3. the ECU that sends the authentic message with ID matching the spoofed message is on a separate bus from the compromised ECU sending the spoof message.

4. ECUs attached to both the CAN bus and a remotely accessible interface are configured to reboot when put in bus-off state.

Currently, it is not typical for ECUs connected to the CAN bus with remotely accessible interfaces to have reboot capability when in a bus-off state. Our proposed approach may require the use of controllers that have this particular feature built-in, and it is in the manufacturer's purview to decide whether or not to include this feature.

6.3.2 ECU Architecture and General System Model

The architecture of a typical ECU node contains a hardware CAN controller and transceiver that interfaces the ECU software to the CAN bus—the controller manages digital bits in frames, and the transceiver implements the physical bus access including bus arbitration. We add IPS logic in parallel to the CAN controller that processes commands and may generate CAN messages as depicted in Figure 6.3. This logic monitors the message frame at the controller level and contains a hardware implementation of an IDS and our proposed recovery approach, which together comprise a CAN bus IPS.

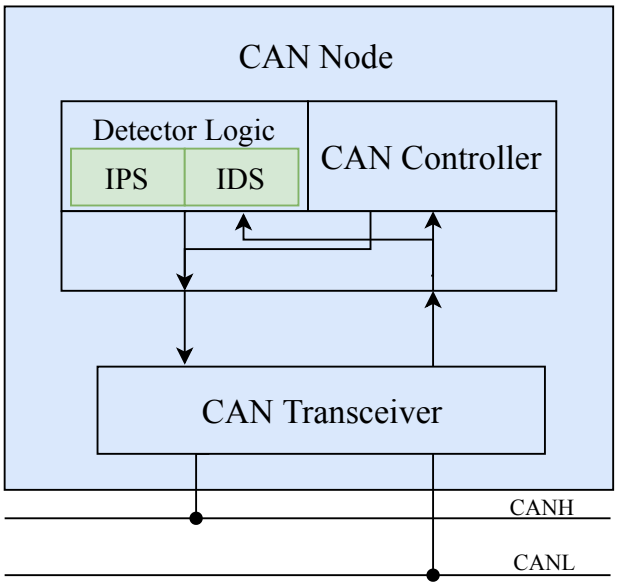


Figure 6.3: ECU Architecture

The primary functionality of the IPS logic is to monitor data transfers of the bus to protect against unauthorized access. The IPS has two operating modes: monitor and react. In the monitoring mode, the IDS component observes the bus operations while receiving message frames without meddling on the bus activities. When the IDS detects an attack, the IPS enters a reactive mode and sends an error frame to defend the system.

6.3.3 Detectors

Our proposed IPS can use any kind of IDS satisfying the requirement that it can detect an attack message before that message finishes transmission on the bus. We encapsulate such an IDS in a *detector* node, which is responsible for triggering the IPS mechanism. In this work, we investigate three IDS algorithms for implementation as a detector: message interval [38], message frequency [35], and message response time analysis [83]. In the following we briefly describe each of these.

6.3.3.1 Message Interval IDS

Song et al. [38] describe an IDS using the interval between messages as a feature. By examining the time interval between messages of the same ID, they evaluate how message injection attacks affect the individual time interval of each message ID. The authors determine that the time interval is a feature capable of detecting message injection attacks in CAN bus traffic by computing the time difference in the arrival of every new message of the same ID transmitted on the bus, and labal a message as injected if the time interval is shorter than the predefined normal.

The detector node operation for the message interval IDS is described in Figure 6.4a. The controller maintains a lookup table of the message IDs, their intervals, and the transmission time of the previous instance of the message. We assume the lookup table contains the list of messages transmitting on the bus as the detector. When a new message frame is transmitted, the IDS checks the CAN ID and computes the time interval from the arrival time of the previous message with the same ID. If the time interval of the new message frame is as expected, the previous transmission time of the ID is updated in the lookup table. Else, if the calculated interval is shorter, i.e., the message frame arrives sooner than expected, the IDS indicates the message is anomalous and transmits the error frame. By updating the previous transmission time, the controller can compute the difference between the received and previous frame transmission time with the stored interval.

6.3.3.2 Message Frequency IDS

Taylor et al. [35] describe an anomaly-based IDS using frequency of messages as a feature. By analyzing the distribution rate of messages over a specified interval, it is possible to detect anomalous messages. The authors applied a flow-based approach to estimate the changes in the data content of the CAN bus and their frequencies, then compared them to the historical value to detect anomalies.

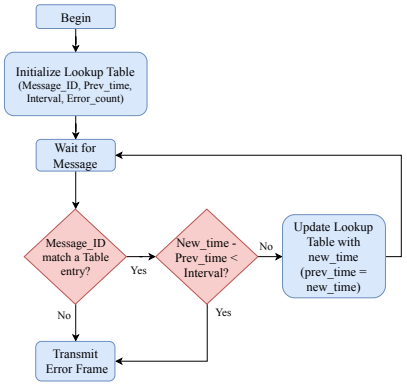
Figure 6.4b shows the system flow diagram of the detector node operation of the fre-

quency IDS. The controller maintains a lookup table of the message IDs, the window size, and their maximum message count for each window. The inputs to the monitoring node are the lookup table and the received message frame. The frequency of a message is the rate of messages transmitted over a time window. For each message ID transmitted in the same window, the detector checks if the count for the ID exceeds the maximum count. When it does, an error frame is transmitted, else it updates the message count and continues the check. The window size comprises the rates of all the message IDs transmitting on the bus for a set size. When the window for the message expires, the detector resets the window size and the message count for the IDs to capture the frequency of the messages in the new window.

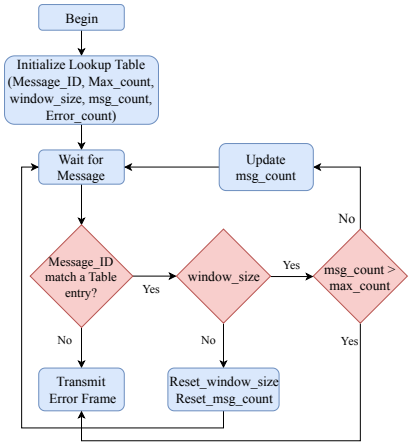
6.3.3.3 Message Response Time Analysis IDS

Olufowobi et al. [83] introduced an IDS based on estimating the real-time model parameters of a set of CAN messages and using response time analysis to derive best- and worst-case response times for each message. These response times are then used to predict the arrival window of periodic messages, and the IDS triggers an attack if messages arrive too soon. An attack is detected when a message with an unknown ID is transmitted, the release time falls outside of the acceptable range, or more than one message is received at a period or in an interval.

