DISSERTATION

ROBUST AND SECURE RESOURCE MANAGEMENT FOR AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

Submitted by

Vipin Kumar Kukkala

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2022

Doctoral Committee:

Advisor: Sudeep Pasricha

Anthony Maciejewski
Ali Pezeshki
Thomas Bradley

ABSTRACT

ROBUST AND SECURE RESOURCE MANAGEMENT FOR

AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

Modern vehicles are examples of complex cyber-physical systems with tens to hundreds of interconnected Electronic Control Units (ECUs) that manage various vehicular subsystems. With the shift towards autonomous driving, emerging vehicles are being characterized by an increase in the number of hardware ECUs, greater complexity of applications (software), and more sophisticated in-vehicle networks. These advances have resulted in numerous challenges that impact the reliability, security, and real-time performance of these emerging automotive systems. Some of the challenges include coping with computation and communication uncertainties (e.g., jitter), developing robust control software, detecting cyber-attacks, ensuring data integrity, and enabling confidentiality during communication. However, solutions to overcome these challenges incur additional overhead, which can catastrophically delay the execution of real-time automotive tasks and message transfers. Hence, there is a need for a holistic approach to a system-level solution for resource management in automotive cyber-physical systems that enables robust and secure automotive system design while satisfying a diverse set of system-wide constraints.

ECUs in vehicles today run a variety of automotive applications ranging from simple vehicle window control to highly complex Advanced Driver Assistance System (ADAS) applications. The aggressive attempts of automakers to make vehicles fully autonomous have increased the complexity and data rate requirements of applications and further led to the adoption of advanced artificial intelligence (AI) based techniques for improved perception and control. Additionally,

modern vehicles are becoming increasingly connected with various external systems to realize more robust vehicle autonomy. These paradigm shifts have resulted in significant overheads in resource constrained ECUs and increased the complexity of the overall automotive system (including heterogeneous ECUs, network architectures, communication protocols, and applications), which has severe performance and safety implications on modern vehicles. The increased complexity of automotive systems introduces several computation and communication uncertainties in automotive subsystems that can cause delays in applications and messages, resulting in missed real-time deadlines. Missing deadlines for safety-critical automotive applications can be catastrophic, and this problem will be further aggravated in the case of future autonomous vehicles. Additionally, due to the harsh operating conditions (such as high temperatures, vibrations, and electromagnetic interference (EMI)) of automotive embedded systems, there is a significant risk to the integrity of the data that is exchanged between ECUs which can lead to faulty vehicle control. These challenges demand a more reliable design of automotive systems that is resilient to uncertainties and supports data integrity goals. Additionally, the increased connectivity of modern vehicles has made them highly vulnerable to various kinds of sophisticated security attacks. Hence, it is also vital to ensure the security of automotive systems, and it will become crucial as connected and autonomous vehicles become more ubiquitous. However, imposing security mechanisms on the resource constrained automotive systems can result in additional computation and communication overhead, potentially leading to further missed deadlines. Therefore, it is crucial to design techniques that incur very minimal overhead (lightweight) when trying to achieve the above-mentioned goals and ensure the real-time performance of the system.

We address these issues by designing a holistic resource management framework called *ROSETTA* that enables robust and secure automotive cyber-physical system design while satisfying a diverse set of constraints related to reliability, security, real-time performance, and energy consumption. To achieve reliability goals, we have developed several techniques for reliability-aware scheduling and multi-level monitoring of signal integrity. To achieve security objectives, we have proposed a lightweight security framework that provides confidentiality and authenticity while meeting both security and real-time constraints. We have also introduced multiple deep learning based intrusion detection systems (IDS) to monitor and detect cyber-attacks in the in-vehicle network. Lastly, we have introduced novel techniques for jitter management and security management and deployed lightweight IDSs on resource constrained automotive ECUs while ensuring the real-time performance of the automotive systems.

ACKNOWLEDGEMENTS

I would like to thank all the individuals whose encouragement and support have made the completion of this thesis possible.

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Sudeep Pasricha, who has guided me through the process of doctoral study with his insightful and valuable advice. It was only with his encouragement and motivation I was able to explore some of the complex yet exciting problems in design optimization of automotive systems. He ensured that I improved my attention to detail while conducting research, which benefited me in both academic and non-academic activities. His devotion to my research and personal development was invaluable to me throughout my doctoral studies. He nurtured my analytical mindset and gave me sufficient time and opportunities to realize my true potential in accomplishing my Ph.D. goals. I appreciate all the help, guidance, and inspiration I received from Prof. Pasricha, who made it possible for me to survive the trial of graduate school with unforgettable memories and broadened horizons.

I would like to take this opportunity to thank the respected members of my Ph.D. committee, Prof. Thomas Bradley, Prof. Anthony Maciejewski, and Prof. Ali Pezeshki. Their feedback helped me rediscover my research and refine my work from different perspectives. I also much appreciate all the help I received from my mentors at National Renewable Energy Labs, Hewlett Packard Enterprise, MathWorks, and General Motors for their feedback in helping me shape my doctoral studies and my career.

A big thanks to my dear friends, current and former lab mates in Prof. Pasricha's EPIC lab: Sai Vineel Reddy Chittamuru, Yaswanth Raparti, Yi Xiang, Nishit Kapadia, Srinivas Desai, Daniel

Dauwe, Saideep Tiku, Ninad Hogade, Ishan Thakkar, Sooryaa Vignesh Thiruloga, Shoumik Maiti, Liping Wang, Sai Kiran Koppu, Jordan Tunnell, Varun Bhatt, Dylan Machovec, Joydeep Dey, Febin Sunny, Asif Anwar Baig Mirza, and Kamil Khan for their support and cooperation through intellectually stimulating conversations that boosted my morale while solving complex research problems.

I am blessed to have a wonderful family – my father Venkata Nageswara Rao Kukkala, my mother Nirmala Devi Kukkala, and my sister Harini Kukkala – for their support that empowered me to pursue my Ph.D. Their generosity and humility have made me continually strive to be a better person.

Lastly, I would like to thank the EcoCAR3 and EcoCAR Mobility Challenge programs and all the organizers and program sponsors, including the U.S. Department of Energy, General Motors, Argonne National Laboratories (ANL), and National Science Foundation (NSF) (through grant CNS-2132385) for supporting my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF RESEARCH PUBLICATIONS

JOURNAL PUBLICATIONS:

- V. K. Kukkala, S. Pasricha, T. Bradley, "Advanced Driver-Assistance Systems: A path toward autonomous vehicles," in **IEEE Consumer Electronics Magazine (CEM), Vol. 7, Iss. 5, September 2018**.
- V. K. Kukkala, S. Pasricha, T. Bradley, "JAMS-SG: A Framework for Jitter-Aware Message Scheduling for Time-Triggered Automotive Networks," in **ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 24, Iss. 6, September 2019**.
- V. Kukkala, S. Pasricha, T. Bradley, "SEDAN: Security-Aware Design of Time-Critical Automotive Networks," in **IEEE Transaction on Vehicular Technology (TVT), Vol. 69, Iss. 8, August 2020**.
- V. K. Kukkala, S. V. Thiruloga, S. Pasricha, "INDRA: Intrusion Detection using Recurrent Autoencoders in Automotive Embedded Systems," in **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 39, Iss. 11, November 2020.**
- V. K. Kukkala, S. V. Thiruloga, S. Pasricha, "LATTE: LSTM Self-Attention based Anomaly Detection in Embedded Automotive Platforms," in **ACM Transactions on Embedded Computing Systems (TECS), Vol. 20, No. 5s, Article 67, August 2021.**
- V. K. Kukkala, S. V. Thiruloga, S. Pasricha, "Roadmap for Cybersecurity in Autonomous Vehicles," in **IEEE Consumer Electronics Magazine (CEM), 2022.**

CONFERENCE PUBLICATIONS:

- V. K. Kukkala, T. Bradley, S. Pasricha, "Priority-based Multi-level Monitoring of Signal Integrity in a Distributed Powertrain Control System," in Proc. of **4th IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling, July 2015**.
- V. K. Kukkala, T. Bradley, S. Pasricha, "Uncertainty Analysis and Propagation for an Auxiliary Power Module," in Proc. of **IEEE Transportation Electrification Conference (TEC), June 2017**.
- V. K. Kukkala, S. Pasricha, T. Bradley, "JAMS: Jitter-Aware Message Scheduling for FlexRay Automotive Networks," in Proc. of **IEEE/ACM International Symposium on Network-on-Chip (NOCS), October 2017**.
- G. C. DiDomenico, J. Bair, V. K. Kukkala, et al., "Colorado State University EcoCAR 3 Final Technical Report," in **SAE World Congress Experience (WCX), April 2019**.
- S. V. Thiruloga, V. K. Kukkala, S. Pasricha, "TENET: Temporal CNN with Attention for Anomaly Detection in Automotive Cyber-Physical Systems," in Proc. of **IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC), January 2022**. (*Best paper award candidate*)

BOOK CHAPTERS:

- V. K. Kukkala, S. V. Thiruloga, S. Pasricha, "AI for Cybersecurity in Distributed Automotive IoT Systems," in **Electronic Design for AI, IoT and Hardware Security** *(to appear)***, Springer, 2022**.

- V. K. Kukkala, S. V. Thiruloga, S. Pasricha, “Machine Learning for Anomaly Detection in Automotive Cyber-Physical Systems,” in **Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing** *(to appear)*, **Springer, 2022.**

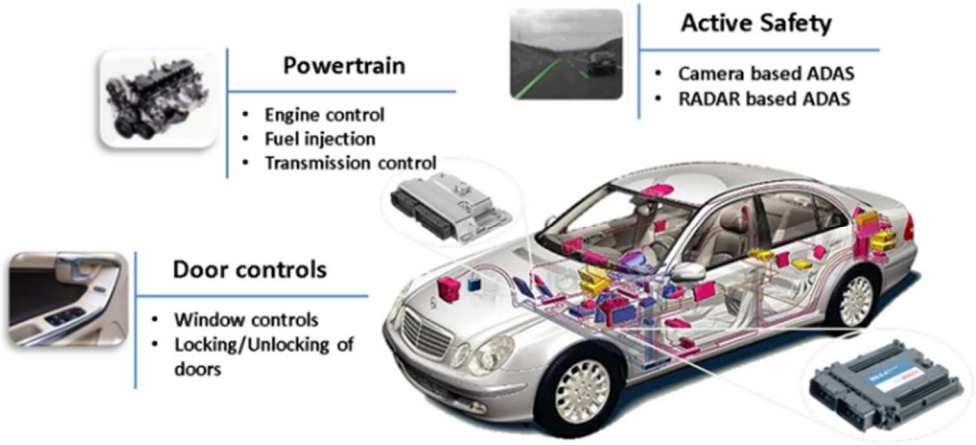
1. INTRODUCTION

Autonomous vehicles are on the horizon and will be revolutionizing the future of transportation safety and comfort. These vehicles will be connected to various external systems and utilize advanced embedded systems to perceive their environment and make intelligent decisions. However, several challenges must be addressed in today's automotive systems to achieve the goal of future vehicle autonomy. In this chapter, we present an overview of the current state-of-the-art automotive systems and trends, and motivate the need for resource management techniques to solve the various design challenges of automotive cyber-physical systems that impact the reliability, security, and real-time performance goals. Moreover, in this chapter, we present an overview of our resource management framework called *ROSETTA* that addresses various design challenges in automotive cyber-physical systems and enables robust (from a real-time performance and reliability perspective), and secure future autonomous vehicles.

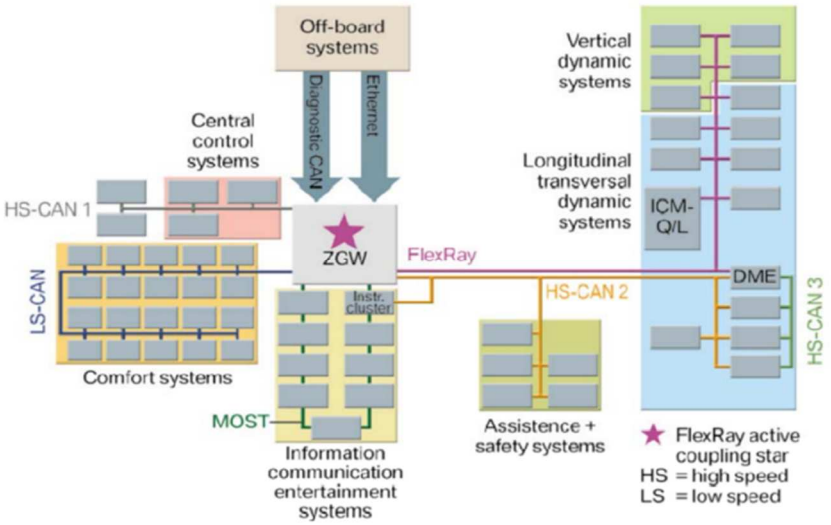
1.1. OVERVIEW OF MODERN AUTOMOTIVE SYSTEMS

Modern vehicles consist of tens to hundreds of processing elements called Electronic Control Units (ECUs) that manage various vehicular subsystems. Examples of ECUs include the engine control unit, transmission control unit, and body control module. To meet the requirements of various vehicular subsystems, a diverse set of ECUs consisting of different compute and memory capacities are used in today's vehicles. These ECUs run various time-critical automotive applications and communicate using different in-vehicle networks, exchanging information in the form of messages which consist of one or more signals. A signal can be a raw data value or control information. Various in-vehicle network protocols such as controller area network (CAN), local

interconnect network (LIN), FlexRay, and Ethernet are employed in today's vehicles to meet the data rate, timing, and reliability requirements of automotive applications. An illustration of a modern vehicle with distributed ECUs and example functionalities, and an in-vehicle network architecture of a real-world vehicle with different in-vehicle network protocols is shown in Figure 1(a) and Figure 1(b), respectively.



(a)



(b)

Figure 1 Illustration of (a) distributed ECUs with example functionalities, and (b) BMW-7 series in-vehicle network architecture [1].

Modern ECUs run various automotive applications ranging from simple vehicle window control to complex safety-critical functions such as airbag deployment and powertrain control. In recent years, intelligent applications such as collision avoidance, lane keep assist, pedestrian detection, and traffic sign detection, which are shown in Figure 2, are becoming increasingly common. These safety-critical applications have strict timing and deadline constraints that need to be met. Failing to meet the deadlines can have catastrophic consequences for the performance and safety of the vehicle (such as delay in deployment of airbags and delay in automatic collision avoidance). Thus, modern vehicles are highly complex distributed hard-real time cyber-physical systems.

(a)

(b)

(c)

Figure 2 Illustration of (a) collision avoidance and lane line detection, (b) pedestrian detection, and (c) traffic sign detection [2].

Table 1 SAE J3016 Levels of driving automation [3].

SAE autonomy level	Definition	Example feature
Level-0	No automation	Blind spot warning, lane departure warning
Level-1	Driver assistance	Lane centering or adaptive cruise control
Level-2	Partial automation	Lane centering and adaptive cruise control
Level-3	Conditional automation	Traffic jam chauffeur
Level-4	High automation	Local driverless taxi
Level-5	Full automation	Same as level 4, but can drive everywhere in all conditions

To standardize vehicle autonomy, the society of automotive engineers (SAE) has defined six levels of automation in the SAE J3016 standard [3] (shown in Table 1), with level-0 having no automation and level-5 being fully autonomous that can drive anywhere in all conditions. Most of today's vehicles are level-2, with a few being level-3. However, there has been an increasing push towards level-3 and level-4 autonomous vehicles by various automakers. With level-3 and level-4 autonomous vehicles expected to hit the roads soon, automotive hardware and software components are experiencing a paradigm shift. Increasing autonomy has led to the integration of

a diverse set of heterogeneous sensors, including cameras, RADAR (radio detection and ranging), LIDAR (light detection and ranging), and ultrasonic sensors to better perceive the environment and make intelligent decisions. These advances have facilitated the development of a new class of intelligent systems called advanced driver assistance systems (ADAS) that improve the vehicle's safety, comfort, and fuel economy while playing a crucial role in enabling future autonomy. Some of the prominent ADAS applications and the sensors used to enable them in today's vehicles are shown in Figure 3.

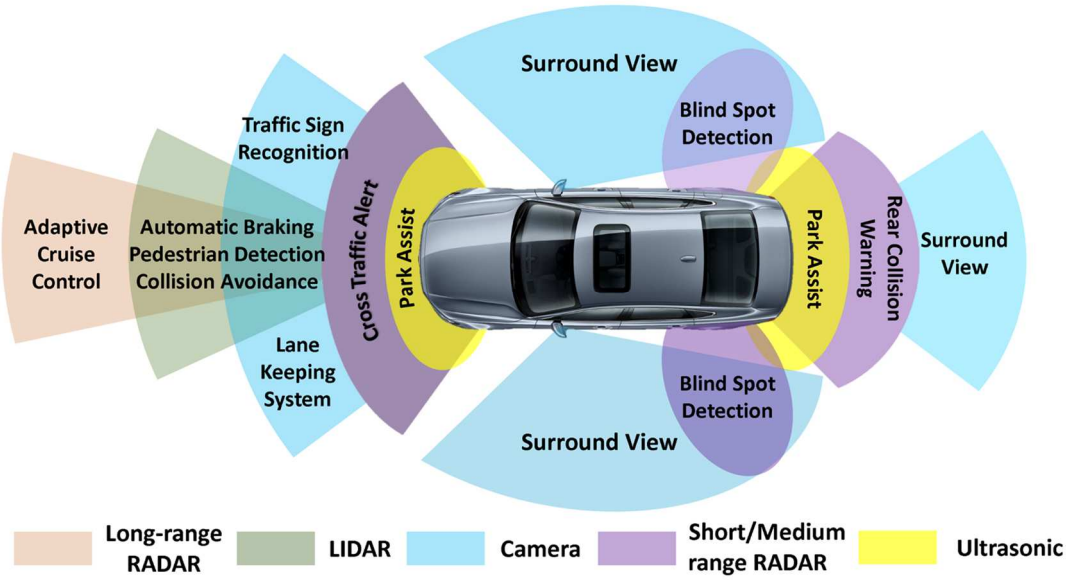


Figure 3 The state-of-the-art ADAS applications and the sensors used to enable them [2].

Many state-of-the-art ADAS applications use various perception tasks such as distance estimation, objection detection, and object tracking to perceive the environment. It is crucial that these applications rely on multiple sensors and different sensor types, as there are inherent limitations with each sensor type. For instance, cameras are highly versatile and can be used for different ADAS tasks but suffer during low light and bright light conditions, and when obstructed by obstacles. Similarly, RADARs are good at detecting objects at long distances but have low

resolution, while LIDARs offer high resolution but suffer during heavy rain or foggy conditions. Thus, it is essential to combine the information from various sensors to achieve different perception goals. This is known as sensor fusion and helps sensors complement each other's limitations and provides high precision, reliability, robustness to uncertainty, extended spatial and temporal coverage, and improved resolution, which are crucial in safety-critical automotive cyber-physical systems. Thus, sensing in modern vehicles is extremely complex and requires handling of large volumes and high rates of data.

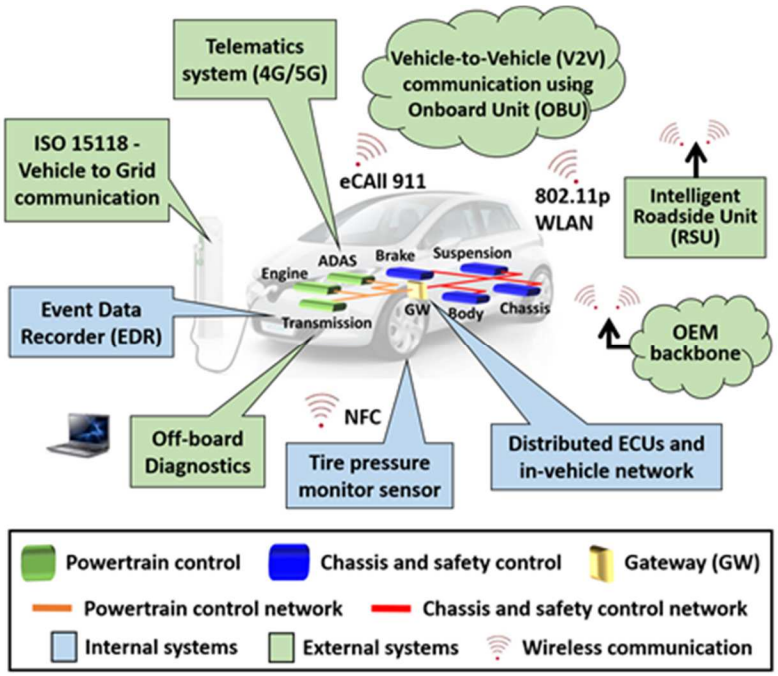


Figure 4 Illustration of an in-vehicle network connecting different ECUs using isolated networks that are connected to a gateway (GW), and external connectivity of modern vehicles with various systems in the environment [4].

In addition to different sensors that are being integrated into today's vehicles, many ADAS solutions and other intelligent applications are becoming increasingly dependent on information from various external systems to function accurately. This has led to the development of intelligent transportation systems (ITS) that provide the vehicle with a variety of information that can be used

to make intelligent decisions. Some of the example ITS applications include traffic monitoring, real-time parking management, and emergency vehicle notification. This communication between vehicle and external systems is facilitated by the onboard units (OBUs) and roadside units (RSUs). The common internal (in blue) and external (in green) communications in modern vehicles are shown in Figure 4. The integration of diverse communication systems has resulted in a complex communication network where vehicles interact with other moving and stationary vehicles, and road infrastructure wirelessly, forming a vehicular ad hoc network (VANET). Modern VANETs are the building blocks of future autonomous vehicles' communication fabric.

Today's VANETs mainly consist of five communication modes: *(i)* vehicle-to-vehicle (V2V), which involves communication between OBUs on vehicles, *(ii)* vehicle-to-infrastructure (V2I), which involves communication between OBUs and RSUs, *(iii)* vehicle-to-pedestrian (V2P), which involves communication between OBUs and personal mobile devices of pedestrians, *(iv)* vehicle-to-cloud (V2C) , which involves communication between OBUs and cloud-based backend systems and, *(v)* vehicle-to-everything (V2X) , which involves communication between OBUs and other objects such as traffic lights and traffic signs. More recently, cellular vehicle-to-everything (C-V2X) communication that enables low latency V2V, V2I, V2P, and V2C using 5G and other cellular standards is gaining popularity. This has resulted in the development of various advanced communication standards such as wireless access in vehicular environment (WAVE) [5] and dedicated short-range communications (DSRC) [6] to realize different communication modes.

As the modern ECUs are becoming increasingly powerful due to the integration of multi-core processors and dedicated hardware accelerators, today's vehicles are increasingly adopting various machine learning (ML) based techniques to better perceive the vehicle environment. Due to the large availability of data and increased computation power, advanced deep learning and

artificial intelligence (AI) based techniques are heavily employed in present-day ADAS and semi-autonomous functionalities to make intelligent decisions. For instance, deep learning models that use convolutional neural networks (CNNs) are widely used for object detection using deep learning models such as You Only Look Once (YOLO) [7] and Region-based CNNs (R-CNN) [8]. Tesla's autopilot is one such system that uses deep learning based techniques to process information from multiple cameras to perceive the environment. An example of Tesla's autopilot machine-perceived view is shown Figure 5 on the right-hand side. These deep learning techniques are transforming the landscape of modern automotive systems and have led to the introduction of Waymo's driverless ride-sharing vehicles and Tesla's full self-driving (FSD) capabilities. Many such systems require advanced control design to precisely control different vehicular subsystems. Thus, AI-based autonomy will be the basis for autonomous vehicles.

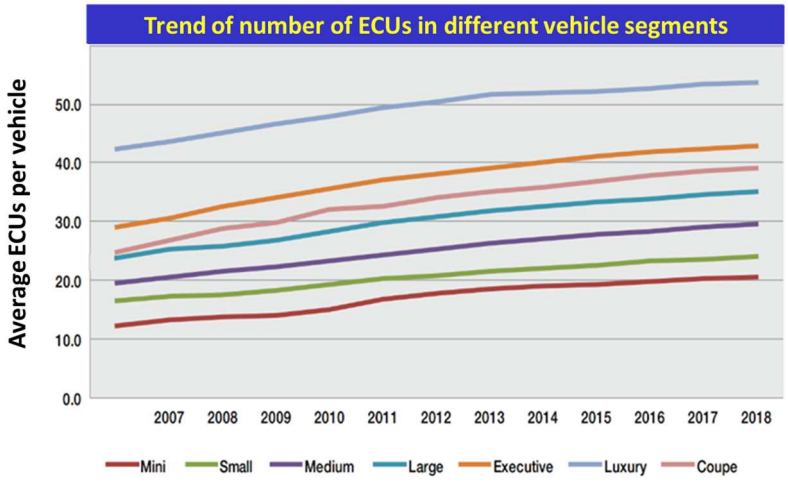
Figure 5 The Tesla's autopilot perceived view of surroundings using different cameras [9].

In summary, the integration of powerful ECUs, heterogeneous vehicular networks, greater connectivity with various external systems, the ability to perceive the environment using different

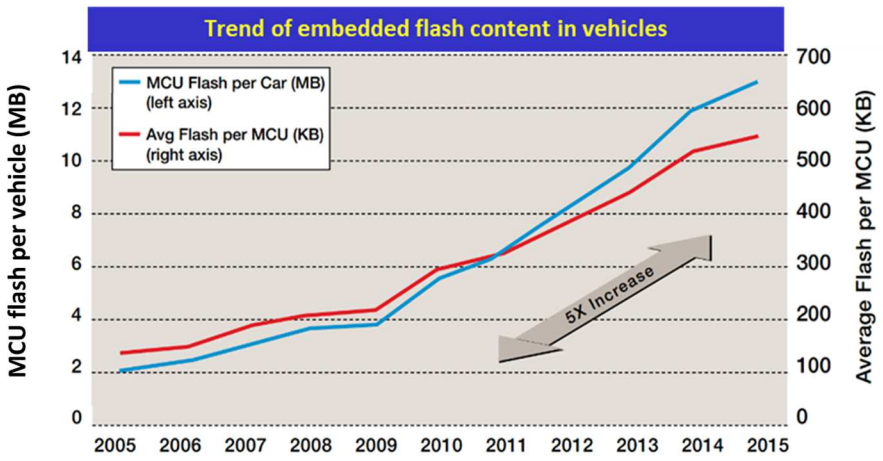
sensors, and advanced deep learning algorithms has revolutionized the capabilities of modern vehicles. These technological advances are paramount for future autonomous vehicles in enabling safe, secure, and reliable transportation.

1.2. MOTIVATION FOR RESOURCE MANAGEMENT IN AUTOMOTIVE SYSTEMS

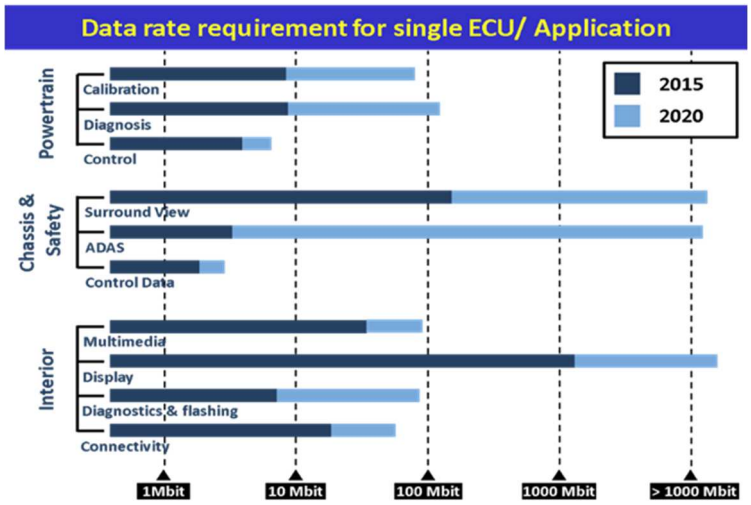
The aggressive attempts of automakers to make vehicles fully autonomous have significantly transformed vehicular subsystems. In recent years, there has been an increased integration of powerful ECUs across various vehicle segments to handle different functionalities to satisfy emerging autonomy needs. For instance, some modern luxury vehicles have more than a hundred different ECUs. Additionally, the advent of state-of-the-art ADAS solutions and other complex control applications has resulted in an increase in the complexity of automotive software. On average, today's high-end vehicles have over 100 million lines of code [10]. Moreover, to meet these advanced applications' increasing data rate requirements, various heterogeneous in-vehicle network protocols are employed in today's vehicles. The trends of increasing ECU count, software complexity, and data rate requirements are illustrated in Figure 6(a), Figure 6(b), and Figure 6(c), respectively.



(a)



(b)



(c)

Figure 6 Trend of (a) number of ECUs in different vehicle segments [11], (b) embedded software in vehicles [11], and (c) data rate requirement for various applications [12].

The increased computation and communication demands and the adoption of various machine learning and AI-based solutions has resulted in increased power and energy consumption, which have significant performance implications particularly for emerging battery-based electric vehicles (EVs). As automotive embedded systems are highly resource-constrained, running these power-hungry applications will further exacerbate the problem. They could lead to various thermal

challenges that severely degrade the application performance and possibly missed deadlines. Thus, the above-mentioned advances have resulted in increasing the overall complexity of the automotive system, which has resulted in three new challenges in designing efficient automotive cyber-physical systems.

Firstly, automotive systems are designed to be deterministic to ensure the timeliness of safety-critical applications. As the complexity of modern automotive systems keeps increasing, the potential for computation and communication uncertainties (such as *jitter*) is also increasing. This presents a great threat to the reliability of automotive systems. These uncertainties are the major cause of delays in applications and messages, resulting in poor application latencies and potentially missed real-time deadlines. Additionally, the integrity of data exchanged between ECUs to run various applications plays a crucial role in the control performance and safety of the vehicle. *With autonomous vehicles expected to hit the roads soon, it is crucial to enable reliable communication in automotive systems that is resilient to uncertainties and also maintains data integrity*.

Secondly, the increased connectivity of modern vehicles has resulted in vehicles becoming highly vulnerable to cyber-attacks. Many attacks have been demonstrated on vehicles using different attack vectors, including WiFi, telematics, Bluetooth, keyless entry systems, and mobile applications [25]-[34]. In the past decade, nearly 79.6% of all automotive attacks have been remote attacks, which do not require the attacker to be within the proximity of the vehicle [13]. *Hence, it is vital to enable secure communication, and this will become crucial as smart and self-driving vehicles become more ubiquitous*. Moreover, the overall increase in the complexity of the automotive systems has resulted in very limited visibility within the in-vehicle network, which poses a significant challenge in detecting cyber-attacks in the in-vehicle network. *It is essential to*

detect cyber-attacks by efficiently monitoring and analyzing data on the in-vehicle network as this often acts as the last line of defense when the attacker breaks through conventional defense mechanisms.

Lastly, modern automotive systems have limited computation and memory capabilities, making them highly resource-constrained embedded systems. Thus, implementing any techniques to improve robustness and security will result in incurring additional overhead on the ECUs. This can adversely affect the safety-critical applications running on the ECUs and potentially lead to missing real-time deadlines. *Therefore, it is crucial to design techniques that incur very minimal overhead on ECUs and ensure that the real-time performance of the system is not compromised.*

Conventional automotive resource management techniques have not been designed to handle such complex system sizes and high application demands. *Thus, there is an urgent need for an efficient resource management framework that can handle complex state-of-the-art automotive systems and address the above-mentioned challenges*. This is imperative for achieving robust and secure future autonomous vehicles.

1.3. REAL-TIME PERFORMANCE CHALLENGES IN AUTOMOTIVE SYSTEMS

Modern automotive systems consist of various subsystems that perform a diverse set of functions with varying criticalities and priority levels. For instance, a safety-critical application such as collision avoidance needs to be executed in a timely manner and must meet the timing and deadline constraints (hard real-time deadline), while a few deadline misses can be tolerated (soft real-time deadline) for infotainment applications. This mixed-criticality nature of automotive systems presents a unique opportunity to efficiently prioritize and schedule the computation and

communication of automotive tasks and messages while meeting all the hard real-time and most of the soft real-time deadlines.

Table 2 Determination of ASIL level in ISO 26262 [14].

Severity class	Probability class		Controllability class		
			C1	C2	C3
	S1	E1	QM	QM	QM
		E2	QM	QM	QM
		E3	QM	QM	A
		E4	QM	A	B
	S2	E1	QM	QM	QM
		E2	QM	QM	A
		E3	QM	A	B
		E4	A	B	C
	S3	E1	QM	QM	A
		E2	QM	A	B
		E3	A	B	C
		E4	B	C	D

Several methodologies, such as the automotive safety integrity level (ASIL) risk classification scheme that is presented in the ISO 26262 [14], can be used to derive the priorities for various functions across different vehicular subsystems. The ISO 26262 standard introduces four ASIL levels, with ASIL-D being the highest criticality level and ASIL-A being the lowest. In addition, another level called quality management (QM) is defined for hazards that do not have safety implications. The ASIL levels are determined based on the severity, controllability, and exposure factors shown in Table 2. The severity level S3 defines life-threatening consequences while S1 defines light and medium injuries. The controllability level is defined as the degree to which the driver can control the vehicle in the event of a failure, with C3 being hard to control and C1 being easy to handle. Lastly, the exposure level defines the probability of the vehicle being involved in that hazard and has four levels, with E4 being highly probable and E1 indicating very

low probability. The priority levels derived from this approach can be used to develop safety requirements for various automotive functions that can be used to efficiently manage the real-time performance of various vehicular subsystems.

Additionally, implementing any mitigation techniques on resource-constrained automotive ECUs incurs additional overhead as they are collocated with the real-time automotive applications. This presents a unique challenge of ensuring real-time performance goals by preventing any mitigation techniques from having a negative impact on the performance of real-time automotive applications. Most of these mitigation techniques are designed offline and deployed at runtime to achieve various goals. However, they seldom work faultlessly due to various systematic and random runtime perturbations that arise from the harsh operating conditions of automotive systems. *Hence, there is a need for runtime management frameworks that can handle real-time perturbations and assist the design time generated techniques while meeting real-time performance goals and deadline requirements*.

1.4. RELIABILITY CHALLENGES IN AUTOMOTIVE SYSTEMS

Modern automotive systems operate under extreme conditions, including high temperatures and vibrations from moving mechanical components, electromagnetic interference (EMI) from high voltage components (such as battery and DC-DC converters in electric vehicles), and radio frequency interference (RFI) from vehicle-to-everything (V2X) communications. These extreme conditions introduce various computation and communication uncertainties in resource-constrained automotive embedded systems that can hurt the performance and safety of the vehicle. Moreover, the growing complexity of automotive systems (due to the reasons discussed in Section 1.2) introduces additional risks that directly impact the system's overall reliability. In this

subsection, we discuss the key aspects of reliability in automotive electronics, primarily related to computation and communication uncertainty (*jitter*), data integrity, and various types of faults.

1.4.1. JITTER

The deterministic nature of automotive safety-critical applications requires the ECUs to execute them as periodic tasks. This results in messages being exchanged between ECUs to possess periodic or time-triggered characteristics. An example of various time-triggered messages in a vehicle powertrain system is shown in Figure 7, where the highlighted cycle time column denotes the message period in milliseconds. Moreover, the de facto vehicle communication protocols such as Controller Area Network (CAN) [15] are increasingly unable to meet the high-speed and time-determinism requirements of modern applications. This has led to the introduction of advanced time-triggered communication protocols such as FlexRay [16], time-triggered Ethernet (TTEthernet) [17], and time sensitive networking (TSN) standards [18].

Name	ID	ID-Format	DLC [...	Cycle Time (ms)	Transmitter
ABSdata	0xC9	CAN Standard	6	50	Engine
Diag_Request	0x200	CAN Standard	8	2	-- No Transmit...
Diag_Response	0x400	CAN Standard	8	2	-- No Transmit...
DiagRequest_...	0x601	CAN Standard	8	2	Gateway
DiagRespons...	0x608	CAN Standard	8	2	Engine
EngineData	0x64	CAN Standard	8	50	Engine
EngineDataIEEE	0x66	CAN Standard	8	50	Engine
EngineStatus	0x65	CAN Standard	1	2	Engine
GearBoxInfo	0x3FC	CAN Standard	1	50	Engine
Ignition_Info	0x67	CAN Standard	2	20	Gateway
NM_Engine	0x51B	CAN Standard	4	2	Engine
NM_Gateway...	0x51A	CAN Standard	4	2	Gateway

Figure 7 Example time-triggered messages in a vehicle powertrain system with the highlighted column denoting the message period in milliseconds [19].

One of the major challenges with time-triggered communication is jitter, which is the unpredictable delay-induced deviation from the actual periodicity of a message. Failure to account

for jitter can lead to missing deadlines and have catastrophic consequences in time-sensitive automotive platforms. The deterministic or bounded jitter that is caused by the systematic occurrences of certain events in the system (such as queuing of messages and clock jitter) has been widely studied and can be easily predicted based on observations. However, the random or unbounded jitter, which is the unpredictable timing noise caused due to factors such as thermal noise in an electrical circuit and external disturbances, is hard to predict. The random jitter causes delayed task executions and message transmissions, resulting in poor application performance and missed real-time deadlines. Only a few works such as [20] and [21] have explored the jitter mitigation techniques for state-of-the-art time-triggered in-vehicle network protocols. However, these techniques consider highly pessimistic scenarios, leading to over designed systems, poor overall application performance, and low scalability. *Thus, designing a robust jitter-aware scheduler that can operate on newer time-triggered protocols and can mitigate the effects of random jitter to ensure reliable communication in automotive systems, remains an open problem*.

1.4.2. DATA INTEGRITY

The ECUs that run different automotive applications are distributed across the vehicle and exchange information using messages. Every message consists of one or more signals that carry raw data values or control information that originates from a sensor, an actuator, or is generated as a result of task execution on the ECU. The extreme operating conditions of automotive systems (described in Section 1.4) can potentially corrupt signal data, resulting in loss of critical information, which can have severe performance and safety implications. For instance, scenarios such as corruption of brake signal data in the event of emergency braking and unintentional acceleration due to faulty acceleration signal can be highly fatal. In addition to the communication

between ECUs, data integrity issues can also arise within the ECUs due to aging hardware components, power supply noise, electromagnetic crosstalk within the SoC, soft and hard errors.

The mixed-criticality nature of automotive applications has resulted in messages and signals having different levels of criticalities both within and across the applications. This leads to different signals having varying levels of impact on vehicle safety and control performance in the event of signal integrity failure. Thus, it is essential to handle signal integrity failures based on the signal criticality and its impact on the safety and performance of the vehicle. Moreover, having one remedial action for all signal integrity failures does not scale well as different messages and signals have different transmission rates and error tolerances. Additionally, the mitigation techniques for handling signal integrity failures need to consider the vehicle's state and the severity of failure when initiating appropriate remedial action. *Therefore, it is vital to preserve signal integrity and ensure that proper remedial action is undertaken in the event of signal integrity failure to ensure control stability and safety of the vehicle*.

1.4.3. FAULTS IN AUTOMOTIVE IP

The increased complexity of automotive systems has led to various types of faults in automotive IP (intellectual property; which includes both software and hardware). These faults pose a significant threat to the control performance and safety of the vehicle. At a high level, these faults can be classified into two categories: *(i) systematic* and *(ii) random* faults. The former is a deterministic fault that occurs as a result of failing to follow the best design practices and can be identified by carefully inspecting the design process and experimentation. In contrast, the latter type of fault occurs erratically and typically follow a probability distribution. The mitigation of systematic faults requires changes in the design of the system, while the random faults cannot be avoided and need to be detected and handled appropriately. A detailed fault reference chart based

on the ISO 26262 [14] standard with the methodologies to handle systematic and random faults in automotive IP is shown in Figure 8. The systematic faults related to the development process and the software bugs can be addressed using approaches highlighted in green boxes. On the other hand, the effects of random hardware faults can be mitigated by implementing various safety mechanisms and using qualitative and quantitative analysis, as shown in Figure 8.

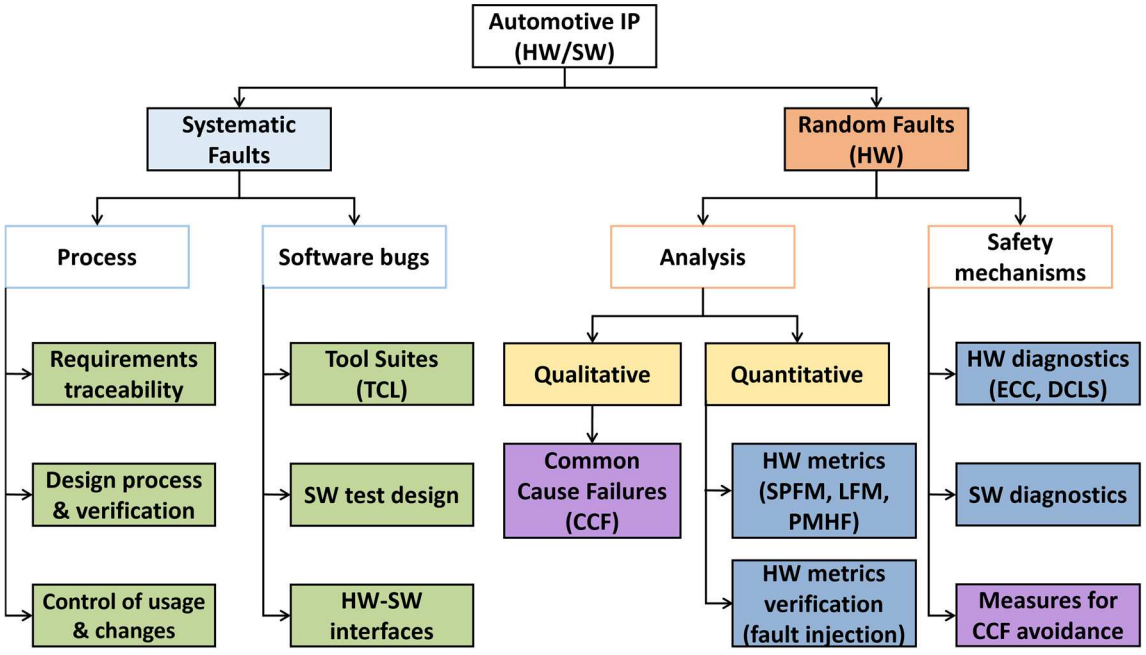


Figure 8 Detailed fault reference chart based on the ISO 26262 [22].

Additionally, soft errors [23] and errors induced due to device aging [24] also pose a significant risk to the reliability of automotive embedded systems. The soft errors are transient faults in semiconductor devices caused by high-energy (e.g., alpha) particle strikes resulting in random bit flips. On the other hand, device aging causes different random faults, including unwanted delays, data corruption, and non-functional devices. It is crucial to handle all of these faults as they can have catastrophic consequences when managed poorly. *Thus, there is a crucial*

need to design effective mitigation techniques that help in minimizing the impact of these faults and avoid adverse effects of faulty vehicular control.

1.5. SECURITY CHALLENGES IN AUTOMOTIVE SYSTEMS

The increased connectivity with various external systems has made modern vehicles highly vulnerable to various security attacks. A variety of attack vectors have been used to gain unauthorized access to the vehicle. A timeline showing some of the prominent cyber-attacks on automotive systems in the past decade (2010 onwards) is presented in Figure 9.

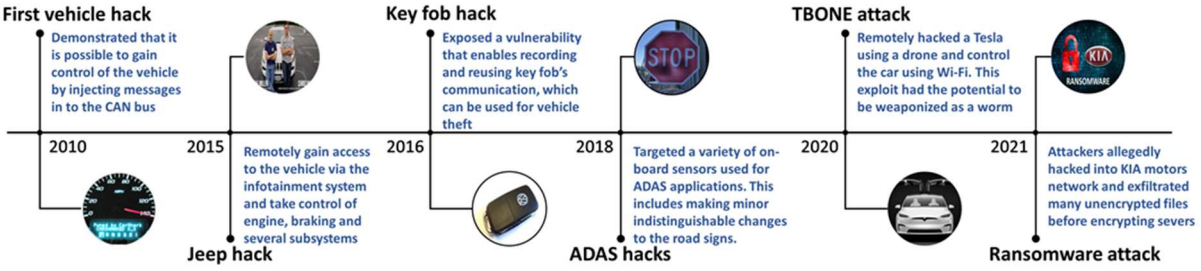


Figure 9 Timeline of major automotive cyber-attacks [4].

The researchers at the University of California at San Diego and the University of Washington demonstrated one of the first vehicle hacks in 2010 [25] by exploiting the onboard telematics system and were able to gain full control of the vehicle. Several other works followed this approach for the next several years, but all of these attacks required the attacker to be physically present inside the target vehicle, which resulted in dismissing as an unlikely situation. This changed in 2015 when the first major remote attack was demonstrated on an unaltered 2014 Jeep Cherokee [26] by two security researchers. The researchers identified a software bug in the vehicle's infotainment system that would allow them to connect to the vehicle remotely over the 4G LTE and send CAN messages to the ECUs in the vehicle. They demonstrated a wide range of

attacks ranging from remotely controlling simple functionality such as the vehicle radio, A/C, and windshield wipers to more critical functionalities such as controlling brakes, transmission, and even killing the engine while the vehicle was on a freeway. This attack completely changed the landscape of automotive cyber-attacks and highlighted the urgency to address cybersecurity in vehicles. Starting around 2016, a new type of attack emerged that focused on hacking the keyless entry system in vehicles. The goal of these attacks was to steal the vehicle rather than remotely control it. The researchers at the University of Birmingham showed how they were able to recover the cryptographic algorithms and keys from the ECUs and clone the VW group remote control by eavesdropping on a single signal sent by the original remote [27]. Similar attacks were conducted on by cloning the Tesla Model S key fob in 2018 [28] and the Tesla Model X key fob in 2019 [29] by capturing the Bluetooth communication between the key fob and the body control module.

A different class of attacks has gained popularity since 2018, that mainly targeted the ADAS systems, and the onboard sensors used for perception. In [30], researchers generated various robust visual adversarial perturbations to a stop sign that resulted in it being misidentified as a 45 mph speed limit sign. More recent attacks include tricking lane change system of a Tesla Model S with bright stickers on the road by Tencent Keen security lab in 2019 [31] and object removal attacks on LiDAR sensors in 2021 [32]. Another recent attack that made the headlines was the T-BONE attack [33] where researchers were able to gain remote code execution (RCE) over Wi-Fi on the infotainment system in a Tesla Model 3 using a drone. They were able to remotely open doors and trunk, change seat positions, steering, and acceleration modes. The researchers also highlighted that adding a privilege escalation exploit such as CVE-2021-3347 to T-BONE would weaponize this exploit and turn it into a worm. This would allow them to load new Wi-Fi firmware and exploit other Tesla cars in the victim car's proximity. More recently, in the beginning of 2021, an online

hacking group by the name DoppelPaymer claimed to conduct a ransomware attack on KIA motors America and have stolen unencrypted confidential data [34].

A common feature among all of the attacks that happened on the vehicles involves gaining unauthorized access to the in-vehicle network. Once the attacks gain access to the vehicular network, malicious payloads are deployed to ECUs to hijack the vehicle or achieve the malicious goals of the attackers. Therefore, in-vehicle network security is a crucial element in achieving secure automotive systems.

1.5.1. OVERHEAD OF SECURITY SCHEMES

Traditional in-vehicle network protocols such as CAN, LIN, and FlexRay, do not have any inherent security features that would prevent unauthorized access to the vehicular networks. To improve vehicular security, three crucial security objectives must be achieved: confidentiality, authentication, and authorization. Confidentiality refers to the practice of protecting information from unauthorized ECUs, and authentication refers to the process of correctly identifying an ECU. In contrast, authorization refers to the process of verifying an ECU's access to a particular resource. Achieving the above-mentioned security objectives require implementing additional security mechanisms in the ECUs. The symmetric key [35] and asymmetric key [36] encryptions are the two most widely used techniques. The former uses the same key (secret key) for both encryption and decryption operations, while the latter uses a public-private key pair that has a strong mathematical relationship. The public key is distributed to all the parties that want to communicate to encrypt the data, and the private key is kept secret and is used to decrypt the data. However, both mechanisms incur additional computational overhead on the ECUs, which may catastrophically delay the execution of real-time automotive tasks and message transfers. For

instance, a delay in the messages from impact sensors to the airbag deployment systems could lead to serious injuries for vehicle occupants. *Thus, security schemes deployed in automotive systems must be carefully designed for minimal overhead.*

When implemented in automotive ECUs, security schemes need to account for the utilization overhead of ECUs and latency overhead on the messages. Failure to do so will result in an over-optimistic design that can suffer in performance or even fail due to missed real-time deadlines. Additionally, considering diverse security requirements such as different key sizes and encryption techniques, heterogeneous ECU architectures, and precedence constraints (ordering requirements) for messages enables further optimization of the security schemes. Moreover, with the increasing adaptation of heterogeneous multi-core ECUs in the automotive domain, dedicated co-processors or hardware accelerators for security schemes need to be considered when designing security schemes. Some of the prior works such as, [37]-[41], tried to address this problem but failed to address a subset or even most of the crucial characteristics, including ECU utilization, latency overhead, precedence constraints for messages and tasks, heterogeneous ECU architectures and diverse set of security requirements. *Thus, there is a need for a lightweight security scheme that would prevent unauthorized access to information traversing over the in-vehicle networks while incurring minimal overhead on the ECUs*.

1.5.2. INTRUSION DETECTION SYSTEM

A well-designed pragmatic security scheme is crucial to prevent cyber-attacks but does not always guarantee 100% protection. Hence, it is essential to deploy active monitoring systems in tandem to detect cyber-attacks in in-vehicle networks. Such systems are knowns as Intrusion Detection Systems (IDSs). Traditionally, such IDSs in computing systems have relied on using

firewalls or rule-based systems to detect cyber-attacks. However, due to the increased complexity of modern automotive attacks, these simple systems fail to detect them. Moreover, the increased complexity of the automotive systems (heterogeneous ECUs, network architectures/protocols, and applications) have resulted in poor attack visibility in the in-vehicle networks, which makes it further challenging for traditional IDSs to detect cyber-attacks. *Thus, there is an urgent need for an intelligent IDS that monitors the in-vehicle network to detect cyber-attacks in vehicles*.

Moreover, modern vehicles have been increasingly adopting various AI-based techniques for different ADAS applications, where environmental perception is required. Such AI techniques can also be deployed in powerful modern-day automotive ECUs to monitor and detect cyber-attacks. As AI-based solutions are well known to be highly efficient in learning complex patterns, they can be used to capture the relationships that exist in high-dimensional time-series vehicular network data. These learned patterns can be used to monitor the vehicular networks and observe for anomalies to detect cyber-attacks. *With fully autonomous vehicles supporting increased connectivity to external subsystems on the horizon, having an efficient IDS that can detect a variety of cyber-attacks using AI-based techniques presents a promising solution to this problem*.

1.6. DISSERTATION OVERVIEW

To address the challenges presented in the previous subsections, in this dissertation, we propose a holistic resource management framework for automotive systems called *ROSETTA* that enables robust and secure automotive cyber-physical system design while satisfying a diverse set of constraints related to reliability, security, real-time performance, energy, and power consumption. An overview of the *ROSETTA* framework is illustrated in Figure 10, which

comprises of contributions published in [2], [4], [19], [42]-[49]. The rest of this dissertation is organized as follows:

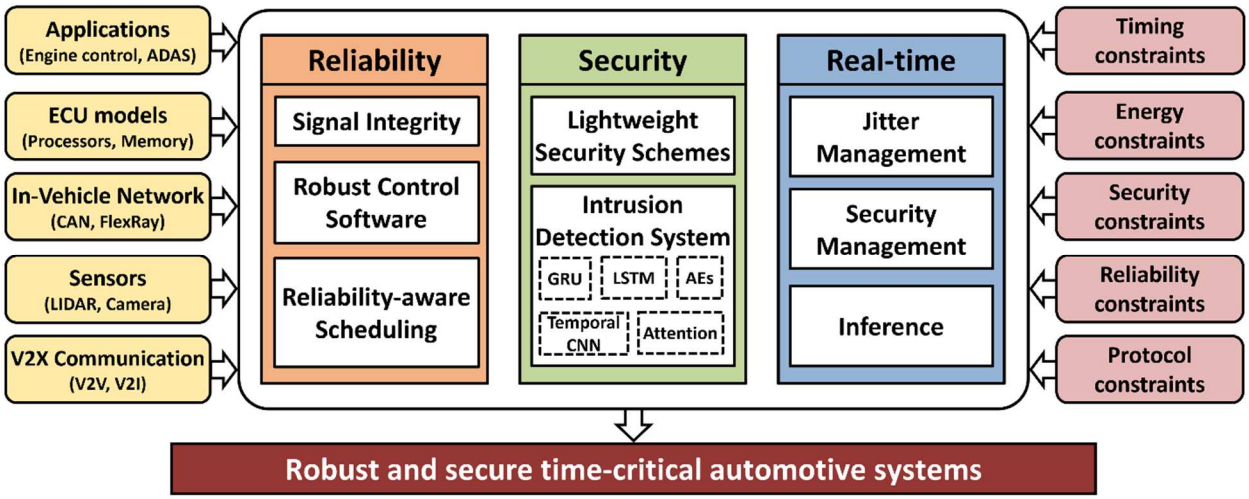


Figure 10 Overview of ROSETTA: robust and secure resource management framework for time-critical automotive systems.

In chapter 2, we discuss the stochastic models that were used to model random jitter and propose a hybrid heuristic based jitter-aware message scheduling framework called *JAMS-SG* [45]. Our proposed *JAMS-SG* framework uses a combination of simulated annealing (SA) and greedy adaptive randomized search procedure (GRASP) to achieve jitter-aware frame packing (packing signals into messages) and a feasible design time schedule. We also introduce a runtime scheduler that can opportunistically pack jitter-affected time-triggered messages and high-priority event-triggered messages in the FlexRay static segment slots using a multi-level feedback queue (MLFQ). Lastly, we present a custom addressing and segmentation scheme within the payload segment of the FlexRay frame to enable partial message transmission by the runtime scheduler.

In chapter 3, we present our proposed priority based multi-level monitoring approach [42] to ensure signal integrity of distributed powertrain control systems. We discuss how our proposed approach prioritizes preserving the signal integrity of torque-related signals (as they have a high impact on vehicle safety and performance) and present the details of handshake mechanisms and performance counters used to monitor signals. We introduce a state machine model that is used to track the maturity of faults and deploy remedial actions (limp modes) based on the severity of the fault. Lastly, we present a hardware-in-the-loop (HIL) testing analysis of the proposed technique as a part of the Colorado State University EcoCAR3 project [19].

In chapter 4, we present a novel holistic security framework called *SEDAN* [46] to improve the security of vehicles with time-triggered network protocols while satisfying security, utilization, and deadline constraints. We introduce a novel methodology to derive security requirements for messages in the system based on ISO 26266 [14] and derive a metric called aggregate security value (ASV) to quantify security. Our proposed *SEDAN* framework utilizes the derived security requirements and the messages in the system to perform a joint exploration for the synthesis of message schedules and security characteristics (e.g., key sizes) using a metaheuristic called greedy randomized adaptive search procedure (GRASP) and ensures that the ECU utilization does not exceed 100%. We present a runtime security framework that employs a session key-based approach using a station-to-station (STS) key agreement protocol and uses an authenticated encryption and decryption scheme to simultaneously provide confidentiality and authenticity to the message data. Furthermore, we present a study of several cryptographic algorithms such as Advanced Encryption Standard (AES), Rivest–Shamir–Adleman (RSA), and elliptic curve cryptography (ECC) in the context of automotive systems.

In chapter 5, we propose a deep learning based lightweight IDS framework called *INDRA* [47] that utilizes a gated recurrent unit (GRU) based recurrent autoencoder network to learn the latent representation of normal operating system behavior at design time. We devise a metric called intrusion score (IS), which is used to quantify the deviation from the learned normal system behavior at runtime, and discuss the details related to the threshold based intrusion detection approach. We also present a comprehensive analysis towards the selection of the threshold for the IS metric and present the hardware implementation on a real-world automotive ECU. Lastly, the details related to implementation overhead and scalability of the proposed *INDRA* framework are discussed in detail.

In chapter 6, we present a deep learning based anomaly detection framework (used as an IDS) called *LATTE* [48] that utilizes a stacked long short-term memory (LSTM) network based model that integrates a novel self-attention mechanism to predict message values in future time steps. We discuss the design of a one-class support vector machine (OCSVM) based detector model that works with the LSTM predictor model to detect cyber-attacks at runtime. We discuss the proposed modifications to the existing vehicle communication controllers to realize the proposed IDS on real-world automotive ECU. We present a comprehensive analysis on the selection of deviation measures that quantify the deviation from normal operating conditions, which are used by the OCSVM to detect cyber-attacks. Lastly, we present a detailed study on inference time, number of model parameters, and memory footprint based on the hardware implementation of the proposed framework on a real-world ECU.

In chapter 7, we present a temporal convolutional neural network (TCN) with a neural attention mechanism (together called TCNA) based anomaly detection framework called *TENET* [49] to actively monitor in-vehicle networks and detect cyber-attacks. We present various details

related to the abilities of TCNA to learn very long-term temporal dependencies between messages to efficiently characterize the normal operating behavior of the system. We introduce a metric called divergence score (DS) to quantify the deviation from the expected behavior and discuss details related to the decision tree classifier that was implemented to detect cyber-attacks at runtime. Lastly, we present a detailed analysis of memory footprint, number of trainable model parameters, and inference time of the proposed technique by implementing it on a real-world automotive ECU.

Chapter 8 concludes this dissertation. We present a comprehensive summary of our research as a part of this dissertation and also make recommendations for future work directions.

2. JAMS-SG: A FRAMEWORK FOR JITTER-AWARE MESSAGE SCHEDULING FOR TIME-TRIGGERED AUTOMOTIVE NETWORKS

Modern automobiles have several processing elements called Electronic Control Units (ECUs) that control different functionalities in a vehicle. ECUs run various types of automotive applications such as anti-lock braking control, cruise control, etc. Most of these automotive applications have strict timing (deadline) and latency constraints and thus they are classified as hard real-time applications [50]. The ECUs on which these applications execute are distributed across the vehicle and communicate with each other by exchanging messages. These messages can be classified as either time-triggered or event-triggered. Time-triggered messages are periodically generated messages originating from safety-critical software applications. In contrast, event-triggered messages are generated asynchronously when a specific event occurs, typically by low priority (e.g., maintenance) applications.

The diverse nature of messages in automotive systems requires unique communication protocols to support them. The Controller Area Network (CAN) protocol is one of the most popular and widely used communication protocols in automotive systems. CAN is a serial protocol which supports a maximum payload of 8 bytes and a typical transmission rate of up to 1 Mbps [15]. Some of the key features of CAN include low cost, a lightweight protocol, broadcast communication capabilities, and support for message priorities and error handling [51], [52]. CAN allows transmission of both time-triggered and event-triggered messages in an event-triggered manner, which is a function of its arbitration scheme. In a CAN based system, when multiple ECUs are trying to transmit messages on the bus at the same time, the message with the lowest CAN message ID gets access to the bus first, while all the other messages (time-triggered or event-triggered) wait

till the next arbitration event. Some of the other commonly used in-vehicle network protocols in today's vehicles include Local Interconnect Network (LIN), FlexRay, Media Oriented Systems Transport (MOST), and Ethernet [54].

The onset of state-of-the-art x-by-wire automotive applications (throttle-by-wire, steer-by-wire, etc.) has led to an increase in the complexity of automotive applications [2]. This has resulted in a demand for an efficient, reliable, and deterministic in-vehicle communication protocol to satisfy the timing constraints of all time-critical applications, while still being able to meet the high bandwidth requirements of these applications [42]. This goal is difficult to achieve using the industry de facto standard CAN bus, as it suffers from limited bandwidth (with a maximum transmission rate of only 1 Mbps, which is insufficient for many high bandwidth vehicular applications such as pedestrian detection, lane tracking, etc.) and lack of time determinism. Moreover, the event-triggered nature of the CAN bus makes it harder to adapt for high bandwidth demanding state-of-the-art safety and time-critical applications. As a result, both automakers and researchers in academia have been actively exploring other automotive communication solutions to achieve these goals. FlexRay is an alternative communication protocol that overcomes the above-mentioned limitations of the CAN protocol and offers added flexibility, higher data rates (at least 10× higher compared to CAN [20]), better time determinism, and support for both time-triggered and event-triggered transmissions. As a result, it is deployed in many state-of-the-art vehicles that implement demanding applications such as Audi A4's electronic stabilization control [55], Volvo XC 90's VDDM [56], etc.

One of the more important challenges with time-triggered transmissions is jitter, which is the stochastic delay-induced deviation from the actual periodicity of a message. At a high level, jitter can be classified into two types: (i) bounded (deterministic) jitter and (ii) unbounded

(random) jitter. The former is a periodic variation that is caused by the systematic occurrences of certain events in the system (such as queuing of messages, clock jitter, etc.) whose peak-to-peak value is bounded. Moreover, due to its deterministic nature, this type of jitter can be easily predicted based on observations. The latter is an unpredictable timing noise whose peak-to-peak value is not bounded, e.g., due to thermal noise in an electrical circuit (resulting in delayed task executions or message transmissions), external disturbances, etc. Unlike deterministic jitter, such random jitter is hard to predict based on system design and simple observations.

In this work, we propose to address the problem of random jitter in automotive systems as it can have a significant impact on the performance and safety of the system. We focus on one of the most important sources of random jitter: delay in the execution of tasks in ECUs. Failure to effectively handle jitter-induced messages from such tasks can severely affect system performance and also be catastrophic in some cases (e.g., when the airbag deployment signal from the impact sensor to the inflation module gets delayed due to jitter). We conjecture that jitter handling must be incorporated from the early design phase, while designing schedules for time-critical automotive applications. At the same time, unexpected jitter variations at runtime must also be carefully handled. There is thus a critical need for an effective jitter handling approach that can be applied when designing and enforcing the schedules for time-critical automotive applications.

In this chapter, we propose a novel message scheduling framework called JAMS-SG to handle both jitter affected time-triggered messages and high priority event-triggered messages in an automotive communication system. Our framework is demonstrated for the FlexRay protocol but it can be extended to other time-triggered protocols relatively easily. JAMS-SG combines design time schedule optimization with a runtime jitter handling mechanism, to minimize the impact of jitter in the FlexRay network.

Our novel contributions in this chapter can be summarized as follows:

- We develop a hybrid heuristic that performs jitter-aware frame packing (packing of different signals from an ECU into messages) for the FlexRay protocol;
- We develop a heuristic approach for the synthesis of jitter-aware design time schedules for the FlexRay-based communication system;
- We introduce a runtime scheduler that opportunistically packs the jitter affected time-triggered and high priority event-triggered messages in the FlexRay static segment slots;
- We compare our JAMS-SG framework with the best-known prior works in the area, and demonstrate its effectiveness and scalability.

The rest of this chapter is organized as follows. Section 2.1 presents an overview of the FlexRay protocol. Related work that addresses message scheduling in FlexRay is discussed in Section 2.2. In Section 2.3, we define the problem statement by introducing the system and jitter models, heuristics used in this work, and important definitions and assumptions. In Section 2.4, we explain our proposed JAMS-SG framework in detail. Section 2.5 discusses the experimental setup and the results from our simulation-based analysis. We conclude with a summary of our work in Section 2.6.

2.1. FLEXRAY OVERVIEW

FlexRay is an in-vehicle communication protocol designed for x-by-wire automotive applications. It supports both time-triggered and event-triggered transmissions. The structure of the FlexRay protocol is shown in Figure 11. According to the FlexRay specification [16], a communication cycle is one complete instance of a communication structure that repeats

periodically, e.g., every 5 milliseconds. Each communication cycle consists of a mandatory static segment, an optional dynamic segment, an optional symbol window, and a mandatory network idle time block.

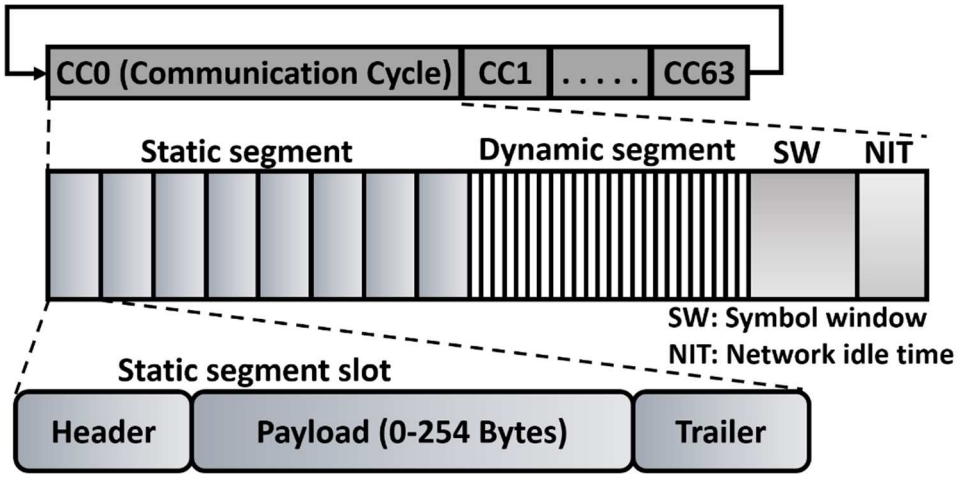


Figure 11 Structure of FlexRay protocol.

The static segment in FlexRay consists of multiple equal-sized slots called static segment slots that are used to transmit time-triggered and critical messages. The static segment enforces a Time Division Multiple Access (TDMA) media access scheme for the transmission of time-triggered messages, which results in a repetition of the schedule periodically. In this TDMA scheme, each ECU is assigned one or more static segment slots and cycle numbers during which its messages can be transmitted on the FlexRay bus, thereby guaranteeing time determinism for message delivery. Each static segment slot incorporates one FlexRay frame, which has three segments: header, payload, and trailer. The *header* segment is 5-bytes long and consists of status bits, frame ID (FID), payload length, header cyclic redundancy check (CRC), and cycle count. The *payload* segment consists of actual data that has to be transmitted and is up to 127 words (254 bytes) long. The *trailer* segment consists of three 8-bit CRC fields to detect errors.

The dynamic segment consists of variable-sized slots called dynamic segment slots that are used to transmit event-triggered and low priority messages. A dynamic segment slot consists of a variable number of minislots (Figure 11), where each minislot is one microtick (usually 1 μs) long. The dynamic segment enforces a Flexible Time Division Multiple Access (FTDMA) media access scheme where ECUs are assigned minislots according to their priorities. If an ECU is selected to transmit a message, then it is assigned the required number of minislots depending on the size of the FlexRay frame, and hence the length of a dynamic segment slot can vary in the dynamic segment (Figure 11). During a transmission, all the other ECUs have to wait until the one that is transmitting finishes. If an ECU chooses not to transmit, then that ECU is assigned only one minislot and the next ECU is assigned the subsequent minislot. The symbol window (SW) is used for network maintenance and signaling for the starting of the communication cycle, while the network idle time (NIT) is used to maintain synchronization between ECUs (Figure 11).

Every automotive ECU has two major components: a host processor and a communication controller. The host processor is responsible for running automotive applications. The communication controller acts as the interface between the host processor and the communication network. The communication controller specifically in a FlexRay ECU has two sub-components: a communication host interface (CHI) and a protocol engine (PE). The CHI handles the message data generated by the host processor and sends the qualified FlexRay frames to the PE, which transmits the frames on a physical FlexRay bus (Figure 12). Each frame has a unique frame ID (FID) that is equal to the slot ID in which the frame is transmitted [16]. A FlexRay frame is considered to be "qualified" when the message data is available at the CHI before the beginning of the allocated static segment slot. Otherwise, a special frame called NULL frame is sent (by

setting a bit in the header segment of the FlexRay frame and setting all the data bytes in the payload to zero).

Jitter is one of the major reasons for the delay in the availability of message data at the CHI. Hence in this work, we focus on a novel frame packing and scheduling framework to overcome the delays and performance losses due to jitter in time-critical automotive systems.

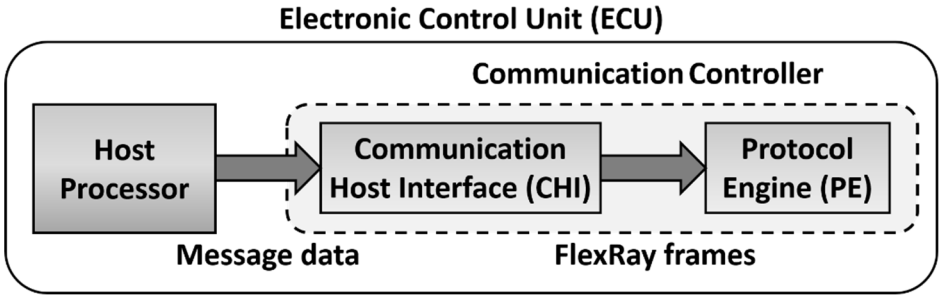


Figure 12 Message generation and transmission in ECUs.

2.2. RELATED WORK

Prior work on message scheduling for the FlexRay protocol can be categorized into two groups: *(i)* time-triggered and *(ii)* event triggered message scheduling. The goal of these works is to synthesize message schedules by optimizing parameters such as bandwidth, number of allocated static segment slots, response time, end-to-end latency, etc., under strict timing constraints.

A common and important step prior to message scheduling is frame packing. Frame packing refers to the process of packing multiple signals into messages, to maximize bandwidth utilization on the bus [53]. The authors in [20] proposed an Integer Linear Programming (ILP) formulation to solve the frame packing problem which requires multiple iterations with ILP to find the optimal solution. A Constraint Logic Programming (CLP) formulation and heuristic were presented for

reliability-aware frame packing in [57], which may require multiple re-transmissions of the packed frames to meet reliability requirements. In [58], the frame packing problem is treated as a one-dimensional allocation problem and an ILP formulation and a heuristic approach were proposed. In [59], a genetic algorithm based frame packing approach was proposed for CAN-FD systems. In [44], the authors proposed a fast greedy heuristic based frame packing approach. The above-mentioned techniques either focus on optimizing bandwidth utilization or minimizing the time taken to generate a frame packing solution. *However, none of the above-mentioned works focus on generating a jitter-aware frame packing solution. Our proposed frame packing technique in this chapter uses a hybrid heuristic approach to generate a near-optimal set of messages that together make the system more resilient to jitter-induced uncertainties*.

In the case of many real-time systems, especially automotive applications, most of the parameters such as period, worst-case execution time, deadline, etc., are known at design time. This facilitates the synthesis of highly optimized design-time schedules that are deployed during runtime to minimize the unpredictability in the system. Many works have addressed the issue of design-time scheduling of the static segment of FlexRay. One of the main objectives in these works is to minimize the number of static segment slots allocated to ensure future extensibility of the system while maximizing bandwidth utilization. In [60], an ILP based approach is proposed to minimize the number of allocated static segment slots by considering task and message scheduling. This work was later extended in [62] by including support for multiple real-time operating systems and using ILP reduction techniques. In [61], the message scheduling problem is transformed into a two dimensional bin-packing problem and an ILP formulation and a heuristic approach were proposed for minimizing the number of allocated slots. In [64], [65] the authors proposed a CLP and ILP formulation respectively for jointly solving the problem of task and message scheduling

in FlexRay systems. A set of algorithms was proposed in [67] to enable scheduling of event-triggered messages in time-triggered communication slots using a virtual communication layer. A few other works solve the same problem with heuristics and variants of ILP and CLP [57], [63], [66], [68], [69]. More recent works such as [75] and [76] combine schedulability analysis and control theory, and were able to achieve fewer FlexRay static segment slots compared to many of the above-mentioned prior works. Additionally, there are works that focus on scheduling time-triggered systems using other network protocols [70], [71], [72]. *However, the above-mentioned works focus on developing scheduling algorithms without incorporating the idea of jitter which makes them unreliable for use in real-time scenarios where jitter can significantly impact scheduling decisions*.

Jitter in FlexRay based systems has largely been ignored and there is limited literature on the topic. The authors in [20] proposed a jitter minimization technique using an ILP formulation. In [21], the frequency of message transmission is increased for the messages that are likely to be affected by jitter, to minimize message response time. However, in both [20] and [21], it is assumed that the jitter value and number of messages that are affected by jitter are known at design time, which is unlikely in real-world scenarios. The authors in [44] proposed a jitter-aware message scheduling technique called JAMS that uses both design time and runtime schedulers to opportunistically pack jitter-affected messages in the system. However, the non-jitter-aware frame packing in [44] results in sub-optimal packing of signals into messages leading to increased message response times in the presence of jitter. Moreover, a simple jitter model is considered in [44] that makes the evaluation process less efficient. In [73] an iterative design time scheduling algorithm was proposed to minimize the impact of jitter on mixed-criticality time-triggered messages. However, [73] does not effectively handle the unpredictabilities due to random jitter at

runtime. As random jitter can affect any message in the system, there is a need for a jitter handling mechanism that can handle jitter more comprehensively at the signal and message level, at both design time and runtime. *In this work, we introduce a realistic jitter model and propose a holistic framework that achieves a jitter-aware frame packing and combines the design time schedule optimization with an improvised runtime jitter handling, to minimize the impact of jitter in FlexRay based systems. We extensively evaluate the proposed JAMS-SG framework to demonstrate its effectiveness and scalability*.

2.3. PROBLEM DEFINITION

2.3.1. SYSTEM MODEL

In this work, we consider a general automotive scenario with multiple ECUs that run different time-critical automotive applications and are connected using a FlexRay bus architecture. Executing an application may result in the generation of signal data at an ECU, which may be required for another application running at a different ECU. A signal can be any raw data value or control pulse. These signals are packed into messages and transmitted as FlexRay frames on the bus. As discussed earlier, there are two types of applications in a typical automotive system: *(i)* time-triggered, and *(ii)* event-triggered. Every ECU or node in the system is capable of transmitting both types of messages (henceforth the terms ECU and node are used interchangeably). Time-triggered messages are transmitted in the static segment slots of the FlexRay while the dynamic segment slots are used for transmitting event-triggered messages. However, in this work, we facilitate the transmission of high priority event-triggered messages in the static segment of the FlexRay (details in Section 2.4.2). Hence, in this work, we focus on the challenging problem of scheduling time-triggered messages and high priority event-triggered messages in the static

segment of FlexRay. We ignore the scheduling of low-priority (and typically low-frequency) event-triggered messages in the FlexRay dynamic segment, which is a much simpler problem and has a negligible impact on vehicle safety.

2.3.2. JITTER MODEL

As discussed earlier, jitter is defined as the delay-induced deviation from the actual periodicity of the message, and there are two types of jitter: *(i)* bounded or deterministic jitter and, *(ii)* un-bounded or random jitter. In this work, we mainly focus on random jitter, as it is hard to predict and can have a significant impact on the performance and safety of the system. Our goal is to mitigate the effect of random jitter on task execution and message transmission delays. We assume that both time-triggered and event-triggered messages are susceptible to such random jitter. However, the impact of random jitter on low priority event-triggered messages is not considered, as such messages have minimal impact on the safety and performance of the system.

Random jitter is also known as Gaussian jitter, because it follows a normal distribution due to the central limit theorem [77]. In this work, we devise a specific jitter model for each signal in the system based on the signal priority and signal period. Signals with a period of less than or equal to 40 ms are treated as high priority signals and other signals are considered as low priority signals. Additionally, the mean jitter associated with each signal is modeled as (*signal_period/5*) for high priority signals and (*signal_period/4*) for low priority signals. These mean jitter values can be tuned based on the designer requirements and system specifications. A similar but more simplistic model is presented in [74] which does not consider the mixed criticality nature of the automotive applications. In a normal distribution representing jitter values (on x-axis) and number of occurrences (on y-axis), the jitter values in the tail region far from the mean occur less frequently

than the values close to the mean. Hence, in this work, we mainly focus on mitigating the effect of mean jitter value associated with each signal on the system performance (i.e., ensuring there are no missed deadlines).

2.3.3. HYBRID SA+GRASP HEURISTIC

A hybrid heuristic is a combination of two or more heuristics. The goal of any hybrid heuristic is to combine the advantages of individual heuristics while minimizing each other's disadvantages. In this work, we propose a hybrid heuristic by combining simulated annealing (SA) and greedy randomized adaptive search procedure (GRASP). Similar attempts were made in the past to combine SA and GRASP and build a hybrid heuristic in [78] and [79]. However, these efforts do not focus on the automotive application problem domain, and they also do not optimize the search space or perform tuning of hyperparameters. Our proposed SA+GRASP hybrid heuristic aims to improve the design space search capability, solution optimality, and computation speed. Moreover, as our proposed framework uses the baseline model from JAMS [44], and the proposed SA + GRASP hybrid heuristic, we name our framework JAMS-SG where S and G represents SA and GRASP, respectively.

2.3.3.1.SIMULATED ANNEALING

Simulated Annealing (SA) is a heuristic that is inspired from the annealing technique in metallurgy. It models the physical process of heating and controlled cooling of a material to strengthen and reduce defects. SA is a used to approximate the global optimum in a very large discrete solution space.

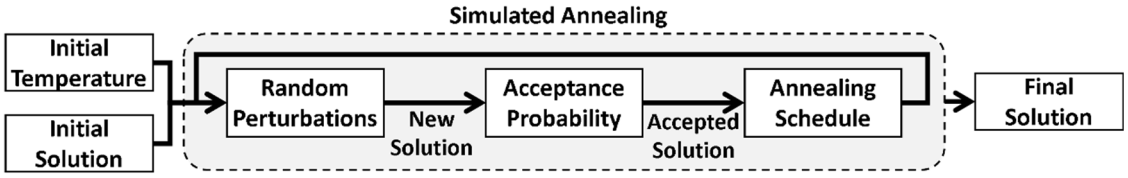


Figure 13 Various steps involved in SA.

There are five phases in any SA problem formulation (shown in Figure 13): *(i) initial solution*, *(ii) initial temperature*, *(iii) random perturbations*, *(iv) acceptance probability*, and *(v) annealing schedule*. The SA begins by taking the initial solution and initial temperature as the inputs and tries to iteratively achieve a better solution at the end of every iteration. The temperature is progressively decreased from an initial positive value until a stopping condition is met (e.g., until temperature > 0). Each iteration begins by constructing a new solution after making random perturbations to the current solution. If the objective function value of the new solution is better than the previous solution, then the new solution is accepted. Otherwise, the new solution is accepted based on the acceptance probability value calculated using an acceptance probability function. The acceptance probability function takes the difference between the objective function values of the new and previous solutions and the current temperature of the system as the inputs, and computes the acceptance probability value. The new solution is accepted when the acceptance probability value is greater than a randomly generated number between 0 and 1. Otherwise, the new solution is discarded. SA tries to accept even a relatively poor solution in the initial stages when the system temperature is high. As the SA progresses, i.e., when the system temperature is lower, SA will favor accepting only those new solutions that are very close to the new solution. When the temperature reaches *zero*, SA behaves like a pure greedy algorithm. Lastly, at the end of each iteration, the temperature of the system is updated using an annealing schedule. The annealing schedule is responsible for the controlled cooling of the system.

SA is quite versatile and can deal with highly non-linear solution spaces. It is also good at dealing with arbitrary systems and cost functions while statistically guarantying an approximate global optimum. However, SA can take a very long time to converge to a good solution and the optimality of the solution is heavily dependent on the chosen hyperparameters (initial temperature, annealing schedule, and acceptance function).

2.3.3.2.GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

The greedy randomized adaptive search procedure (GRASP) is a multi-start metaheuristic. The main objective of GRASP is to repeatedly sample stochastically greedy solutions, and then use an adaptive local search to refine them to a local optimum. At the end, the best of the local optima is chosen as the final solution.

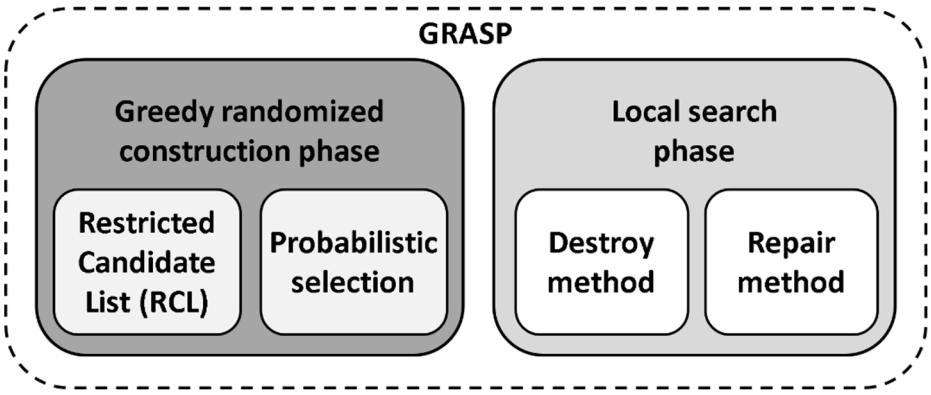


Figure 14 Various steps involved in GRASP.

Figure 14 shows the two phases in GRASP: *(i) the greedy randomized construction phase* that tries to build a feasible solution and *(ii) the local search phase* that tries to explore a defined neighborhood for a local optimum. The best of the local optima is chosen as the final solution at the end. The two important aspects of the greedy randomized construction phase are its *greedy aspect* and *probabilistic aspect*. The greedy aspect involves generating a Restricted Candidate List

(RCL), which consists of the best elements that will improve the partial solution (solution within the greedy randomized construction phase). The probabilistic aspect is the random selection of an element from the RCL, to be incorporated into the partial solution. However, the solutions generated during the greedy randomized construction phase are not necessarily optimal. Hence, the local search phase tries to improve the constructed solution by iteratively using destroy and repair mechanisms, which are used to perturb the current solution and reconstruct a new solution, respectively. They help in searching for the local optimum within a defined neighborhood. Lastly, when an improved solution is found, then the best solution is updated.

GRASP is simple to construct and can be used for large optimization problems. However, as GRASP depends on the greedy algorithm to evaluate the quality of the solution, it can get stuck at a local optima. Moreover, it might restart at the same solution multiple times leading to rediscovering of the same local solution.

2.3.3.3.HYBRID HEURISTIC FORMULATION

To overcome the individual limitations of SA and GRASP, we propose a hybrid heuristic that combines both of them. The proposed hybrid heuristic uses SA to explore the large solution space and GRASP to find an improved local solution within a smaller neighborhood around the solution obtained from SA. In particular, the greedy construction phase of GRASP is used to make perturbations in the SA and the local search phase is used to explore the neighborhood with a goal of finding a better solution. The details of our hybrid heuristic are discussed in detail in Section 2.4.1.

2.3.4. INPUTS AND DEFINITIONS

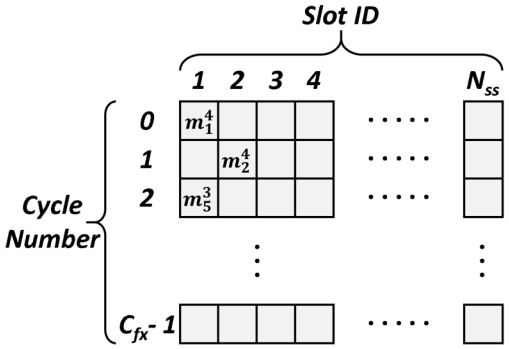
We consider an automotive system with the following inputs:

- $\mathbb{N}$ represents the set of nodes, where $\mathbb{N}$ = {1, 2, 3, …, N};
- For each node $n \in \mathbb{N}$, $S^n = \{s_1^n, s_2^n, \ldots, s_{K_n}^n\}$ denotes the set of signals transmitted from that node and K_n represents the maximum number of signals in node n;
- Every signal $s_i^n \in S^n$, (i = 1, 2 …, K_n) is characterized by the tuple $\{\bar{p}_i^n, \bar{d}_i^n, \bar{b}_i^n, \bar{\gamma}_i^n\}$, where $\bar{p}_i^n$, $\bar{d}_i^n$, $\bar{b}_i^n$ and $\bar{\gamma}_i^n$ denote the period, deadline, data size (in bytes), and mean jitter of the signal s_i^n respectively;
- After frame packing, every node maintains a set of messages $M_n = \{m_1^n, m_2^n, \cdots, m_{R_n}^n\}$ in which every message $m_j^n \in M_n$, (j = 1, 2, …, R_n) (where R_n represents the maximum number of messages in node n) is characterized by the tuple $\{a_j^n, p_j^n, d_j^n, b_j^n, \mu_j^n\}$, where a_j^n, p_j^n, d_j^n, b_j^n and μ_j^n denote the arrival time, period, deadline, data size (in bytes), and mean jitter of the message m_j^n respectively.

We assume the following <u>definitions</u>:

- *Slot number or Slot identifier (slot ID)*: A number used to identify a specific slot within a communication cycle;
- *Cycle number*: A number used to identify a specific communication cycle in the schedule;
- To transmit a message m_j^n on the FlexRay bus, it needs to be allocated a slot ID $sl \in$ {1, 2, …, N_{ss}} and a cycle number $bc \in$ {0, 1, …, C_{fx}} where N_{ss} and C_{fx} are the total number of static segment slots in a cycle and the total number of cycles, respectively. This is referred to as *message-to-slot assignment*;
- If a message m_j^n is assigned to a particular slot and a cycle then the source node n of that message is allocated ownership of that slot. This is known as *node-to-slot assignment*.