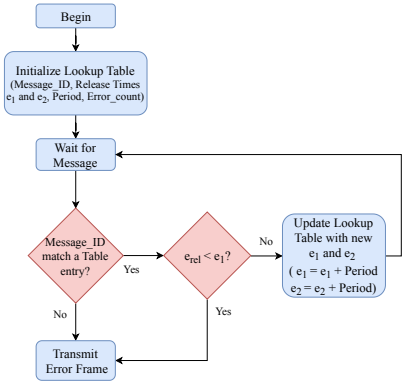
Figure 6.4c shows the system flow diagram of the detector node operation for the message response time analysis IDS. The controller maintains a lookup table of message IDs, their earliest and latest release times of the next frame, and their period or inter-arrival time. We assume that the lookup table contains the list of all messages that transmit on the same bus as the detector node. By our attack model, these are non-safety-critical nodes (message IDs) that can be accessed remotely by an adversary to gain access to the safety-critical nodes. The inputs to the monitoring node are the lookup table and the received frame observed through the controller. When a valid frame is received and is transmitted successfully, the next expected release time of the ID is updated in the lookup table. Oth-



(a) Message Interval IDS.



(b) Message Frequency IDS.



(c) Message Response Time IDS.

Figure 6.4: Flowcharts of the *detector* nodes for each IDS.

erwise, the error frame is transmitted if the message violates its periodicity or sporadicity. By incrementing the counter, the controller can compare the next received message with the sequence of the message in the updated table. The sequence helps in validating the authenticity of the message as the counter should be consistent with the received message.

6.3.4 Attack Mitigation and Recovery

When the IDS detects an attack, the IPS reaction is to immediately enqueue an error frame for transmission. This frame starts with six consecutive dominant bits, which will have the highest priority during the next bus arbitration. The nodes in receipt of the error frame will discard the message they received.

Each time a message is flagged as an attack, the IPS will transmit the error frame causing the sending node to increase its TEC by 8, and every other node on the bus will increase its REC by 1. If the attack continues until the TEC of the compromised node is higher than 255, it enters the bus-off state. This method is similar to the attack proposed by Cho and Shin [52] to drive a node to bus-off. Figure 6.5 shows the process of steps the compromised node goes through before entering the bus-off state.

ECUs in the bus-off can either go through reset or observe 128 times 11 consecutive recessive bits on the bus before they can transition into the error active state. We suggest that ECUs with remote interface capability should undergo a reboot process. A reboot process represents a recovery procedure that provides a way to restore the initial system state. The reboot process does not depend on the correct functioning of the rebooted system, is easy to implement and automate, and returns the software to its initial state, which is often its best understood and best-tested state [98]. Power-cycling is fast with minimal impact on the time it takes for the system to recover, and can provide high availability of these ECUs even when a detection algorithm is prone to false alarms or when it is unknown if the reboot process can correct the failure. Also, it should be noted that these ECUs are not the safety-critical ones, so the impact of rebooting them will not affect system safety, but perhaps will negatively affect user experience.

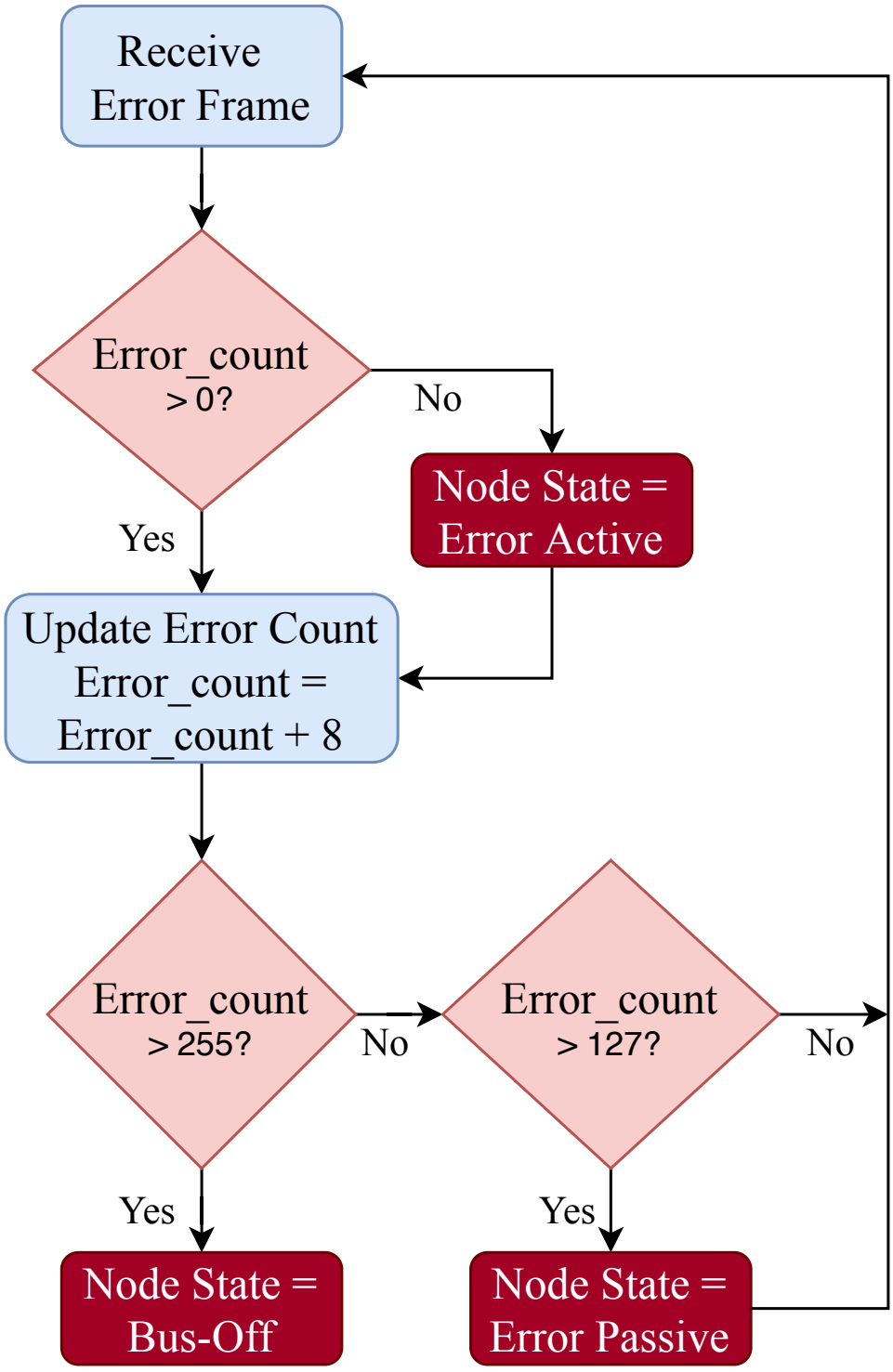


Figure 6.5: Flowchart of the behavior of the victim and malicious ECUs

6.4 IPS Implementation

To demonstrate the effectiveness of our approach, we developed a proof of concept implementation using the Xilinx LogiCORE IP CAN v5.0 as a reference [99]. This core conforms to the ISO 11898-1, CAN 2.0A, and CAN 2.0B standards. It is particularly suited for automotive applications and has user-configurable options that provide flexibility for multiple ECU applications. Options also exist to easily modify the proposed detector node from standard frame size to support a system with extended message frames. It also supports message prioritization via its high priority buffer (TX HPB) and readable error counters. As such, it allows for seamless integration of our detector functionalities. Our implementation targeted the Zynq-7000 SoC family devices which enable analytics and hardware acceleration.