All of the above-mentioned definitions are illustrated in Figure 15 with an example FlexRay 3.0.1 schedule. In the example, message (m_1) is allocated a slot ID = 1 and cycle number = 0 (*message-to-slot assignment*). This implies that the source node (ECU_4) sending the message (m_1) is allocated ownership of the slot (*node-to-slot assignment*).



(slot, cycle)	Ownership	Message
(1 , 0)	ECU_4	m_1
(2 , 1)	ECU_4	m_2
(1, 2)	ECU_3	m_5

Figure 15 Illustration of an example FlexRay 3.0.1 schedule (on the left) with slot IDs and cycle numbers; and the table (on the right) showing the message-to-slot and node-to-slot allocation.

Problem Objective: For the above inputs, the goal of our work is to satisfy deadline constraints for time-triggered and high priority event-triggered messages sent over the FlexRay protocol, by enabling jitter resilience during communication, which includes: *(i)* performing jitter-aware frame packing, and design time scheduling (message-to-slot assignment, node-to-slot assignment) for the time-triggered messages without violating timing constraints, and *(ii)* minimizing the effect of jitter on time-triggered messages and high priority event-triggered messages at runtime.

2.4. JAMS-SG FRAMEWORK OVERVIEW

We propose the JAMS-SG framework to enable jitter-aware scheduling of time-triggered messages and collocation of high priority event-triggered messages in the static segment of a FlexRay-based automotive system. An overview of the proposed framework is shown in Figure

16. At a high level, the steps in JAMS-SG are categorized into design-time and runtime steps. At design time, JAMS-SG uses the proposed hybrid heuristic approach that combines Simulated Annealing (SA) and Greedy Randomized Adaptive Search Procedure (GRASP) to achieve jitter-aware frame packing of time-triggered messages and a feasible design-time schedule. At runtime, JAMS-SG facilitates the handling of both jitter affected time-triggered and high-priority event-triggered messages using a multi-level feedback queue (MLFQ). The output of MLFQ and the design time schedule are given as the inputs to a runtime scheduler that opportunistically packs these jitter affected messages into the already allocated FlexRay slots based on the available slack. Each of these steps is discussed in detail in the following subsections.

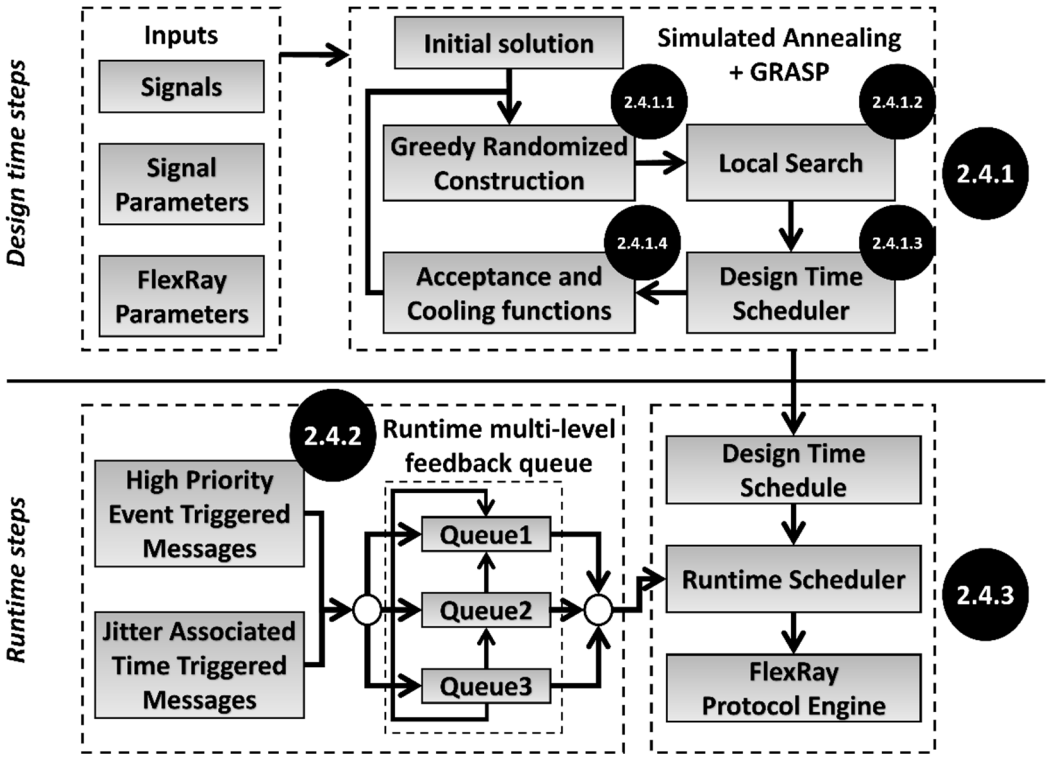


Figure 16 Overview of JAMS-SG framework.

2.4.1. JITTER-AWARE DESIGN TIME FRAME PACKING

Frame packing refers to packing of multiple periodic signals in a node into messages. This step is important to maximize bandwidth utilization, which not only improves system performance but also enhances the extensibility of the system by utilizing fewer slots than without frame packing. However, frame packing with a goal of just maximizing the bandwidth utilization can result in sub-optimal results at run-time. For instance, if one of the signals packed in the message is not available before the start of the message's allocated static segment slot due to jitter-induced delays, the entire message has to be delayed and the CHI has to send out a NULL frame because of the lack of availability of complete message data. Once all of the signals in the message are available, the message can be transmitted in the next allocated static segment slot. This delayed transfer will result in increased response time of the messages and can potentially lead to missed deadlines, which can be catastrophic for safety-critical application. Thus it is important to have a jitter-aware frame packing technique that co-optimizes bandwidth utilization and mitigation of the impact of jitter.

In this work, we define the following four necessary conditions that govern how signals can be packed into the same message. The first is called the source node condition and is expressed as:

$$m_k^n = \{s_i^{n_1}, s_j^{n_2}\} \quad iff\ src(s_i^{n_1}) == src(s_j^{n_2}) \quad i,j \in [1, K_n]\ \&\ i \neq j \tag{1}$$

This condition states that if two signals $(s_i^{n_1}, s_j^{n_2})$ are packed into the same message (m_k^n) they should belong to the same source node (*src*() returns the source node of the signal) (as shown in equation (1). This is because, according to the FlexRay protocol specification [16], any slot in a given FlexRay cycle can be assigned to at most one node, which restricts the packing of signals

from different nodes into the same message. This makes the frame packing problem solvable independently for different nodes.

$$m_k^n = \{s_i^n, s_j^n\} \quad iff \quad \overline{p}_i^n == \overline{p}_j^n \;\; i,j \in [1, K_n] \;\; \& \, i \neq j \tag{2}$$

The periodicity condition in equation (2) states that only the signals with the same periods should be packed into the same message. This is done to minimize the re-transmissions of the message frames, which in turn minimizes the number of allocated static segment slots. For example, if two signals with periods 5ms and 15 ms are packed into the same message, the resulting message with period 5ms retransmits the signal with 15ms period twice and results in inefficient bandwidth utilization.

$$\sum_{i \in sigIDs(m_k^n)} \overline{b}_i^n \;\leq\; B_{slot} \tag{3}$$

The payload condition in equation (3) states that the sum of all signal sizes packed in a message (sigIDs() returns the set of IDs of the signals packed in the message) should not exceed the maximum payload (B_{slot}) of the FlexRay static segment slot.

$$ResponseTime(m_k^n) \leq \; d_k^n \qquad \forall \, n, k = 1,2, \ldots, R_n \tag{4}$$

Lastly, the deadline condition in equation (4) states that the set of messages generated from frame packing should result in a feasible schedule, i.e., the response time (end-to-end latency) for all the messages should not exceed their deadline requirements. In addition, if there are any specific timing requirements associated with any signal (such as latency, worst-case response time, etc.), they become additional constraints to the problem. The timing constraints are further discussed in Section 2.4.1.3.

Given the above-mentioned constraints, the goal of jitter-aware frame packing is to maximize the laxity of each resulting message while minimizing the total number of message frames. In other words, we prioritize packing of signals with similar jitter profile into the same messages. This is because of the reason discussed next.

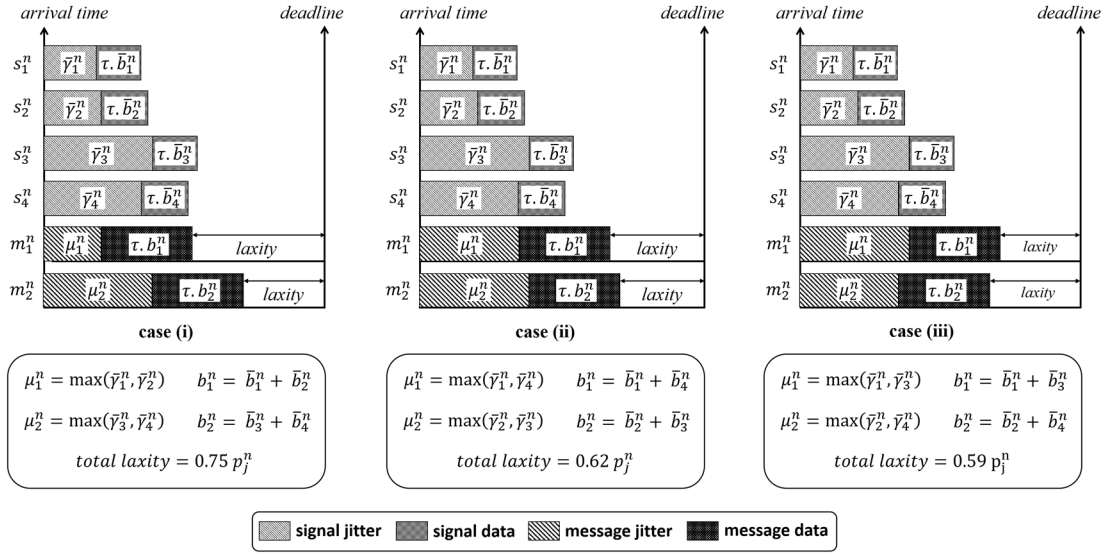


Figure 17 Motivation for objective function selection.

Consider an example scenario consisting of three ways of packing four different signals in a node as shown in Figure 17. For simplicity, we considered the packing of a maximum of two signals per message in this example. However, this constraint is not enforced anywhere else in the framework. In case (i), signals with similar jitter profile ($\bar{\gamma}_i^n$) are packed together ((s_1^n, s_2^n) and (s_3^n, s_4^n)) resulting in two messages (m_1^n, m_2^n) with the effective mean jitter value (μ_k^n) close to the individual signal jitter values. This results in maximizing the laxity of the messages, which is important as it provides more opportunities to pack and transmit other jitter-affected messages and also to better cope with unpredictable runtime jitter. In contrast, in cases (ii) and (iii), the signals with very different jitter profile are packed together resulting in lower laxity values. This makes

the messages less resilient to jitter compared to the one in case (i). Note that to avoid the degenerate case of packing one signal per message (which would maximize laxity but would lead to very inefficient bus utilization), we formulated a weighted objective function that achieves jitter-aware frame packing while also effectively minimizing the total number of messages in the system.

The objective function is a weighted harmonic sum of average laxity of the messages in each node and is used to achieve jitter-aware frame packing, as is shown in equation (5). Laxity of any message is defined as the difference between the message deadline and sum of mean jitter value of the message and time required to transmit the message payload, as shown in equation (6). The mean jitter value of a message (as shown in equation (7)) is the maximum mean jitter value of all signals packed in that message. Our proposed hybrid heuristic that combines SA and GRASP aims to minimize the objective function (in equation (5); with equations (6) and (7) further describing some of the parameters of this function) while satisfying all of the constraints mentioned above. The parameters R_n, b_i^n and τ are the number of messages in the node n, data size of signal i in node n and time taken to transmit 1 byte of data on FlexRay bus, respectively.

$$minimize \ \sum_{n=1}^{N} \left(w * \frac{R_n}{\sum_{j=1}^{R_n} laxity_j^n} \right) \tag{5}$$

$$laxity_j^n \ = \ d_j^n - \left(\mu_j^n + \ \tau . \sum_{i \in sigIDs\left(m_j^n\right)} \overline{b}_i^n \right) \tag{6}$$

$$\mu_j^n = \max\left(\left\{ \bar{\gamma}_i^n \ | \ i \ \in sigIDs\left(m_j^n\right) \right\} \right) \tag{7}$$

We use SA for exploring the overall solution space and uses GRASP for creating and refining new solutions at every iteration. The solution here refers to the signal to message packing for all the nodes in the system. To the best of our knowledge, this is the first work in this area that

attempts to achieve a jitter-aware frame packing. Algorithm 1 shows the pseudo-code of the hybrid heuristic.

Algorithm 1: SA + GRASP based frame packing

Inputs: Set of nodes ($\mathcal{N}$), Set of time-triggered signals in each node (S^n), GRASP parameters (α, β), temperature (T), cooling rate (C_r)

```
1:  Initialize: cur_sol, prev_sol, best_sol ← initial_solution(S^n)
2:  for each iteration until max_iterations do
3:  |   δ = random_int(1, N)
4:  |   λ = random_selection(N, δ)
5:  |   gr_sol ← greedy_randomized_construction(α, λ, cur_sol)
6:  |   ls_sol ← local_search(β, N, gr_sol)
7:  |   cur_sol ← choose_solution(gr_sol, ls_sol)
8:  |   if feasiblity(cur_sol) then
9:  |   |   P_acc = acceptance_probability(cur_sol, prev_sol, T)
10: |   |   if P_acc > random(0,1) then
11: |   |   |   prev_sol ← cur_sol
12: |   |   |   if Φ(cur_sol) < Φ(best_sol) then
13: |   |   |   |   best_sol ← cur_sol
14: |   |   |   end if
15: |   |   end if
16: |   end if
17: |   T *= C_r
18: end for
19: output_sol ← best_sol
20: output_schedule ← design_time_schedule(output_sol)
```

Output: Set of messages in each node and a feasible design time schedule

The inputs to the Algorithm 1 are: a set of nodes ($\mathcal{N}$), set of time-triggered signals for each node (S^n), GRASP control parameters (α – which is the RCL threshold discussed later in Section 2.4.1.1, β – which is the destroy-repair threshold discussed later in Section 2.4.1.2), and the SA hyper-parameters: temperature (T) and cooling rate (C_r). The algorithm begins by initializing the current (*cur_sol*), previous (*prev_sol*), and best (*best_sol*) solutions with a one signal per message (one-to-one) frame packing (step **1**), which also acts as the initial solution for the SA. All solutions are data structures that have information about the signal to message packing for each node (solution← [M_1, M_2,…, M_N]). Each element in the list has a frame packing configuration for each

node. At the beginning of each iteration, the algorithm selects a random number (δ) of nodes (λ) to be perturbed as shown in steps **3**, **4**. A new solution (*gr_sol*) is constructed from the current solution (*cur_sol*) using the greedy randomized construction phase of GRASP (step **5**). In step **6**, this new solution (*gr_sol*) is given as the input to the local search phase of GRASP to look for a local op-timum solution (*ls_sol*) within the defined neighborhood. The better of the two solutions (*gr_sol*, *ls_sol*), i.e., the solution that results in a minimal objective function value is chosen as the current solution (*cur_sol*) in step 7. The feasibility of the chosen solution is checked using the *feasibility*() function (step **8**), which returns true when there are no missed deadlines for any message in the given solution; otherwise, it returns false. When a solution is feasible, the decision of accepting or discarding it is dependent on the probability of acceptance (P_{acc}). This value is computed using the *acceptance_probability*() function, which takes the current system temperature (T), and both current and previous solutions as the input (step **9**). If the current solution is accepted, then the previous solution is assigned the current solution and the best solution is also updated if the current solution has lower objective function value compared to the best solution (steps **10**-**16**). The function $\Phi()$ is used to compute the objective function value of the solution. Additionally, at the end of each iteration, the annealing schedule performs a controlled cool down of the system (step **17**). At the end of *max_iterations*, the best solution is chosen as the output solution (*output_sol*) and a design-time schedule is synthesized using it (steps **19**, **20**).

<u>*Note*</u>: The resulting output messages will have their period be the same as the signal period and their deadline will be equal to the lowest signal deadline packed in that message. In this work, we assume that all the messages have deadlines equal to their periods. The output of Algorithm 1 is a jitter-aware frame packing and a feasible design time schedule.

2.4.1.1. GREEDY RANDOMIZED CONSTRUCTION

In this subsection, we discuss the greedy randomized construction phase of GRASP that is used to perturb the solution in SA, in step **5** of Algorithm 1.

Algorithm 2: greedy_randomized_construction (α, λ, *cur_sol*)
Inputs: RCL threshold (α), perturbed nodes (λ), and *cur_sol*

```
1:  function greedy_construct(α, Ŝ^n, partial_sol)
2:      while Ŝ^n ≠ { } do
3:          s = random_selection(Ŝ^n, 1)
4:          Ω ← feasible_frame_ids(s, partial_sol)
5:          if Ω ≠ { } then
6:              C_fids ← cost(s, Ω)
7:              C_min = min(C_fids); C_max = max(C_fids)
8:              RCL ← {fid ∈ Ω | C_fids(fid) ≤ C_min + α*(C_max - C_min)}
9:              chosen_fid = random_select(RCL,1)
10:             assign_fid(s, chosen_fid, partial_sol);
11:         end if
12:         Remove s from Ŝ^n
13:     end while
14:     return partial_sol
15: end function
16: Initialize: greedy_randomized_sol ← cur_sol
17: for each node n in λ do
18:     ρ = random_int(1,length(S^n))
19:     Ŝ^n = random_selection(S^n, ρ)
20:     greedy_randomized_sol(n) ← greedy_construct(α, Ŝ^n, cur_sol(n))
21: end for
22: return greedy_randomized_sol
```

Output: greedy randomized constructed solution; ∀ nodes $n \in \lambda$

Algorithm 2 shows the pseudo code of the greedy randomized construction where the inputs are: RCL threshold (α; discussed in more detail below), set of nodes whose frame packing will be perturbed (λ), and the solution that needs to be perturbed (*cur_sol*). The algorithm begins by initializing the current solution (*cur_sol*) to a greedy randomized solution (*greedy_randomized_sol*) (step **16**). For each node n whose current frame packing needs to be perturbed, the algorithm selects a random number (ρ) of signals ($\hat{S}^n$) in that node and tries to

greedily construct a new solution using the *greedy_construct*() function, in steps **17-21**. The newly generated solution for the node *n* is updated in the *greedy_randomized_sol*(*n*) function (step **20**), and at the end, the final solution for all nodes is returned (step **22**).

The function *greedy_construct*() in steps **1-15** takes the RCL threshold (α), set of signals whose frame packing will be changed ($\hat{S}^n$), and the current frame packing (*partial_sol*) as the inputs and tries to assign the signal to a new message. The addition of signals to new messages happens in a greedy manner that would result in minimizing the value of the objective function. Until the set $\hat{S}^n$ is empty, in every iteration a signal is randomly chosen from ($\hat{S}^n$) (step **3**), and a list of feasible frames (Ω) to which the signal (*s*) can be packed into is generated (step **4**). For a frame ID (*fid*) to be feasible, it has to satisfy the conditions mentioned in equations (1) – (3). If there exist no feasible frame IDs for the signal (*s*), its frame packing configuration is left unchanged. Otherwise, the individual cost of adding that signal to each frame is computed using the *cost*() in step **6**. The minimum cost (C_{min}) and maximum cost (C_{max}) computed in step **7** are used in generating the restricted candidate list (*RCL*) (step **8**). The *RCL* consists of the feasible frame IDs whose associated cost of adding the signal is within the interval $[C_{min}, C_{min} + \alpha*(C_{max} - C_{min})]$. This is the greedy aspect of the algorithm. The quality of RCL depends on the RCL threshold ($0 \leq \alpha \leq 1$). The threshold (α) controls the amount of randomness and greediness in the algorithm. For instance, when $\alpha = 0$, the algorithm exhibits a pure greedy behavior and when $\alpha = 1$, the algorithm exhibits a purely random behavior. A random frame ID (*chosen_fid*) is selected from *RCL* in step **9**, and the signal *s* is assigned to that frame ID (step **10**). Furthermore, after an attempt to change the frame packing for signal *s*, it is removed from $\hat{S}^n$ (step **12**). When all the signals in $\hat{S}^n$ are explored, the function terminates and returns the perturbed solution (*partial_sol*) (step **14**).

2.4.1.2. LOCAL SEARCH

Local search is the second phase of the GRASP metaheuristic, invoked in step **6** of Algorithm 1. It iteratively explores the defined neighborhood around the greedy randomized constructed solution in the search for a local optimum. This is accomplished using *destroy and repair* mechanisms that try to randomly remove a part of the solution and reconstruct it. In this work, we define *neighborhood* as the set of solutions that are generated by randomly changing the frame packing of β signals. The parameter β is known as the destroy-repair threshold and it controls the amount of destroy and repair operations in each iteration. The pseudo-code for the local search is shown below in Algorithm 3.

Algorithm 3: local_search (β, N, *gr_sol*)

Inputs: Destroy-repair threshold (β), set of nodes (N), and greedy randomized constructed solution (*gr_sol*)

1: **Initialize**: *interm_sol*, *new_sol* ← *gr_sol*
2: **for** each *ls_iteration* **until** *max_ls_iterations* **do**
3: η = **random_selection**(N, *1*)
4: $\hat{S}_{ls}^{\eta}$ = **random_selection**(S^{η}, β)
5: *new_sol*(η) = **greedy_construct** (α, $\hat{S}_{ls}^{\eta}$, *interm_sol*(η))
6: **if** Φ(*new_sol*) < Φ(*interm_sol*) **then**
7: *interm_sol* ← *new_sol*
8: **end if**
9: **end for**
10: **return** *interm_sol*

Output: Local optimum with in the defined neighborhood- if there exists one; Otherwise, same solution as greedy_randomized_construction( ).

The local search in Algorithm 3 begins by taking the destroy-repair threshold (β), set of nodes (N) and greedy randomized constructed solution (*gr_sol*) as the inputs and then initializes the intermediate solution (*interm_sol*), and new solution (*new_sol*) with the greedy randomized constructed solution (in step **1**). The frame packing for β random signals ($\hat{S}_{ls}^{\eta}$) belonging to a random node (η) are chosen to be changed (destroy mechanism) in steps **3, 4**. A new solution

(*new_sol*) is reconstructed using the *greedy_construct*() function in step **5** (repair mechanism), and it is accepted if the new solution (*new_sol*) resulted in a smaller objective function value compared to the prior solution (*interm_solution*) (steps **6**-**8**). At the end of *max_ls_iterations*, the algorithm returns the final local search solution (*interm_sol*) in step **10**.

2.4.1.3. DESIGN TIME SCHEDULING

In this subsection, we present the jitter-aware design time scheduling heuristic that is invoked in step **20** of Algorithm 1. The heuristic takes the frame packing solution (i.e., set of all the time-triggered messages) in the system as input and generates a design-time schedule. The design time schedule consists of an assignment of slot ID and cycle numbers to messages (*message-to-slot allocation*) and their source nodes (*node-to-slot allocation*). In designing this schedule, we aim to allocate the messages as early as possible in the schedule to minimize the response time. Additionally, we try to minimize the number of allocated static segment slots for effective bandwidth utilization while meeting all the deadline constraints. Moreover, we take advantage of cycle multiplexing in FlexRay 3.0.1 where multiple nodes can be assigned slots with the same slot ID in different communication cycles. This helps to maximize the static segment utilization while using only a minimal number of slots [61]. Moreover, jitter awareness is added to the scheduling by considering the previously computed mean jitter of the message (μ_j^n) and a control parameter called coefficient of jitter resilience (σ) that dictates the resiliency of the design time schedule to jitter. The parameter σ is a non-negative real number that dictates how resilient the schedule is for jitter. When $\sigma = 0$, it reflects a special case called zero-jitter (ZJ) scheduling. However, in real-time systems ZJ scheduling is not encouraged as it has no resilience to jitter. Having a higher value for σ results in longer response times and leads to potentially missing message deadlines. Hence, it is important to choose an appropriate value of σ that provides

sufficient jitter resilience while not resulting in longer response times and missed deadlines. In this work, we empirically set the value of σ as 0.8. In addition, we consider the concept of message repetition and slot ID utilization in a FlexRay system. For any time-triggered message in FlexRay, message repetition (rm_j^n) is the ratio of message period to the cycle time of the FlexRay as shown below:

$$rm_j^n = \frac{p_j^n}{C_{fx}} \tag{8}$$

This number is an integer value as the FlexRay cycle time is chosen to be the greatest common divisor of all the message periods in the system. Moreover, any time-triggered message that is assigned a particular slot ID will end up using $(1/rm_j^n)$ of the available slots within that slot ID.

Algorithm 4 shows how the above-mentioned metrics are utilized in the synthesizing of a jitter-aware design time schedule. The heuristic begins by taking the set of time-triggered messages in the system (*M*) and FlexRay parameters as the inputs, and initializes all slot utilizations to zero. In addition, a slot cycle list (*SCL*) is defined to keep track of the list of available cycles in a particular slot ID and each element in it is initialized with a list [0, 1,…,63] as C_{fx} = 64 (Section 2.3.4). After initializing in step **1**, all the time-triggered messages (*M*) in the system are sorted in increasing order of message periods (step **2**). For each time-triggered message (m_j^n) in the system, we begin the search for slot ID and cycle number allocation from the computed *slot* and *cyc* in steps **4**, **5**. The calculations for beginning slot ID and cycle number are based on the message parameters arrival time (a_j^n), mean jitter value (μ_j^n), and the design parameters: coefficient of jitter resilience (σ), static segment slot duration (t_{ds}), and cycle time (t_{dc}). The computed *slot* and *cyc* are subjected to checks for three defined constraints in steps **7-9**: (*i*) arrival time constraint

(*constraint1*) – checks if the current slot (*slot, cyc*) begins after the arrival time plus the effective jitter ($\sigma * \mu_j^n$) of the message; *(ii)* allocation constraint (*constraint2*) – checks if the current slot is not allocated to any other message; and *(iii)* utilization constraint (*constraint3*) – checks if the slot ID (*slot*) utilization is below 100% after adding the current message.

Algorithm 4: design_time_schedule (*solution*)

Inputs: Set of all time-triggered messages in the system (*M*), FlexRay parameters (N_{ss}, C_{fx}, B_{slot}, t_{ds}, t_{dc}), and coefficient of jitter resilience (σ)

Initialize: all slot utilizations ← 0; *SCL*= [SC_1,..,SC_{62}]; SC_x= [0,..,63]
Sort *M* in the increasing order of message periods
for each message m_j^n in *M* **do**
    $cyc = \lfloor (a_j^n + \sigma * \mu_j^n)/t_{dc} \rfloor$
    $slot = \lceil (a_j^n + (\sigma * \mu_j^n) - (cyc * t_{dc}))/t_{dc} \rceil + 1$
    while m_j^n is **not** allocated **do**
        constraint1 = (**start**(*slot*, *cyc*) $\geq a_j^n + \sigma * \mu_j^n$)
        constraint2 = ((*slot*, *cyc*) is **not** allocated to any message)
        constraint3 = (**slot_util**(*slot*) $+ 1/rm_j^n \leq 1$)
        if *constraint1*, *constraint2*, *constraint3* are all **True then**
            if start(*slot*, *cyc*) + $t_{ds} > d_j^n$ **then**
                exit("*No feasible solution*")
            else
                feasible, *cyc_list* = **sc_allocation**(m_j^n, *slot*, *cyc*, SC_{slot})
                if *feasible* **then**
                    $slot_{m_j^n}$←*slot*; $cyc_{m_j^n}$←*cyc*; $cyc\_list_{m_j^n}$←*cyc_list*
                    Assign ownership($slot_{m_j^n}, cyc\_list_{m_j^n}$) → **src**($m_j^n$)
                    Remove elements in *cyc_list* from SC_{slot}
                    m_j^n allocated ← **True**; **break**( )
                end if
            end if
        end if
        slot += 1;
        if *slot* > N_{ss} **then** *slot* = 1; *cyc* += 1
    end while
end for

Output: Message-to-slot assignment ($slot_{m_j^n}$, $cyc_{m_j^n}$) for each time-triggered message m_j^n, and slot ownership of each node *n*.

If all these constraints are satisfied (step **10**) and the finish time of the (*slot*, *cyc*) exceeds the message deadline (step **11**), the algorithm terminates with no feasible solution for the given input message set (*M*). Otherwise, *sc_allocation*( ) is used to check for the feasibility of allocation of the current *slot* and *cyc* to the message. The function returns a binary variable indicating feasibility (*feasible*) and a list of cycles (*cyc_list*) that can be allocated to the message (step **14**). If the current *slot* and *cyc* are feasible, they are allocated as slot ID and base cycle respectively for the current message. Additionally, other cycles in the *cyc_list* are allocated to the message and the ownership of the allocated slot ID and cycles are assigned to the message and its source node (steps **15**-**17**). In step **18**, the *SCL* for the allocated slot ID (*slot*) is updated by removing the allocated cycles (*cyc_list*) and the search for allocation of slot ID and cycle number for the next message is initiated. If the computed slot ID (*slot*) and cycle number (*cyc*) fails to meet any of the three constraints mentioned in steps **7**-**9**, the slot ID and (if needed) the cycle number are incremented accordingly (steps **23**, **24**). When all the messages in the system are allocated slot and cycle numbers, the algorithm terminates successfully.

The pseudo code for *sc_allocation*( ) is shown in Algorithm 5. It takes the current message (m_j^n), slot ID (*slot*), base cycle (*cyc*), and *SCL* corresponding to the slot ID (SC_{slot}) as the inputs and checks for the feasibility of allocating the slot ID and base cycle to the current message. In step **1**, the function initializes a feasibility flag (*feasible*) to zero and cycle list (*cyc_list*) with an empty list, and then computes the minimum number of instances (*num_instances*) of the message in C_{fx} cycles. From steps **2**-**14**, the function tries to find a feasible cycle number for each instance of the message. The search begins by initializing the feasible cycle exists flag (*fc_exists*) to zero and computing the first cycle ($k = 0$) under consideration (step **3**). In steps **4**-**11**, the function tries to find a cycle before the message deadline (i.e., rm_j^n- 1 cycles) by checking three different

conditions (steps **5**-**7**): *(i)* allocation condition – checks if the cycle number in the current slot ID is unallocated; *(ii)* arrival time condition – checks if the slot begins after the message arrival time (a_j^n) and effective jitter ($\sigma * \mu_j^n$) and; *(iii)* deadline constraint – checks if the finish time of the slot is before the deadline of the (k+1)th instance. If all the three conditions are satisfied, the cycle number is added to the *cyc_list*, *fc_exists* is changed to 1 (steps **8**-**10**), and the search for next instance begins. When all the instances of the message are allocated a feasible cycle, the function returns *feasible* as 1 and the list of allocated cycles (*cyc_list*). Otherwise, the current slot ID and cycle number are infeasible for allocating to the current message.

Algorithm 5: sc_allocation (m_j^n, *slot*, *cyc*, SC_{slot})

Inputs: current message (m_j^n), slot ID (*slot*), base cycle (*cyc*), and *SCL* corresponding to slot ID (SC_{slot})

1: **Initialize**: *feasible* = 0; *cyc_list* = [ ]; k = 0; *num_instances* = $\left\lfloor \frac{C_{fx}}{rm_j^n} \right\rfloor$;
2: **while** k < *num_instances* **do**
3: | *fc_exists* = 0; *test_cyc* = *cyc* + k* rm_j^n
4: | **for** i from 0 to (rm_j^n- 1) **do**
5: | | *condition1* = ((*test_cyc*+i) **in** SC_{slot})
6: | | *condition2* = ($a_j^n + \sigma * \mu_j^n$) ≤ **start**(*slot*, *test_cyc* + i)
7: | | *condition3* = **start**(*slot*, *test_cyc* + i) + t_{ds} ≤ (k+1)*d_j^n
8: | | **if** *conditions 1*, *2*, *3* and *4* are all **True then**
9: | | | **Append** *cyc_list* ← (*test_cyc* + i); *fc_exists* = 1; **break**( )
10: | | **end if**
11: | **end for**
12: | **if** *fc_exists* == 0 **then break**( )
13: | k += 1
14: **end while**
15: *feasible* = 1 **if length**(*cyc_list*) == *num_instances*; **else** *feasible* = 0
16: **return** *feasible*, *cyc_list*

Output: feasibility flag (*feasible*) and list of communication cycles allocated to message (m_j^n) for the given *slot* and *cyc*.

2.4.1.4. ACCEPTANCE AND COOLING FUNCTIONS

The acceptance function is used to probabilistically accept the solution created in every iteration (step **9** in Algorithm 1). The probability of acceptance of a new solution is computed using equation (9) below. The term ΔE is the difference between the objective function value of the current solution (*cur_sol*) and the previous solution (*prev_sol*) as shown in equation (10). This is analogous to the energy difference between the new state and previous state in SA. Lastly, the cooling function defines the controlled cooling of the system. In this work, a simple cooling function is employed as shown in equation (11), where C_r is the cooling rate.

$$P_{acceptance} = e^{-\left(\frac{\Delta E}{T}\right)} \tag{9}$$

$$E = \boldsymbol{\Phi}(cur\_sol) - \boldsymbol{\Phi}(prev\_sol) \tag{10}$$

$$Temperature\ (T) = C_r * T \tag{11}$$

2.4.2. RUNTIME MULTI-LEVEL FEEDBACK QUEUE

The schedule generated by the design time scheduler will only guarantee latencies for time-triggered messages when the runtime jitter experienced by the messages does not exceed their effective jitter ($\sigma * \mu_j^n$) value. However, at runtime, various internal and external disturbances may interfere with the normal operation of the FlexRay bus and might result in additional, larger jitter. Hence, it is important to handle a multitude of jitter values during runtime. We focus on handling jitter at runtime using a runtime scheduler that re-schedules jitter affected time-triggered messages using the design-time generated schedule and the output of the Multi-Level Feedback Queue (MLFQ; discussed next) as the inputs. Moreover, in this work we allow the transmission of high priority event-triggered messages within the static segment of FlexRay. Let us consider an example

scenario where a high priority event-triggered message arrives just after the beginning of dynamic segment. Suppose that there is already a low priority event-triggered message that is being transmitted and ends up taking the entire duration of the dynamic segment due to its large message size. This results the high priority event-triggered message having to wait until the beginning of the dynamic segment in the next communication cycle to start transmission if there are no other higher priority messages. This could result in a missed deadline. Hence, we facilitate the transmission of high priority event-triggered messages in the static segment of FlexRay by treating them similar to jitter affected time-triggered messages within the MLFQ, but with a priority lower than time-triggered messages during the runtime scheduling. This facilitates the easy rescheduling of high priority event-triggered messages in the static segment of the FlexRay. Moreover, as the dynamic segment is optional in FlexRay, our framework can be used in the situations where there is no dynamic segment to handle high priority event-triggered messages.

The MLFQ consists of a system of queues (usually two or more) that have different priorities and are capable of exchanging messages between different levels using feedback connections (as shown earlier in Figure 16). The number of queues in an MLFQ defines the number of levels; each level queue can have a different prioritization scheme and scheduling policy compared to other queues. Moreover, the MLFQ attempts to resolve the issues associated with the traditional scheduling schemes such as first come first serve (FCFS), shortest job first (SJF), etc., especially with minimizing inefficient turnaround times for the messages and preventing message starvation.

In this work, we considered an MLFQ consisting of three level queues (Figure 16), with queue 1 (Q1) having the highest priority followed by queue 2 (Q2) and queue 3 (Q3) with lower priorities. In addition to prioritization between different level queues, we set priorities between different types of messages and within the messages of the same type. We prioritize time-triggered

messages over event-triggered ones. Moreover, within the time-triggered messages, we assign static priorities using a Rate Monotonic (RM) policy to prioritize messages with high frequency of occurrence. In case of a tie, priorities are resolved using a First Come First Serve (FCFS) strategy. Event-triggered messages inherit the priority of their generating node. In cases of multiple event-triggered messages from the same node, an Earliest Deadline First (EDF) scheme is employed to prioritize messages. These static priorities of the messages are used to reorder the messages in the queues and promote messages to upper level queues. In addition, there are two separate buffers that are used to handle jitter affected time-triggered messages and high priority event-triggered messages, which are later fed to the MLFQ.

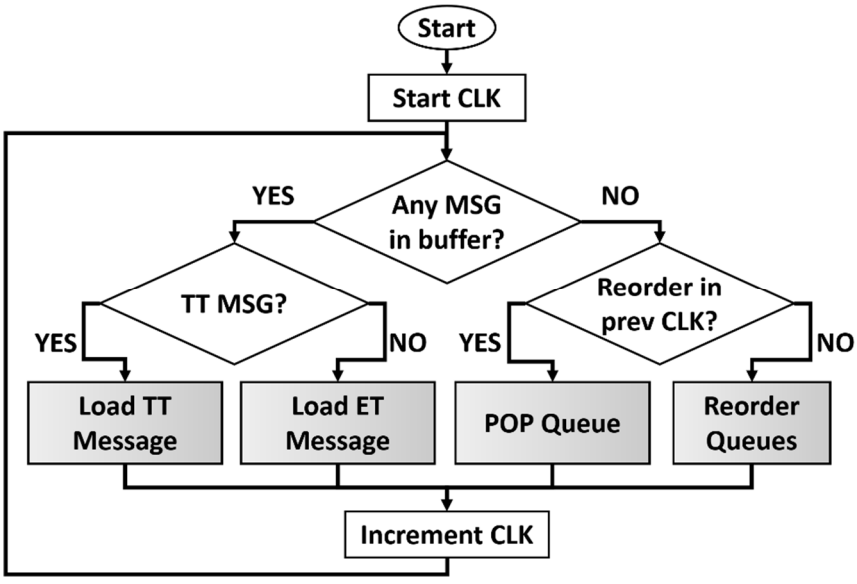


Figure 18 Overview of MLFQ operation.

The operation of the MLFQ is depicted in the flowchart in Figure 18. It begins by checking the time-triggered (TT) message buffer for jitter-affected messages. If a TT message is available, the *load TT message* function is executed, which checks for a vacancy in the queues in the order Q1, Q2, and Q3; and stores the TT message in the first available queue. If the TT message buffer

is empty and an event-triggered (ET) message is available in the ET message buffer, the *load ET message* function is executed which checks for a vacancy in the queues in the order Q2, Q3, and Q1 and stores the ET message in the first available queue. In either case, when all three queues are full, the message is held in the corresponding buffer and the same function is executed in the next clock cycle. Whenever there are no messages available in both the buffers and, the reorder queue function is not executed in the preceding clock cycle, messages in the queues are reordered in the order of their priorities, by executing the *reorder queues* function. Otherwise, the queues are checked in the order Q1, Q2 and Q3 by executing the *POP queue* function. The conditions for popping a queue are discussed in the next subsection.

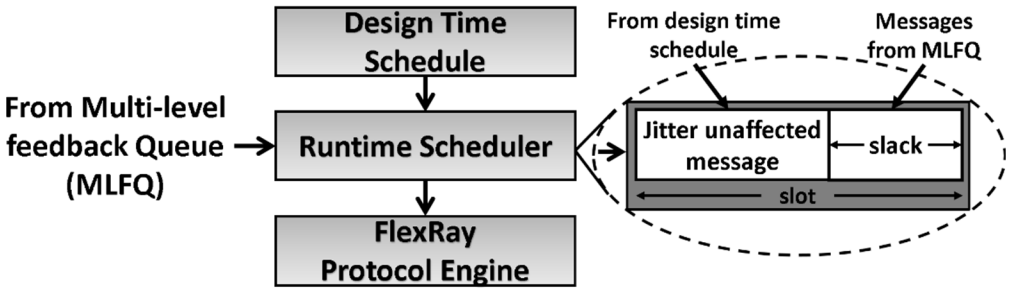


Figure 19 Packing of jitter-affected messages during runtime.

2.4.3. RUNTIME SCHEDULER

After handling the jitter-affected messages in the MLFQ as discussed in the previous subsection, the next step is to re-schedule them during runtime to attempt to meet their deadline constraints. To achieve this, we introduce a runtime scheduler that handles multiple inputs coming from the output of the MLFQ, and the design-time generated schedule. Additionally, the runtime scheduler also has information on the available slack in each of the static segment slots from the design-time generated schedule. Thus, whenever there is a message in the MLFQ, the runtime

scheduler checks the ownership of the next incoming slot. If the incoming slot is owned by the source node of the jitter-affected message in the MLFQ, the runtime scheduler computes the available slack in the incoming slot using the design time schedule. If there is non-zero slack in the incoming slot, the jitter-affected message is collocated with the jitter unaffected message as shown in Figure 19. The entire jitter-affected message is rescheduled in the incoming slot if there is sufficient slack to accommodate the full jitter-affected message. Otherwise, the jitter-affected message is partitioned into two parts, with the size of the first part equal to the available slack in the incoming slot, and the remaining as the size of the second part. The second part of the message remains in the queue and is transmitted in the next feasible incoming slot by bumping up its priority.

Similarly, when a high priority event-triggered message is available at the MLFQ, the high priority event-triggered message is treated similar to the jitter-affected time-triggered message with a lower priority than the regular time-triggered message and the above-mentioned series of steps are followed. This results in packing of two different message data into the payload segment of one FlexRay frame, which unfortunately leads to two major challenges. Firstly, there is a need for a mechanism at the receiver node to decode the payload segment correctly and distinguish between the two messages. Secondly, the implicit addressing scheme of FlexRay is lost because of combining two different messages, as the receiving nodes will not be able to identify to which specific node the message is meant for.

To overcome the above-mentioned challenges, we propose a segmentation and addressing scheme to differentiate multiple messages packed into the same frame. This scheme introduces one additional segment in the payload segment of the FlexRay frame that is common to multiple jitter-affected messages packed in that frame, and two more segments for each jitter-affected

message that is packed into the frame. An illustration of two jitter-affected messages being collocated with a jitter unaffected message with the proposed segmentation and addressing scheme is illustrated in Figure 20.

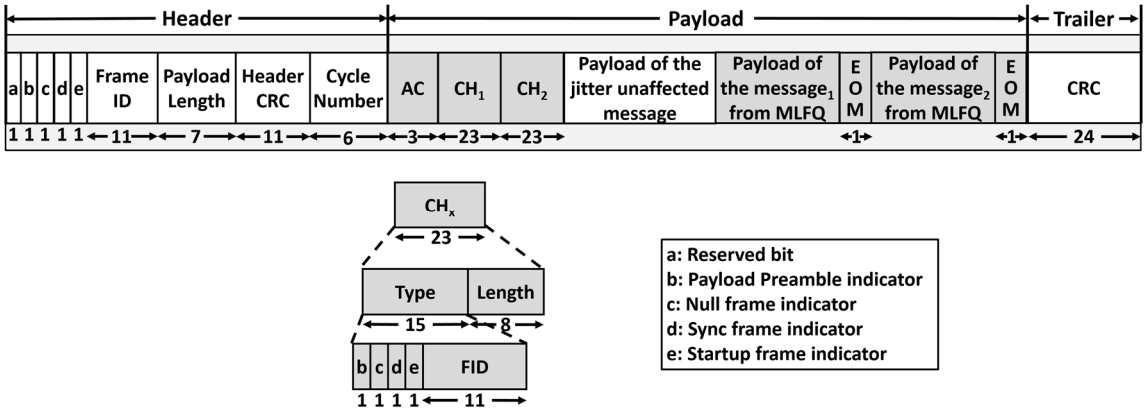


Figure 20 Updated frame format of the FlexRay frame using the proposed segmentation and addressing scheme (sizes of individual segments are shown in bits). The parts of the frame highlighted in gray represent our modifications.

The first common segment is called *AC* or append counter, which is also the first segment in the payload. It is a 3-bit field indicating the number of different jitter-affected messages that are packed in the current frame. In this work we support partial message transmission to fully utilize the available bandwidth in allocated static segment slots. The second segment is called *CH* or custom header, which is private for every jitter-affected message in the FlexRay frame. Every *CH* segment further consists of two fields: a *type* field and a *length* field. The *type* is a 15-bit field used to specify different message types (defined in [16]). The *type* field consists of one bit each for payload preamble indicator, null frame indicator, sync frame indicator, and startup frame indicator, and an 11-bit frame ID (*FID*) field for specifying the *FID* of the jitter-affected message. The *length* field is 8-bit long and specifies the data length of the jitter-affected message in bytes. The *length* field in the custom header along with the payload length field in the frame header is used to find

the start bit of the jitter affected message in the payload segment. The third segment we introduced in the payload is called *EOM* or end of message, which is a 1-bit segment that is private to each jitter-affected message. The *EOM* field is 1 when the entire message is transmitted; otherwise the *EOM* field is set to 0 indicating a partial message transmission. The remaining message data that is transmitted in the next feasible slot will have the remaining data size in the *length* field of its custom header. If there is more than one jitter-affected message packed in the FlexRay frame, the headers of all the messages are in the beginning of the payload segment. This gives the receiver node information about all the jitter-affected messages that are packed in the FlexRay frame. Moreover, the regular operation of the FlexRay protocol is not altered in any way by implementing these changes.

2.5. EXPERIMENTS

2.5.1. EXPERIMENTAL SETUP

To evaluate the effectiveness of the proposed JAMS-SG framework, we first contrast it against two variants of the same framework: JAMS-SA and JAMS-ACC-SG. The first variant JAMS-SA, uses a simulated annealing (SA) approach with no GRASP based local search. This is implemented to test the effectiveness of local search. In JAMS-SA the solution is subjected to random perturbations and a new solution is created every iteration with the randomly chosen signals having an equal likelihood of grouping or splitting. The second variant is called JAMS-ACC-SG (accelerated SA+GRASP). JAMS-ACC-SG behaves similar to JAMS-SA in the beginning but then it switches to a JAMS-SG behavior (i.e., including GRASP based local search) when the temperature of the system is sufficiently low. This threshold temperature is set to 30% of the initial temperature based on empirical analysis. The motivation for an accelerated version is

to save the computation time spent in looking for the local optimum in the beginning and only perform the local search after a reasonable solution is achieved. A comparison between these three techniques is presented in Section 2.5.2. In addition, we also perform a series of experiments with different weight values to determine the optimal weight parameters for the best variant of our framework.

Subsequently, we compare the best variant of our framework with several prior works: Optimal Message Scheduling with Jitter minimization (OMSC-JM [21]), Optimal Message Scheduling with FID minimization, (OMSC-FM [21]), Policy-based Message Scheduling (PMSC [67]), and JAMS-greedy [44]. OMSC-JM [21] and OMSC-FM [21] use the ILP based frame packing technique from [20] and change the message repetition to minimize the effect of jitter. OMSC-JM and OMSC-FM differ in the weights associated with the objective function in their optimization problem formulation. OMSC-JM tries to minimize the effect of jitter by allocating more slots and performing more frequent message transmissions while OMSC-FM aims at minimizing the number of allocated slots. PMSC [67] uses a priority based runtime scheduler that supports preemption based on the message arrival time and priority. Messages are scheduled based on the slot assignment generated using heuristics. JAMS-greedy [44] uses a greedy frame packing approach to generate the set of messages and uses a heuristic based scheduler to synthesize design time schedules. In addition, JAMS-greedy also supports a runtime scheduler to reschedule jitter-affected messages similar to JAMS-SG. However, JAMS-greedy lacks the ability to send multiple jitter affected messages in one FlexRay frame and also does not support partial message transmission. Additionally, we also implemented a genetic algorithm (GA) based frame packing approach for FlexRay-based systems using the frame packing technique proposed for CAN-FD in [59]. We further adapted the scheduling policy proposed in [44] and combined it with [59] (and

hence the name JAMS-GA) to compare it with our framework. All experiments conducted with these prior works are discussed in detail in the next subsections.

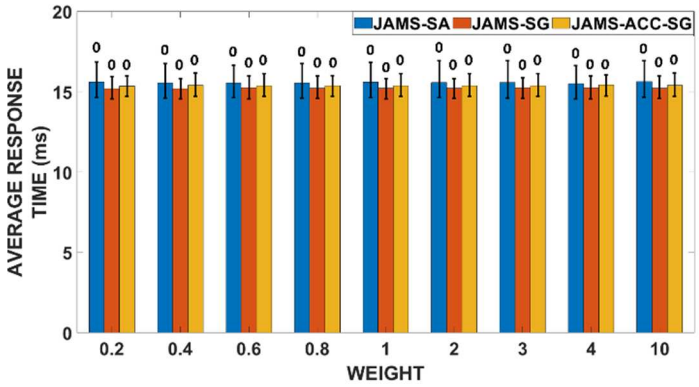
To evaluate our proposed framework with its variants and against prior work, a set of test cases was derived using different combinations of the number of nodes, number of signals in the system, and the periods of the signals based on automotive network data extracted from a real-world 2016 Chevrolet Camaro vehicle that we have access to. For all our experiments we considered a FlexRay 3.0.1 based system with the following network parameters: cycle duration of 5ms (t_{dc}) with 62 static segment slots (N_{ss}), with a slot size of 42 bytes (B_{slot}) and 64 communication cycles (C_{fx}). Moreover, each experiment was run for 1000 iterations with an initial temperature = 10000 and the cooling rate (C_r) set to 0.993. The RCL threshold parameter (α) of GRASP was chosen as 0.4 which resulted in a relatively near greedy solution in the presence of relatively large variance. The destroy-repair parameter (β) is set to 2 which helped in avoiding an exploration of a larger neighborhood around the greedy randomized constructed solution. We randomly sampled jitter values as a function of the message period to modify the arrival times of randomly selected messages originating from a set of randomly selected jitter-affected nodes. Moreover, in this work, we set the coefficient of jitter resilience (σ) = 0.8, as discussed earlier. To model the overhead of MLFQ operations, we assume that each message affected by jitter experiences additional latency (which we consider in our experiments) that is a function of static priority of the message (derived using the RM scheme), message data size, and the queue it is in. All the simulations are run on an Intel Core i7 3.6GHz server with 16 GB RAM.

2.5.2. COMPARISON OF JAMS-SG VARIANTS

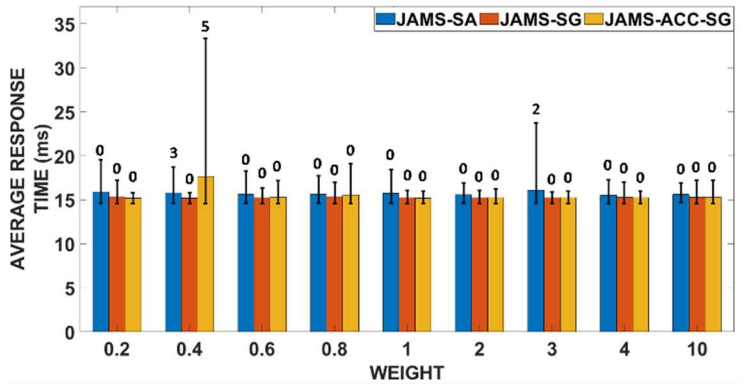
In this subsection, we present a comparison of the proposed JAMS-SG framework with the two other proposed variants, JAMS-SA and JAMS-ACC-SG. A series of experiments were conducted by changing the weight in the objective function (as shown in equation (5)). The results were analyzed under four different scenarios: (i) zero, (ii) low (iii) medium and, (iv) high jitter. Under zero jitter, none of the messages in the system are affected by jitter. Hence, their arrival times remain unchanged. Under the next three different jitter scenarios, the arrival times of randomly selected time-triggered messages originating from a randomly selected set of jitter-affected nodes are modified as a function of the message period. In low, medium and high jitter scenarios, the randomly chosen messages are subjected to jitter values equal to $p_j^n/8$, $p_j^n/5$ and $p_j^n/4$, respectively (where p_j^n represents the period of message j belonging to node n). We considered a real world automotive case study (as discussed in Section 2.5.1) consisting of 19 ECUs and 248 signals.

Figure 21 (a)-(d) show the average response time of all the messages in the system for the three framework variants with different objective function weights under zero, low, medium and high jitter conditions, respectively. The confidence interval on top of each bar represents the minimum and maximum of the average response time, and the number on top of the bar represents the number of deadline misses. It is evident from Figure 21(a)-(d) that JAMS-SG has superior performance in response time compared with both JAMS-SA and JAMS-ACC-SG in most of the cases, across all weight values and jitter scenarios. Most importantly, JAMS-SG never misses any deadline for any weight value and jitter scenario. This is because JAMS-SG is able to find a better jitter-aware frame packing solution from the beginning due to its more effective GRASP based optimization. This results in achieving a solution that efficiently balances between minimizing the

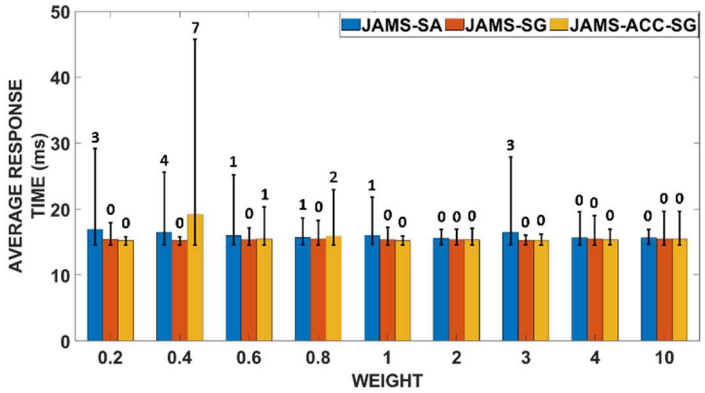
number of FlexRay messages and maximizing laxity of the messages. JAMS-SA fails to find a comparable solution because of the lack of local search mechanisms to improve the quality of the pure SA based solution. JAMS-ACC-SG suffers similarly as JAMS-SA until the local search process is initiated. But, when the local search process begins, the system temperature is already low. This forces the system to only accept the better solutions, as the acceptance probability function results in a smaller probability value in case of a relatively bad solution. This in turn often results in the approach getting stuck at a local minima. From Figure 21 it can also be observed that as the weight value increases, the number of missed deadlines decreases across the frameworks and under different jitter scenarios. This is due to the increasing emphasis on minimizing the number of FlexRay frames in all three frameworks, resulting in fewer frames to be scheduled, which simplifies the problem. Moreover, Figure 21 also shows that choosing a very high or a very low weight value makes the system heavily biased towards optimizing either the number of FlexRay frames or towards optimizing laxity. In order to avoid this, we select an intermediate weight value of 2. Henceforth, all the other comparisons are done against JAMS-SG (the best variant in our analysis) with weight (w) = 2.



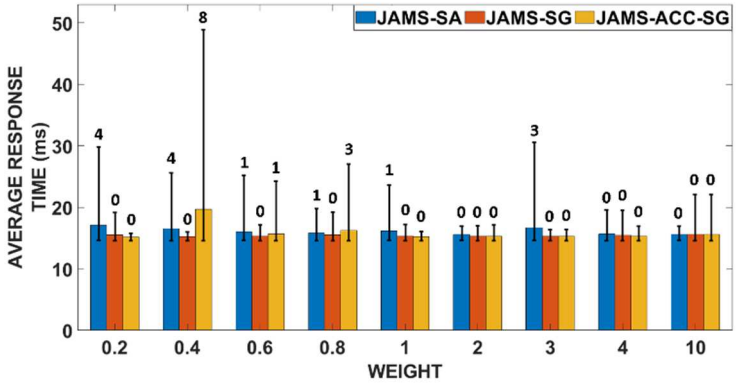
(a)



(b)



(c)

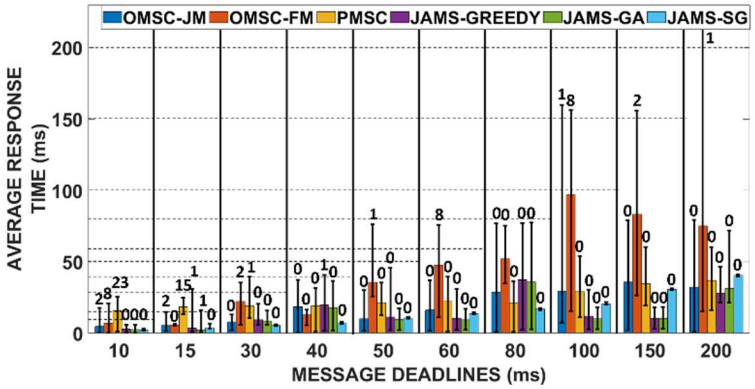


(d)

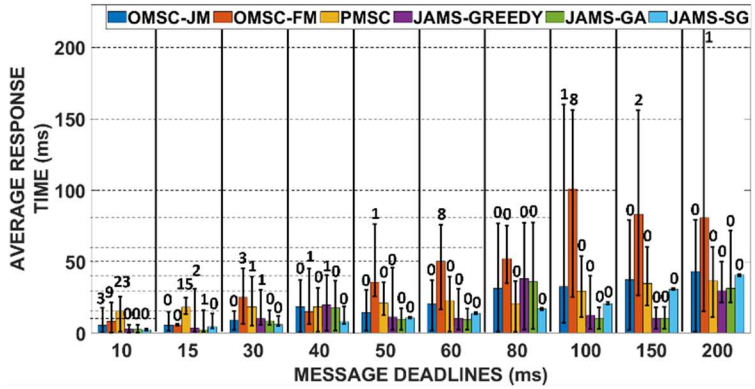
Figure 21 Average response time of all signals for different objective function weights (with number of missed deadlines shown on top of the bars) under (a) zero, (b) low, (c) medium, and, (d) high jitter conditions; for JAMS-SA, JAMS-SG and JAMS-ACC-SG.

2.5.3. RESPONSE TIME ANALYSIS

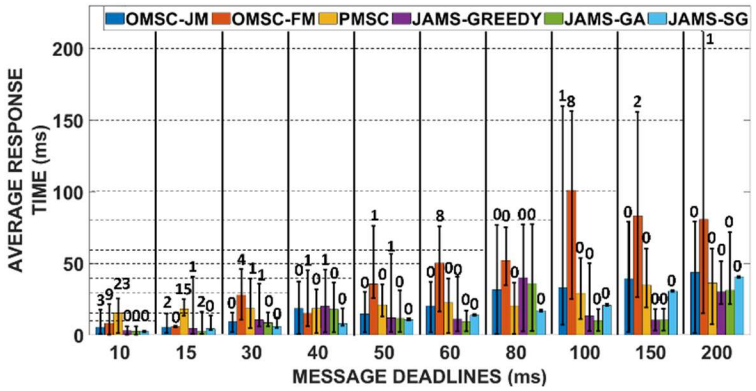
Next, we present a comparison study of JAMS-SG with message scheduling frameworks from prior work. We use the same vehicle test case as used in the previous subsection. The arrival times of randomly selected messages from the randomly chosen nodes are delayed, to induce jitter in simulations. Moreover, all the nodes and the messages in them have an equal probability of being selected to be subjected to jitter.



(a)



(b)



(c)

Figure 22 Message deadlines vs average response time (with number of missed deadlines shown on top of the bars) under (a) low, (b) medium and (c) high jitter conditions; for the comparison frameworks (OMSC-JM [21], OMSC-FM [21], PMSC [67], JAMS-GREEDY [44], JAMS-GA [59], [44]), and our proposed JAMS-SG framework. The number of instances of missed deadlines typically increases as jitter goes from low to high for the comparison frameworks, and the messages with smaller (more stringent) deadlines are generally more susceptible to missing deadlines.

Figure 22 (a)-(c) show the average response time of the messages under low, medium and high jitter scenarios (configured as discussed in the previous subsection). The confidence interval on each bar represents the minimum and maximum average response time of the messages achieved using each technique and the number on top of each bar represents the number of deadline misses. The response time results are clustered into groups based on the message deadlines (on x-axis) and the dashed horizontal line represents the deadlines. It can be observed that using OMSC-FM results in high response times in the presence of jitter. This is because of the high emphasis on minimizing the number of static segment slots, which resulted in the generated solution having only a few static segment slots, but poor jitter resilience. On the other hand, OMSC-JM performs relatively better as it allocates extra slots for transmission. However, it still has issues handling random jitter during runtime, especially for high priority messages. In the PMSC technique, jitter has a strong impact on the high priority messages, because of the frame packing approach used in

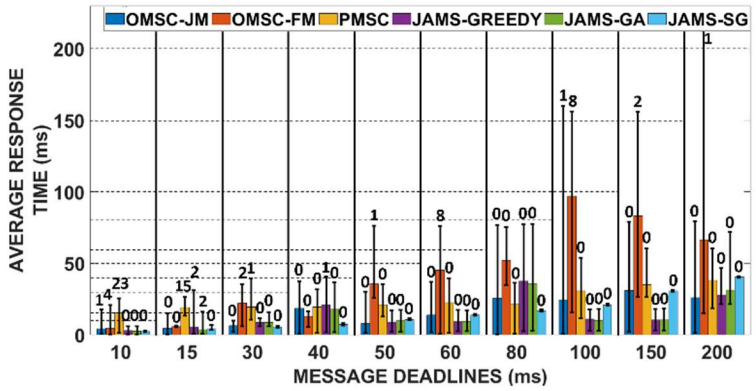
it. PMSC aims to use the entire static segment slot by packing the signals that are larger than the slot size and uses EDF-based preemption at the beginning of each slot. In the presence of jitter, especially for high priority messages, the arrival of these messages are delayed causing the node to wait for the next transmitting slot to preempt existing transmissions of low priority messages. This delay along with jitter can sometimes exceed the message deadline and lead to missed deadlines. Additionally, JAMS-greedy and JAMS-GA result in suboptimal frame packing that focuses on minimizing the number of FlexRay frames. The jitter-awareness due to the runtime scheduler in these works help them to be relatively jitter resilient compared to other prior works. However, these frameworks start missing deadlines when there is high jitter. It is evident that under all three jitter scenarios, JAMS-SG outperforms all the other prior works with *no deadline misses*. This is accomplished by JAMS-SG's ability to find a balanced solution that results in optimal frame packing and jitter resilience. Moreover, the support for partial message transmission helps JAMS-SG to meet the deadline constraints under different jitter scenarios.

2.5.4. SENSITIVITY ANALYSIS

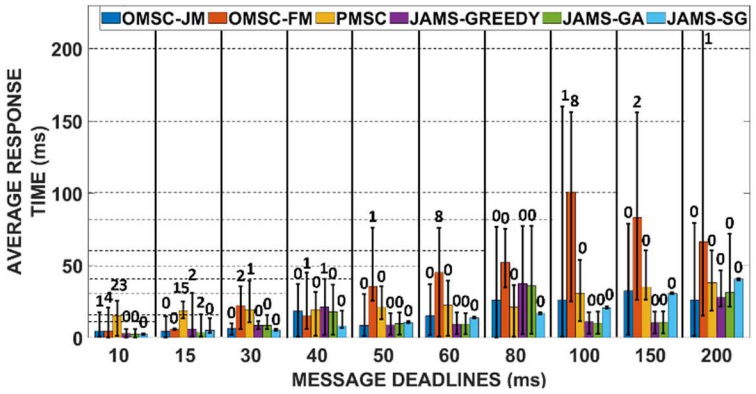
In this subsection, we analyze the impact of jitter on a specific subset of messages and study the behavior of the system. The same test case considered in the previous subsection is used and the results are compared with the prior works described previously.

Figure 23 (a)-(c) illustrates the message deadline vs average response time plots for the considered test case under low, medium, and high jitter, for the case where only the high priority messages (messages with deadline ≤ 40ms) are subjected to jitter. The messages affected by jitter are randomly chosen from the set of high priority messages belonging to the randomly chosen set of nodes. It can be seen that the impact of jitter results in higher response times and deadline misses

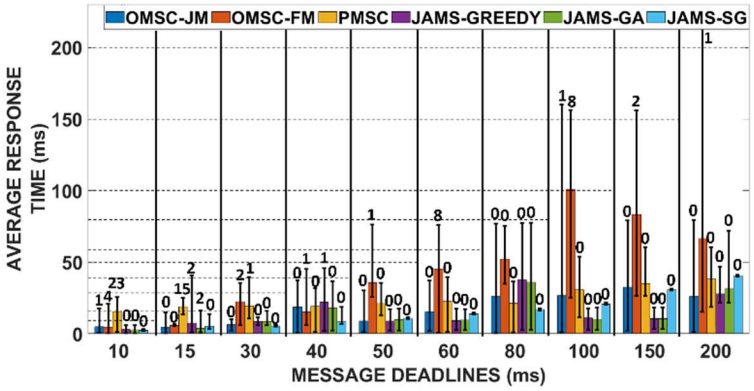
for the high priority messages in most of the frameworks from prior work. In particular, for OMSC-JM and OMSC-FM, some of the low priority messages suffer from very long response times and deadline misses. JAMS-SG not only results in minimal response times for most of the cases but also results in *no deadline misses*.



(a)



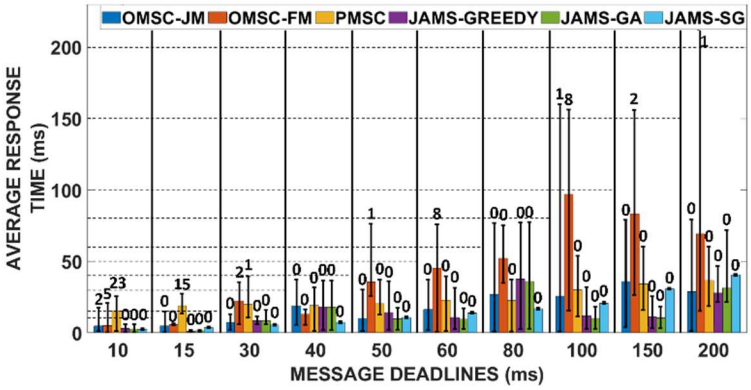
(b)



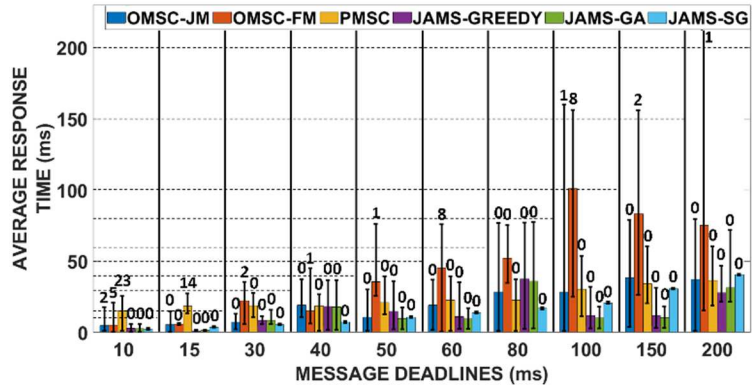
(c)

Figure 23 Message deadlines vs average response time with jitter affecting high priority messages only; (with number of missed deadlines on top of bars) under (a) low, (b) medium and (c) high jitter conditions; for the comparison frameworks (OMSC-JM [21], OMSC-FM[21], PMSC [67], JAMS-GREEDY[44], JAMS-GA [59], [44]), and JAMS-SG. The number of instances of missed deadlines typically increases as jitter goes from low to high for the comparison frameworks, and the messages with smaller (more stringent) deadlines are generally more susceptible to missing deadlines.

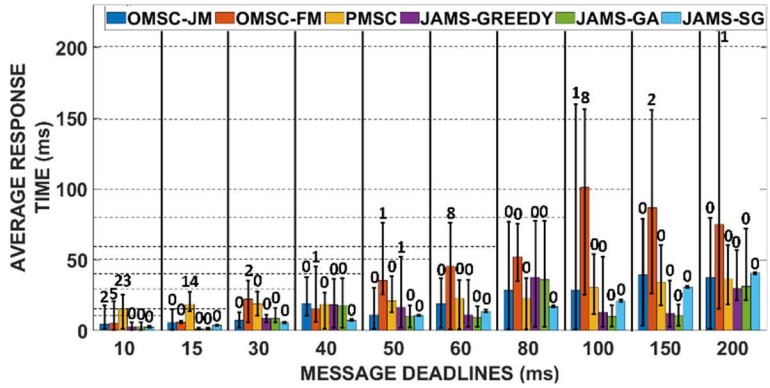
Figure 24 (a)-(c) shows the message deadline vs average response time plots for three jitter scenarios where only the low priority messages (messages with deadline > 40ms) are subjected to jitter. It is clear that almost all of the frameworks except JAMS-SG fail to meet the deadline constraint for specific scenarios.



(a)



(b)



(c)

Figure 24 Message deadlines vs average response time with jitter affecting low priority messages only; (with number of missed deadlines on top of bars) under (a) low, (b) medium and (c) high jitter conditions; for the comparison frameworks (OMSC-JM [21], OMSC-FM[21], PMSC [67], JAMS-GREEDY[44], JAMS-GA [59], [44]), and JAMS-SG. The number of instances of missed deadlines typically increases as jitter goes from low to high for the comparison frameworks, and the messages with smaller (more stringent) deadlines are generally more susceptible to missing deadlines.

From Figure 22,Figure 23 and Figure 24, it is evident that JAMS-SG can handle a wide variety of jitter patterns and is still able to meet the deadline constraints for all the messages in the system.

2.5.5. SCALABILITY ANALYSIS

To evaluate the effectiveness of JAMS-SG for system configurations with different complexities, we analyzed our framework against the frameworks from prior works by selecting test cases with varying combinations of number of nodes and number of signals. Figure 25 presents the average response times of all the messages for the high jitter scenario, for different system configurations $\{p, q\}$ where p denotes the number of nodes and q is the number of signals (x-axis).

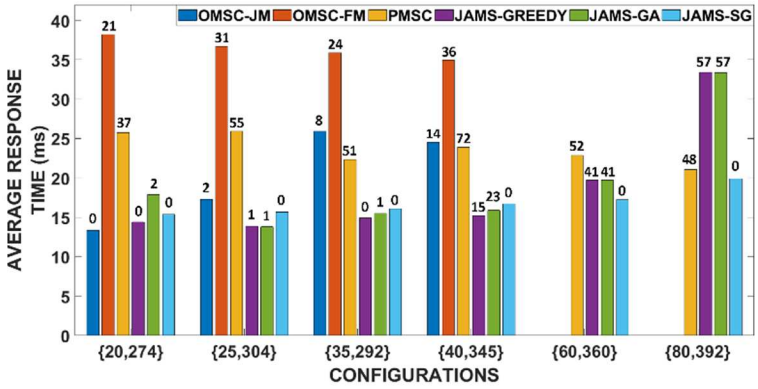


Figure 25 Average response time for different system configurations (with number of missed deadlines on top of bars) under high jitter; OMSC-JM [21], OMSC-FM[21], PMSC [67], JAMS-GREEDY [44], JAMS-GA [59], [44], and our JAMS-SG framework.

The number on the top of each bar indicates the number of signals that missed the deadline in that configuration. For larger test cases, some of the prior works (OMSC-JM, OMSC-FM) failed to result in a feasible solution within the 24 hour time limit. It can be observed that even with increasing system size, JAMS-SG is able to meet all message deadlines for every configuration. On the other hand, it can be seen that prior works lacking jitter awareness suffer from multiple deadline misses. Among them OMSC-FM seems to perform particularly poorly compared to all other works because of its heavy emphasis on minimizing number of allocated slots, thereby resulting in minimal number of available slots, but a lack of jitter resilience.

Table 3 Time taken to generate the solution (in seconds) for different configurations: OMSC-JM [21], OMSC-FM [21], PMSC [67], JAMS-Greedy [44], JAMS-GA [59], [44] and our JAMS-SG framework.

	Configurations					
	{20,274}	{25,304}	{35,292}	{40,345}	{60,360}	{80,392}
OMSC-JM	2700.54	2700.89	2700.94	2700.93	-	-
OMSC-FM	2700.49	2700.85	1116.09	2700.95	-	-
PMSC	0.72	1.89	1.947	2.58	1.31	1.65
JAMS-GREEDY	0.71	1.811	2.52	3.48	3.01	5.93
JAMS-GA	66.05	142.6	152.15	178.75	122.08	157.97
JAMS-SG	94.15	134.96	298.62	449.71	576.02	1149.80

Lastly, Table 3 presents the time taken (at design time) for JAMS-SG and other frameworks to generate the best solution for the different system configurations. It can be seen that JAMS-SG is able to achieve a better solution (jitter resilient frame packing and schedule) with no deadline misses under 20 minutes even for the largest test case configuration. Thus, our proposed JAMS-SG framework is highly scalable across a variety of system complexities and jitter profiles, and unlike the frameworks proposed in prior work, results in no message deadline misses for the test cases considered. This makes the framework a promising approach to cope with random jitter in emerging automotive systems.

2.6. CONCLUSIONS

In this chapter, we presented a novel message scheduling framework called JAMS-SG that utilizes both design time and runtime scheduling to mitigate the effect of jitter in time-triggered automotive systems. At design time, our framework uses a hybrid SA+GRASP heuristic for jitter-aware frame packing and to synthesize a design time schedule. At runtime, our framework effectively handles both jitter-affected time-triggered and high priority event-triggered messages using the proposed MLFQs and runtime scheduler. We also proposed a custom frame format to solve the addressing and segmentation problem associated with packing multiple jitter-affected

messages. We compared our JAMS-SG framework with the best known prior works in the area by studying performance under varying jitter conditions. Our experimental analysis indicate that JAMS-SG is able to achieve lower response times for most of the cases and more importantly, without missing any message deadlines. Our experiments also show that JAMS-SG is highly scalable and outperforms the best-known prior works for various system sizes and under a variety of jitter patterns. Lastly, our framework can be extended to other time-triggered protocols with minimal changes.

3. PRIORITY-BASED MULTI-LEVEL MONITORING OF SIGNAL INTEGRITY IN A DISTRIBUTED POWERTRAIN CONTROL SYSTEM

As the number of Electronic Control Units (ECUs) in modern vehicles increases, the in-vehicle control and communication network has become highly complex. This problem is perhaps more acute in the case of a Hybrid Electric Vehicle (HEV) as there are a greater number of electrical components such as motors, batteries, and DC-DC converters, than in conventional vehicles. In most modern HEVs and conventional vehicles, Controller Area Network (CAN) is the most widely used communication protocol because of its simplicity, low cost, noise immunity and ease of implementation. However, it suffers from low bandwidth, poor security, message delays etc. as discussed in [80]. The typical communication system in a HEV is shown in Figure 26. The driver inputs are sent to the supervisory controller via CAN, and it sends the appropriate signals to all the other local controllers via CAN messages. If there are multiple nodes connected to the same CAN channel, then the network congestion increases which leads to many issues such as signal delay, loss of signal integrity, jitter, and other failures as mentioned in [81]. All of these communication system limitations must be detected and remedied to avoid catastrophic vehicle-level malfunction.

Comprehensively monitoring the integrity of 100% of the signals in the vehicle would greatly increase the required bandwidth of the communication system. To avoid this naïve approach to signal integrity monitoring, we seek in this study to monitor signal integrity for only those signals that are of importance to vehicle safety and performance. In particular, we focus on monitoring the signal integrity of torque-related signals because of their importance in preserving the safety and performance of the vehicles.

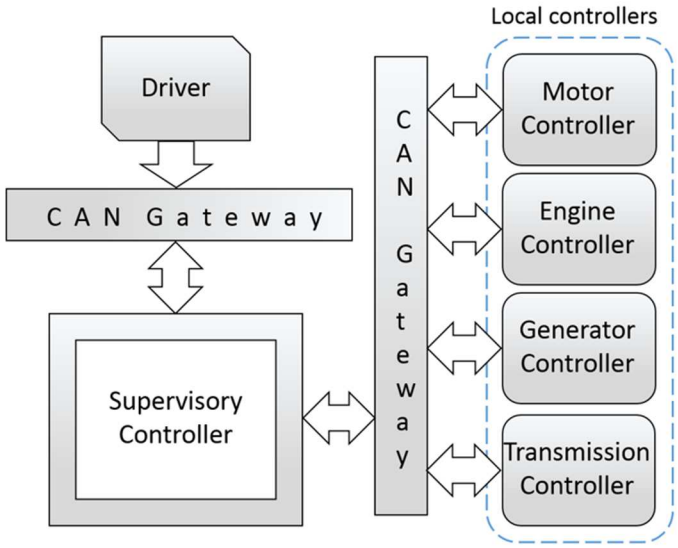


Figure 26 Typical communication system in an HEV.

3.1 RELATED WORK

Various techniques have been proposed to protect the signal integrity in CAN and other busses. The authors in [82] present a delayed data authentication technique using the KASUMI encryption algorithm in CBC-MAC (Cipher Block Chaining- Message Authentication Code) mode to generate a 64-bit compound MAC for a group of four messages, which is further divided into four 16-bit MACs and each of them is stored in the 16-bit CRC (Cyclic Redundancy Check) field of the next four CAN messages. This technique assures message integrity, but suffers significant delay as the first four messages are validated by comparing the generated MAC with the received MAC in the CRC field of the next four messages. Furthermore, the delay in receiving the second group of messages adds to the overall delay, limiting this algorithm's usefulness in time-critical powertrain control applications.

The authors in [83] discuss different controller integrity techniques and propose asymmetric

and extended asymmetric controller strategies. In these strategies an auxiliary controller is used to check the integrity of the primary controller. Some of these techniques have synchronization hardships, size and power overhead, and also encounter overhead in making changes to the existing network. [84] introduce a signal integrity technique in which the original signal, along with a redundant signal in encrypted form is sent in the same message. This signal is validated by comparing the original signal and decrypted redundant signal. The main limitation of this technique is that the bandwidth requirements get doubled, leading to a high bus load. Also, encrypting and decrypting signals can incur high computational overhead in the system.

The motivation for our research is to come up with a signal integrity technique that helps to improve the signal integrity in the CSU EcoCAR3 project without incurring significant overhead on the controller while trying to minimize bus traffic. In this chapter we introduce a priority based, multi-level signal integrity technique in which, error tolerance for different signals are set according to the order of priority and are monitored using performance counters. The rest of the chapter is organized as follows: In Section 3.2, the proposed technique is presented in detail and in Sections 3.3 and 3.4, the experimental setup and results are discussed. In Section 3.5, this method is illustrated by performing analysis on bus parameters and vehicle performance and Section 3.6 presents our conclusions.

3.2 PRIORITY BASED MULTI-LEVEL SIGNAL INTEGRITY TECHNIQUE

An illustration of the proposed technique is shown in Figure 27. In the first step, we divide different controller signals into different groups based on their criticality. In the next step, messages are transmitted and a handshake signal is sent from the supervisory controller to the local controller to which the torque command is sent. To monitor the signal integrity we make use of

performance counters in the supervisory controller and set a threshold for the number of negative acknowledgements that command the local controller to discard the message and take appropriate remediation actions. If the number of negative acknowledgements exceed the threshold, the vehicle is moved into one of the limp modes depending on the criticality level of the failed signal, otherwise the vehicle operates normally.

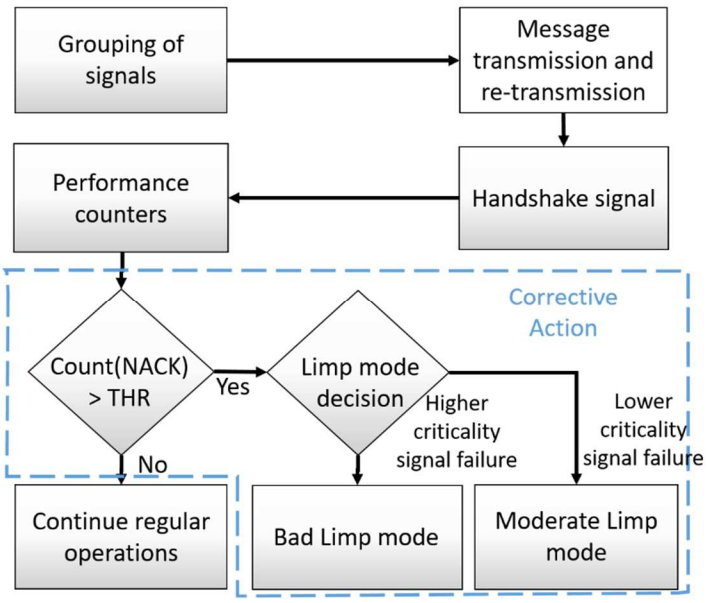


Figure 27 Flow chart of proposed technique. (NACK = Negative Acknowledgement, THR = Threshold).

3.2.1. GROUPING OF TORQUE RELATED SIGNALS

In this subsection, we discuss how different torque–related signals are considered from various components in the vehicle and grouped together based on their criticality. Each group is assigned different levels of criticality with level-1 being the most critical group and level-4 being the lowest. The signals in different levels impact the vehicle performance and safety in different ways. Detection of failures in signal integrity should therefore be handled differently at each level of criticality. Thus, the limit of fault tolerance varies from level to level as discussed in the next

subsection. Table 4 shows the grouping of different controller signals and their criticality levels. Inefficient grouping of signals can have a negative impact on the safety and performance of the vehicle.

Table 4 Grouping of signals and their criticality levels.

Criticality level	Signals
Level- 1	Physical brake request, Motor regen request
Level- 2	Motor torque request, Engine torque request, Acceleration pedal position sensors (A & B)
Level- 3	PRNDL, Ignition
Level- 4	Infotainment

3.2.2. MESSAGE TRANSMISSION WITH HANDSHAKE SIGNALS

In this subsection, the concept of measuring and verifying signal integrity is discussed in detail. [85] states that in order to preserve signal integrity, it is necessary to add some redundancy to the message which inevitably increases the bandwidth required. In the signal integrity preservation technique proposed by [84], every CAN signal is transmitted twice, at every sample, which doubles the required bandwidth and increases delays. In our technique we propose a multi-channel CAN setup, in which there is a dedicated CAN channel for all the redundant and handshake signals labelled Control Bus (CB) and there can be one or multiple CAN channels for regular communications. When a CAN message is transmitted from the supervisory controller to the local controller via the CAN bus, the local controller re-sends the received message back to the supervisory controller via the CB. Then the supervisory controller compares the sent and received messages and sends a handshake signal to the local controller via CB, indicating faulty or not-faulty signal transmission. Performing this signal integrity monitoring and confirmation for every transmission can be expensive and bandwidth-intense. In this study, for a particular criticality level, the signal integrity monitoring and conformation is performed periodically at a

rate equal to the time window of that particular level. In other words, we do not monitor all the samples of the signal, instead we perform our technique only on every n^{th} sample of the signal (where n = Time window (ms)/ Rate of transmission of original signal (ms)). The CAN network setup for our technique is shown in Figure 28.

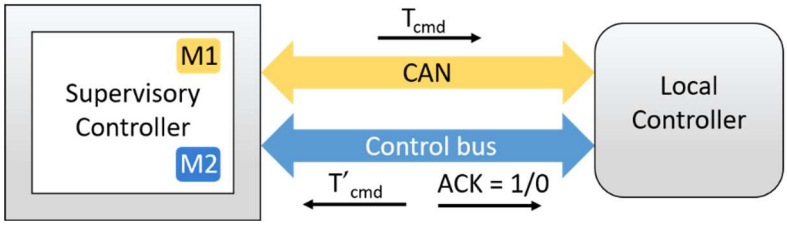


Figure 28 CAN network setup for our proposed technique.

In Figure 28, the driver torque request is sent to the local controller continuously via CAN and is stored in the memory (M1) of the supervisory controller at the beginning of each time window. In the same time window the local controller immediately sends the received torque command back to the supervisory controller via the control bus (CB), which is then stored at a different memory location (M2). The supervisory controller then sends out a signal ACK = 1 (positive acknowledgement) if values in both memory locations M1 and M2 match, else, it sends ACK = 0 (negative acknowledgement) via CB, which are referred to as handshake signals. Thus, ACK = 0 notifies the local controller that the received message is faulty while ACK = 1 indicates a flawless transmission. The torque command is not executed at the local controller until it encounters an ACK=1 (positive acknowledgement) signal from the supervisory controller.

3.2.3. PERFORMANCE COUNTERS

Any erroneous transmissions are reported to the local controller via negative acknowledgement signals from the supervisory controller. The supervisory controller keeps a record of these signals by making use of different performance counters, which monitor the number of NACKs (Negative Acknowledgements) and the THR (Threshold). The values of NACK and THR are specific for each signal type. For example, a motor torque request signal has a dedicated counter MG_torque_NACK, which counts all the negative acknowledgements (ACK = 0) from the supervisory controller that are associated with the motor torque request. MG_torque_NACK increments every time a negative acknowledgement is encountered until it exceeds the threshold, MG_torque_THR. This threshold is set to different values for different criticality levels. The counter is reset only when the vehicle moves from either of the limp modes (discussed in the next subsection) to a normal operating state. This happens only when a set of positive acknowledgements are sent to the local controller over a set of time windows.

Table 5 Time window and threshold for different levels.