The CAN nodes in our network are assumed to be connected to the CAN bus via the physical interface (CAN PHY). Each Xilinx CAN node can operate in stand-alone mode or connected to a Control block or processor using its AXI4-Lite Interface located inside the CAN Controller. The Controller has an Object Layer for message storage, filtering, and status updating. It also has a transfer layer where the CAN protocol engine resides.

The CAN protocol engine consists primarily of the bit timing logic (BTL), the bit stream processor (BSP) and the clock prescalar modules. The BTL synchronizes the operation of the CAN bus and the BSP. At the appropriate clock tick, the BTL captures a received bit or places a transmitted data bit from or to the CAN bus. It also produces a sampling clock signal for the BSP. The BSP analyzes bus traffic during transmission and reception, updates the error counters and the error state when necessary, and manages operations dealing with CAN message transmission and reception. It captures message frames from the high priority buffer (TX HPB) or from the transmission queue. It inserts the error flags (bit, stuff, form, CRC and ACK errors). The frame's bits are serialized and constructed into fields per the CAN core messaging protocols at this stage. The reverse operation is also completed by the BSP when data is received. Message frames are deconstructed and

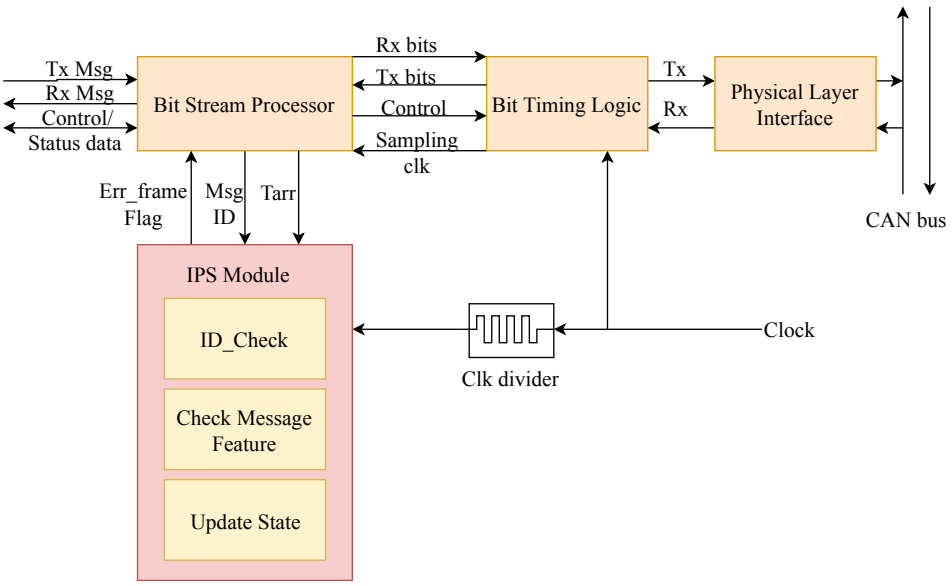


Figure 6.6: CAN controller with IPS module

stored in the receive queue (RX FIFO) where the identifier of the received message (IDR) can be accessed by the detector node. The IDR is four bytes long and is located at the head of the message frame. Its 11 most significant bits represent the message ID (Msg_ID) of the sender node. The arrival time stamp (Tarr) of each message is also recorded and submitted to the IPS inside the detector node along with the clock signal (clk), which can be derived from the main CAN engine protocol clock. In our approach, the IPS module sits at the base of the BSP in the CAN protocol engine as shown in Figure 6.6.

The implementation features three modules: Message_ID_Check, Check_Message_Feature and Update_State. After the initial system reset, the Message_ID_Check module is activated once a message is received and ready to be read from the RX FIFO. The sender node identification Msg_ID is extracted and compared against entries of a look-up table containing all known nodes in the network. If a match is not found in the table, the error_frame signal is activated to indicate that the node from which the message is received is unknown so invalidate the message. Upon a successful match, the detector examines the received message inside the Check_Message_Feature, which is where we implement each of the three IDSs used for evaluation.

	Interval IDS	Response Time IDS
Slice Registers	65	163
Slice LUTs	71	138
Occupied Slices	30	81
LUT Flip-Flops	95	228

Table 6.1: Synthesis area results

When Check_Message_Feature detects an anomaly, the message is invalidated by signaling the error frame and the error count update is triggered at the BSP level. In the case Tarr's range is valid, the Update_State module becomes active. It updates the stored state depending on the IDS in use, for example programming the next expected arrival time of a message or incrementing message counters.

6.5 Experimental Validation

Here, we describe the evaluation criteria for our IPS to illustrate its effectiveness and performance. First, we focus on the latency required for the detector node to issue a decision after a message frame has been received at the controller level. We compare the implementation results of three different IDSs which includes Interval, frequency and response time analysis based approaches. We also offer an analysis of the minimum operating speed suitable for various CAN bus speed, using different data lengths for a standard message frames.

6.5.1 Area

The area requirements for the message interval and message response time IDS are shown in Table 6.1.

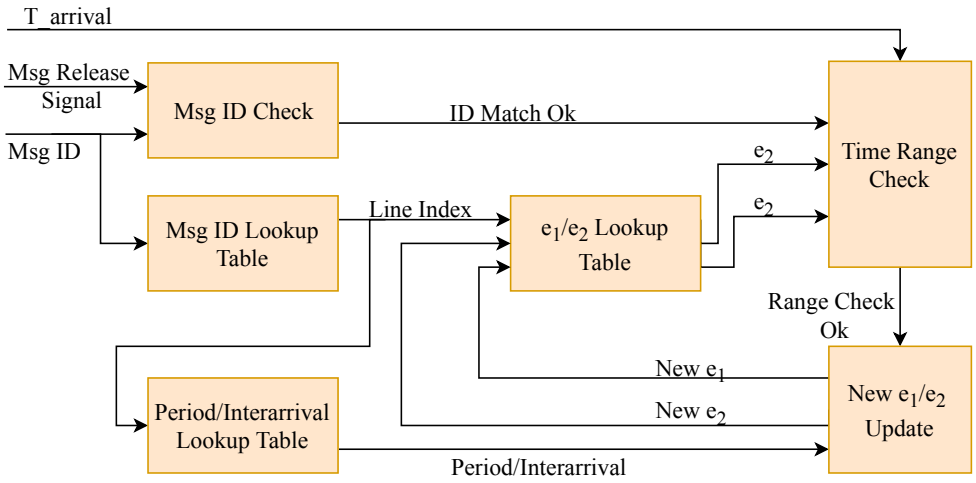


Figure 6.7: Process check of the *detector* node using the message response time analysis IDS

6.5.2 Detection Latency

This evaluation considers only the computation time for a single message at the detector level. The clock signal of the detector block is derived from the CAN controller clock.

For each message received, the detector node performs various checks which include message instance ID checks and a check based on the arrival time. When the message instance pass both checks, the detector node updates the arrival time of the next instance of the message. This process is depicted in Figure 6.7 in detail for the message response time analysis IDS. If the message fails one of the checks, the detector node transmits an error frame and increase its REC by one while the TEC of the transmitting node is increased by 8.

Regardless of the IDS in use, if a message frame is received with an ID that is not in the *detector's* lookup table, it takes the detector four clock cycles to check for validity. The detector node queues the error frame for transmission, which will occupy the bus immediately after the injected message completes, thus invalidating the message frame of that unknown ID.