Criticality Level	Time window (ms)	Threshold
Level- 1	150	5
Level- 2	100	10
Level- 3	2000	3
Level- 4	-	-

The time windows and thresholds (shown in Table 5) for different levels of signals are chosen by taking into account both safety and bus load. For example, for signals of criticality level-2, the signal integrity check must be failed 10 times to enable a corrective action. With a time window of 100ms, the time between signal integrity failure and a corrective action is 1000 ms.

The choice of the time window values is discussed in more detail in the analysis section (Section 3.5.1).

3.2.4. CORRECTIVE ACTION

The corrective or remedial action involves taking an appropriate action once a failure in a controller is detected because of erroneous message transmissions. In developing the corrective action, the first step is to keep a track of NACK counters for all the torque related signals that are monitored. The NACK counter value is less than THR for normal operation of the controller. When this condition is violated, the corrective action mechanism is initiated. As there are different levels of signal criticalities and thresholds associated with them, there can be multiple safety modes (we label them as *limp* modes) in the vehicle. The two different types of limp modes in our technique are *moderate* limp mode and *bad* limp mode.

The vehicle enters moderate limp mode when a low criticality level signal (e.g. level-3 or level-4) loses signal integrity. The function of the moderate limp mode depends on the failure that is detected. Consider the example of an accelerator pedal position sensor (PPS-A) value going out of range while the other sensor (PPS- B) is in range. In this case, the error is only associated with PPS-A, which is discarded and the vehicle is moved to moderate limp mode. The PPS-B signal is taken as the valid torque command signal, and only 50% of the total requested value is sent to the local controller. In contrast, when a higher criticality level signal (e.g., brake request at level-1) is faulty, the vehicle is moved to bad limp mode in which only 20% of the maximum allowable acceleration is commanded to the vehicle, so that the driver can safely pull off the road. In the case of multiple signal failures, depending on the criticality of the failed signal, one of these limp modes is chosen. In some cases, if multiple low level signals are faulty, a bad limp mode can be chosen

over a moderate limp mode. This can be in the case when, for example, when both the acceleration pedal position sensor values are faulty, whereby the vehicle is moved into bad limp mode and only 20% of the maximum allowable acceleration is given to the vehicle. The supervisory controller decides which limp mode the vehicle should enter in the case of a signal integrity failure to protect the powertrain and other actuators from deterioration. Figure 29 illustrates the Stateflow logic of a level-2 signal (motor torque request) performance counter and associated remedial action.

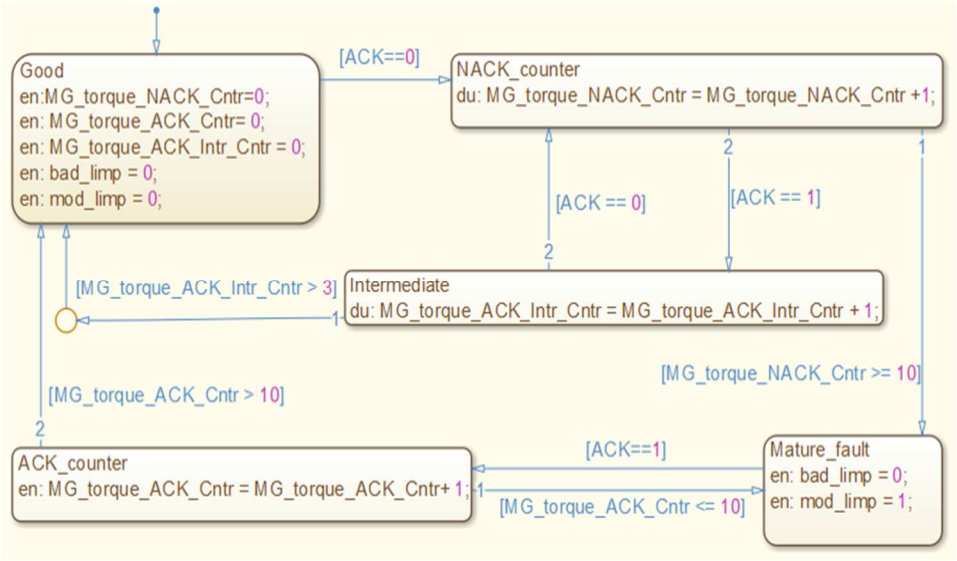


Figure 29 Stateflow logic of a level-2 signal (motor torque request) performance counter and corrective action.

In the initial Stateflow state, all the counters and the bad and moderate limp mode variables are set to zero. The vehicle remains in this state, until a NACK (ACK=0) is encountered. The controller checks for the maturity of the fault in the NACK_counter state. During this, if a series of positive acknowledgements are encountered, the vehicle goes back to normal operation state. If the fault is matured, the vehicle is moved into an appropriate limp mode. The vehicle goes back to normal operation state only after ensuring there's no erroneous transmission in the network.

3.3 EXPERIMENTAL SETUP

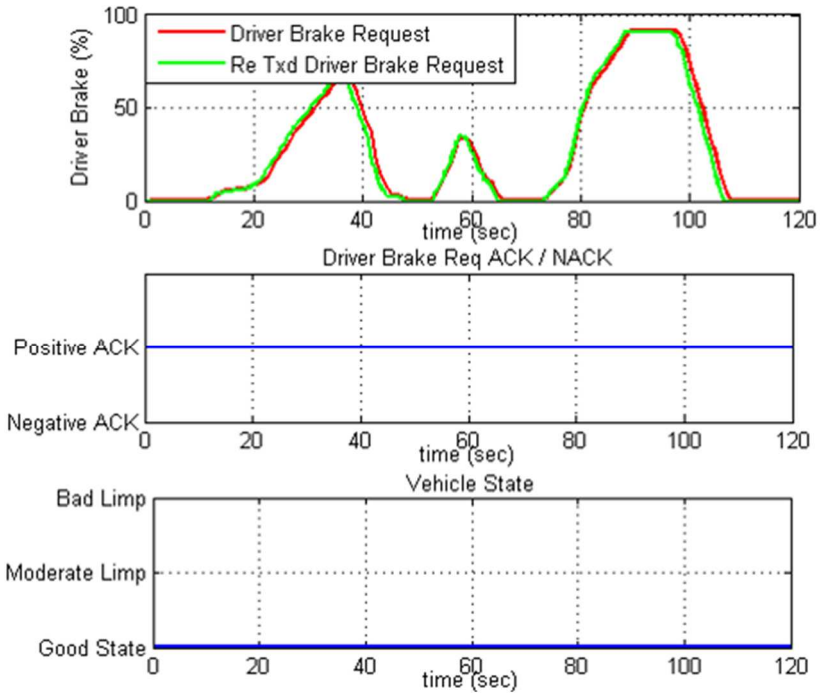
The function and performance of the proposed technique was verified using HIL testing. As a part of HIL testing, we used the Woodward Motohawk SECM-112 controller [86] as the supervisory controller. We adapted model based design approach to build a P2 type HEV fuel economy and powertrain control model using MATLAB/SIMULINK™. The sizing of various components such as electric motor, engine, battery etc. are determined so that they meet the EcoCAR3 competition requirements such as range of the vehicle, time for 0-60 mph, time for 50-70 mph etc. The control software was developed using Simulink and the Motohawk library and was tested using various driver inputs from hardware, including acceleration and brake pedals, PRNDL, ignition etc. Stateflow charts were used to implement performance counters and to define corrective action for various failures as shown in Figure 29. Since signals of different criticality levels have different time windows, their integrity testing mechanisms are triggered at a different rate. We used the dSPACE mid-size real time simulator (RTS) [87] to run the vehicle model in real time and established two channel CAN communication between the Motohawk controller and the dSPACE RTS as illustrated in Figure 28.

3.4 RESULTS

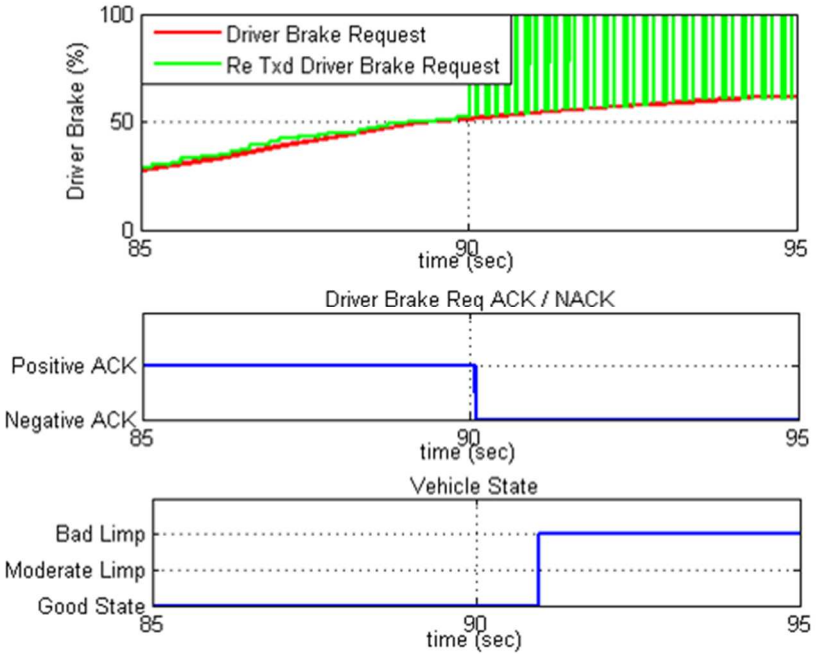
In this section we discuss the vehicle control systems' behavior with respect to the various signals subjected to the integrity test.

Figure 30(a) shows the normal operation of the vehicle with no erroneous transmissions. The first subplot in Figure 30(a) shows the driver brake command issued by the supervisory controller (in red) and the re-transmitted command received from the local controller (in green). The second and the third subplots show the acknowledgements and vehicle state respectively. Since both of

the signals in first subplot are equivalent all the time, there are no negative acknowledgements and hence, the vehicle remains in its normal operation state (Good State). In the next case, noise is introduced into the CAN channel and the driver command received at the local controller is therefore different from what is actually commanded. In this case, the supervisory controller sets a negative acknowledgement (NACK) as shown in Figure 30(b) indicating an erroneous transmission. The NACK prevents the local controller from executing the faulty torque command. Since the mismatch persists between the requested and re-transmitted signals, the vehicle is moved to a bad limp mode. It can be seen that there is a small time interval between the vehicle going to one of the limp mode and the beginning of the series of negative acknowledgements. This indicates the buffer period during which the performance counters check for the maturity of the fault.



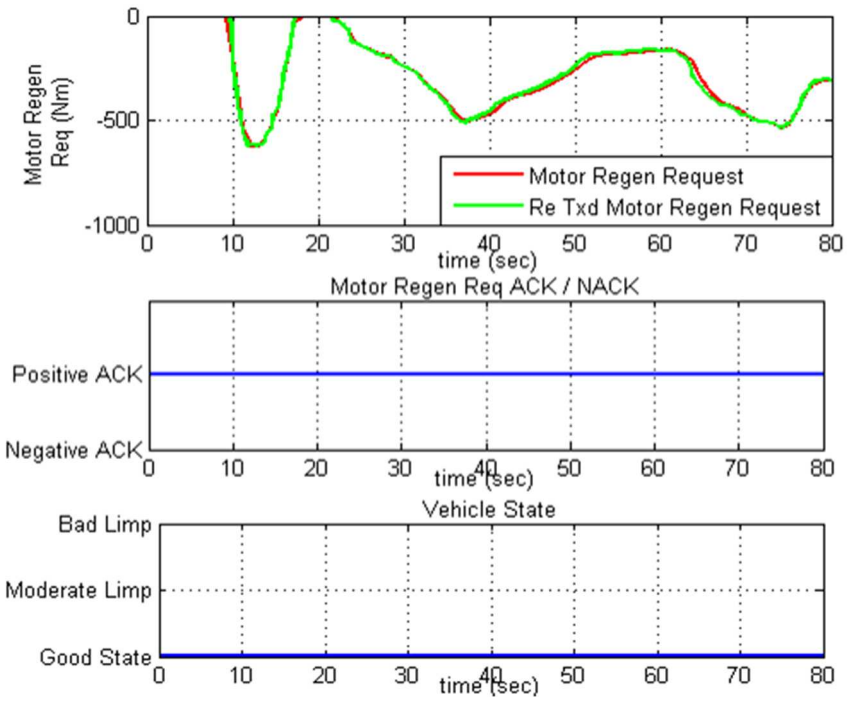
(a)



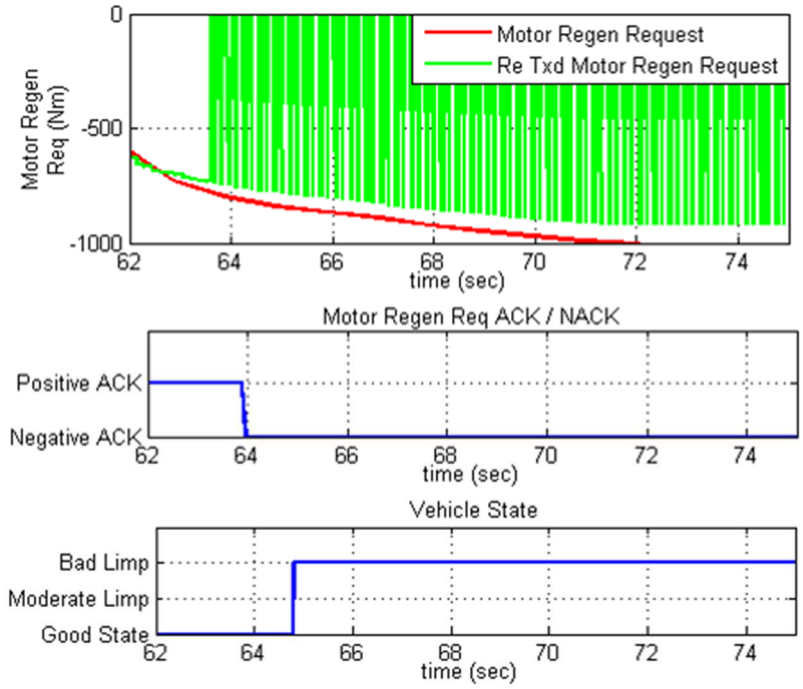
(b)

Figure 30 Signal integrity of a level-1 signal (Driver brake request) with (a) no faults in transmission; (b) noise during transmission.

Similar results are obtained for the signals at different criticality levels which are shown in figures Figure 31-Figure 34. In the case of failure of signal integrity test for level- 2 and level- 3 signals, it can be seen that the vehicle is moved into a moderate limp mode as shown in figures Figure 32(b), Figure 33(b) and Figure 34(b).

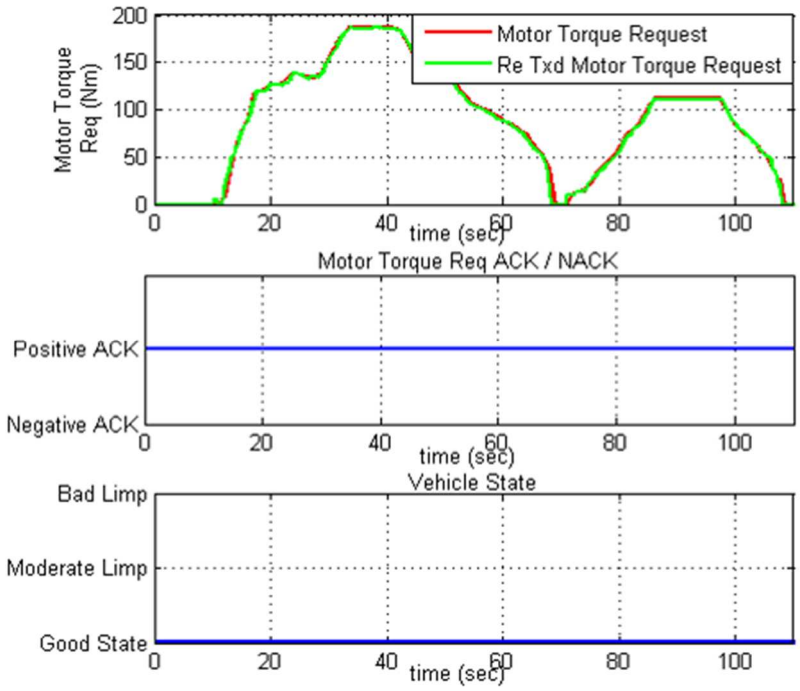


(a)

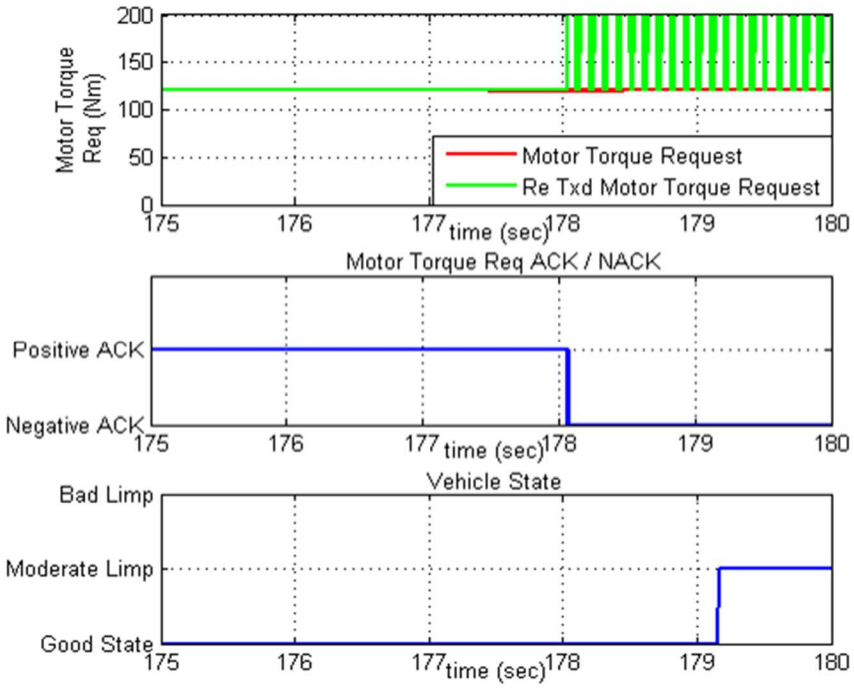


(b)

Figure 31 Signal integrity of a level-1 signal (Motor regen request) with (a) no faults in transmission; (b) noise during transmission.

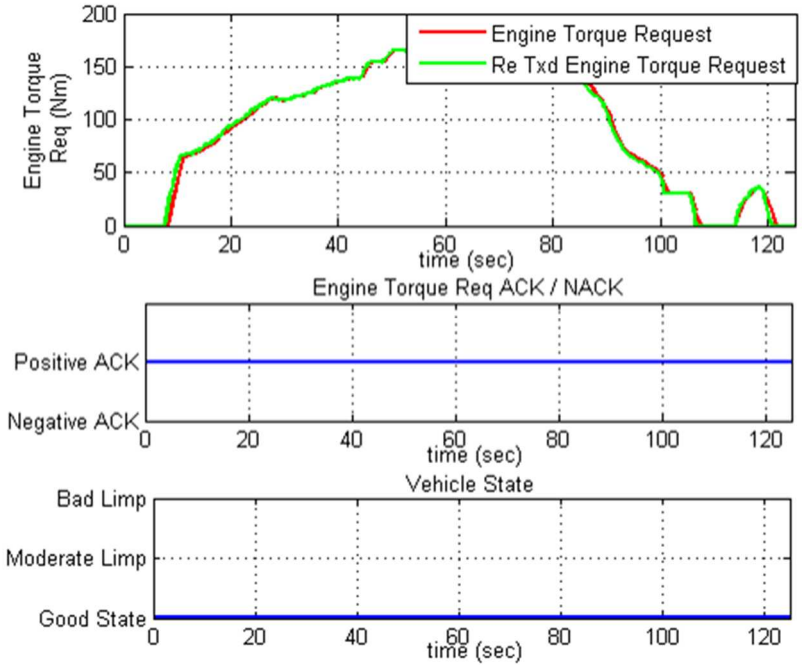


(a)

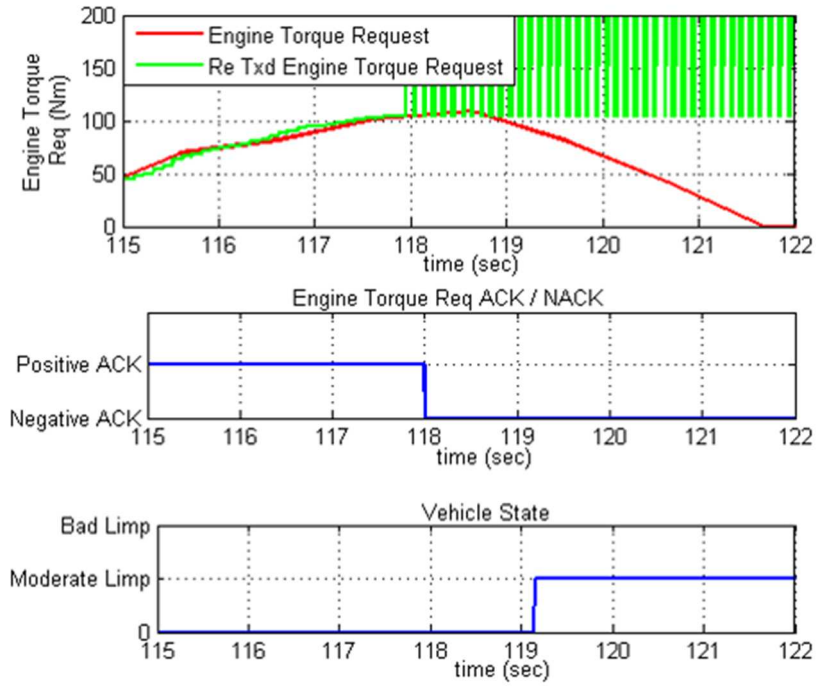


(b)

Figure 32 Signal integrity of a level-2 signal (Motor torque request) with (a) no faults in transmission; (b) noise during transmission.

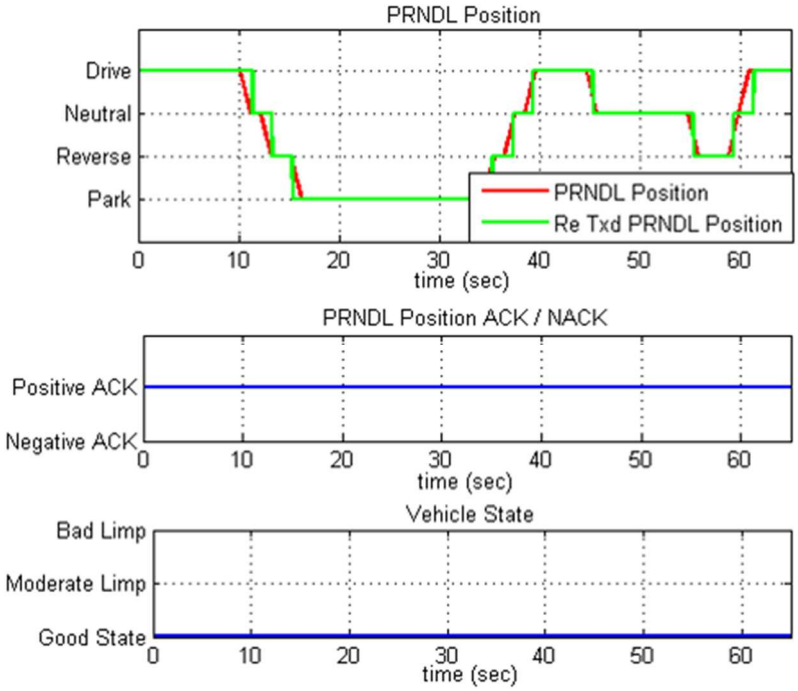


(a)

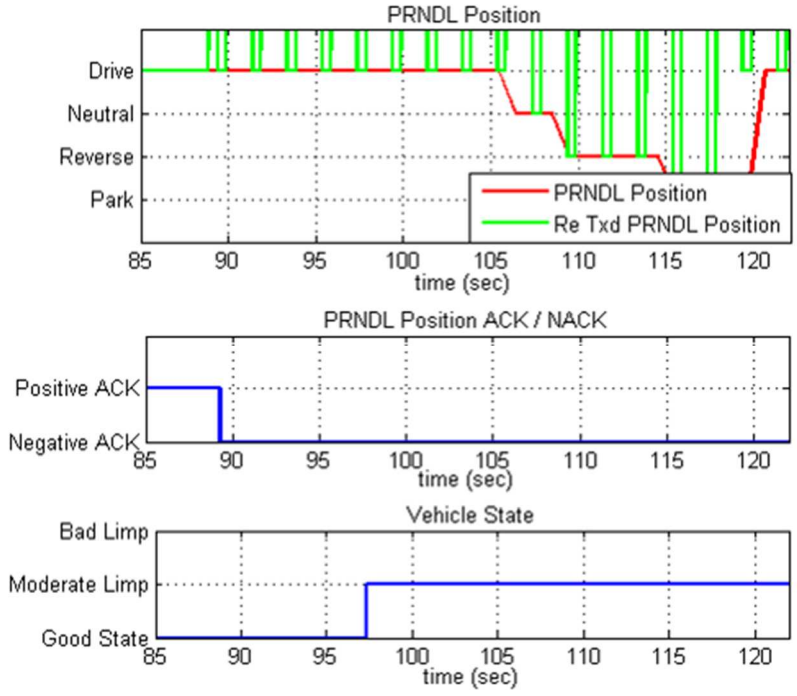


(b)

Figure 33 Signal integrity of a level-2 signal (Engine torque request) with (a) no faults in transmission; (b) noise during transmission.



(a)



(b)

Figure 34 Signal integrity of a level-3 signal (PRNDL position) with (a) no faults in transmission; (b) noise during transmission.

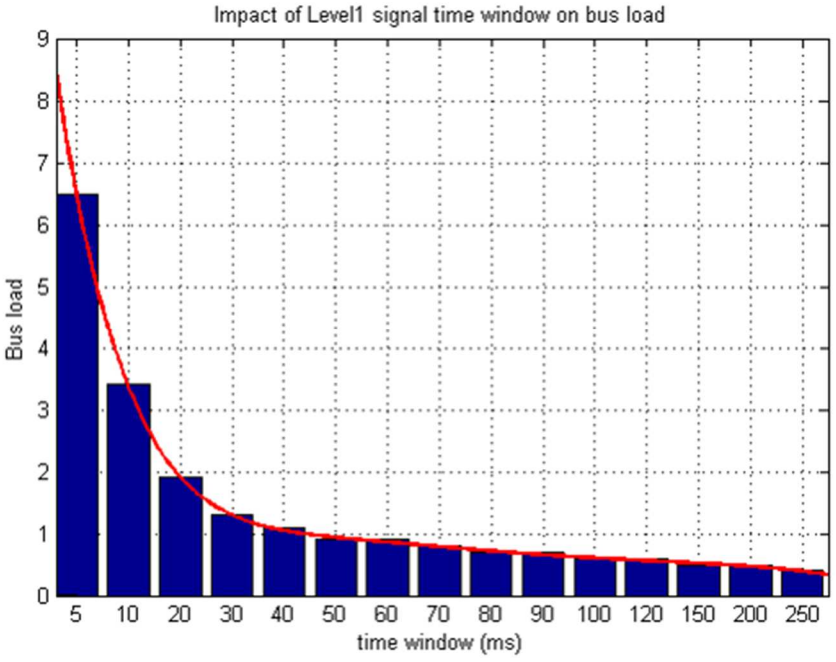
By implementing our technique, the total bus load in the control bus was found to be significantly lower and is approximately 0.5% for all the signals considered.

3.5 PARAMETER ANALYSIS

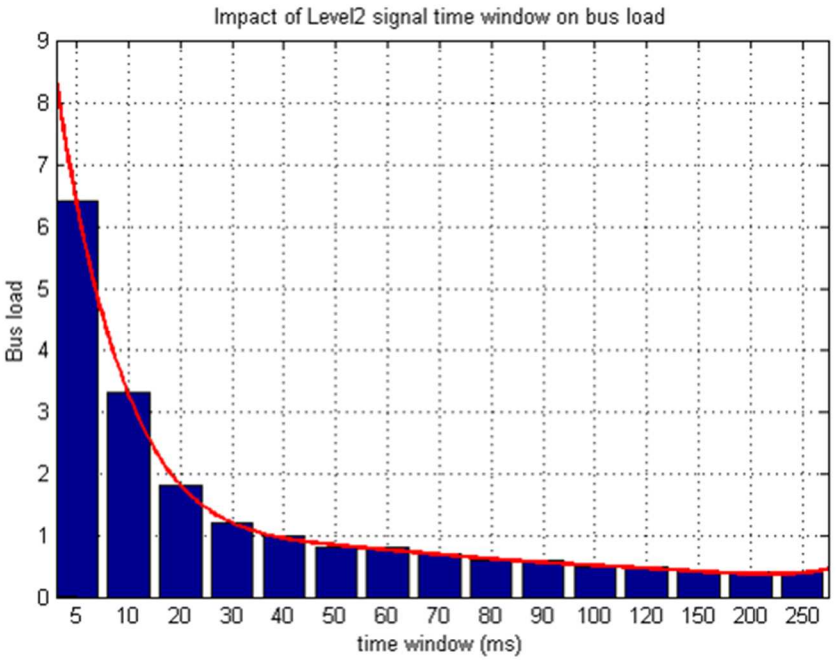
In order to determine the efficiency and overhead of our proposed technique, we carried out a detailed analysis on two key parameters as discussed below.

3.5.1. IMPACT OF TIME WINDOW SIZE ON BUS LOAD

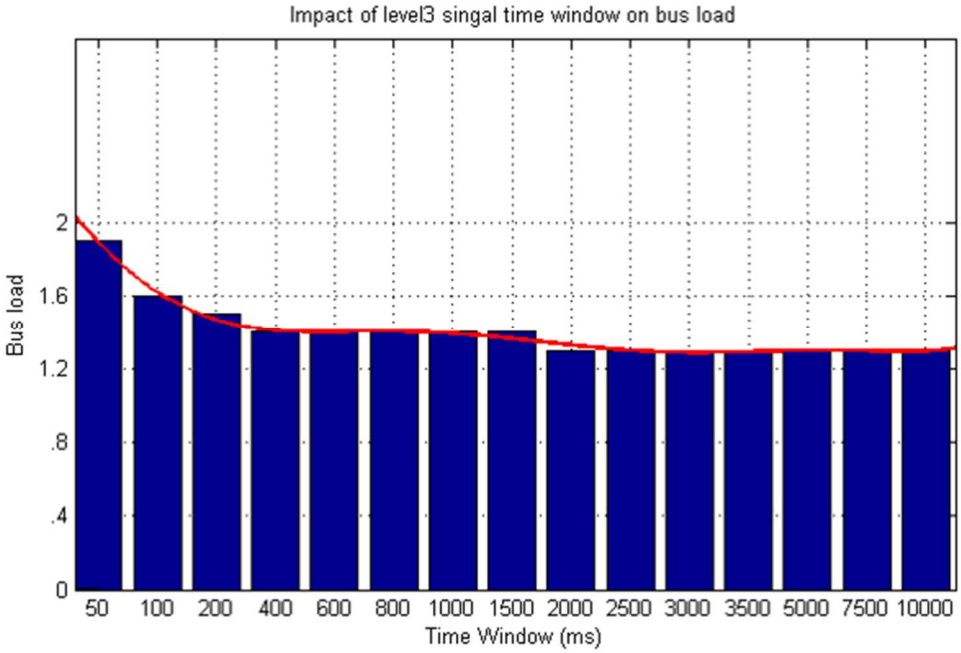
In this subsection, we present the results of how the bus load varies with different time window sizes for a particular signal level. Time window size determines the frequency of monitoring the signal. Having a higher time window size decreases the responsiveness of the vehicle to a signal integrity faulty, but a smaller time window size increases the bus load.



(a)



(b)



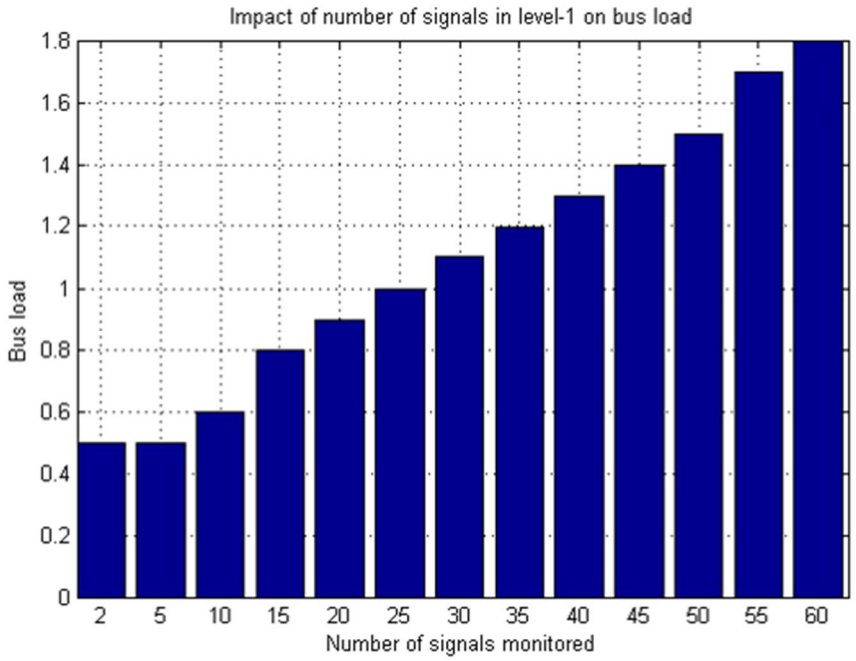
(c)

Figure 35 Bus load of control bus for varying time windows of (a) level-1 signals; (b) level-2 signals; (c) level-3 signals.

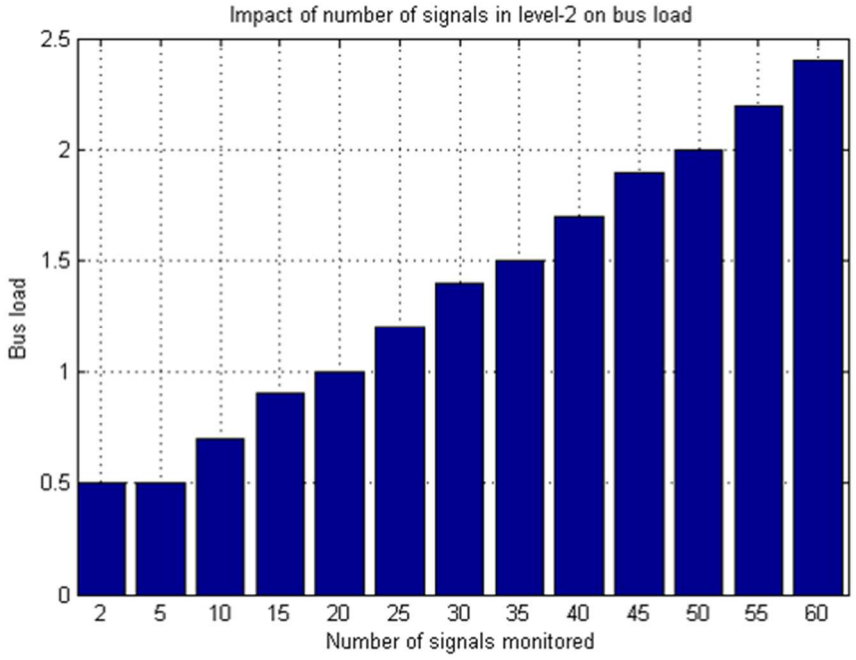
From the figures Figure 35(a), Figure 35(b) and Figure 35(c) it is evident that bus load drops almost hyperbolically with increasing time window size. Based on these tradeoffs it is important to choose the appropriate time window for different levels. Time window for different levels are chosen such that the bus load is minimal while the vehicle is being responsive. For example, in the case of level-1, signals are transmitted every 50ms and the time window size of 150ms is chosen as the drop in bus load is insubstantial after this point (as shown in fig 10(a)) and checking every third sample makes the system sufficiently responsive to failures in signal integrity. Also, from our study on Electric Power Research Institute (EPRI) databases of various HEVs it is observed that the rate of change of the accelerator pedal is higher than the rate of change of the brake pedal, PRNDL or ignition switch and hence the transmission rate of their associated signals. Since the signals that are transmitted at a higher rate should be checked more often for signal integrity, level-2 has the smallest time window, followed by level-1 and level-3. Analysis on level-4 signals is not presented as our technique is not applied to signals in this lowest level of priority. Passenger comfort, Infotainment related signals fall under this level.

3.5.2. IMPACT OF NUMBER OF MONITORED SIGNALS ON BUS LOAD

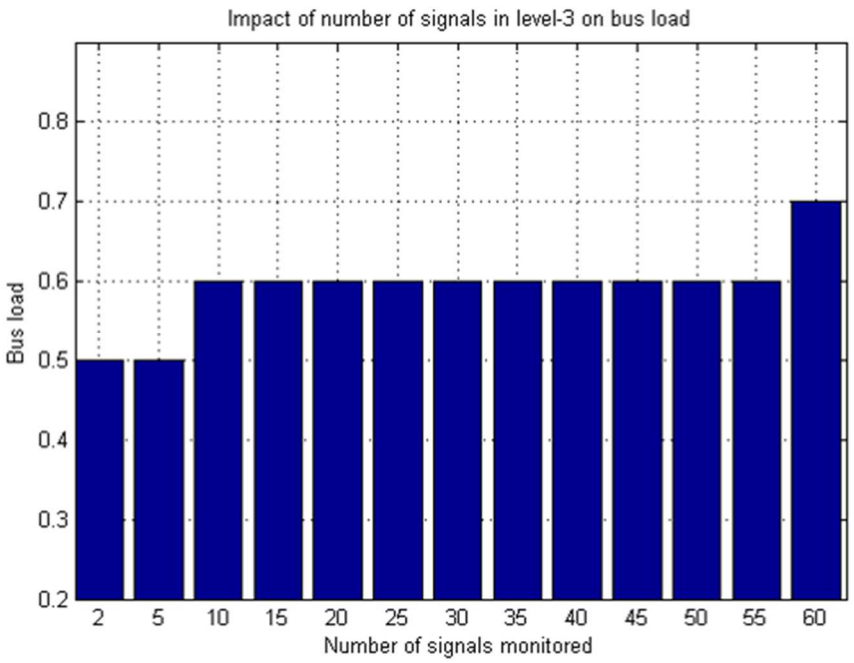
In this subsection, we discuss how the bus load is affected by the number of signals that are being monitored. This helps to understand the communication overhead of the proposed technique. The number of signals under consideration are varied for different levels and the bus load of the control bus (CB) is obtained.


Impact of number of signals in level-1 on bus load
Bus load
Number of signals monitored
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
5
10
15
20
25
30
35
40
45
50
55
60

(a)


Impact of number of signals in level-2 on bus load
Bus load
Number of signals monitored
0
0.5
1
1.5
2
2.5
2
5
10
15
20
25
30
35
40
45
50
55
60

(b)



(c)

Figure 36 Bus load of control bus for different number of (a) level-1 signals; (b) level-2 signals; (c) level-3 signals.

Figures Figure 36(a), Figure 36(b) and Figure 36(c) illustrate the change in bus load as a function of the number of signals monitored in each level by maintaining the base line values of time windows for each level (shown in table 2). It can be seen that the bus load increases at a faster rate in the case of level-2 signals than level-1 and it almost negligible in level-3 signals. This is because, the time windows defined for each of these levels dictate the rate of increase in bus load with increase in number of signals in that group. From this study, it can be seen that the overhead incurred by implementing our technique is minimal and also, the proposed technique is linearly scalable with the number of signals to be monitored.

3.6 CONCLUSION

In this chapter, we have proposed a priority based multi-level signal integrity monitoring and remediation technique which groups the controller signals based on their criticality and uses performance counters and handshake signals to monitor signal integrity. Our technique effectively handles different possible faults by changing the vehicle state to appropriate limp modes that ensure the safety of the vehicle. We verified our technique using HIL testing as a part of the CSU EcoCAR3 project and verified the signal integrity of various torque associated signals.

4. SEDAN: SECURITY-AWARE DESIGN OF TIME-CRITICAL AUTOMOTIVE NETWORKS

Modern vehicles are examples of complex cyber-physical systems with tens of interconnected Electronic Control Units (ECUs) that control various operations. The advent of Advanced Driver Assistance Systems (ADAS) in vehicles has resulted in an increase in the number of ECUs, which in turn has increased the complexity of the in-vehicle network and the entire automotive system. It is projected that in the near future, improving ADAS effectiveness will require connecting to external systems using vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) protocols [2]. This increased connectivity will make vehicles more vulnerable to sophisticated security attacks. *Ensuring the security of automotive systems will thus become crucial as smart and self-driving vehicles become more ubiquitous.*

Some of the most common attacks on vehicles include masquerade, replay, and denial of service (DoS) attacks [88]. In a *masquerade* attack, an attacker ECU pretends to be an existing ECU in the system. In a *replay* attack, the attacker eavesdrops on the in-vehicle network, captures valid messages transmitted by other ECUs, and sends them on the network in the future. In a *DoS* attack, the attacker ECU floods the network with random messages, thereby preventing the normal operation of valid ECUs. Most of these attacks require access to the in-vehicle network either physically e.g., using on-board diagnostics (OBD-II), or remotely e.g., using LTE or Bluetooth. Some efforts, e.g., [25], [26], [89], [90], have demonstrated different ways to gain access to the in-vehicle network and send malicious messages to take control of the vehicle. As wireless V2V/V2I transfers become common, vehicle security will be further compromised.

Traditional in-vehicle network protocols such as CAN, FlexRay, etc., do not have any inherent security features to address security concerns such as confidentiality, authentication, and authorization. Hence, preventing unauthorized access to the in-vehicle network requires implementing additional security mechanisms in ECUs. The two most widely used techniques involve symmetric key and asymmetric key encryption. The former uses the same key for both encryption and decryption, while the latter uses a public-private key pair that has a strong mathematical relation. *Both mechanisms incur computational overhead, which may catastrophically delay the execution of real-time automotive tasks and message transfers*, e.g., a delay in the messages from impact sensors to airbag deployment systems could lead to serious injuries for vehicle occupants. Thus, security mechanisms must be introduced very carefully in vehicles.

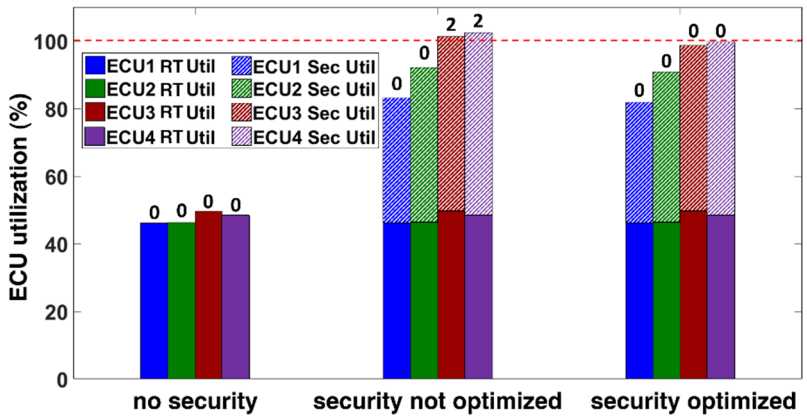


Figure 37 Motivation for an in-vehicle security framework with low overhead; numbers on top of bars indicate missed application real-time deadlines.

Figure 37 illustrates the individual ECU utilizations of a FlexRay-based automotive system consisting of four ECUs running 12 different time-critical automotive applications (each with multiple tasks). Each ECU has a utilization because of the execution of real-time automotive tasks

(*RT Util*) and a security utilization because of the execution of security-specific tasks (*Sec Util*). The numbers on top of each bar show the number of applications that miss their deadlines when executed on the corresponding ECU. Along the x-axis, the *no security* case has no security mechanism implemented, while the *security not optimized* case uses AES-256 in the ECUs for encryption and decryption of messages. In the latter case, the total utilization for ECUs 3 and 4 (sum of real-time and security task utilizations) exceeds 100% because of the overhead of security-specific encryption/decryption task execution, resulting in missed deadlines for four applications. The *security-optimized* case represents our goal in this work, to integrate all required security mechanisms while keeping utilization below 100% for all ECUs, without any deadline violations.

In this chapter, we propose a novel framework called *SEDAN* to improve security in time-triggered automotive systems with a minimal overhead on the in-vehicle ECUs. As symmetric key cryptography is less computationally intense than asymmetric key cryptography, we adapt it as part of *SEDAN* to enhance vehicle security; *however, note that the use of symmetric encryption is not the main novelty of our work*. *SEDAN* aims to maximize overall system security without violating real-time deadline constraints and per-message security constraints in the system. Our novel contributions in this work are:

- We introduce a novel methodology to derive the security requirements for the different messages in an automotive system based on ISO 26262 and an empirically derived metric to quantify the security of a system;
- We devise a meta-heuristic based key management technique to provide effective security for various message types and ensure ECU utilizations do not exceed 100%;
- We develop an approach for the joint exploration and synthesis of message schedules and security characteristics in TDMA-based automotive systems, and also propose a technique

to efficiently map tasks to ECUs while meeting real-time message deadlines and ECU utilization goals;

- We extract network traffic and ECU execution data from a real-world 2016 Chevrolet Camaro vehicle, to compare *SEDAN* with [37] which is the best-known prior work in the area, and also demonstrate the scalability of our work.

4.1 RELATED WORK

Security in automotive systems was not a major concern until recently. The first full vehicle hack in 2010 was demonstrated in [25] where the authors had physical access to the vehicle and were able to control various systems in the vehicle by injecting custom messages into the CAN bus. Moreover, they reverse engineered a subset of the ECUs and were able to update the firmware via the CAN bus. They were also able to perform the same attacks remotely [89]. The researchers in [26] hacked the radio in a 2014 Jeep Cherokee which was connected to both the CAN buses in that vehicle. They used the telematics system in the radio to send remote messages to the vehicle, which were injected into the CAN bus. Most recently in [90], the authors developed a Trojan app that was executed on a smartphone connected to the vehicle infotainment system via Bluetooth. They used this app to send custom CAN messages into the vehicle network. All of these attacks have raised serious concerns about security in automotive systems.

It is hard to prevent unauthorized access to the vehicle bus as the traditional in-vehicle network protocols do not provide any security features. However, one of the popular solutions in the literature to prevent unauthorized access is by authenticating the sender ECU using message authentication codes (MACs). Several works [91]-[95], [97], [98] advocate the use of MACs to improve security in automotive systems. A mixed integer linear programming (MILP) formulation

to minimize the overhead for MAC computation and end-to-end application latency in a CAN-based system was proposed in [92], where the same MAC was used for a group of ECUs. This work was extended in [93] to minimize the security risks associated with grouping of different ECUs. In [95] an authentication protocol called LCAP was presented to encrypt messages, with hash functions generating hashed MACs to authenticate ECUs. An RC4 encryption based authentication is implemented in [97] to improve security in CAN-based systems. Another lightweight authentication scheme based on PRESENT [96] is introduced in [98] and evaluated on FPGAs. However, cryptanalysts have demonstrated successful attacks on both RC4 and PRESENT. In [99], a technique is presented to protect a fleet of vehicles by obfuscating CAN bus message identifiers (IDs). *However, all of the above mentioned techniques are designed for event-triggered protocols (such as CAN), and are not applicable to more scalable and sophisticated time-triggered protocols.*

In [91], a lightweight authentication technique is proposed which uses cipher-based MACs that are generated using the ECU local time stamp and a secret key. However, this technique requires strong synchronization between the ECUs and any uncertainty can result in a full system failure. A device level technique is presented in [101] that uses an enhanced network interface (NI) to authenticate ECUs in the system by making use of hardware-based security modules (HSMs). In [94], FPGAs are used as co-processors for ECUs to handle all the security operations implemented based on the TESLA [100] protocol. But both techniques in [94] and [101] require additional computing resources and many modifications to the existing automotive systems, which is not cost efficient. The authors in [102] proposed a virtual local network (VLAN) based solution for improving security in Ethernet-based automotive systems. They proposed an integer linear programming (ILP) model to minimize message routing times and authenticate the messages by

making multiple message transmissions on different routes. However, this technique results in inefficient bandwidth utilization and also lacks scalability. In [103], a co-design framework is proposed to improve message response times while meeting the security concerns. However, the authors only consider encrypting a small subset of messages to guarantee control performance, which makes the system vulnerable.

An interesting framework is proposed in [37] that uses a time delayed release of keys approach (adapted from the TESLA protocol [100]) in conjunction with simulated annealing to minimize the end-to-end latency of messages by co-optimizing task allocation and message scheduling. This is one of the very few holistic frameworks that integrates the concept of security with real-time system design from the beginning of the system design phase. This work is extended in [38] by including V2V communication, using dedicated short-range communication (DSRC). A lightweight authentication technique for vehicles called LASAN is proposed in [39] that is based on the Kerberos protocol which is a popular network authentication protocol in the client-server environment. The authors extended this work in [40] by performing a detailed analysis and comparison with the TESLA [100] protocol. Though the LASAN technique demonstrated superior performance over others, it has stringent requirements for a trusted centralized ECU, which creates a single point of failure. In [41], a security mechanism using different authentication methods was proposed for real time systems. In [104], a group-based security service model is presented with a goal to maximize the combined security of the system. However, as the model ignores time-critical constraints, it cannot be implemented in automotive systems.

An intrusion detection system based on principal component analysis (PCA) is proposed in [105]. An in-vehicle network monitoring system that detects the presence of an attacker by monitoring the increased transmission rates of the messages is proposed in [106]. In [107], the

usage of reactive runtime enforcers called safety guards is proposed, to detect the discrepancies between the input data from sensors and output of the controllers. A challenge response authentication approach was proposed in [108] to detect the presence of attackers and estimate the values of the attacked signals. However, this technique requires prior (sometimes proprietary) information about the sensors and also cannot be used for passive safety sensors.

The above mentioned prior works for securing time-triggered systems have various limitations: (i) they do not consider the utilization overhead on ECUs and latency overhead on messages due to the implemented security mechanisms, which leads to over-optimistic results; (ii) they use only one key size for all messages, which ignores heterogeneous security goals in real systems; (iii) they ignore precedence constraints between tasks and messages and; (iv) they consider homogenous single core ECUs which do not accurately represent today's vehicles. In this work, we present the *SEDAN* framework that addresses these limitations of prior work. *SEDAN* improves security in vehicles with time-triggered network protocols (we demonstrate it for the FlexRay protocol, but it can be easily extended to other time-triggered protocols as well, e.g., TTEthernet), while satisfying all designer-imposed security, utilization, and message timing constraints.

4.2 PROBLEM DEFINITION

4.2.1. SYSTEM AND APPLICATION MODEL

We consider a general automotive system where multiple ECUs execute different time-critical applications and are connected using a FlexRay bus-based network, as shown in Figure 38 Each ECU consists of two major components: a host processor (HP) and a communication controller (CC). The HP is responsible for running automotive and security applications, whereas

a CC acts as an interface between the HP and the FlexRay bus, and is responsible for packing message data into frames, sending and receiving messages, and filtering unwanted messages. We consider heterogeneous HPs that have different numbers of cores, which aligns with the state-of-the-art. Note that the heterogeneity is limited to varying the number of homogeneous cores per HP (i.e., multicore parallelism).

Every automotive application consists of both dependent and independent tasks that are mapped to different ECUs and executed in the corresponding HPs. If two dependent tasks are mapped to the same ECU, they exchange information using shared memory. Otherwise, the tasks communicate with each other by exchanging messages over the FlexRay bus. A message can contain control or data signal values generated by an ECU as a result of task execution. Signals are packed into messages by the HP and are given to the CC to transmit as FlexRay frames on the bus. The automotive applications can be classified as one of two types: *(i)* time-triggered (periodic), or *(ii)* event-triggered (aperiodic). Most safety-critical applications, e.g., anti-lock braking, collision detection, etc., are time-triggered and generate time-triggered messages. Event-triggered messages are generated by maintenance and diagnostic applications. Much like real-time applications across other domains, the execution characteristics of these applications are known at design time. In this work, we focus on time-triggered applications as they have a significant impact on system performance and vehicle safety. Additionally, time-triggered messages generated by these applications have strict timing and deadline constraints. Thus, it is vital to optimize the security of the time-triggered messages while also meeting their real-time deadline constraints. We adapt state-of-the-art standards, Advanced Encryption Standard (AES) with key sizes 128,192 and 256 bits, and evaluate Rivet-Shamir-Adleman (RSA) with key sizes 512, 1024, 2048 and 4096

bits, and Elliptic Curve Cryptography (ECC) with key sizes 256 and 384 bits to improve system security.

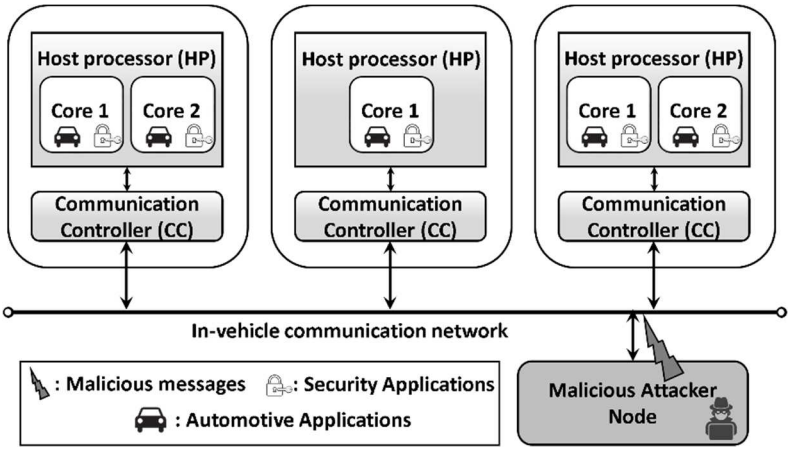


Figure 38 Overview of our assumed automotive system model.

4.2.2. ATTACK MODEL

We focus on protecting a vehicle from masquerade and replay attacks as they are most common, hard to detect, and can have a severe impact. The increased external connectivity of modern vehicles creates multiple pathways (attack vectors) to gain access to the in-vehicle network and ECUs. An attacker can employ available attack vectors to gain access to the in-vehicle network and masquerade as an existing ECU or replay valid message transmissions to achieve their goal. In our study, we considered the most common and feasible attack vectors in vehicles, which include connecting to systems that communicate with the outside world (such as infotainment systems), connecting to the OBD-II port, probe-based snooping on the vehicle bus, and replacing an existing ECU. Our framework can still be effective even when the attacker gains access to the in-vehicle network via other attack vectors. However, handling some of the advanced attack

vectors such as core tampering and hardware Trojans requires additional resources and is beyond the scope of our work.

4.2.3. SECURITY MODEL

We aim to achieve the following security objectives in vehicles: *(i)* confidentiality of message data, and *(ii)* the authentication of ECUs. Meeting these objectives can prevent masquerade and replay attacks. *Confidentiality* refers to the practice of protecting information from unauthorized ECUs, whereas *authentication* refers to the process of correctly identifying an ECU. We use AES to achieve confidentiality by encrypting message data using a shared secret key. We evaluate the choice of using RSA and ECC for setting up shared secret keys. However, it should be noted that neither RSA nor ECC is used for message encryption as they are much slower than AES. While AES with 128-bit keys (AES-128) is considered very secure today, the advent of quantum computing may challenge this assumption, hence we also consider AES-192 and AES-256. As each ECU can have messages of various criticalities, every ECU in the system can run all three variants of AES. Section 4.3.6 presents the entire encryption/decryption flow in detail. The key size for encrypting/decrypting messages is assigned based on security requirements of a message, which is discussed in Section 4.3.3.

4.2.4. DEFINITIONS

Our system model has the following inputs:

- Set of heterogeneous (1 or 2 core) ECUs $N = \{1, 2, ..., N\}$;
- Set of applications A = {1, 2, …, λ} and set of tasks in the system $T = \{T_1 \cup T_2 ... \cup T_\lambda\}$, where T_a is the set of tasks in application $a \in A$;
- Every task in T has a unique task ID $T_{ID} = \{1, 2, ..., G\}$;

- After task allocation, each task t is represented as $t_{q,n}$ where $q \in T_{ID}$ is the task ID, and $n \in N$ is the ECU to which the task t is mapped;
- Every task t is characterized by the 4-tuple $\{\tilde{a}_{q,n}, \tilde{p}_{q,n}, \tilde{d}_{q,n}, \tilde{e}_{q,n}\}$where $\tilde{a}_{q,n}, \tilde{p}_{q,n}, \tilde{d}_{q,n}, \tilde{e}_{q,n}$ denote the arrival time, period, deadline, and execution time of the task respectively;
- For each ECU $n \in N$, $S_n = \{s_{1,n}, s_{2,n} \ldots, s_{K_n,n}\}$ is the set of signals transmitted from the ECU; K_n is the total number of signals in n;
- Every signal $s_{i,n} \in S_n$, $(i = 1, 2 \ldots, K_n)$ is characterized by the 4-tuple $\{\bar{a}_{i,n}, \bar{p}_{i,n}, \bar{b}_{i,n}, \bar{d}_{i,n}\}$, where $\bar{a}_{i,n}, \bar{p}_{i,n}, \bar{b}_{i,n}, \bar{d}_{i,n}$ are the arrival time, period, deadline, and data size (in bytes) of signal $s_{i,n}$ respectively;
- After frame packing, every ECU has a set of messages $M_n = \{m_{1,n}, m_{2,n}, \ldots, m_{R_n,n}\}$, where R_n is the total number of messages in n;
- Every message $m_{j,n} \in M_n$, $(j = 1, 2, \ldots, R_n)$ is characterized by the 5-tuple $\{a_{j,n}, p_{j,n}, d_{j,n}, b_{j,n}, \Delta_{j,n}, \psi_{j,n}\}$ where $a_{j,n}, p_{j,n}, d_{j,n}, b_{j,n}, \Delta_{j,n}, \psi_{j,n}$ are the arrival time, period, deadline, data size (in bytes), and minimum security requirement of the message $m_{j,n}$ (see Section 4.3.3, respectively. $\psi_{j,n}$ is a binary variable that has a value = 1 when the security constraints of the message are satisfied. Otherwise $\psi_{j,n} = 0$;

Problem Objective: Our goal is to maximize security (aggregate security value, described in Section 4.3.3) while synthesizing a design time schedule for time-triggered tasks and messages that meet three types of constraints: *(i)* real-time deadline constraints for tasks and messages in all applications; *(ii)* minimum security constraints for each message in the system, *(iii)* ensure utilization of an ECU does not exceed 100%.

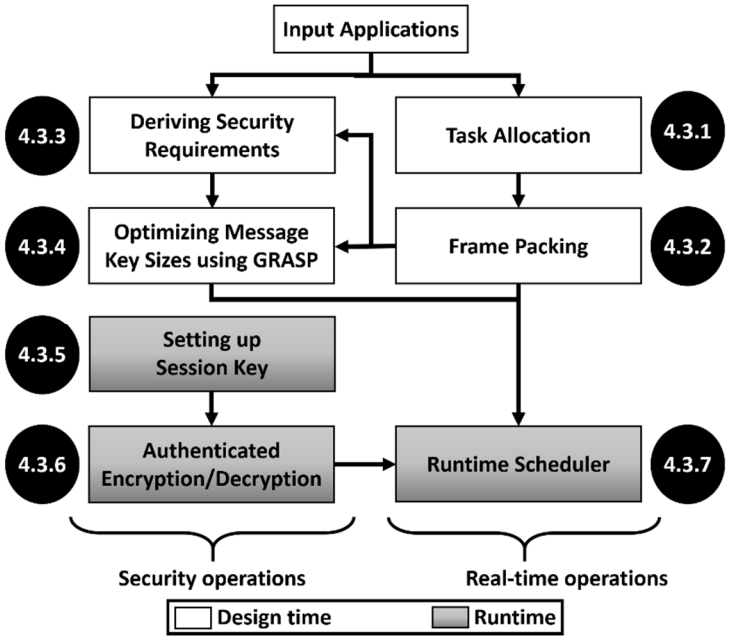


Figure 39 Overview of the proposed *SEDAN* framework.

4.3 SEDAN FRAMEWORK: OVERVIEW

A high level illustration of our *SEDAN* framework is shown in Figure 39 with all of the design time steps shown with white boxes and the runtime steps shown with gray boxes. The steps involved in *SEDAN* can be classified into two categories: *(i)* security-related operations to improve system security, and *(ii)* real-time operations to satisfy the application real-time performance objectives. At *design time*, *SEDAN* begins by allocating tasks to available ECUs in the system and generates the set of signals needed for inter-task communication. These signals are packed into messages using a frame packing approach, and security requirements are derived for each message. The size of the keys used for encryption and decryption of the messages are optimized using a greedy randomized adaptive search procedure (GRASP) metaheuristic. At *runtime*, *SEDAN* first sets up the session keys that will be used for generating keys to perform authenticated encryption and decryption of messages. A runtime scheduler then makes use of the previously generated keys

and the optimal design time schedule to schedule messages online. Each of these steps is discussed in detail in the following subsections.

4.3.1. TASK ALLOCATION

This is the first step of the *SEDAN* framework and occurs at design time. The goal here is to quickly allocate each task in the system to an available ECU resulting in a balanced real-time utilization across ECUs, which makes the *load-balancing task allocation scheme* a good choice for this step. Note that if there are some tasks that need to be allocated to certain ECUs, e.g., due to being in close proximity to sensors or actuators that they use heavily (or exclusively), we pre-allocate those tasks and do not include them in the set of mappable tasks for allocation.

For any task t_q, the real-time utilization of the task ($\widetilde{U}_{t_q}$) is defined as the ratio of execution time ($\tilde{e}_q$) and the period ($\tilde{p}_q$) of the task, as shown in (12) below. The real-time utilization of any given ECU ($\widetilde{U}_n$) is the sum of the real-time utilizations of the tasks ($\widetilde{U}_{t_{q,n}}$) allocated to that ECU, as shown in (13):

$$\widetilde{U}_{t_q} = \frac{\tilde{e}_q}{\tilde{p}_q} \tag{12}$$

$$\widetilde{U}_n = \sum_{q=1}^{G_n} \left(\widetilde{U}_{t_{q,n}} \right) \tag{13}$$

Our proposed *load-balancing task allocation* scheme begins by initializing all the ECUs' real-time utilization ($\widetilde{U}_n$) to zero and computing the real-time utilization of all the tasks' ($\widetilde{U}_{t_q}$) using (12). The allocation subsequently occurs in three steps: *(i)* the set of ECUs in the system is sorted in the increasing order of the ECU real-time utilization ($\widetilde{U}_n$); *(ii)* the first unallocated task in the set of tasks (T), sorted in decreasing order of real-time utilization, is selected and allocated

to the least loaded ECU; and *(iii)* the task's real-time utilization ($\widetilde{U}_{t_q}$) is added to the allocated ECU's real-time utilization ($\widetilde{U}_n$). These three steps are repeated until all the unallocated tasks in T are allocated. If any task $t \in T$, cannot be allocated to an ECU during this process, then there exists no solution for the given configuration. Otherwise, at the end, each task in the system is allocated to an available ECU. After the task allocation step, it is trivial to generate the set of signals S_n for each ECU, based on the precedence constraints of tasks in the application.

Note: As an alternative to a load-balancing task allocation, we also explored an allocation approach with a goal of minimizing total communication volume between ECUs. However, it resulted in non-uniform load allocation across ECUs, which led to violations in ECU utilization constraints after implementing security mechanisms.

4.3.2. FRAME PACKING

Frame packing refers to the grouping of generated signals at each ECU into messages. This is done to maximize bus bandwidth utilization. The set of signals generated by the mapped tasks on each ECU are given as the input to this step and are packed into messages based on three conditions: *(i)* for any two signals to be packed into the same message, they must originate from the same source ECU; *(ii)* signals with the same periods are packed together to avoid multiple message transmissions; and *(iii)* the total computed payload of the message is the sum of the size of the cipher generated by AES and the size of the MAC; and should not exceed the maximum possible FlexRay payload size. Because of the nature of AES, *the size of the cipher is independent of the key size used*. But it is dependent on the input size to AES, which is the sum of signal sizes grouped in that message. Thus the cipher size can be expressed as ⌈*sum of signal sizes in the message*/16⌉ and the size of MAC is set to the maximum of the minimum required MAC size (49

bits, explained further in Section 4.3.3; a designer can also use a value greater than 49). We adapted a fast *greedy frame packing* heuristic proposed in [44] by integrating the computed payload size definition to generate a set of messages to be transmitted and received, for each ECU.

4.3.3. DERIVING SECURITY REQUIREMENTS

We now present a methodology to derive security requirements for each message obtained from the output of the frame packing step. A risk classification scheme defined in ISO 26262 [14] known as the Automotive Safety Integrity Level (ASIL) is adapted to derive security requirements in our work. Four different ASILs: ASIL-A, ASIL-B, ASIL-C, and ASIL-D, are defined in the standard to classify applications based on their risk upon failure. Applications classified as ASIL-D have the lowest failure rate limit indicating high criticality, while ASIL-A applications are less critical and subject to fewer security requirements. The underlying assumption for deriving security requirements based on ASIL groups is that the applications that demand high safety levels are more critical and need to be better protected from malicious attackers (hence, higher the safety requirement, higher the security requirement).

We define two security requirements for every message based on their ASIL classification.

The first requirement is the *minimum key size* required to encrypt the message depending on its ASIL group, which is as follows: ASIL-A (128 bits), ASIL-B (128 bits), ASIL-C (192 bits), and ASIL-D (256 bits). The messages in the system are assigned ASIL groups as follows. Every application is associated with an ASIL group depending on the criticality and tolerance to failure. Each task in that application inherits the same ASIL group and so do the signals generated by these tasks. When these signals are packed into messages, the highest ASIL group among the signals in that message is assigned as the ASIL group ($m_{j,n}^{AG}$) of the message. We assign a security score

($m_{j,n}^{SS}$) to each safety-critical message depending on its assigned key size, as follows: 128 bit key (score=1), 192 bit key (score=2), and 256 bit key (score=3). Additionally, each message is assigned a weight value called *ASIL weight* ($m_{j,n}^{AW}$). A high *ASIL weight* value indicates a high message criticality and is analogous to a Risk Priority Number (RPN) that can be calculated using Hazard Analysis and Risk Assessment (HARA) approaches [111]. Using these metrics that we have defined above, a *security value* ($m_{j,n}^{SV}$) is derived for each message as shown in (14) below. The overall security of the system can then be quantified using an empirically derived metric called *Aggregate Security Value (ASV)*, as shown in (15):

$$m_{j,n}^{SV} = m_{j,n}^{AW} * m_{j.n}^{SS} \tag{14}$$

$$Aggregate\ Security\ Value\ (ASV) = \frac{\sum_{n=1}^{N}\sum_{j=1}^{R_n}\left(\psi_{j,n} * m_{j,n}^{SV}\right)}{\sum_{n=1}^{N} R_n} \tag{15}$$

where $\psi_{j,n}$ and R_n are defined in Section 4.2.4. ASV is essentially the ratio of the sum of security values of all messages in the system for which minimum security requirements are satisfied, to the total number of messages in the system. *ASV can be used to compare the security of multiple systems using the same encryption standard. A system with a higher ASV value is more secure than a system with a lower value.*

The second requirement is the minimum number of Message Authentication Code (MAC) bits required for a message based on the assigned ASIL group. This is derived using the failure rate limit of the ASIL group of the message. The failure rate limit is usually expressed as FIT (Failure in Time), which denotes the maximum number of acceptable failures per 1 billion hours of usage. According to specifications, ASIL-D has 10 FIT, ASIL-B and C have 100 FIT, and ASIL-A has 1000 FIT as their maximum limits. In other words, ASIL-D applications need to have less

than 10^{-8} failures per hour while ASIL-A applications can have up to 10^{-5} failures per hour. The security requirements for each message in the system are then derived as follows:

- Consider a message $m_{j,n}$ with period $p_{j,n}$ (in milliseconds)
- Number of transmissions of $m_{j,n}$ per second are $(10^3/p_{j,n})$
- Number of transmissions of $m_{j,n}$ per hour are $((3600*10^3)/p_{j,n})$
- If there are k bits in the MAC field of a message, the *probability of failure due to an attacker guessing a valid MAC* (e.g., using brute-forcing or other methods) is 2^{-k} for one transmission of that message;
- Thus the probability of failure due to a compromised MAC for an hour-long transmission is $((3600*10^3)/p_{j,n})*2^{-k}$
- For an ASIL-D application, the probability of failure needs to be less than 10^{-8} per hour, i.e., $((3600*10^3)/p_{j,n})*2^{-k} \leq 10^{-8}$
- Thus, the minimum number of MAC bits ($\Delta_{j,n}$) required for the message ($m_{j,n}$) according to the ASIL-D requirement is:

$$\Delta_{j,n}(D) = k \geq \left\lceil Q + \log_2\left(\frac{1}{p_{j,n}}\right)\right\rceil \quad (16)$$

where constant Q has a value of 48.35 for ASIL-D. Similarly, the minimum number of MAC bits required ($\Delta_{j,n}$) for other ASIL groups are calculated using (16) by using Q=45.04 for ASIL-B and ASIL-C, and Q=41.72 for ASIL-A. The different values of Q for each ASIL group are computed based on the FIT limit corresponding to that ASIL. Thus, for an ASIL-D message, for the most stringent (smallest) period we observed (=1ms), $\Delta_{j,n}(D)$ = 49 bits (thus this is used in frame packing).

4.3.4. OPTIMIZING MESSAGE KEY SIZES USING GRASP

This is the last step of the design time process. The goal here is to assign an optimal key size for each message in the system that maximizes the ASV while meeting the security requirements and real-time deadline constraints. Additionally, in this work, we model the overhead caused by the security applications (i.e., encryption and decryption) in terms of the additional ECU utilization (security-induced utilization) and latency (response time) of the message. For any message ($m_{j,n}$), encrypted or decrypted using a block cipher, the security-induced ECU utilization ($\overline{U}_{m_{j,n}}$) due to the message is:

$$\overline{U}_{m_{j,n}} = \left(\left\lceil \frac{b_j}{b_{size}} \right\rceil * \frac{T_{encr/decr}}{p_j} \right) \tag{17}$$

where b_{size} denotes the block size in bytes and $T_{encr/decr}$ represents the time to encrypt or decrypt one block of data. As AES is the encryption algorithm used in this work, the above equation can be re-written as:

$$\overline{U}_{m_{j,n}} = \left(\left\lceil \frac{b_j}{16} \right\rceil * \frac{T_{AES(X)}}{p_j} \right) \tag{18}$$

where $T_{AES(X)}$ is the time to encrypt or decrypt one block (16 Bytes) of data using AES with an X bit long key (where X can be 128, 192 or 256). The security-induced utilization of any given ECU ($\overline{U}_n$) is the sum of the security-induced utilizations due to all transmitted and received messages ($\overline{U}_{m_j}$) for that ECU, as shown in (19) below. Thus, for an ECU n, its total utilization (U_n) is the sum of the real-time utilization ($\widetilde{U}_n$) and security-induced utilization ($\overline{U}_n$) as shown in (20):

$$\overline{U}_n = \sum_{j=1}^{R_n} \overline{U}_{m_{j,n}} \tag{19}$$

$$U_n = \widetilde{U}_n + \overline{U}_n \tag{20}$$

To avoid undesirable latency overheads and uncertainty, we always constrain the utilization for any ECU to be below 100%.

A *greedy randomized adaptive search procedure* (*GRASP*) metaheuristic [109] is developed and utilized to achieve this goal. An overview of this approach is illustrated in Figure 40. The optimal message key size allocation step takes the set of messages from the output of frame packing (Section 4.3.2) and the derived security requirements (Section 4.3.3) as inputs. An *initial solution* is generated by assigning the minimum required key sizes for all the messages based on the derived security requirements. A feasibility check is performed later to determine the feasibility of the initial solution. The feasibility check investigates the *(i)* total ECU utilization (U_n) for all ECUs and *(ii)* number of missed deadlines using a design time scheduler. In this work we adapt the fast design time scheduling heuristic proposed in [44] to synthesize an optimal design time schedule. If there are no utilization violations at any ECU ($U_n \leq 100\%\ \forall$ ECUs) and deadline misses for any message, the initial solution is given as the input to the GRASP metaheuristic. If any of the above mentioned conditions fail, the optimal message key size allocation step terminates and the system does not have a feasible solution. GRASP explores various design time schedule configurations (assigning messages and ECUs to FlexRay static segment slots) and message key sizes (that are greater than or equal to the minimum key size requirement for a message) to select a solution that maximizes ASV, with no security violations, real-time deadline misses, and without exceeding 100% utilization for any ECU.

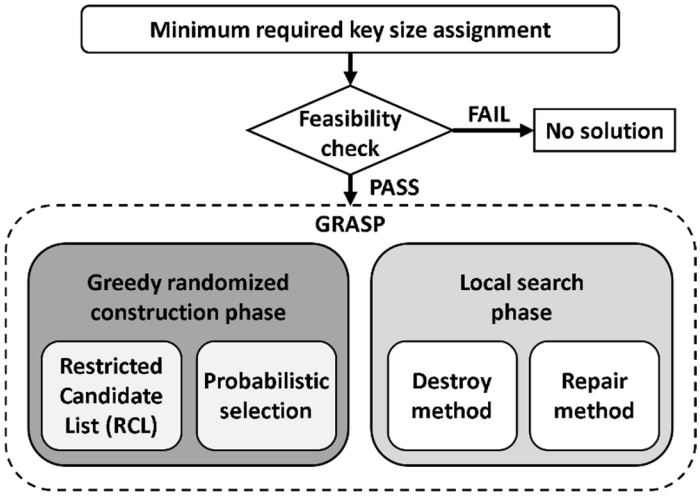


Figure 40 Overview of optimal message key size allocation step using GRASP.

The GRASP metaheuristic is an iterative process in which each iteration has two major phases: *(i) greedy randomized construction phase* that tries to build a local feasible solution and *(ii) local search phase* that tries to investigate the neighborhood for a local optimum. At the end, the best overall solution is chosen as the final solution. Two important aspects of the greedy randomized construction phase are the *greedy aspect* and *probabilistic aspect*. The greedy aspect involves generating a Restricted Candidate List (RCL), which consists of best elements that will improve the partial solution (solution within the greedy randomized construction phase). The random selection of an element from the RCL, to be incorporated into the partial solution, is the probabilistic aspect. The solutions generated during the greedy randomized construction phase are not necessarily optimal. Hence a local search phase is used to improve the constructed solution. The local search is an iterative process that uses destroy and repair mechanisms to search for local optimum within a defined neighborhood. If an improved solution is found, then the best solution is updated.

Algorithm 6: GRASP based optimal message key size assignment

Inputs: Set of nodes (*N*), Set of all messages (*M*), *init_solution*, *max_iterations*, RCL threshold (α), and destroy-repair threshold (β)

```
1:  best_solution ← init_solution
2:  for iteration = 1, …, max_iterations do
3:  |   current_solution ← greedy_randomized_construction(α, N, M)
4:  |   current_solution ← local_search(β, N, M, current_solution)
5:  |   if current_solution > best_solution do
6:  |   |   best_solution ← current_solution
7:  |   end if
8:  end for
```

Output: Optimal message key sizes for every message that results in maximum ASV and a feasible design time schedule with no deadline misses, security violations and utilization of all ECUs below 100%.

Algorithm 6 presents an overview of our GRASP based optimal message key size assignment where the inputs are: set of nodes (N), set of all the messages in the system (M), ASV of the system and minimum required message key size assignment (*init_solution*) which is the initial solution given to GRASP to reduce the search space, maximum iterations (*max_iterations*), RCL threshold (α), and a destroy-repair threshold (β). The algorithm begins by assigning the *init_solution* to the *best_solution* (step **1**). GRASP iteratively tries to find a better solution (steps **2**-**8**) until *max_iterations* is reached. In each iteration *greedy_randomized_construction()* (step **3**) generates a local feasible solution (*current_solution*) which is updated using *local_search()* (step **4**). If a better solution is found at the end of local search phase, the *best_solution* is updated (steps **5**-**7**). The output of the algorithm is an optimal message key size for every message and a feasible design time schedule with no deadline misses, no security violations, and with utilization for each ECU in the system below 100%. *Note*: Every solution in GRASP consists of two attributes *(i)* key sizes for all the messages and *(ii)* ASV of the system as a result of the key size assignment. Every solution generated by GRASP ensures that no message is allocated a key size less than the key

size assigned in the initial solution and the overall system ASV is always greater than the ASV of the initial solution.

4.3.4.1 GREEDY RANDOMIZED CONSTRUCTION PHASE

The greedy randomized construction phase tries to generate a feasible solution in every iteration of GRASP by increasing the key sizes of some of the non ASIL-D messages with an aim to maximize the ASV of the system without violating deadline, security, and ECU utilization constraints. It also ensures that no message is allocated a key size less than the key size allocated in the initial solution (minimum required key size). The solution generated by the greedy randomized construction phase will be given as the input to the local search phase for refinement.

Algorithm 7: greedy_randomized_construction (α, N, M)

Inputs: RCL threshold (α), set of nodes (N), and set of all messages (M)

1: $\widetilde{M} \leftarrow \{m \in M \mid m^{AG} \neq \text{ASIL-D}\}$
2: **Increment** m^{SS} by 1 $\forall\, m \in \widetilde{M}$ and compute m^{SV}
3: **Sort** $\widetilde{M}$ in the increasing order of m^{SV}
4: **while** $\widetilde{M} \neq \{\ \}$ **do**
5: | $SV_{min} = \min(\{m^{SV}\ \forall\, m \in \widetilde{M}\})$
6: | $SV_{max} = \max(\{m^{SV}\ \forall\, m \in \widetilde{M}\})$
7: | $RCL \leftarrow \{m \in \widetilde{M} \mid m^{SV} \geq SV_{min} + \alpha * (SV_{max} - SV_{min})\}$
8: | $\overline{m} \leftarrow$ random element from RCL
9: | **Increment** the key size of $\overline{m}$ to the next higher key size
10: | **if feasibility_check**( ) == *fasle* **do**
11: | | **Revert** the key size of $\overline{m}$ back to its previous key size
12: | | **Decrement** m^{SS} by 1 for $\overline{m}$ and compute m^{SV}
13: | **end if**
14: | **Remove** $\overline{m}$ from $\widetilde{M}$
15: **end while**
16: *current_solution* $\leftarrow$ {**calculate_ASV**( ), message key size assignment}

Output: Local feasible solution that results in a feasible schedule with no deadline misses, security violations and utilization of all ECUs below 100%.

Algorithm 7 shows the pseudocode of the greedy randomized construction phase where the inputs are: set of nodes (N), set of messages (M), and RCL threshold (α). A set of non ASIL-D

messages ($\widetilde{M}$) is created in step **1**. The security score of each message (m^{SS}) in $\widetilde{M}$ is incremented by one and the security values of the messages (m^{SV}) are updated using (14) (step **2**). In step **3**, $\widetilde{M}$ is sorted in the increasing order of m^{SV} and the ties are resolved based on the message period. In steps **4**-**15**, the algorithm tries to find a local solution by incrementing key sizes for some of the messages that would result in no deadline, security, and ECU utilization violations. In steps **5**, **6** the minimum (SV_{min}) and maximum (SV_{max}) security values are computed respectively. The *RCL* consists of messages in $\widetilde{M}$, that will result in increased ASV when their key size is incremented. The messages whose security value (m^{SV}) is within the interval [$SV_{min} + \alpha\ (SV_{max} - SV_{min})$, SV_{max}] are added to the *RCL* in step **7** (this is the greedy aspect of greedy randomized construction). The quality of *RCL* is regulated using an RCL threshold ($\alpha \in [0, 1]$). The threshold (α) controls the amount of greediness and randomness in the algorithm. The case $\alpha = 0$ corresponds to a pure random approach, while $\alpha = 1$ is equivalent to pure greedy approach. In step **8**, a random message ($\overline{m}$) is selected from the *RCL* (probabilistic selection) and its key size is incremented to the next higher key size in step **9** (i.e., 128 → 192 or 192 → 256). The *feasibility_check()* in step **10**, checks for any *(i)* ECU utilization violations (i.e., any ECU utilization > 100%) and *(ii)* deadline misses using the design time scheduling heuristic proposed in [44]. If any of them fails, the *feasibility_check()* returns *false*, and reverts the key size of ($\overline{m}$) back to its previous key size (step **11**) and re-computes m^{SV} of $\overline{m}$ after decrementing the m^{SS} by one (step **12**). Otherwise, the key size increment is left unchanged. The message ($\overline{m}$) is removed from $\widetilde{M}$ and the steps **5**-**14** are repeated until there are no messages left in $\widetilde{M}$. Lastly, in step **16**, the ASV of the system and the current message key size assignment are assigned to the *current_solution*. The function *calculate_ASV()* is implemented using (15).

4.3.4.2 LOCAL SEARCH PHASE

The local search phase iteratively improves the solution found in the greedy randomized construction phase by investigating a defined neighborhood in the problem space. This is achieved by using *destroy* and *repair* methods which remove a part of the solution and recreate a feasible solution, respectively. In this work, we define the neighborhood as the set of solutions that are generated by randomly changing key sizes for β number of messages. The parameter β known as the destroy-repair threshold specifies how much to destroy or repair in each iteration of the local search. These random changes in message key sizes help in recovering from suboptimal ordering (sorting in the increasing order of m^{sv}) of messages in the greedy randomized construction phase.

Algorithm 8 shows the pseudocode of the local search procedure. The *destroy()* function (steps **1**-**4**) randomly selects a message from the set of messages that are allocated a key size higher than the minimum required key size, and decreases the key size to the next smaller key size. *min_score()* returns the minimal security score demanded by the assigned ASIL group. *repair()* (steps **5**-**18**) aims to increase the key size for β non ASIL-D messages and computes *local_solution()*. Every time a message needs to be selected for incrementing the key size in *repair()*, the one message that results in maximum increase in the ASV of the system is selected (step **8**). Ties are resolved based on the ASIL group and if multiple messages have the same ASIL group, one message is selected at random.

The local search algorithm iteratively explores the neighborhood around the *current_solution* using *destroy()* and *repair()* to find a better solution (steps **19**-**29**). In each iteration, β is chosen randomly from [2, β_{max}] (step **20**). *destroy()* is modeled as a stochastic process which is controlled by the key decrease probability (p_{kd}) (steps **21**-**24**). Lastly, the *current_solution* is updated if a better *local_solution* is found in the repair method (steps **25**-**28**). In each iteration

of GRASP, at the end of local search phase a local optimum is found if there exits one. Otherwise, the solution remains unchanged from the greedy randomized construction phase.

Algorithm 8 : local_search(*β*, *N*, *M*, *current_solution*)

Inputs: Destroy-repair threshold (β), set of nodes (*N*), set of all messages (*M*), and *current_solution*

function *destroy* (*M*)
  $M_d = \{m \in M \mid m^{SS} > \textbf{min\_score}(m^{AG})\}$
  Decrement the key size of a random message ($\bar{m}$) in M_d
end function

function *repair* (*β*, *M*, *N*)
  $M_r = \{m \in M \mid m^{AG} \neq$ ASIL-D$\}$
  while ($\beta > 0$) or ($M_r \neq \{\ \}$) **do**
    $\bar{\bar{m}} = \{m \in M_r \mid \Delta$ASV is maximum$\}$
    Increment the key size of message ($\bar{\bar{m}}$)
    if feasiblity_check( ) == *false* **do**
      Revert the key size of ($\bar{\bar{m}}$) back to previous key size
    else do
      $\beta = \beta - 1$
    end if
    Remove ($\bar{\bar{m}}$) from M_r
  end while
  return {**calculate_ASV( )**, message key size assignment}
end function

for *local_iteration* = 1,…, *max_local_iterations* **do**
  β = **random_integer**(*2*, β_{max})
  if p_{kd} > **random**(0,1) **do**
    destroy (*M*)
    $\beta = \beta - 1$
  end if
  local_solution ← **repair** (*β*, *M*, N)
  if *local_solution* > *current_solution* **do**
    current_solution ← *local_solution*
  end if
end for

Output: Local optimum with in the defined neighborhood- if there exists one; Otherwise, same solution as greedy_randomized_construction( ).

Note: When the message key size is changed, the size of the output cipher and MAC (or the message size) remains unchanged. The key size only affects the time taken to encrypt/decrypt the message and the security induced utilization of the sender and receiver ECUs. The real-time task set induced utilization of the ECUs also remains unchanged, as the time-triggered task execution times do not change with changing message key sizes.

4.3.5. SETTING UP SESSION KEY

This is the first step at runtime in the *SEDAN* framework. It involves settings up session keys required for generating keys that will be used for the encryption and decryption of messages. In general, *using the same key every time for encryption and decryption for the entirety of the vehicle lifetime makes the system highly vulnerable*. Therefore, during runtime, we generate a new key for every session (called session key).