For each IDS, the time needed for checking the message feature for an anomaly and

updating the state may differ. The following characterizes these costs for each IDS we consider.

1. **Message Interval IDS.** This IDS uses five clock cycles to check the difference between the previous message transmission time and the transmission time of the new message frame compared with the saved interval. The lookup table is updated with the record of the new transmission time denoting the previous time for the next arriving message.

2. **Message Frequency IDS.** This process takes 6 clock cycles for the IDS to check the sender's node ID against the list of known IDs in the network. Then the arrival time is used along with the starting time of the observation window to determine whether a new observation window should be started. The message count is updated and checked against the maximum number of messages that is expected to be received from that node within that time frame.

3. **Message Response Time Analysis IDS.** This IDS uses five clock cycles to check whether the interarrival time of the message fits in the expected arrival time range of the next message for the matching ID. The interarrival time of the next message in the lookup table is updated with the record of the next arrival time, which also included in the 5 clock cycles.

The static timing analysis of the post-route implementation shows that the detector node can operate with a minimum clock period of $3.46ns$. However, we have only been able to achieve a clock period of $3.5ns$. In other words, the maximum operating clock frequency achieved without violating any time constraints is $286MHz$, which is about 12 to 35 times faster than the IP CAN controller clock of 8 to $24MHz$. In addition, per [99], the characterization results of the IP Core runs between $160MHz$ to $220MHz$ for a Zynq board running at a speed of -3. Thus, our detector block can run fast enough to generate the necessary signals for the node.

Assuming a bus speed of $1Mbps$, the transmission time for a single bit is $1/busspeed$ and the transmission time for a message frame with 8 bytes of data length code is $1.35 \times 10^{-4} seconds$. This implies that to transmit 1-bit of the data frame will take approximately $2109ns$ to complete. Recall that per the previously described scenarios, it only takes a maximum of $8ns$ for the detector to complete its check. Thus, assuming the detector is operating at full speed ($286MHz$), it will have completed its operation and decided on the still-queued data frame (bit 19 to 63) before the completion of the 13th bit. In an attack scenario where the message ID is not in the lookup table or the timing is not as expected, the detector node will have decided on the message, and an error frame will have been queued well before the end of the transmission of the spoofed message. Furthermore, even if the detector is operating at a frequency of approximately $51KHz$, it takes a maximum of $69597ns$ to decide on the message frame in the worst case. This is still enough time to issue the error frame right before the last bit in the end-of-frame of the message frame and before the message completes. In Figure 6.8 we show the minimum frequency the detector can operate in at different bus speed, while varying the number of data bits (0 to 8 bytes) being transmitted. First, for every message frame transmitted, the detector has a minimum time frame corresponding to 33 bits transmission time to issue a decision. This minimum occurs when 0 bytes of data are included in the message frame. For each additional data byte, the decision window increases by 8-bit transition time, effectively decreasing the required operating frequency to be imposed on the detector node. Therefore, as can be seen on the chart, for different bus speeds, the minimum frequency required gradually decreases to approximately to a third when the number of data bytes increased from 0 to 8 bytes.

These minimum operating frequencies are similar to the case when the Interval IDS is used. This is expected since a maximum of 5 clock cycles is needed by the detector. However, for the Frequency IDS, the minimum frequency required is $62KHz$, which is higher than the others because the observation window must be reset when necessary. Figure 6.9 shows the behavior of the frequency IDS for different bus speeds.

Additionally, in many vehicles, the non-safety-critical ECUs are attached to the medium

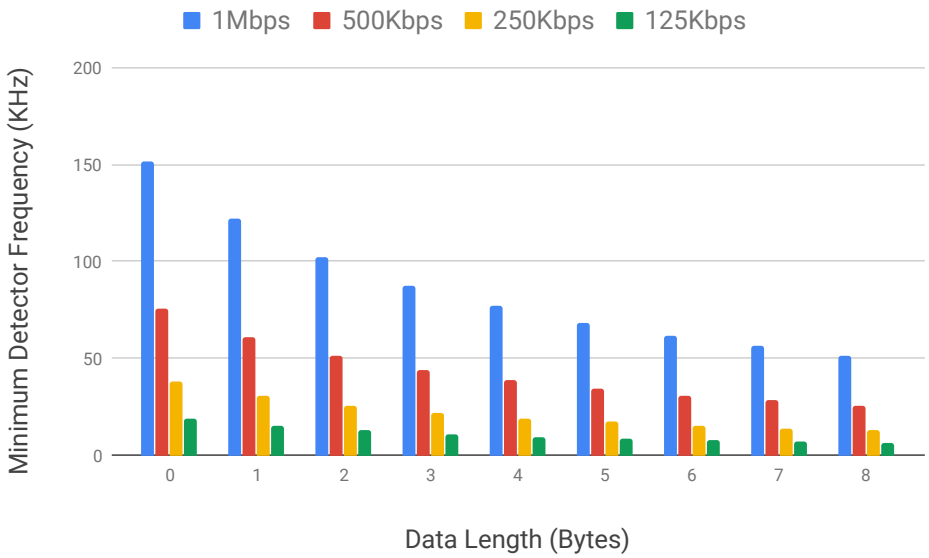


Figure 6.8: Detector Behavior for Different Bus Speed for Response Time Analysis IDS

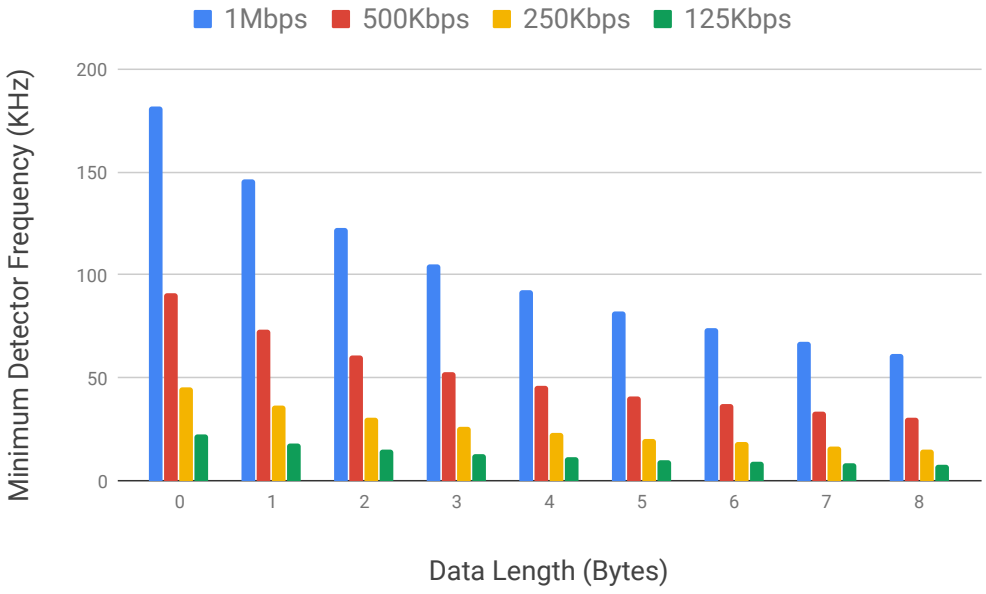


Figure 6.9: Detector Behavior for Different Bus Speed for Frequency IDS

and lower speed buses in which the transmission time is even longer. This also implies that our detector node operating in the worst case frequency of $51KHz$ have much more time to complete its operations and send an invalidation message. By this, our approach to detecting anomalous behavior in the vehicular network is practical, and the proposed algorithm is lightweight and computationally efficient.