A session is defined as the time duration between the start of a vehicle to turning off the vehicle. As we use symmetric key encryption, all ECUs in the system need the same secret key to function properly. As traditional automotive networks do not have any inbuilt security features, the major challenge here is in exchanging the session keys between ECUs over an unsecure channel. We adapt the Station-to-Station (STS) key agreement protocol [110] which is based on the famous Diffie-Hellman key exchange method [111] to the automotive domain (as simple Diffie-Hellman is vulnerable to man-in-the-middle attacks), to securely transfer session keys between ECUs over an unsecured FlexRay bus. Moreover, within the STS protocol, we utilize elliptic curve cryptography (ECC) operations as the basis for key agreement instead of RSA, as the former is faster and has lower memory footprint for the same level of security compared to the

latter (as discussed in Section 4.4.2). The STS protocol with ECC is illustrated in Figure 41, and discussed below.

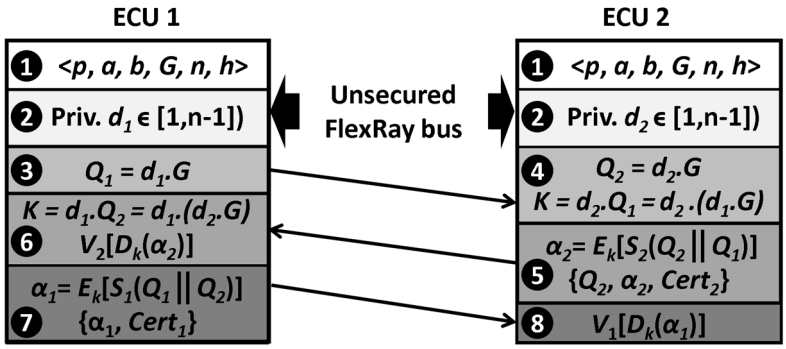


Figure 41 Steps involved in setting up a session key using the STS protocol using ECC operations over an unsecure FlexRay bus.

The approach begins with two ECUs agreeing upon a set of parameters known as domain parameters that define the elliptic curve. In the first step in Figure 41, the parameter p defines the field, a and b define the elliptic curve, G is the generator and n is its order, and h is the co-factor. Additionally, each ECU has an asymmetric key pair used for authentication (sign and verify). In the second step, each ECU generates a random private number (d_1 in ECU 1 and d_2 in ECU 2), which is not shared with any other ECU in the system. In the next step (step 3), one of the ECUs (e.g., ECU1) performs an elliptic curve scalar multiplication (hereafter referred to as scalar multiplication) of the private number d_1 and generator G. The output Q_1 is transmitted to ECU2 over an unsecured FlexRay bus. A similar scalar multiplication (between d_2 and G) is performed at ECU2 but the output Q_2 is not sent to ECU1 (step 4). Additionally, ECU2 also computes the scalar multiplication of the private number d_2 and the received output Q_1 resulting in the common secret key K (session key). In the next step (step 5), ECU2 computes the signature ($S_2()$) of the concatenation of Q_2 and Q_1 (represented as $Q_2 \| Q_1$) using its private key of the asymmetric key pair. The output signature is encrypted ($E_k()$) using the computed session key from the previous

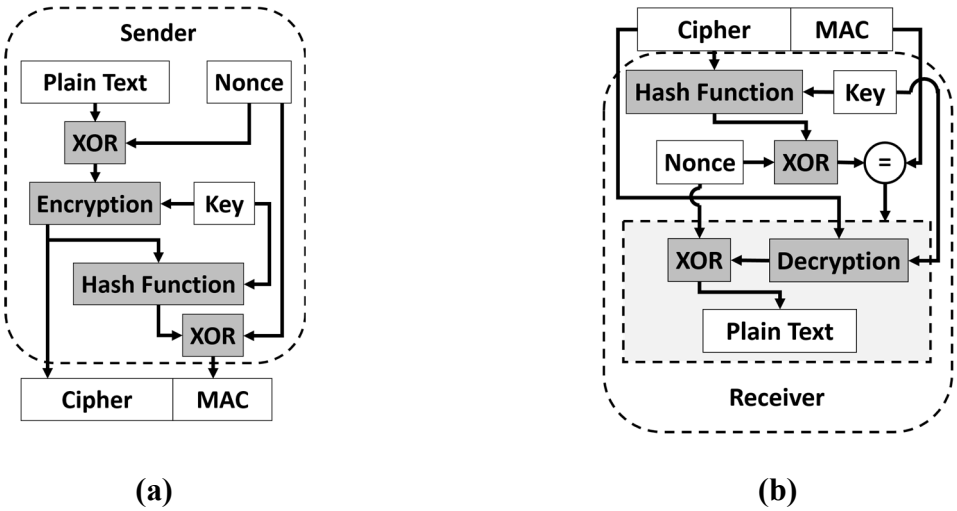
step resulting in the cipher α_2. The scalar multiplication output (Q_2), output cipher (α_2), and the certificate ($Cert_2$) are all transmitted to ECU1 over the unsecured FlexRay bus. The certificate is used to prove the ownership of a public key, which is issued by a trusted certificate authority (CA) and programmed in the ECUs by the manufacturer. It consists of the public key of the owner and signature of the CA. The public key of the CA is used to verify the certificate and extract the public key of the owner. When the ECU1 receives them (in step 6), it performs a scalar multiplication of private number d_1 and Q_2 to produce the shared secret key K (session key). Moreover, ECU1 utilizes the key K to decrypt ($D_k()$) the received cipher (α_2) and verifies ($V_1()$) the decrypted output using the public key extracted from the certificate of ECU2 ($Cert_2$). ECU1 agrees to use the key K for a session only when the verification is successful thereby authenticating ECU2. In the next step (step 7), ECU1 computes the signature ($S_1()$) of the concatenation of Q_1 and Q_2 (represented as $Q_1 \parallel Q_2$) using its private key of the asymmetric key pair. The output is encrypted using the key K resulting in the cipher (α_1) which is transmitted to ECU2 along with the certificate ($Cert_1$). Lastly (in step 8), at ECU 2, the received cipher (α_1) is decrypted using the key K and the output is verified using the public key extracted from certificate of ECU1 ($Cert_1$). The key K is accepted to use for the session only when the verification is successful. Thus, all the ECUs are authenticated and a common secret key (session key) is established at every ECU without actually exchanging the key over the bus. Additionally, the STS protocol uses no timestamps and provides a perfect forward secrecy. This session key is used to generate 128-bit, 192-bit and 256-bit keys using a standard AES key schedule at every ECU. These resulting keys are used for encrypting and decrypting messages at runtime. Moreover, in order to avoid interference with the time-critical messages, the messages related to the security operations utilize a small number of reserved FlexRay frames.

Note: Even if there was an attacker already in the system during the key setup phase, the attacker cannot compute the secret key with the publicly available results due to the discrete logarithm problem [112]. Moreover, the common type of man-in-the-middle attack that has been performed on the simple Diffie-Hellman approach [113] fails with STS as the attacker cannot authenticate successfully. To speed up the startup process, we assume that the manufacturer pre-programs some of the session keys during manufacturing. New keys are generated continuously during the idle time of an ECU, saved in local memory, and used in future sessions. To further speedup this process, the public keys of the trusted ECUs can be pre-programmed in the ECU's tamper proof memory thereby avoiding the verification of the certificate, which saves both computation time and network bandwidth.

4.3.6. AUTHENTICATED ENCRYPTION/ DECRYPTION

Authenticated encryption refers to simultaneously providing a message with confidentiality and authenticity. This is a well-known technique in the literature and is not a novelty of our work. However, we discuss it here to highlight how *SEDAN* leverages this process to achieve a more secure runtime system.

The keys computed using the session key (Section 4.3.5) and the message to be encrypted are given as the inputs to this step. The authenticated encryption and decryption phases are illustrated in Figure 42(a) and Figure 42(b) and discussed next.



(a) **(b)**

Figure 42 (a) Authenticated encryption at the sender ECU; (b) Authenticated decryption at the receiver ECU.

The authenticated encryption at the sender ECU begins with an XOR operation between the message data (plain text) and a nonce (random number), and the result is encrypted using AES with the key size allocated (as discussed in Section 4.3.4). The XOR operation is performed to avoid generating the same cipher every time in cases where input data remains unchanged for long durations. Even though protecting the system from side channel attacks is not within the scope of this work, this simple step could be the first step in preventing information leakage. The output cipher and the key used for encryption are given to a cryptographic hash function (MD5). The result is XORed with a nonce to generate the MAC. The output MAC size is truncated if needed and set to be at least the size computed in Section 4.3.3. It is then transmitted with the encrypted message data in the payload section of the FlexRay frame.

At the receiver ECU, the first step is authenticating the sender ECU of a received message. The received cipher and the selected key are given to the same cryptographic hash function whose result is XORed with a nonce to generate a local MAC. The sender ECU is successfully authenticated when the local MAC matches with the received MAC. Otherwise, the authentication

process fails and the received message is discarded. After successful authentication of a sender ECU, AES decryption is initiated, and the output is XORed with the nonce to extract the original message data (plain text).

As discussed in Section 4.2.2, we mainly focus on protecting the system from masquerade and replay attacks as they are the most common, hard to detect, and have a severe impact on the system. The system is protected against masquerade or impersonation attacks by authenticating the ECUs in the system using the STS protocol, which also establishes the session keys used for encryption and decryption only after a successful authentication. The attacker fails to authenticate due to the lack of trusted certificates, and hence cannot masquerade as a legitimate ECU. The MAC generated in the authenticated encryption protects the system from replay attacks. During the MAC generation, it is important to XOR the output of the hash function with the nonce as it makes the messages resilient to replay attacks. Whenever an attacker tries to perform a replay attack, the authenticity of the replayed message fails as the nonce used in computing the local MAC at the receiver is different from the nonce used in generating the received MAC at the sender. This results in a MAC mismatch leading to discarding of the message sent by the attacker. Moreover, in the event of a man-in-the middle attack, where the attacker tries to make any changes to the payload content, the MAC comparison fails, thereby protecting the integrity of the messages. Due to the broadcast nature of communication in automotive systems, an ECU or attacker can eavesdrop on the network using any of the attack vectors mentioned in Section 4.2.2. However, the attacker would not be able to decrypt the encrypted messages in the network. In this manner, we achieve confidentiality of the message data. Hence, using the proposed *SEDAN* framework, we were able to achieve all the security objectives- message confidentiality and integrity, and ECU authenticity (as discussed in Section 4.2.3). The other common type of attack in automotive systems is

distributed denial of service (*DDoS*) attack. However, we do not focus on this attack as they can be easily detected using a rule based intrusion detection systems (IDS) and can be prevented by designing the network with appropriate gateways and proper bus isolation. Some of the other complex attacks such as side channel attacks, core tampering and hardware trojans require additional resources and are outside the scope of our current work. We plan to investigate these attacks in our future work.

4.3.7. RUNTIME MESSAGE SCHEDULER

Runtime message scheduling is the last step in our framework that takes the unique values of the cipher and MAC generated in the previous step and inserts them into FlexRay frames generated during the frame packing step (Section 4.3.2). Other controls fields, such as the fields in the header and trailer segments, that are required for the transmission of FlexRay frames are also added by the scheduler at this point. The runtime scheduler uses the design time generated message schedule and interacts with the FlexRay protocol engine to schedule messages on to the FlexRay bus at runtime.

4.4 EXPERIMENTS

4.4.1. EXPERIMENTAL SETUP

We evaluated the performance of our proposed *SEDAN* framework by comparing it with [37], which uses simulated annealing to minimize the end-to-end latencies of all in-vehicle messages and uses symmetric key encryption and time-delayed release of keys to improve security in a vehicle system. As [37] does not support variable key sizes, three different variants of [37] are implemented using AES encryption with fixed key sizes of 128, 192 and 256 bits and referred to

as ‘Lin et al. AES-128’, ‘Lin et al. AES-192’, and ‘Lin et al. AES-256’ respectively in the results. We generated test cases based on automotive network and ECU computation data extracted from a real-world 2016 Chevrolet Camaro vehicle that we have access to. Directed acyclic graphs (DAGs) were generated using TGFF [114] and annotated with this data. We generated multiple test cases by scaling this data based on different combinations of the number of ECUs, number of applications, number of tasks in each application, and the range of periods. Also, we assume that the deadline for both tasks and messages are equal to their period. For all experiments, FlexRay 3.0.1 protocol [16] is used with the following network parameters: cycle duration of 5ms with 62 static segment slots, with a slot size of 42 bytes, and 64 communication cycles.

Table 6 AES, RSA and ECC execution times (ms) on ARM Cortex A9.

Cryptographic scheme	Key size	Encryption / Decryption	
AES	128	0.35	
	192	0.393	
	256	0.415	
Cryptographic scheme	**Key size**	**Public key operation**	**Private key operation**
RSA	512	2.01	19.89
	1024	6.48	139.15
	2048	23.65	911.8
	4096	91.52	6283.2
ECC	256	59.8	17.1
	384	182.4	50.4

4.4.2. BENCHMARKING ENCRYPTION ALGORITHMS

To accurately model the runtime behavior of session key generation and authenticated encryption/decryption we implemented these algorithms in software. We implemented AES-CBC with key sizes of 128, 192 and 256 bits, RSA with key sizes of 512, 1024, 2048 and 4096 bits, and the ECC with key sizes of 256 and 384 bits using OpenSSL [115]. The algorithms were executed

on an ARM Cortex-A9 CPU on a ZedBoard, which has similar specifications compared to many of the state-of-the art ECUs [116], [117].

The average AES encryption/decryption times with different standard key sizes for one block of data (16 Bytes) are shown in Table 6. These values are used at design time to model the latency overhead on each message due to the added security mechanisms. They are also used in computing the response time of the messages and when making scheduling decisions. The encryption and decryption times of RSA with 512, 1024, 2048 and 4096 bit keys and ECC with 256 and 384 bit keys are also shown in Table 6. These values are used to decide between RSA or ECC as the scheme for cryptographic operations in STS protocol. The NIST recommends a keys size of 2048 bits for RSA [118], while NSA recommends a 256 bit key size for SECRET level and 384 bit key size for TOP SECRET level using ECC [119]. Moreover, ECC with 224, 256 and 384 bit key sizes provides a similar security as RSA with 2048, 3072 and 7680 key sizes respectively [120]. In this work, we consider the minimum key sizes based on the above mentioned recommendations. From Table 6, it can be observed that RSA is faster for verifying signatures (operation performed using public key) and much slower for generating signatures (operation performed using private key). However, on the other hand, ECC is much faster for generating signatures while being relatively slower for verifying signatures. It is important to observe that the security (provided by RSA using the equivalent key size) doubles when the ECC key size is increased from 256 to 384. However, since the automotive systems are resource constrained, we choose to employ ECC with a 256-bit key size (which still provides higher security than the minimum recommended key size for RSA) for cryptographic operations in the STS key agreement protocol. Additionally, the ECC values are used in estimating the worst-case time required for setting up a session key, which is 0.24s for a 256-bit key, while an equivalent RSA 2048 takes

3.72s. Thus, ECC is much faster compared to RSA to achieve a similar level of security. Moreover, the key size required to achieve a similar security is much shorter in ECC compared to RSA. The latency associated with computing the hash value using MD5 for one block of data is 2.68µs. Moreover, with the increasing complexity of automotive applications it is important to design the security mechanisms that result in minimal power overhead. Hence, we profiled the security mechanisms used in this work and presented the power consumption results in Table 7. Other overheads such as memory consumption are not explicitly modeled as most modern day ECUs have sufficient memory to store the small keys needed for secure transfers. However, the designer can place an upper limit on the number of pre-computed session keys that can be stored to minimize memory overhead. Based on the results shown in Table 6 and Table 7, ECC has lower computation and memory overhead compared to RSA for the same level of security. Therefore, in *SEDAN* we authenticate the ECUs in the system and setup session keys using the STS protocol with ECC, instead of RSA. Additionally, we use AES to encrypt and decrypt the messages in the system using the keys derived from the session key.

Table 7 AES, RSA and ECC power consumption on ARM Cortex A9.

Cryptographic scheme	Key size	Encryption / Decryption	
AES (mW)	128	57.76	
	192	58.04	
	256	60.19	
Cryptographic scheme	**Key size**	**Public key operation**	**Private key operation**
RSA (W)	512	0.28	0.65
	1024	0.34	1.22
	2048	0.72	1.91
	4096	1.08	2.58
ECC (W)	256	0.62	0.33
	384	0.93	0.58

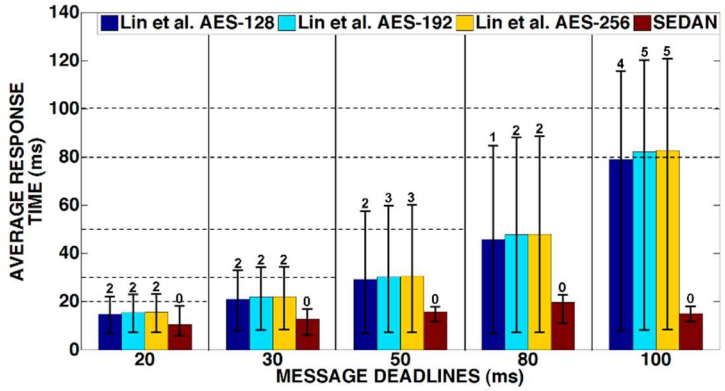
4.4.3. GRASP PARAMETER SELECTION

To get an efficient solution using the GRASP metaheuristic, it is important to choose the appropriate values for the threshold parameters α and β_{max}. We ran a series of simulations by changing the value of α from 0 to 1 with an increment of 0.2 and the greedy randomized construction phase was run for 1000 times using different input test cases. We noticed that the mean solution approached a greedy solution, while the variance approached zero as α tends to 1. In contrast, when α is small and close to zero, the mean solution approaches a random solution with high variance. In order to provide a good quality solution to the local search phase, we chose $\alpha = 0.8$ which leads to a near greedy solution in the presence of relatively large variance. Also, we observed that $\beta_{max} = 3$ provided enough randomness to look for other solutions in each iteration of the local search phase. A higher value of β_{max} could result in an exhaustive local search leading to unreasonably long computation times. Also, the minimum value of β needs to be 2, in order to increase the key size of at least one message when the key size is reduced in the event of a destroy operation. This prevents the generation of a solution with lower ASV compared to the solutions in previous iterations. Moreover, a relative small value for $p_{kd} = 0.3$ is chosen to avoid frequent key size decrements.

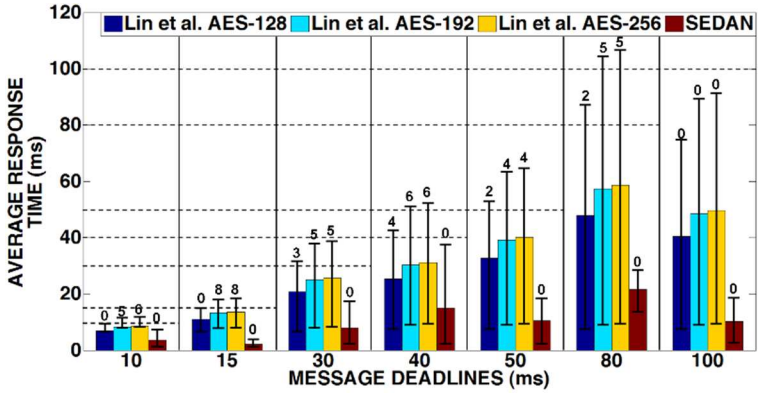
4.4.4. RESPONSE TIME ANALYSIS

We tested the *SEDAN* framework and the three variants from Lin et al. [37] with three different test cases: (1) low input load, in a system with 5 ECUs (3 single-core and 2 dual-core) and 77 tasks that produced 57 (time-triggered) signals; (2) medium input load, in a system with 12 ECUs (9 single-core and 3 dual-core) and 126 tasks with 93 signals; and (3) high input load, in a system with 16 ECUs (12 single-core and 4 dual-core) and 243 tasks with 196 signals. Figure 43

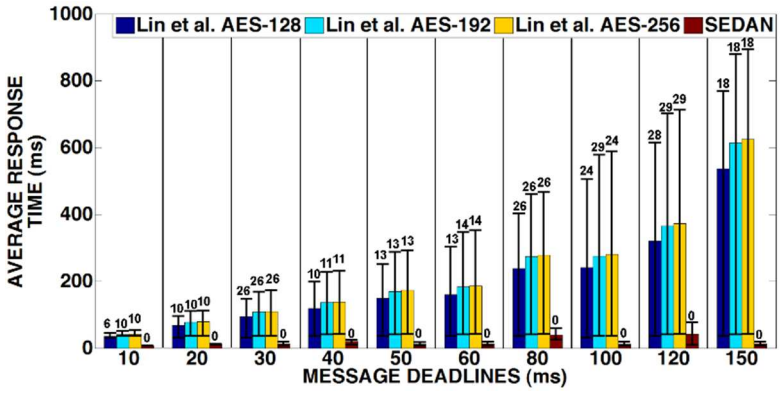
(a)-(c) show the average message response time for the low, medium and high input load cases with their deadlines on the x-axis. Response time of a message is its end-to-end latency, which is the aggregate of the time for encryption and MAC generation, and queuing delay at the sender ECU; transmission time on the Flexray bus, and the time for MAC verification and decryption at the receiver ECU. The confidence interval on each bar shows the minimum and maximum average response time of messages. The dashed horizontal lines represent different message deadlines. The number on top of each bar is the number of deadlines misses.



(a)



(b)



(c)

Figure 43 Average response time of all messages (with number of missed deadlines shown on top of bars) for (a) low; (b) medium and (c) high input application load conditions, for Lin et al. AES-128, AES-192, AES-256 [37]; and *SEDAN*.

It is evident that *SEDAN* outperforms the three variants of [37] and achieves lower average response times for all of the messages in the different input load cases. *SEDAN* achieves this by balancing security and real-time performance goals by optimizing key sizes while meeting message security requirements and ensuring that all ECU utilizations are below 100%. This prevents the messages from experiencing additional delays on top of the latency caused by the encryption-decryption processes. Moreover, compared to *SEDAN*, all the three variants of [37] experience significant authentication delays (time taken from the transmission of the message to decryption of the message), which results in increased response time of the messages. These high authentication delays are because of the time delayed release of keys in all three variants of [37]. Also, the periodic computation of keys in every session at each ECU in all three variants of [37] results in high ECU utilization overhead resulting in increased response time and power consumption. Moreover, the requirement of large message buffers to hold multiple messages for longer durations (due to time-delayed release of keys) further increases the power consumption and response time.

Table 8 Number of security violations for each input load configuration.

Framework	Lin et al. 128	Lin et al. 192	Lin et al. 256	SEDAN
Low load	28	12	0	0
Medium load	45	16	0	0
High load	96	31	0	0

4.4.5. SECURITY ANALYSIS

Table 8 shows the number of security violations in each technique, for the three different input load cases (as discussed in the previous sub-section). A security violation is defined as an instance when the derived security constraints (Section 4.3.3) for a message are not met. It can be seen that the *SEDAN* and Lin et al. AES-256 are the only techniques that do not violate any security requirements. It is important to note that unlike *SEDAN*, Lin et al. AES-256 has no smart key size assignment scheme and assigns all the messages with 256-bit keys irrespective of their ASIL group, which helps in meeting the message security requirements. But this results in increased ECU utilization, which in turn incurs additional latency overheads for messages. Moreover, unlike all the three variants of [37], *SEDAN* does not exchange or release keys on an unsecured communication bus, thereby preventing an attacker from gaining knowledge about the current and previously used keys. *SEDAN* also does not require frequent key computation at each ECU within a single session, as done in [37], which helps reduce utilization overheads in ECUs when *SEDAN* is used.

Lastly, Figure 44 depicts the ASV for the three input load cases, with numbers on top of each bar showing the number of messages that missed deadlines. Lin et al. AES-256 achieves the highest ASV, however this comes at the cost of multiple missed real-time deadlines. *SEDAN* is able to satisfy minimum message security requirements (i.e., all messages have at least the

minimum key size required by the designer) and all real-time deadlines, while providing an ASV value that is higher than that for Lin et al. AES-128 and Lin et al. AES-192.

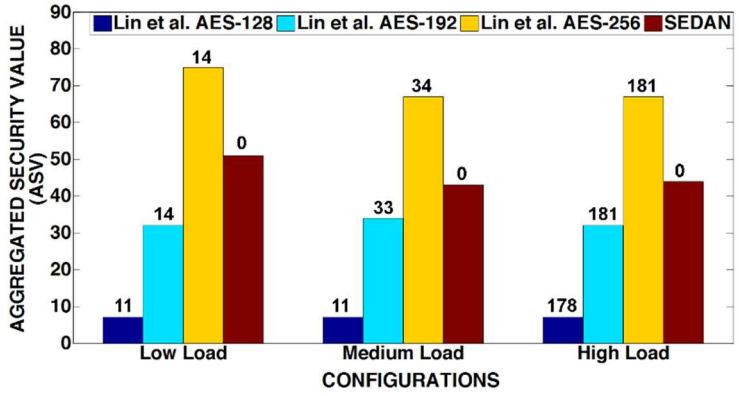


Figure 44 Aggregate Security Value (ASV) for each input load configuration (with number of missed deadlines on top of bars).

In summary, *SEDAN* represents a promising framework that can intelligently manage the limited computing resources in vehicles while improving the security of the overall system. *SEDAN* is able to do a better job of balancing security and real-time performance goals than [37] as shown in Figure 44 and Table 8. It does so by intelligently optimizing key sizes and accurately integrating overheads of security primitives while making task and message scheduling decisions.

4.5 CONCLUSION

In this work, we presented a novel security framework (*SEDAN*) that combines design time schedule optimization with runtime symmetric key management to improve security in time-critical automotive systems without utilizing any additional hardware. We demonstrated the feasibility of our SEDAN framework by implementing the cryptographic algorithms on real processors. Moreover, our experimental results indicate that *SEDAN* is able to reason about security overheads to intelligently adapt security primitives during message and task scheduling,

ultimately ensuring that both security and real-time performance goals are met. Such a framework promises to be extremely useful as we move towards connected autonomous vehicles with large attack surfaces, by *enabling security to be a first-class design objective* without sacrificing real-time performance objectives.

5. INDRA: INTRUSION DETECTION USING RECURRENT AUTOENCODERS IN AUTOMOTIVE EMBEDDED SYSTEMS

Modern vehicles can be considered as complex distributed embedded systems that consist of tens of interconnected Electronic Control Units (ECUs). These ECUs control various components in the vehicle and communicate with each other using the in-vehicle network. The number of ECUs and the complexity of software running on these ECUs has been steadily increasing in emerging vehicles, to support state-of-the-art Advanced Driver Assistance Systems (ADAS) features such as lane keep assist, collision warning, blind spot warning, parking assist, etc. This in turn has resulted in an increase in the complexity of the in-vehicle network, which is the backbone over which massive volumes of heterogeneous sensor and real-time decision data and control directives are communicated.

The trend in recent ADAS solutions has been to interact with external systems using advanced communication standards such as Vehicle-to-X (V2X) and 5G technology [2]. Unfortunately, this makes modern vehicles highly vulnerable to various security attacks that can be catastrophic. Several attacks have been demonstrated in [25], [26], [90] showing different ways to gain access to the in-vehicle network and take control of the vehicle via malicious messages. With connected and autonomous vehicles becoming increasingly ubiquitous, these security issues will only get worse. Hence, it is crucial to prevent unauthorized access of in-vehicle networks from external attackers to ensure the security of automotive systems.

Traditionally, firewalls are used to defend a network from various external attackers. *However, no firewall is perfect and no network is impenetrable*. Hence, there is a need for an active monitoring system that scans the network to detect the presence of an attacker in the system. This

can be achieved using an intrusion detection system (IDS) which monitors' network traffic and triggers alerts when suspicious activity or known threats are detected. The IDS is often the last line of defense in automotive systems.

IDSs can be classified into two types: *(i) signature-based*, and *(ii) anomaly-based*. The former observes for traces of any known attack signatures while the latter observes for the deviation from the known normal system behavior to indicate the presence of an attacker. Signature-based IDS can have faster detection times and very few false positives, but can only detect known attacks. On the other hand, anomaly-based IDS can detect both known and unknown attacks, but can suffer from higher false positives and relatively slower detection times. An efficient IDS needs to be *lightweight, robust* and *scalable* with different system sizes. Moreover, a pragmatic IDS needs to have a *large coverage of attacks* (able to detect both known and unknown attacks), *high confidence in detection,* and *low false positive rate* as recovery from false positives can be expensive.

Since getting the signature of every possible attack is impractical and would limit us to only detecting known attacks, we conjecture that using anomaly-based IDS is a more practical approach to this problem. Additionally, due to the ease of in-vehicle network data acquisition (from test driving), there can be a large amount of in-vehicle message data to work with, which facilitates the use of advanced deep learning models for detecting the presence of an attacker in the system.

In this chapter, we propose a novel IDS framework called *INDRA* that monitors the messages in Controller Area Network (CAN) based automotive systems for the presence of an attacker. In the offline phase, *INDRA* uses deep learning to learn the normal system behavior in an unsupervised fashion. At runtime, *INDRA* monitors the network and indicates the presence of an

attacker if any anomalies (any deviation from the normal behavior learned during the offline phase) are detected. *INDRA* aims to maximize the detection accuracy and minimize false positive rate with minimal overhead on the ECUs.

Our novel contributions in this work are as follows:

- We propose a Gated Recurrent Unit (GRU) based recurrent autoencoder network to learn the latent representation of normal system behavior during the offline phase;
- We propose a metric called intrusion score (IS), which is a measure of deviation from the normal system behavior;
- We perform a thorough analysis towards the selection of thresholds for this intrusion score metric;
- We compare our proposed *INDRA* framework with the best known prior works in the area, to show its effectiveness.

5.1 RELATED WORK

Several techniques have been proposed to design IDS for time-critical automotive systems. The goal of these works is to detect various types of attacks by monitoring the in-vehicle network traffic.

Signature-based IDS relies on detecting known and pre-modeled attack signatures. The authors in [121], used a language theory-based model to derive attack signatures. However, this technique fails to detect intrusions when it misses the packets transmitted during the early stages of an attack. In [122], the authors used transition matrices to detect intrusions in a CAN bus. In spite of achieving a low false-positive rate for trivial attacks, this technique failed to detect realistic replay attacks. The authors in [123] identify notable attack patterns such as an increase in message

frequency and missing messages to detect intrusions. A specification-based approach to detect intrusions is proposed in [124], where the authors analyze the behavior of the system and compare it with the predefined attack patterns to detect intrusions. Nonetheless, their system fails to detect unknown attacks. In [125], an IDS technique using the Myers algorithm [126] was proposed under the map-reduce framework. A time-frequency analysis of CAN messages is used in [127] to detect multiple intrusions. A rule-based regular operating mode region is derived in [106] by analyzing the message frequency at design time. This region is observed for deviations at runtime to detect anomalies. In [128], fingerprints of the sender ECU's clock skew and the messages are used to detect intrusions by observing for variations in the clock-skew at runtime. In [129], a formal analysis is presented for clock-skew based IDS and evaluated on a real vehicle. A memory heat map is used to characterize the memory behavior of the operating system to detect intrusions in [130]. An entropy-based IDS is proposed in [131] that observes for change in system entropy to detect intrusions. However, the technique fails to detect small scale attacks where the entropy change is minimal. *In summary, signature-based techniques offer a solution to the intrusion detection problem with low false positive rates but cannot detect more complex and novel attacks. Moreover, modeling signatures of every possible attack is impractical.*

Anomaly-based IDS aims to learn the normal system behavior in an offline phase and observe for any deviation from the learned normal behavior to detect intrusions at runtime. A sensor-based IDS was proposed in [132], that utilizes attack detection sensors to monitor various system events to observe for deviations from normal behavior. However, this approach is not only expensive but also suffers from poor detection rates. A One-Class Support Vector Machine (OCSVM) based IDS was proposed in [133]. However, this approach suffers from poor detection latency. In [134], the authors used four different nearest neighbor classifiers to distinguish between

a normal and an attack induced CAN payload. In [135], a decision-tree based detection model is proposed to monitor the physical features of the vehicle to detect intrusions. However, this model is not realistic and suffers from high detection latencies. A Hidden Markov Model (HMM) based technique was proposed in [136] that monitors the temporal relationships between messages to detect intrusions. A deep neural network based approach was proposed to examine the messages in the in-vehicle network in [137]. This approach is tuned for a low priority tire pressure monitoring system (TPMS), which makes it hard to adapt to high priority safety-critical powertrain applications. A Long Short-Term Memory (LSTM) based IDS for multi-message ID detection was proposed in [138]. However, the model architecture is highly complex, which incurs high overhead on the ECUs. In [139], the authors use LSTM based IDS to detect insertion and dropping attacks (explained in Section 5.3.3). An LSTM based predictor model is proposed in [140] that predicts the next time step message value at a bit level and observes for large variations in loss to detect intrusions. In [141], a recurrent neural network (RNN) based IDS was proposed to learn the normal patterns in CAN messages in the in-vehicle network. A hybrid IDS was proposed in [142], which utilizes a specification-based system in the first stage and an RNN based model in the second stage to detect anomalies in time-series data. *However, none of these techniques provides a holistic system-level solution that is lightweight, scalable, and reliable to detect multiple types of attacks for in-vehicle networks.*

In this chapter, we propose a lightweight recurrent autoencoder based IDS using gated recurrent units (GRUs) that monitors messages at a signal level granularity to detect multiple types of attacks more effectively and successfully than the state of the art. Table 9 summarizes some of the state-of-the-art works' performance under different metrics and shows how our proposed *INDRA* framework fills the existing research gap. The *INDRA* framework aims at improving

multiple performance metrics compared to the state-of-the art IDS works that target a subset of performance metrics. A detailed analysis of each metric and evaluation results are presented later in Section 5.5.

Table 9 Comparison between our proposed *INDRA* framework and state-of-the-art works.

Technique	Performance metrics			
	Lightweight	Low False Positive Rate	High accuracy	Fast Inference
PLSTM [140]	X	✓	X	X
RepNet [141]	✓	X	X	✓
CANet [138]	X	✓	✓	X
INDRA	✓	✓	✓	✓

5.2 SEQUENCE LEARNING BACKGROUND

With the availability of increased computing power from GPUs and custom accelerators, training neural networks with many hidden layers (known as *deep neural networks*) has led to the creation of powerful models for solving difficult problems in many domains. One such problem is detecting intrusions in the in-vehicle network. In an in-vehicle network, the communication between ECUs happens in a time-dependent manner. Hence, there exist temporal relationships between the messages, which is essential to exploit, in order to detect intrusions. However, this cannot be achieved using traditional feedforward neural networks where the output of any input is independent of the other inputs. Sequence models are a more appropriate approach for such problems, as they are designed to handle sequences and time-series data.

5.2.1. SEQUENCE MODELS

A sequence model can be understood as a function which ensures that the output is dependent not only on the current input, but also on the previous inputs. An example of such a sequence

model is the recurrent neural network (RNN), which was introduced in [143]. In recent years, other sequence models such as long short-term memory (LSTM) and gated recurrent unit (GRU) have also been developed.

5.2.1.1. RECURRENT NEURAL NETWORKS (RNN)

An RNN is a type of neural network which takes sequential data as the input and learns the relationship between data sequences. RNNs have hidden states, which allows learned information to persist over time steps. The hidden states enable the RNN to connect previous information to current inputs. An RNN cell with feedback is shown in Figure 45(a), and an unrolled RNN in time is shown in Figure 45(b).

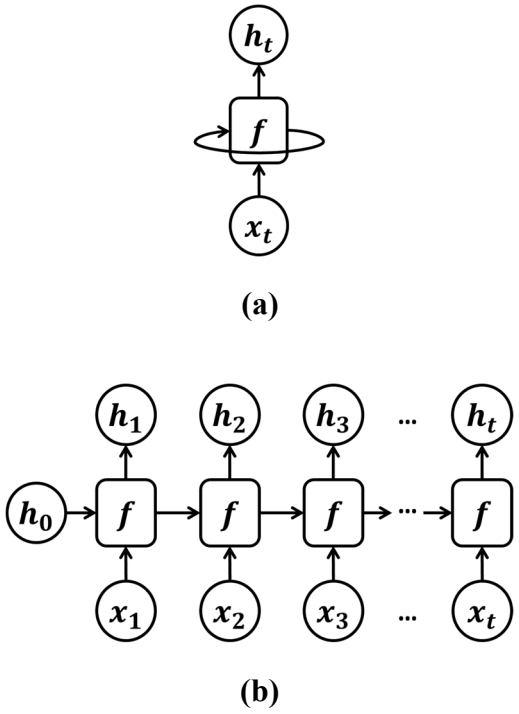


Figure 45 (a) A single RNN cell and (b) unrolled RNN unit; where, f is the RNN cell, x is the input, and h represents hidden states.

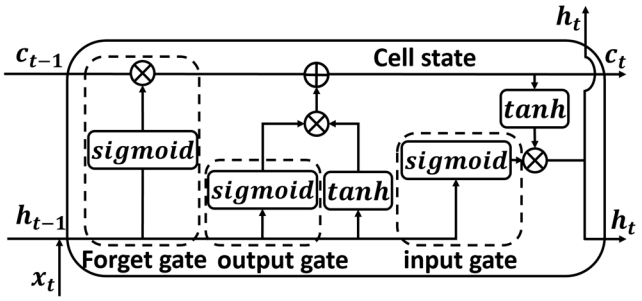
The output (h_t) of an RNN cell is a function of both the input (x_t) and the previous output (h_{t-1}) as shown in (21):

$$h_t = f(Wx_t + Uh_{t-1} + b) \tag{21}$$

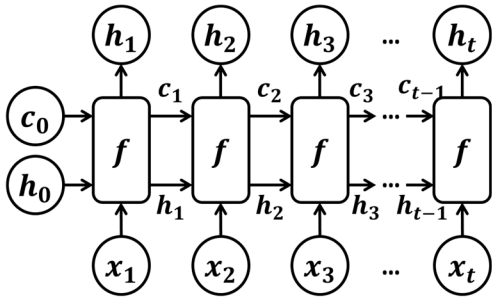
where *W, U* are weight matrices, *b* is a bias term, and *f* is a nonlinear activation function (e.g. sigmoid or tanh). One of the limitations of RNNs is that they are very hard to train. Since RNNs and other sequence models deal with sequence or time-series inputs, backpropagation happens through various time samples (known as backpropagation through time). During this process, the feedback loop in RNNs causes the errors to shrink or grow rapidly (creating vanishing or exploding gradients respectively), destroying the information in backpropagation. This problem of vanishing gradients hampers the RNNs from learning *long term dependencies*. This problem was solved in [144] with the introduction of additional states and gates in the RNN cell to remember long term dependencies, which led to the introduction of Long Short-Term Memory Networks.

5.2.1.2. LONG SHORT-TERM MEMORY (LSTM) NETWORKS

LSTMs are modified RNNs that use cell state and hidden state information along with multiple gates to remember long term dependencies. The cell state can be thought of as a transport highway, that carries relevant information throughout the processing of a sequence. The state accommodates the information from earlier time steps, which can be used in the later time steps, thereby reducing the effects of short-term memory. The information in the cell state is modified via gates. Hence, the gates in LSTM help the model decide which information has to be retained and which information to forget.



(a)



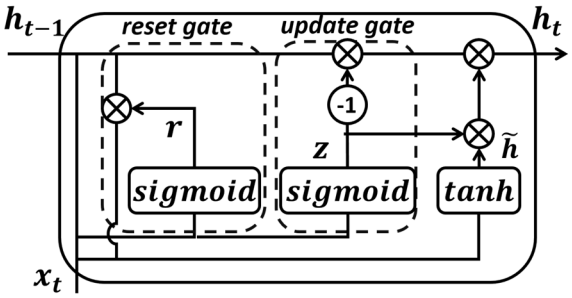
(b)

Figure 46 (a) A single LSTM cell with different gates and (b) unrolled LSTM unit; where, *f* is an LSTM cell, *x* is input, *c* is cell state and *h* is the hidden state.

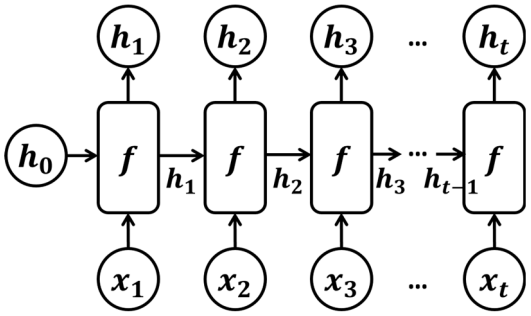
An LSTM cell consists of three gates: *(i)* forget gate (f_t) *(ii)* input gate (i_t), and *(iii)* output gate (o_t) as shown in Figure 46(a). The forget gate is a binary gate that chooses which information to retain from the previous cell state (c_{t-1}). The input gate adds relevant information to the cell state (c_t). Lastly, the output layer is controlled by the output gate, which uses information from the previous two gates to produce an output. An unrolled LSTM unit is shown in Figure 46(b).

By using the combination of different gates and hidden states, LSTMs can learn long term dependencies in a sequence. However, they are not computationally efficient as the sequence path is more complicated than in RNNs, due to the addition of multiple gates, requiring more memory at runtime. Moreover, training LSTMs is compute intensive even with advanced training methods such as truncated backpropagation. To overcome these limitations, a simpler recurrent neural

network called gated recurrent unit (GRU) network was introduced in [145] that can be trained faster than LSTMs and also remembers dependencies in long sequences with low memory overhead, while solving the vanishing gradient problem.



(a)



(b)

Figure 47 (a) A single GRU cell with different gates and (b) unrolled GRU unit; where, f is a GRU cell, x is input, and h represents hidden states.

5.2.1.3. GATED RECURRENT UNIT (GRU)

A GRU cell uses an alternate route for gating information when compared to LSTMs. It combines the input and forget gate of the LSTM into a solitary *update gate* and furthermore combines hidden and cell state, as shown in Figure 47(a) and Figure 47(b).

A traditional GRU cell has two gates *(i)* reset gate, and *(ii)* update gate. The reset gate combines new input with past memory while the update layer chooses the amount of pertinent data that should be held. Thus, a GRU cell can control the data stream like an LSTM by uncovering its hidden layer contents. Moreover, GRUs achieve this using fewer gates and states, which makes them computationally more efficient with low memory overhead. As real-time automotive ECUs are highly resource-constrained embedded systems with tight energy and power budgets, it is critical to use low overhead models for inferencing tasks. Thus, GRU based networks are an ideal fit for inference in automotive systems. Additionally, GRUs are relatively new, less explored and have a lot of potential to offer compared to the RNNs and LSTMs. Hence, in this work, we chose to use a lightweight GRU based model to implement our IDS (explained in detail in Section 5.4).

One of the advantages of sequence models is that they can be trained using both supervised and unsupervised learning approaches. As there is a large volume of CAN message data in a vehicle, labeling the data can become very tedious. Additionally, the variability in the messages between vehicle models from the same manufacturer and the proprietary nature of this information, makes it even more challenging to label messages correctly. However, due to the ease of availability to CAN message data via onboard diagnostics (OBD-II), large amounts of unlabeled data can be collected easily. Thus, we use GRUs in an unsupervised learning setting in this work.