6.5.3 Time to Error Passive and Bus Off States

We calculate the time for the compromised node to transition to error passive and bus off state. Using $1Mbps$ bus speed, the transmission time, C_m for a single bit is $\tau_{bit} = \frac{1}{busspeed}$, and the error frame has a maximum of $23bits$. Assuming the message is of the highest priority and the error frame experience no interference during its transmission when the anomalous message is detected, the total time consumed is calculated by:

$$16(C_m + 23 \times \tau_{bit}) \tag{6.1}$$

Where 16 represents the total number of anomalous messages required to transition into another state, C_m is the maximum transmission time of a CAN message including the stuff bits and the inter-frame space. C_m of a message with an 11-bit identifier containing s_m data bytes is given:

$$C_m = (55 + 10s_m)\tau_{bit} \tag{6.2}$$

Similarly, if the anomalous message is not of the highest priority transmitting in the bus, we incorporate the notion of message delay which is due to higher priority messages that may win arbitration and get transmitted before the message. The recurrence relation (equation 6.3) gives the message delay where k are set of messages with higher priority than message i and T_k is their respective periods. The starting value for $w_i^0 = 0$ and terminates when $w_i^{n+1} = w_i^n$.

$$w_i^{n+1} = \sum_{k<i} \left\lceil \frac{w_i^n + \tau_{bit}}{T_k} \right\rceil C_k \tag{6.3}$$

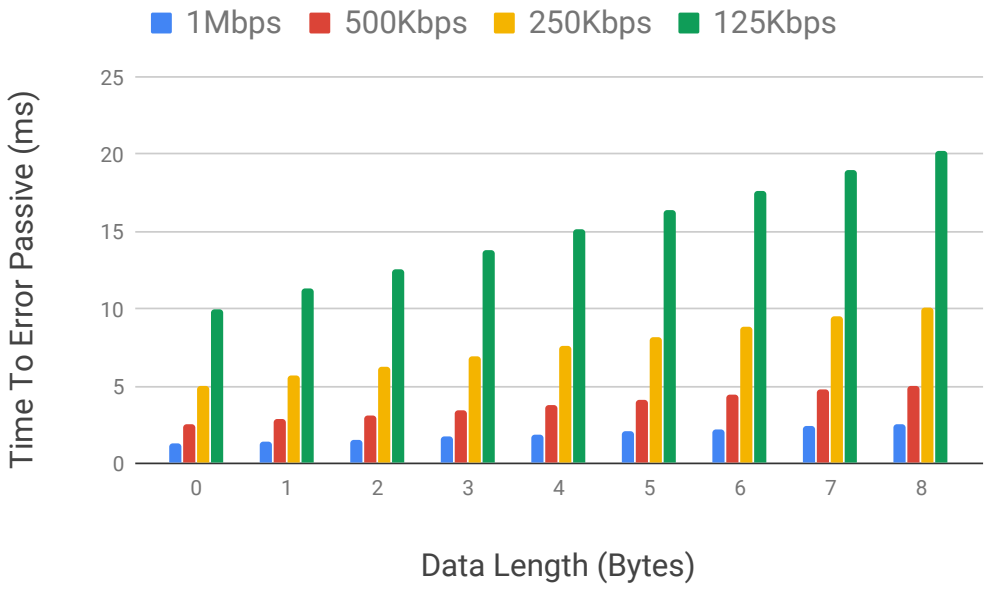


Figure 6.10: Time to error passive for different bus speed

Therefore, from these calculations, it takes on the average 2.53ms for a message with the highest priority to transition into the error passive state and approximately the same time to transition into the bus off state. Figure6.10 shows the time it takes for a message with the highest priority and different data bytes to transition into error passive state for different bus speeds.

Furthermore, we confirm this assumption on the high speed CAN bus of a vehicle with 500$kbps$ bus speed to measure how long it takes for a node to transition into an error passive and bus off state. Figure6.11 shows the logs from our sniffer, and the anomalous message (ID 82) is the 6th highest message transmitting on the bus. As shown, it takes approximately 2.685ms for this node to transition into the error passive state. Similar transition time was measured from error passive state to bus off.

Abs Time(Sec)	Rel Time (Sec)	Network	ID	B1-256
0.764554024	0.000236273	HS CAN	202	4
0.764798343	0.000244319	HS CAN	204	DC
0.765046656	0.000217319	HS CAN	43E	0
0.765737414	0.000656068	HS CAN	82	7F
0.767215133	0.001477718	HS CAN	41	0
0.76746732	0.000252187	HS CAN	42	0
0.767709434	0.000255108	HS CAN	14B	79
0.7679497	0.000258744	HS CAN	167	B2
0.768197894	0.000248194	HS CAN	185	0
0.768432081	0.000205338	HS CAN	7A	BA
0.76867038	0.000238299	HS CAN	25C	1
0.7689147	0.000244319	HS CAN	367	7
0.76916492	0.000237942	HS CAN	474	0
0.77161628	0.000197351	HS CAN	91	FF
0.771779835	0.00012058	HS CAN	CAN Rx/Tx REGS - TEC: 8 - REC: 0	0
0.771860242	8.04E-05	HS CAN	CAN Rx/Tx REGS - TEC: 16 - REC: 0	0
0.772003889	0.000143647	HS CAN	CAN Rx/Tx REGS - TEC: 24 - REC: 0	0
0.77212131	0.000117421	HS CAN	CAN Rx/Tx REGS - TEC: 32 - REC: 0	0
0.772238672	0.000117362	HS CAN	CAN Rx/Tx REGS - TEC: 40 - REC: 0	0
0.772393048	0.000154376	HS CAN	CAN Rx/Tx REGS - TEC: 48 - REC: 0	0
0.772510469	0.000117421	HS CAN	CAN Rx/Tx REGS - TEC: 56 - REC: 0	0
0.772627831	0.000117362	HS CAN	CAN Rx/Tx REGS - TEC: 64 - REC: 0	0
0.772786319	0.000158489	HS CAN	CAN Rx/Tx REGS - TEC: 72 - REC: 0	0
0.77290374	0.000117421	HS CAN	CAN Rx/Tx REGS - TEC: 80 - REC: 0	0
0.773053885	0.00012058	HS CAN	CAN Rx/Tx REGS - TEC: 88 - REC: 0	0
0.773292005	0.000112474	HS CAN	76	3E
0.773406088	0.000114083	HS CAN	Tx Error Warning - TEC: 96 - REC: 0	5
0.773667991	5.96E-06	HS CAN	77	0
0.773802042	0.000134051	HS CAN	Tx Error Warning - TEC: 104 - REC: 0	5
0.77404207	0.000135839	HS CAN	7D	0
0.774143517	0.000101447	HS CAN	Tx Error Warning - TEC: 112 - REC: 0	5
0.77430433	0.000157893	HS CAN	Tx Error Warning - TEC: 120 - REC: 0	5
0.774465144	7.05E-05	HS CAN	Tx Error Passive - TEC: 128 - REC: 0	10
0.774672031	3.91E-05	HS CAN	82	7F
0.774914801	4.16E-05	HS CAN	82 (Msg Error- Lost Arbitration)	7F
0.774936736	2.19E-05	HS CAN	Tx Error Warning - TEC: 127 - REC: 0	5
			Total time = 0.002685 seconds	

Figure 6.11: Log showing node transition into error passive state

6.5.4 Practical Consideration of FO-IDS

For the placements of the detector node, we propose that a detector node should be added to each bus that has remotely accessible ECUs communicating on it. This placement will allow for effective detection of anomalies or malicious node behavior on the bus. Figure 6.12 shows the practical placement of the detector node on the network.

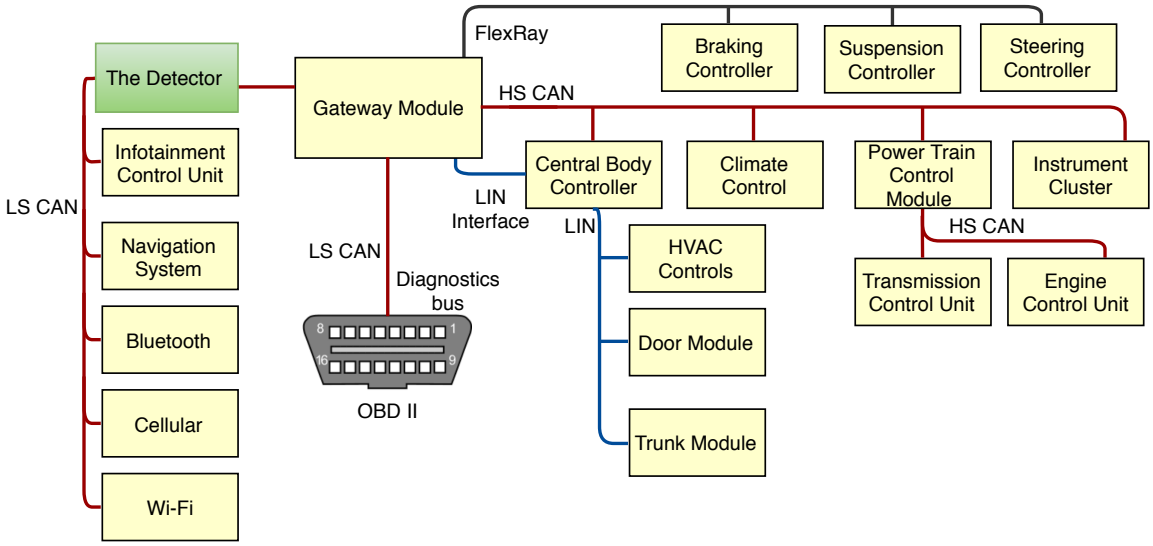


Figure 6.12: Practical placement of the *detector* node

6.6 Summary

In this chapter, we have described the fault-tolerant approach of the CAN bus and the system architecture our proposed reboot-based recovery approach. We outlined the procedure for detecting message IDs with anomalous behavior and the process of recovering for the victim node. To evaluate the effectiveness of our approach, we developed a simulation model to measure the time the compromised node takes to recover from going through all the stages of recovery. Despite this performance, the practical use of this approach depends on the speed of detection and recovery. Also, the performance of this approach depends

on other external factors and the bus configuration. Furthermore, we examine the optimal placement of the detector node on the vehicular network.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this dissertation, we have motivated the need for securing automotive in-vehicle networks against cyber attacks using intrusion detection system that integrates reboot-based fault-tolerance approach. The automotive in-vehicle networks are susceptible to various cybersecurity risk and attacks because of the heterogeneous devices embedded in the vehicles for safety, comfort, and automation. The result of these interconnected and interacting devices is the consequence of device failure which can arise from the malicious objective of an adversary against the networked automotive systems introduced to cause harm. Knowing that a failure in one of the systems cannot affect another significantly or compromise the entire operation of the system gives confidence in the use of these devices. To achieve this, we developed a fail-operational intrusion detection system using the real-time schedulability analysis of the CAN bus and a reboot-based recovery approach to detect and recover from cybersecurity threats and attacks targeting the networked component. In our approach, we model the ideal behavior of the CAN bus and use the worst-case response time analysis to develop a specification of the standard bus operation used to identify violations. Furthermore, we leverage the essential characteristics of the bus error management capabilities for invalidating spoofed messages, stopped the propagation and prevent subsequent damages while taking action to reboot the impersonated node to its initial state.

7.2 Future Work

The developed FO-IDS will require comprehensive testing in real-world attack and driving conditions that include different driving modes and attack scenarios. Some of the possible directions for future work are as follows:

1. **Evaluation of FO-IDS on a real vehicle**: Current implementation of FO-IDS run in a simulation and testbed environment. In our future work, we aim to implement and examine our approach on a real network of a vehicle to measure the performance, latency of detection and recovery while reducing false positives.

2. **Experiment in another environment and attack scenario**: We aim to continue the experiment in other environments and evaluate our approach on different attack scenarios to decrease the efficiency of malicious attacks on the vehicles. Also, we will evaluate various attack intensity to improve the overall design of the algorithm.

3. **Extension of the FO-IDS to other in-vehicle buses**: We aim to adapt our approach to other in-vehicle network buses such as LIN and MOST buses to enable a comprehensive solution in attack detection and recovery.

References

[1] SNS Research, "The connected car ecosystem: 2015 - 2030 - opportunities, challenges, strategies forecasts," Tech. Rep., 2015. [Online]. Available: http://www.researchandmarkets.com/reports/3300941/

[2] TU-Automotive, "Cyber security in the connected vehicle report 2016," Tech. Rep., 2016. [Online]. Available: http://tu-auto.com/cybersecurity-report/

[3] H. Olufowobi and G. Bloom, "Connected cars: Automotive cybersecurity and privacy for smart cities," in *Smart Cities Cybersecurity and Privacy*. Elsevier, 2019, pp. 227–240.

[4] S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *USENIX SECURITY*. USENIX, 2011.

[5] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle." BlackHat USA, 2015.

[6] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 447–462.

[7] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *DEF CON*, vol. 21, pp. 260–264, 2013.

[8] R. GmbH, *Bosch Automotive Electrics and Automotive Electronics: Systems and Components, Networking and Hybrid Drive*, ser. Bosch Professional Automotive

Information. Springer Fachmedien Wiesbaden, 2013. [Online]. Available: https://books.google.com/books?id=EJe4BAAAQBAJ

[9] G. Bloom, G. Cena, I. C. Bertolotti, T. Hu, and A. Valenzano, "Optimized event notification in can through in-frame replies and bloom filters," in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, May 2017, pp. 1–10.

[10] K. Etschberger, "Controller area network: basics, protocols, chips and applications," 2001.

[11] S. Punnekkat, H. Hansson, and C. Norstrom, "Response time analysis under errors for can," in *Real-Time Technology and Applications Symposium, 2000. RTAS 2000. Proceedings. Sixth IEEE.* IEEE, 2000.

[12] K. Tindell, A. Burns, and A. Wellings, "Calculating controller area network (can) message response times," in *Distributed Computer Control Systems 1994.* Elsevier, 1995, pp. 29–34.

[13] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," National Institute of Standards and Technology, Tech. Rep., 2012.

[14] J. Zhang and M. Zulkernine, "Anomaly based network intrusion detection with unsupervised outlier detection," in *Communications, 2006. ICC'06. IEEE International Conference on*, vol. 5. IEEE, 2006, pp. 2388–2393.

[15] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16 – 24, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804512001944

[16] R. Mitchell and R. Chen, "A survey of intrusion detection in wireless network applications," *Computer Communications*, vol. 42, pp. 1–23, 2014.

[17] C.-Y. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. Levitt, "A specification-based intrusion detection system for aodv," in *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*, ser. SASN '03. New York, NY, USA: ACM, 2003, pp. 125–134. [Online]. Available: http://doi.acm.org/10.1145/986858.986876

[18] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 266–282, First 2014.

[19] R. Bace and P. Mell, "Nist special publication on intrusion detection systems," BOOZ-ALLEN AND HAMILTON INC MCLEAN VA, Tech. Rep., 2001.

[20] K. Zaraska, "Ids active response mechanisms: Countermeasure subsytem for prelude ids," *Citeseer*, 2002.

[21] A. Shameli-Sendi, N. Ezzati-Jivan, M. Jabbarifar, and M. Dagenais, "Intrusion response systems: survey and taxonomy," *IJCSNS*, vol. 12, no. 1, pp. 1–14, 2012.

[22] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 528–533.

[23] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[24] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks–practical examples and selected short-term countermeasures," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2008, pp. 235–248.

[25] J. A. Bruton, "Securing can bus communication: An analysis of cryptographic approaches," *M. Sc., National University of Ireland, Galway*, 2014.

[26] W. A. Farag, "Cantrack: Enhancing automotive can bus security using intuitive encryption algorithms," in *Modeling, Simulation, and Applied Optimization (ICMSAO), 2017 7th International Conference on*. IEEE, 2017, pp. 1–5.

[27] B. Groza, P.-S. Murvay, A. Van Herrewege, and I. Verbauwhede, "Libra-can: A lightweight broadcast authentication protocol for controller area networks." in *CANS*. Springer, 2012, pp. 185–200.

[28] O. Hartkopp and R. M. SCHILLING, "Message authenticated can," in *Escar Conference, Berlin, Germany*, 2012.

[29] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "Canauth-a simple, backward compatible broadcast authentication protocol for can bus," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011.

[30] M. Wolf, A. Weimerskirch, and C. Paar, "Secure in-vehicle communication," *Embedded Security in Cars*, pp. 95–109, 2006.

[31] L. Yu, J. Deng, R. R. Brooks, and S. B. Yun, "Automobile ecu design to avoid data tampering," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*. ACM, 2015, p. 10.

[32] M. Müter, A. Groll, and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *Information Assurance and Security (IAS), 2010 Sixth International Conference on*. IEEE, 2010, pp. 92–98.

[33] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 1110–1115.

[34] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi, "Survey on security threats and protection mechanisms in embedded automotive networks," in *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*. IEEE, 2013, pp. 1–12.

[35] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive can bus," in *Industrial Control Systems Security (WCICSS), 2015 World Congress on*. IEEE, 2015, pp. 45–49.

[36] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PloS one*, vol. 11, no. 6, p. e0155781, 2016.

[37] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection." in *USENIX Security Symposium*, 2016, pp. 911–927.

[38] H. M. Song, H. R. Kim, and H. K. 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)*. IEEE, 2016, pp. 63–68.

[39] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Sept 2016, pp. 1–6.

[40] M. Salem, M. Crowley, and S. Fischmeister, "Anomaly detection using inter-arrival curves for real-time systems," in *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*. IEEE, 2016, pp. 97–106.

[41] A. Boudguiga, W. Klaudel, A. Boulanger, and P. Chiron, "A simple intrusion detection method for controller area network," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–7.

[42] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, and S. J. Prowell, "Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks: A data-driven approach to in-vehicle intrusion detection," in *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, ser. CISRC '17. New York, NY, USA: ACM, 2017, pp. 11:1–11:4.

[43] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "Voltageids: Low-level communication characteristics for automotive intrusion detection system," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2114–2129, 2018.

[44] T. Kuwahara, Y. Baba, H. Kashima, T. Kishikawa, J. Tsurumi, T. Haga, Y. Ujiie, T. Sasaki, and H. Matsushima, "Supervised and unsupervised intrusion detection based on can message frequencies for in-vehicle network," *Journal of Information Processing*, vol. 26, pp. 306–313, 2018.

[45] C. Young, J. Zambreno, H. Olufowobi, and G. Bloom, "Survey of automotive controller area network intrusion detection systems," *IEEE Design Test*, pp. 1–1, 2019.

[46] M. Gmiden, M. H. Gmiden, and H. Trabelsi, "An intrusion detection method for securing in-vehicle can bus," in *2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*. IEEE, 2016, pp. 176–180.

[47] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive can bus," in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, Dec 2015, pp. 45–49.

[48] C. Young, H. Olufowobi, G. Bloom, and J. Zambreno, "Automotive intrusion detection based on constant can message frequencies across vehicle driving modes," in *Proceedings of the ACM Workshop on Automotive Cybersecurity*, ser. AutoSec '19. New York, NY, USA: ACM, 2019, pp. 9–14. [Online]. Available: http://doi.acm.org/10.1145/3309171.3309179

[49] H. Olufowobi, U. Ezeobi, E. Muhati, G. Robinson, C. Young, J. Zambreno, and G. Bloom, "Anomaly detection approach using adaptive cumulative sum algorithm for controller area network," in *Proceedings of the ACM Workshop on Automotive Cybersecurity*, ser. AutoSec '19. New York, NY, USA: ACM, 2019, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/3309171.3309178

[50] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networkspractical examples and selected short-term countermeasures," *Reliability Engineering and System Safety*, vol. 96, no. 1, pp. 11 – 25, 2011, special Issue on Safecomp 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832010001602

[51] U. E. Larson, D. K. Nilsson, and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *Intelligent Vehicles Symposium, 2008 IEEE*. IEEE, 2008, pp. 220–225.

[52] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1044–1055.

[53] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, June 2011, pp. 1110–1115.

[54] H. Ji, Y. Wang, H. Qin, X. Wu, and G. Yu, "Investigating the effects of attack detection for in-vehicle networks based on clock drift of ecus," *IEEE Access*, 2018.

[55] M. Markovitz and A. Wool, "Field classification, modeling and anomaly detection in unknown can bus networks," *Vehicular Communications*, vol. 9, pp. 43–52, 2017.

[56] C. Jichici, B. Groza, and P.-S. Murvay, "Examining the use of neural networks for intrusion detection in controller area networks," in *International Conference on Security for Information Technology and Communications*. Springer, 2018, pp. 109–125.

[57] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on.* IEEE, 2016, pp. 130–139.

[58] C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu, and X. Cheng, "A distributed anomaly detection system for in-vehicle network using htm," *IEEE ACCESS*, vol. 6, pp. 9091–9098, 2018.

[59] F. Martinelli, F. Mercaldo, V. Nardone, and A. Santone, "Car hacking identification through fuzzy logic algorithms," in *Fuzzy Systems (FUZZ-IEEE), 2017 IEEE International Conference on.* IEEE, 2017, pp. 1–7.

[60] H. M. Song, H. R. Kim, and H. K. 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)*, Jan 2016, pp. 63–68.

[61] U. E. Larson, D. K. Nilsson, and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *2008 IEEE Intelligent Vehicles Symposium*, June 2008, pp. 220–225.

[62] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi, "A language-based intrusion detection approach for automotive embedded networks," in *The 21st IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2015)*, 2014.

[63] H. Lee, S. H. Jeong, and H. K. Kim, "Otids: A novel intrusion detection system for in-vehicle network by using remote frame," in *Privacy, Security and Trust (PST 2017)*, 2017.

[64] M. J. Kang and J. W. Kang, "A novel intrusion detection method using deep neural network for in-vehicle network security," in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, May 2016, pp. 1–5.

[65] C. Li, H. Li, and H. Dai, "Collaborative quickest detection in adhoc networks with delay constraint-part ii: Multi-node network," in *Information Sciences and Systems, 2008. CISS 2008. 42nd Annual Conference on.* IEEE, 2008, pp. 600–605.

[66] J. Tang, Y. Cheng, and W. Zhuang, "Real-time misbehavior detection in ieee 802.11-based wireless networks: An analytical approach," *IEEE Transactions on Mobile Computing*, vol. 13, no. 1, pp. 146–158, 2014.

[67] Y. Huang, J. Tang, Y. Cheng, H. Li, K. A. Campbell, and Z. Han, "Real-time detection of false data injection in smart grid networks: an adaptive cusum method and analysis," *IEEE Systems Journal*, vol. 10, no. 2, pp. 532–543, 2016.

[68] H. Yang, O. Hadjiliadis, and M. Ludkovski, "Quickest detection in the wiener disorder problem with post-change uncertainty," *Stochastics*, vol. 89, no. 3-4, pp. 654–685, 2017.

[69] M. N. Kurt, Y. Ylmaz, and X. Wang, "Distributed quickest detection of cyber-attacks in smart grid," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2015–2030, Aug 2018.

[70] S. Banerjee and G. Fellouris, "Decentralized sequential change detection with ordered cusums," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 36–40.

[71] R. Mitchell and R. Chen, "Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 1, pp. 16–30, 2015.

[72] R. Mitchell and I.-R. Chen, "Specification based intrusion detection for unmanned aircraft systems," in *Proceedings of the first ACM MobiHoc workshop on Airborne Networks and Communications*. ACM, 2012, pp. 31–36.

[73] P. Uppuluri and R. Sekar, "Experiences with specification-based intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001, pp. 172–189.

[74] D. Fauri, D. R. dos Santos, E. Costante, J. den Hartog, S. Etalle, and S. Tonetta, "From system specification to anomaly detection (and back)," in *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*. ACM, 2017, pp. 13–24.

[75] H. Esquivel-Vargas, M. Caselli, and A. Peter, "Automatic deployment of specification-based intrusion detection in the bacnet protocol," in *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*. ACM, 2017, pp. 25–36.

[76] H. Wang, D. Zhang, and K. G. Shin, "Change-point monitoring for the detection of dos attacks," *IEEE Transactions on dependable and secure computing*, vol. 1, no. 4, pp. 193–208, 2004.

[77] A. G. Tartakovsky, "Rapid detection of attacks in computer networks by quickest changepoint detection methods," in *Data analysis for network cyber-security*. World Scientific, 2014, pp. 33–70.

[78] R. B. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky, "A novel approach to detection of denial-of-service attacks via adaptive sequential and batch-sequential change-point detection methods," in *Proceedings of IEEE systems, man and cybernetics information assurance workshop*. Citeseer, 2001, pp. 220–226.

[79] A. G. Tartakovsky, B. L. Rozovskii, R. B. Blazek, and H. Kim, "A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-

sequential change-point detection methods," *IEEE Transactions on Signal Processing*, vol. 54, no. 9, pp. 3372–3382, 2006.

[80] O. Osanaiye, K.-K. R. Choo, and M. Dlodlo, "Change-point cloud ddos detection using packet inter-arrival time," in *Computer Science and Electronic Engineering (CEEC), 2016 8th*. IEEE, 2016, pp. 204–209.

[81] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[82] P. Granjon, "The cusum algorithm-a small review," 2013.

[83] H. Olufowobi, G. Bloom, C. Young, and J. Zambreno, "Work-in-progress: Real-time modeling for intrusion detection in automotive controller area network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 161–164.

[84] K. Tindell, H. Hanssmon, and A. J. Wellings, "Analysing real-time communications: Controller area network (can)." in *RTSS*, 1994.

[85] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[86] I. Broster, A. Burns, and G. Rodriguez-Navas, "Timing analysis of real-time communication under electromagnetic interference," *Real-Time Systems*, vol. 30, no. 1-2, pp. 55–81, 2005.

[87] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th international conference on software engineering*. ACM, 2002, pp. 547–550.

[88] J. Kienzle, "Software fault tolerance: An overview," in *International Conference on Reliable Software Technologies*. Springer, 2003, pp. 45–67.

[89] W. L. Heimerdinger and C. B. Weinstock, "A conceptual framework for system fault tolerance," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 1992.

[90] W. Gabsi and B. Zalila, "Fault tolerance for distributed real time dynamically reconfigurable systems from modeling to implementation," in *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE, 2013, pp. 98–103.

[91] W. Gabsi, B. Zalila, and J. Hugues, "A development process for the design, implementation and code generation of fault tolerant reconfigurable real time systems," *International Journal of Autonomous and Adaptive Communications Systems*, vol. 9, no. 3-4, pp. 269–287, 2016.

[92] R. Bosch *et al.*, "Can specification version 2.0," *Rober Bousch GmbH, Postfach*, vol. 300240, p. 72, 1991.

[93] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata, "Cacan-centralized authentication system in can (controller area network)," in *14th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*, 2014.

[94] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, and K. Oishi, "A method of preventing unauthorized data transmission in controller area network," in *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*. IEEE, 2012, pp. 1–5.

[95] T. Dagan and A. Wool, "Parrot, a software-only anti-spoofing defense system for the can bus," *ESCAR EUROPE*, 2016.

[96] S. Abbott-McCune and L. A. Shay, "Intrusion prevention system of automotive network can bus," in *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*, Oct 2016, pp. 1–8.

[97] D. Souma, A. Mori, H. Yamamoto, and Y. Hata, "Counter attacks for bus-off attacks," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018, pp. 319–330.

[98] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot–a technique for cheap recovery," *arXiv preprint cs/0406005*, 2004.

[99] Xilinx, "Can v5.0 logicore ip product guide," August 2016. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/can/v5_0/pg096-can.pdf