5.2.2. AUTOENCODERS

An autoencoder is an unsupervised learning algorithm whose goal is to reconstruct the input by learning latent input features. It achieves this by encoding the input data (x) towards a hidden layer, and finally decodes it to produce a reconstruction $\tilde{x}$ (as shown in Figure 48). This encoding at the hidden layer is called an embedding. The layers that create this embedding are called the

encoder, and the layers that reconstruct the embedding into the original input are called the decoder. During training, the encoder tries to learn a nonlinear mapping of the inputs, while the decoder tries to learn the nonlinear mapping of the embedding to the inputs. Both encoder and decoder achieve this with the help of non-linear activation functions, e.g., tanh, and relu. Moreover, the autoencoder aims to recreate the input as accurately as possible by extracting the key features from the inputs with a goal of minimizing reconstruction loss. The most commonly used loss functions in autoencoders are mean squared error (MSE) and Kullback-Leibler (KL) divergence.

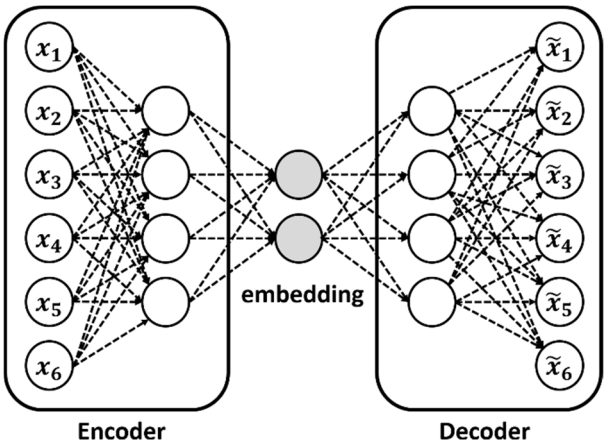


Figure 48 Autoencoders.

As autoencoders aim to reconstruct the input by learning the underlying distribution of the input data, it makes them an ideal choice to learn and reconstruct highly correlated time-series data efficiently by learning the temporal relations between signals. *Hence, our proposed INDRA framework uses light weight GRUs in an autoencoder to learn latent representations of CAN messages in an unsupervised learning setting.*

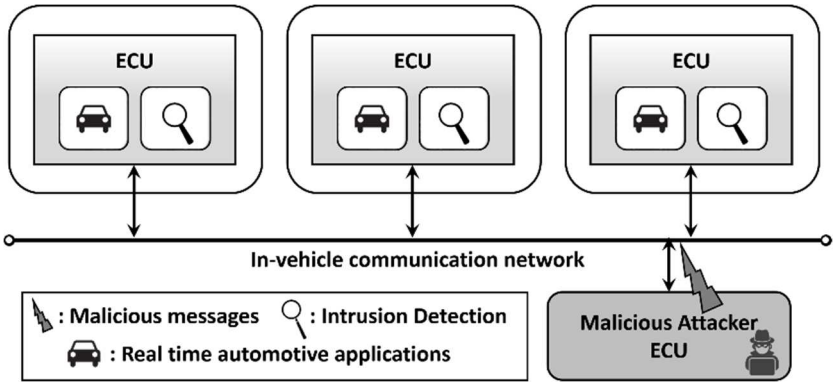


Figure 49 Overview of the system model.

5.3 PROBLEM DEFINITION

5.3.1 SYSTEM MODEL

We consider a generic automotive system consisting of multiple ECUs connected using a CAN based in-vehicle network, as shown in Figure 49. Each ECU is responsible for running a set of automotive applications that are hard-real time in nature, meaning they have strict timing and deadline constraints. In addition, we assume that each ECU also executes intrusion detection applications that are responsible for monitoring and detecting intrusions in the in-vehicle network. We consider a distributed IDS approach (intrusion applications collocated with automotive applications) as opposed to a centralized IDS approach where one central ECU handles all intrusion detection tasks due to the following reasons:

- A centralized IDS approach is prone to single-point failures, which can completely open up the system to the attacker.
- In extreme scenarios such as during a flooding attack (explained in Section 5.3.3), the in-vehicle network can get highly congested and the centralized system might not be able to communicate with the victim ECUs.

- If an attacker succeeds in fooling the centralized IDS ECU, attacks can go undetected by the other ECUs, resulting in compromising the entire system; whereas with a distributed IDS, fooling multiple ECUs is required which is much harder, and even if an ECU is compromised, this can still be detected by the decentralized intelligence.
- In a distributed IDS, ECUs can stop accepting messages as soon as an intrusion is detected without waiting for a centralized system to notify them, leading to faster response.
- The computation load of IDS is split among the ECUs with a distributed IDS, and the monitoring can be limited to only the required messages. Thus, multiple ECUs can monitor a subset of messages independently, with lower overhead.

Many prior works, e.g., in [121] and [106], consider a distributed IDS approach for these reasons. Moreover, with automotive ECUs becoming increasingly powerful, the collocation of IDS applications with real-time automotive applications in a distributed manner should not be a problem, provided the overhead from the IDS is minimal. Our proposed framework is not only lightweight, but also scalable, and achieves high intrusion detection performance, as discussed in Section 5.5.

The design of an IDS should have low susceptibility to noise, low cost, and a low power/energy footprint. The following are some of the goals that we considered for our IDS:

- *Lightweight*: Intrusion detection tasks can incur overhead on the ECUs that could result in poor application performance or missed deadlines for real-time applications. This can be catastrophic in some cases. Hence, we aim to have a lightweight IDS that incurs low overhead on the system.

- *Few false positives*: This is a highly desired quality in any kind of IDS (even outside of the automotive domain), as handling false positives can become expensive very quickly. A good IDS needs to have few false positives or false alarms.
- *Coverage*: This is the range of attacks an IDS can detect. A good IDS needs to be able to detect more than one type of attack. A high coverage for IDS will make the system resilient to multiple attack surfaces.
- *Scalability*: This is an important requirement as emerging vehicles have increasing numbers of ECUs, and high software and network complexity. A good IDS should be highly scalable and be able to support multiple system sizes.

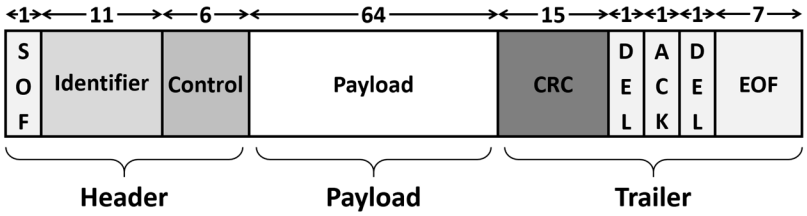


Figure 50 Standard CAN frame format.

5.3.2 COMMUNICATION MODEL

In this subsection, we discuss the vehicle communication model that was considered. We primarily focus on detecting intrusions in a CAN bus-based automotive system. Controller Area Network (CAN) is the defacto industry standard in-vehicle network protocol for automotive systems today. CAN is a lightweight, low cost, and event-triggered communication protocol that transmits messages in the form of frames. The structure of a standard CAN frame is shown in Figure 50 and the length of each field (in bits) is shown on the top. The standard CAN frame consists of header, payload, and trailer segments. The header consists of information such as the

message identifier (ID) and the length of the message. The actual data that needs to be transmitted is in the payload segment. The trailer section is mainly used for error checking at the receiver. A variation of the standard CAN, called CAN-extended or CAN 2.0B is also becoming increasingly common in modern vehicles. The major difference is that CAN extended has a 29-bit identifier allowing for more number of messages IDs.

Signal Name	Message	Start bit	Length	Byte Order	Value Type
Battery_Current	Status	0	16	Intel	Signed
Battery_Voltage	Status	16	16	Intel	Unsigned
Motor_Current	Status	32	16	Intel	Signed
Motor_Speed	Status	48	8	Intel	Signed
Motor_Direction	Status	56	8	Intel	Unsigned

Figure 51 Real-world CAN message with signal information.

In this work, we design our IDS with a focus on monitoring the message payload and observe for anomalies to detect intrusions. This is because an attacker needs to modify the message payload to accomplish a malicious activity. While an attacker could target the header or trailer segments, it would result in the message getting rejected at the receiver. The payload segment consists of multiple data entities called signals. An example real-world CAN message with the signals is shown in Figure 51 [19]. Each signal has a fixed size (in bits), a particular data type, and a start bit that specifies its location in the 64-bit payload segment of the CAN message.

In this work, we focus on monitoring individual signals within message payloads to observe for anomalies and detect intrusions. Our model learns the temporal dependencies between the messages at a signal level during training and observes for deviations at runtime to detect intrusions. Signal level monitoring would give us the capability to not only detect the presence of an intruder but also helps in identifying the signal within the message that is being targeted during an attack. This can be valuable information for understanding the intentions of the attacker, which

can be used for developing countermeasures. The details about the signal level monitoring of our IDS are discussed in Section 5.4.2. *Note*: Even though in this work we focus on detecting intrusions by monitoring CAN messages, our approach is protocol-agnostic and can be used with other in-vehicle network protocols.

5.3.3 ATTACK MODEL

Our proposed IDS aims to protect the vehicle from multiple types of attacks listed below. These are some of the most common and hard to detect attacks, and have been widely considered in literature to evaluate IDS models.

(1) Flooding attack: This is the most common and easy to launch attack and requires no knowledge about the system. In this attack, the attacker floods the in-vehicle network with a random or specific message and prevents the other ECUs from communicating. These attacks are generally detected and prevented by the bridges and gateways in the in-vehicle network and often do not reach the last line of defense (the IDS). However, it is important to consider these attacks as they can have a severe impact when not handled correctly.

(2) Plateau attack: In this attack, an attacker overwrites a signal value with a constant value over a period of time. The severity of this attack depends on the magnitude of the jump (increase in signal value) and the duration for which it is held. Large jumps are easier to detect compared to shorter jumps.

(3) Continuous attack: In this attack, an attacker slowly overwrites the signal value with the goal of achieving some target value and avoid triggering of an IDS in the system. This attack is hard to detect and can be sensitive to the IDS parameters (discussed in Section 5.4.2).

(4) Suppress attack: In this attack, the attacker suppresses the signal value(s) by either disabling the communication controller of the target ECU or by powering off the ECU. These attacks can be easily detected, as they shut down message transmission for long durations, but are harder to detect for shorter durations.

(5) Playback attack: In this attack, the attacker replays a valid series of message transmissions from the past trying to trick the IDS. This attack is hard to detect if the IDS does not have the ability to capture the temporal relationships between messages.

In this work, we assume that the attacker can gain access to the vehicle using the most common attack vectors, which include connecting to V2X systems that communicate with the outside world (such as infotainment and connected ADAS systems), connecting to the OBD-II port, probe-based snooping on the in-vehicle bus, and via replacing an existing ECU. We also assume that the attacker has access to the bus parameters (such as BAUD rate, parity, flow control, etc.) that can help in gaining access to the in-vehicle network.

Problem objective: The goal of our work is to implement a lightweight IDS that can detect multiple types of attacks (as mentioned above) in a CAN based automotive system, with a high detection accuracy and low false positive rate, and while maintaining a large attack coverage.

5.4 INDRA FRAMEWORK OVERVIEW

We propose the *INDRA* framework to enable a signal level anomaly-based IDS for monitoring CAN messages in automotive embedded systems. An overview of the proposed framework is shown in Figure 52. At a high level, the *INDRA* framework consists of design-time and runtime components. At design time, *INDRA* uses trusted CAN message data to train a recurrent autoencoder based model to learn the normal behavior of the system. At runtime, the

trained recurrent autoencoder model is used for observing deviations from normal behavior (inference) and detect intrusions based on the deviation computed using the proposed intrusion score metric (detection). The following subsections describe these steps in more detail.

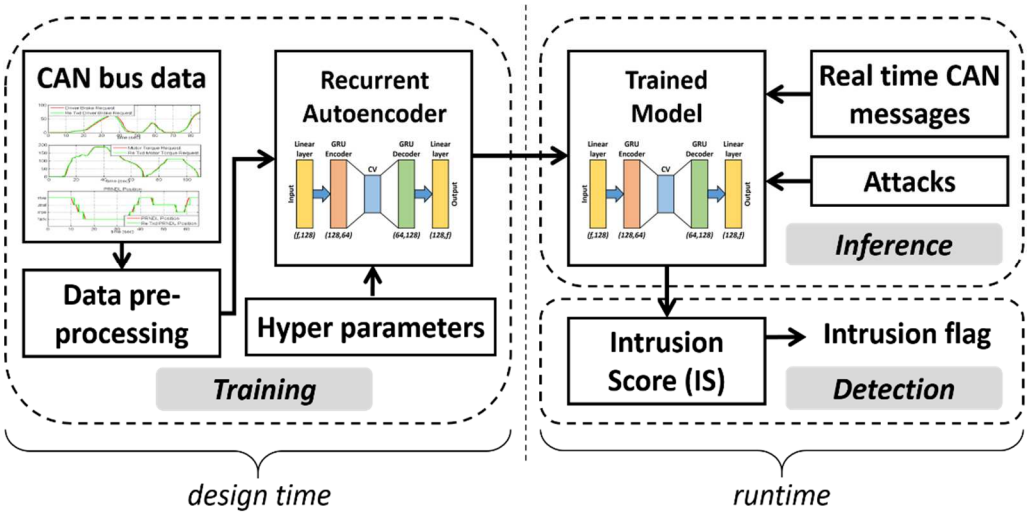


Figure 52 Overview of proposed *INDRA* framework.

5.4.1. RECURRENT AUTOENCODER

Recurrent autoencoders are powerful neural networks that are designed to behave like an encoder-decoder but handle time-series or sequence data as inputs. They can be visualized as regular feed-forward neural network based autoencoders, with the neurons being RNN, LSTM or GRU cells (discussed in Section 5.2). Similar to regular autoencoders, the recurrent autoencoders have an encoder and a decoder stage. The encoder is responsible for generating a latent representation of the input data in an n-dimensional space. The decoder uses the latent representation from the encoder and tries to reconstruct the input data with minimal error. In this work, we propose a new lightweight recurrent autoencoder model, that is customized for the design

of IDS to detect intrusions in the in-vehicle network data. The details of the proposed model architecture and the stages involved in its training are discussed next.

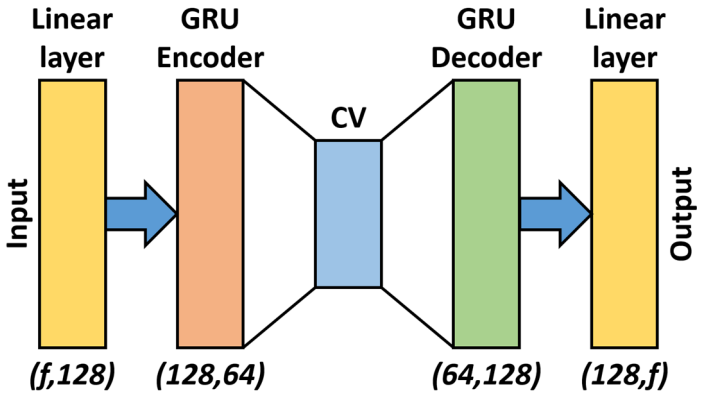


Figure 53 Proposed recurrent autoencoder network (*f* is number of features i.e., number of signals in the input CAN message, MCV is message context vector).

5.4.1.1. MODEL ARCHITECTURE

The proposed recurrent autoencoder model architecture with the dimensions (input, output) of each layer is illustrated in Figure 53. The model consists of a linear layer at the input, GRU based encoder, GRU based decoder and a final linear layer before the output. The input to the first linear layer is the time-series of CAN message data with signal level values with *f* features (where *f* is the number of signals in that particular message). The output of the linear layer is given to the GRU based encoder to generate the latent representation of the time-series signal inputs. We call this latent representation as a message context vector (MCV). The MCV captures the context of different signals in the input message data, and hence has a vector form. Each value in the MCV can be thought of as a point in an n-dimensional space that contains the context of the series of signal values given as input. The MCV is fed into a GRU based decoder, which is then followed by a linear layer to reconstruct the input time-series of CAN message data with individual signal

level values. Mean square error (MSE) is used to compute the loss between the input and the reconstructed input. Weights are updated using backpropagation through time. We design a recurrent autoencoder model for each message ID.

5.4.1.2. TRAINING PROCESS

The training process begins with the pre-processing of the CAN message data. Each sample in the dataset consists of a message ID and corresponding values of the signals within that message ID. The signal values are scaled between 0 to 1 for each signal type, as the range of signal values can be very large in some cases. Using such unscaled inputs can result in an extremely slow or very unstable training process. Moreover, as our goal is to reconstruct the input, scaling signal values also helps us avoid the problem of exploding gradients.

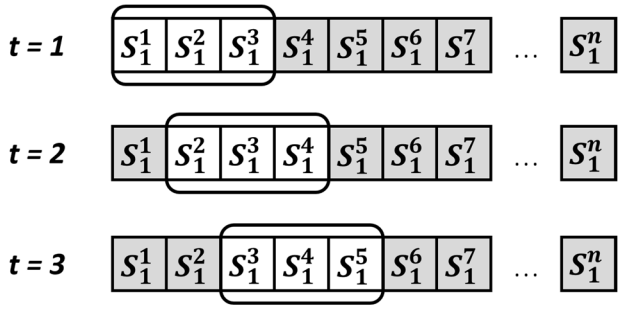


Figure 54 Rolling window based approach.

After pre-processing the available data for training, it is split into training data (85%) and validation data (15%), and is prepared for training using a rolling window based approach. This involves selecting a window of fixed size and rolling it to the right by one time sample every time step. A rolling window size of three samples for three time steps is illustrated in Figure 54, where the term S_i^j represents the i^{th} signal value at j^{th} sample. The elements in the rolling window are collectively called as a subsequence and the subsequence size is equal to the size of the rolling

window. As each subsequence consists of a set of signal values over time, the proposed recurrent autoencoder model tries to learn the temporal relationships that exist between the series of signal values. These signal level temporal relationships help in identifying more complex attacks such as continuous and playback (as discussed in Section 5.3.3). The process of training using subsequences is done iteratively until the end of the sequence in training data.

During training, each iteration consists of a forward pass and a backward pass using backpropagation through time to update the weights and biases of the neurons (discussed in Section 5.2) based on the error value. At the end of the training, the model's learning is evaluated (forward pass only) using the validation data, which was not seen during the training. By the end of validation, the model has seen the full dataset once and this is known as an epoch. The model is trained for multiple epochs until the model reaches convergence. Moreover, the process of training and validation using subsequences is sped up by training the input data in groups subsequences known as mini-batches. Each mini-batch consists of multiple consecutive subsequences that are given to the model in parallel. The size of each mini-batch is commonly called batch size and it is a common practice to choose the batch size as a power of two. Lastly, to control the rate of update of parameters during backpropagation, a learning rate needs to be specified to the model. These hyperparameters such as subsequence size, batch size, learning rate, etc., are presented later in Section 5.5.1.

5.4.2. INFERENCE AND DETECTION

At runtime, the trained model is set to evaluation mode, meaning only the forward passes occur and the weights are not updated. In this phase, we can test for multiple attack scenarios

(mentioned in Section 5.3.3), by simulating appropriate attack condition in the CAN message dataset.

Every data sample that goes through the model gets reconstructed and the reconstruction loss is sent to the detection module to compute a metric called *intrusion score* (IS). The IS helps us in identifying whether a signal is malicious or normal. We compute IS at a signal level to predict the signal that is under attack. The IS is computed at every iteration during inference, as a *squared error* to estimate the prediction deviation from the input signal value, as shown in (22).

$$IS_i = \left(\left(S_i^j - \hat{S}_i^j\right)^2\right) \quad \forall\, i \in [1, m] \tag{22}$$

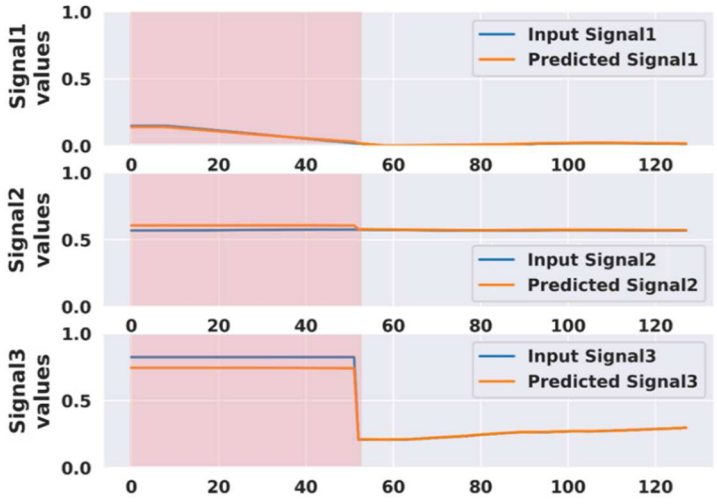
where, S_i^j represents i^{th} signal value at j^{th} sample, $\hat{S}_i^j$ denotes its reconstruction, and m is the number of signals in the message. We observe a large deviation for predicted value from the input signal value (i.e., large IS value), when the signal pattern is not seen during the training phase, and a minimal IS value otherwise. This is the basis for our detection phase.

As we do not have a signal level intrusion label information in the dataset, we combine the signal level IS information into a message-level IS, by taking the maximum IS of the signals in that message as shown in (23).

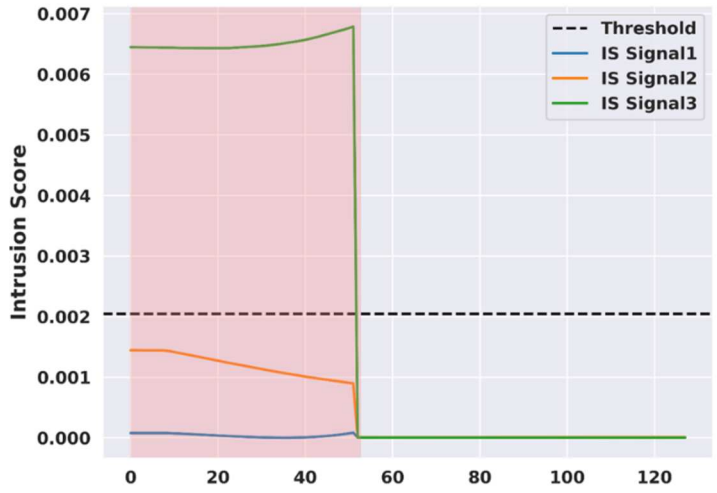
$$MIS = \max(IS_1, IS_2 \ldots, IS_m) \tag{23}$$

In order to get adequate detection accuracy, the intrusion threshold (IT) for flagging messages needs to be selected carefully. We explored multiple choices for IT, using the best model from the training process. The best model is defined as the model with the lowest validation running loss during the training process. From this model, we log multiple metrics such as

maximum, mean, median, 99.99%, 99.9%, 99% and 90% validation loss across all iterations as the choices for the IT. The analysis of IT metrics is presented in Section 5.5.2.



(a)



(b)

Figure 55 Snapshot of our proposed IDS checking a message with three signals under a plateau attack, where (a) shows the signal comparisons and (b) shows IS for signals and IS for the message and Intrusion flag.

A snapshot of our IDS working in an environment with attacks is illustrated in Figure 55(a) and Figure 55(b) with a plateau attack on a message with three signals, between time 0 and 50. Figure 55(a) shows the input (true) vs IDS predicted signal value comparisons for 3 signals. The red highlighted area represents the attack interval. It can be seen that for most of the time, the reconstruction is close for almost all signals except during the attack interval. Signal 3 is subjected to a plateau attack where the attacker held a constant value until the end of attack interval as shown in the third subplot of Figure 55(a) (note the larger difference between the predicted and actual input signal values in that subplot, compared to for signals 1 and 2). Figure 55(b) shows the different signal intrusion scores for the 3 signals. The dotted black line is the intrusion threshold (IT). As mentioned earlier, the maximum of signal intrusion scores is chosen as message intrusion score (MIS), which in this case is the IS of signal 3. It can be observed from Figure 55(b) that the intrusion score of signal 3 is above the IT, for the entire duration of the attack interval, highlighting the ability of *INDRA* to detect such attacks. The value of IT (equal to 0.002) in Figure 55(b) is computed using the method discussed in Section 5.5.2. Note that this value is specific to the example case shown in, Figure 55 and is not the threshold value used for our remaining experiments. Section 5.5.2 describes how we select the IT value for our framework.

5.5 EXPERIMENTS

5.5.1. EXPERIMENTAL SETUP

To evaluate the performance of the *INDRA* framework, we first present an analysis for the selection of intrusion threshold (IT). Using the derived IT, we contrast it against the two variants of the same framework: *INDRA*-LED and *INDRA*-LD. The former removes the linear layer before the output and essentially leaving the GRU to decode the context vector. The term LED implies,

(L)linear layer, (E) encoder GRU and (D) decoder GRU. The second variation replaces the GRU and the linear layer at the decoder with a series of linear layers (LD implies linear decoder). These experiments were conducted to test the importance of different layers in the network. However, the encoder end of the network is not changed because we require a sequence model to generate an encoding of the time-series data. We explored other variants as well, but they are not included in the discussion as their performance was poor compared to the LED and LD variants.

Subsequently, we compare the best variant of our framework with three prior works: Predictor LSTM (PLSTM [140]), Replicator Neural Network (RepNet [141]), and CANet [138]. The first comparison work (PLSTM) uses an LSTM based network that is trained to predict the signal values in the next message transmission. PLSTM achieves this by taking the 64-bit CAN message payload as the input, and learns to predict the signal at a bit-level granularity by minimizing the prediction loss. A log loss or binary cross-entropy loss function is used to monitor the bit level deviations between the real next signal values and the predicted next signal values, and the gradient of this loss function is computed using backpropagation to update the weights in the network. During runtime, PLSTM uses the prediction loss value to decide if that particular message is malicious or not. The second comparison work (RepNet) uses a series of RNN layers to increase the dimensionality of the input data and reconstruct the signal values by reducing back to the original dimensionality. RepNet achieves this by minimizing the mean squared error between the input and the reconstructed signal values. At runtime, large deviations between the input received signal and the reconstructed signal values are used to detect intrusions. Lastly, CANet unifies multiple LSTMs and linear layers in an autoencoder architecture and uses a quadratic loss function to minimize the signal reconstruction error. All experiments conducted with the *INDRA* variations and prior works are discussed in further subsections.

To evaluate our proposed framework with its variants and against prior works we used the SynCAN dataset that was developed by ETAS and Robert Bosch GmbH [138]. The dataset consists of CAN message data for 10 different IDs modeled after real-world CAN message data. The dataset comes with both training and test data with multiple attacks as discussed in Section 5.3.3. Each row in the dataset consists of a timestamp, message ID, and individual signal values. Additionally, there is a label column in the test data with either 0 or 1 values indicating normal or malicious messages. The label information is available on a per message basis and does not indicate which signal within the message is subjected to the attack. We use this label information to evaluate our proposed IDS over several metrics such as detection accuracy and false positive rate, as is discussed in detail in the next subsections. Moreover, to simulate a more realistic attack scenario in the in-vehicle networks, the test data has normal CAN traffic between the attack injections. *Note*: We do not use the label information in the training data when training our model, as our model learns the patterns in the input data in an unsupervised manner.

All the machine learning based frameworks (*INDRA* and its variants, and comparison works) are implemented using Pytorch 1.4. We conducted several experiments to select the best performing model hyperparameters (number of layers, hidden unit sizes, and activation functions). The final model discussed in Section 5.4.1 was trained using the SynCAN data set by splitting 85% of train data for training and the remaining for validation. The validation data is mainly used to evaluate the performance of the model at the end of every epoch. We trained the model for 500 epochs, using a rolling window approach (as discussed in Section 5.4.1.2) with the subsequence size of 20 messages and the batch size of 128. We also implemented an early stopping mechanism that monitors the validation loss across epochs and stops the training process if there is no improvement after 10 (patience) epochs. We chose the initial learning rate as 0.0001, and apply

tanh activations after each linear and GRU layers. Moreover, we used the ADAM optimizer with the mean squared error (MSE) as the loss criterion. During testing, we used the trained model parameters and considered multiple test data inputs to simulate attack scenarios. We monitored the intrusion score metric (as described in Section 5.4.2) and the computed intrusion threshold to flag the message as malicious or normal. We computed several performance metrics such as detection accuracy, false positives, etc. to evaluate the performance of our model. All the simulations are run on an AMD Ryzen 9 3900X server with an Nvidia GeForce RTX 2080Ti GPU.

Lastly, before showing the experimental results, we present the following definitions in the context of IDS:

- *True Positive (TP)*- when the IDS detects an actual malicious message as malicious;
- *False Negative (FN)*- when the IDS detects an actual malicious message as normal;
- *False Positive (FP)*- when the IDS detects a normal message as malicious (also known as false alarm);
- *True Negative (TN)*- when the IDS detects an actual normal message as normal.

We focus on two key performance metrics: *(i) Detection accuracy*- a measure of IDS ability to detect intrusions correctly, and *(ii) False positive rate*: also known as false alarm rate. These metrics are calculated as shown in (24) and (25):

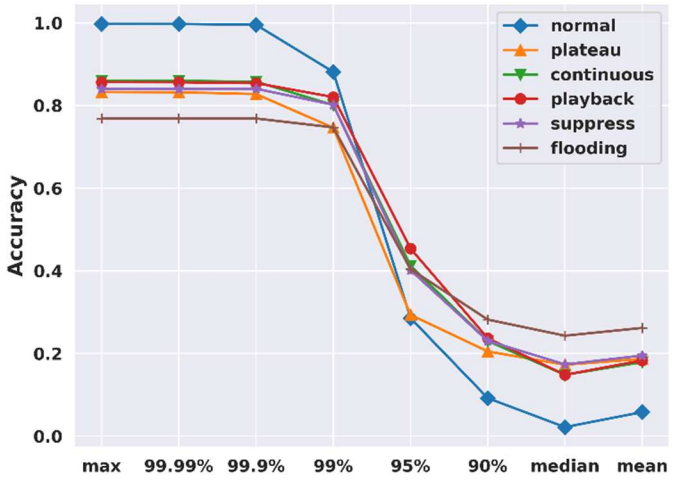
$$Detection\ Accuracy\ = \frac{TP+TN}{TP+FN+FP+TN} \tag{24}$$

$$False\ Positive\ Rate\ = \frac{FP}{FP+TN} \tag{25}$$

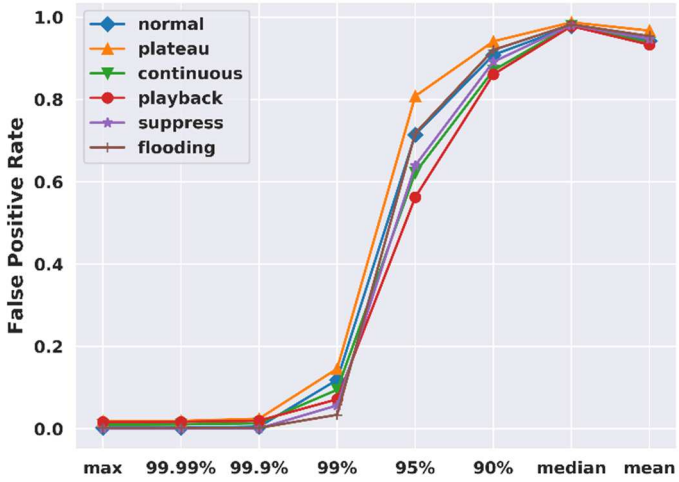
5.5.2. INTRUSION THRESHOLD SELECTION

In this subsection, we present an analysis for the selection of intrusion threshold (IT) by considering various options such as max, median, mean, and different quantile bins of validation loss of the final model. The model reconstruction error for the normal message should be much smaller than the error for malicious messages. Hence, we want to explore several candidate options to achieve this goal, that would work across multiple attack and no-attack scenarios. Having a large threshold value can make it harder for the model to detect the attacks that change the input pattern minimally (e.g., continuous attack). On the other hand, having a small threshold value can trigger multiple false alarms, which is highly undesirable. Hence it is important to select an appropriate threshold value to optimize the performance of the model.

Figure 56(a) and Figure 56(b) illustrate the detection accuracy and false positive rate respectively for various candidate options to calculate IT, under different attack scenario. It is clear from the results in the figure that selecting higher validation loss as the IT can lead to a high accuracy and low false alarm rate. However, choosing a very high value (e.g., ‘max’ or ‘99.99 percentile’) can sometimes result in missing small variations in the input patterns that are found in more sophisticated attacks. From our experiments we found the maximum and 99.99 percentile values to be very close. In order to capture the attacks that produce small deviations, we selected a slightly smaller threshold that would still perform similar to max and 99.99 percentile thresholds on all of our current attack scenarios. Hence, in this work, we choose the 99.9th percentile value of the validation loss as the value of the intrusion threshold (IT). We use the same IT value for the remainder of the experiments discussed in the next subsections.



(a)



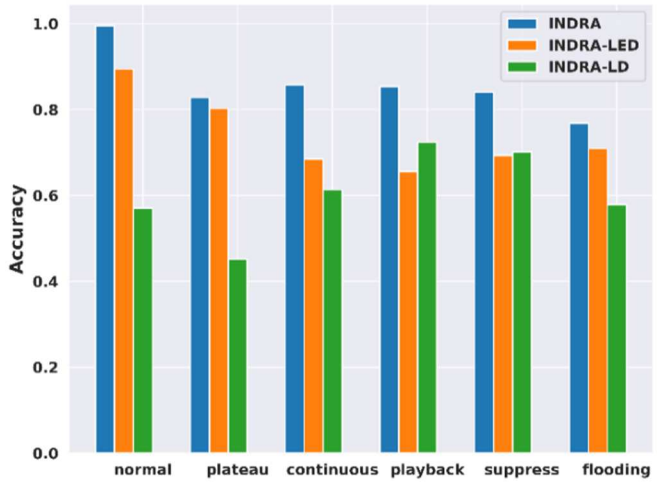
(b)

Figure 56 Comparison of (a) detection accuracy, and (b) false positive rate for various candidate options of intrusion threshold (IT) as a function of validation loss under different attack scenarios. (% refers to percentile not percentage).

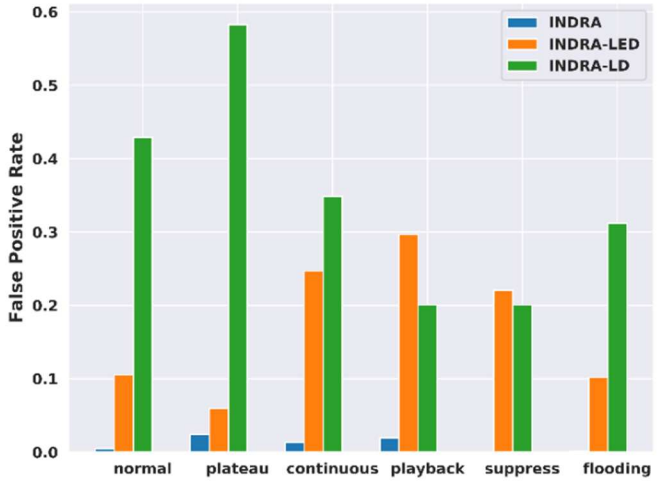
5.5.3. COMPARISION OF INDRA VARIANTS

After selecting the correct intrusion threshold from the previous subsection, we use that criterion and evaluate our proposed *INDRA* framework with two other variants: *INDRA*-LED, and

INDRA-LD. The main intuition behind evaluating different variants of *INDRA* is to analyze the impact of different types of layers in the model on the performance metrics discussed in Section 5.5.1.



(a)



(b)

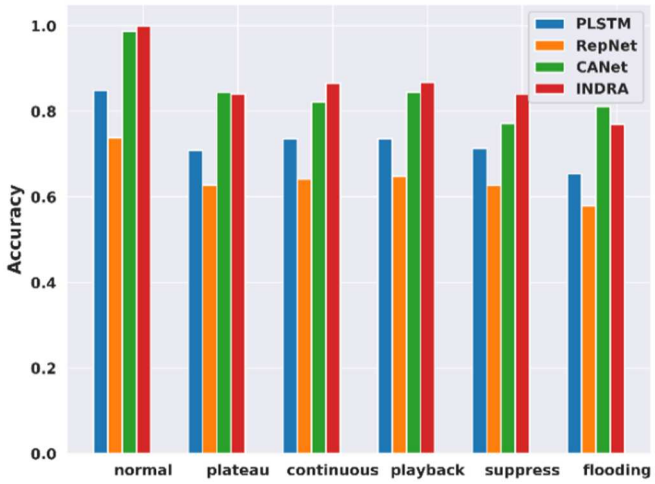
Figure 57 Comparison of (a) detection accuracy, and (b) false positive rate for *INDRA* and its variants *INDRA*-LED and *INDRA*-LD under different attack scenarios.

Figure 57(a) shows the detection accuracy for our *INDRA* framework and its variants on y-axis with different attack types and for a no-attack scenario (normal) on the x-axis. We can observe that *INDRA* outperforms the other two variants and has high accuracy in most of the attack

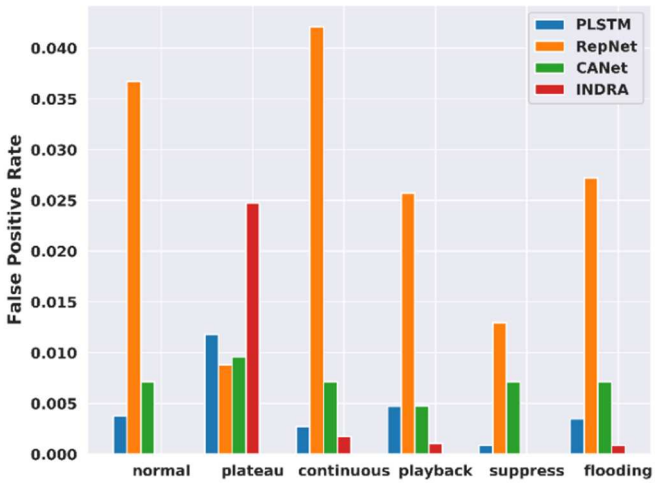
scenarios. It is to be noted that the high accuracy is achieved by monitoring at a signal level unlike prior works that monitor at the message level. The false positive rate or false alarm rate of *INDRA* and other variants under different attack scenarios is shown in Figure 57(b). It is evident that *INDRA* has the lowest false positive rate and highest detection accuracy compared to the other variants. Moreover, *INDRA*-LED which is just short of a linear layer at the decoder end is the second best performing model after *INDRA*. *INDRA*-LED's ability to use a GRU based decoder helps in reconstructing the MCV back to original signals. It can be clearly seen in both Figure 57(a) and Figure 57(b), that the lack of GRU layers on the output decoder end for *INDRA*-LD leads to a significant performance degradation. Hence, we chose *INDRA* as our candidate model for subsequent experiments.

5.5.4. COMPARISION WITH PRIOR WORKS

We compare our *INDRA* framework with PLSTM [140], RepNet [141] and CANet [138], which are some of the best known prior works in the IDS area. Figure 58(a)-(b) show the detection accuracy and false positive rate respectively for the various techniques under different attack scenarios.



(a)



(b)

Figure 58 Comparison of (a) detection accuracy, and (b) false positive rate of *INDRA* and the prior works PLSTM [140], RepNet [141] and CANet [138].

From the results shown in Figure 58, it is evident that *INDRA* achieves high accuracy for each attack scenario and also achieves low positive rates for most of the scenarios. The ability to monitor signal level variations along with the more cautious selection of intrusion threshold gives *INDRA* an advantage over comparison works. PLSTM and RepNet use the maximum validation loss in the final model as the threshold to detect intrusions in the system, while CANet uses interval based monitoring to detect attacks. Selecting a larger threshold helped PLSTM to achieve slightly lower false positive rates for some scenarios, but it hurt the ability of both PLSTM and RepNet to detect attacks with small variations in the input data. This is because the deviations produced by some of the complex attacks are small and due to the large thresholds the attacks go undetected. Moreover, the interval based monitoring in CANet struggles with finding an optimal value for the thresholds. Lastly, the false positive rates of *INDRA* are still significantly low with the maximum of 2.5% for plateau attacks. Please note that the y-axis in Figure 58(b) has a much smaller scale than in Figure 58(a), and the magnitude of the false positive rate is very small.

5.5.5. IDS OVERHEAD ANALYSIS

In this subsection, we present a detailed analysis of the overhead incurred by our proposed IDS. We quantify the overhead in terms of both memory footprint and time taken to process an incoming message i.e., inference time. The former metric is important as the resource constrained automotive ECUs have limited available memory, and it is crucial to have a low memory overhead to avoid interference with real-time automotive applications. The inference time not only provides important information about the time taken to detect the attacks, but also can be used to compute the utilization overhead on the ECU. Thus, we choose the above-mentioned two metrics to analyze the overhead and quantify the lightweight nature of our proposed IDS.

To accurately capture the overhead of our proposed *INDRA* framework and the prior works, we implemented our proposed IDS approach on an ARM Cortex- A57 CPU on a Jetson TX2 board, which has similar specifications to the state-of-the-art multi-core ECUs. Table 10 presents the memory footprint of our proposed *INDRA* framework and the prior works mentioned in the previous subsections. It is clear that our proposed *INDRA* framework has a low memory footprint compared to the prior works, except for the RepNet [141]. However, it is important to observe that even though our proposed framework has slightly higher memory footprint compared to the RepNet [141], we outperform all of the prior works including RepNet [141] in all performance metrics under different attack scenarios, as shown in Figure 58. The heavier (high memory footprint) models can provide the ability to capture a large variety of details about the system behavior, but they are not an ideal choice for resource constrained automotive systems. On the other hand, a much lighter model such as RepNet, fails to capture key details about the system behavior due to its limited parameters and therefore suffers from performance issues.

Table 10 Memory footprint comparison between *INDRA* framework and the prior works PLSTM [140], RepNet [141] and CANet [138].

Framework	Memory footprint (KB)
PLSTM [140]	13,417
RepNet [141]	55
CANet [138]	8,718
INDRA	443

In order to understand the inference overhead, we benchmarked the different IDS frameworks on an ARM Cortex- A57 CPU. In this experiment, we consider different system configurations to encompass a wide variety of ECU hardware that is available in the state-of-the-art vehicles. Based on the available hardware resources on the Jetson TX2, we selected two different system configurations. The first configuration utilizes only one CPU core (single core), while the second configuration uses two CPU cores.

Table 11 Inference time comparisons between *INDRA* framework and the prior works PLSTM [140], RepNet [141] and CANet [138] using single core, and dual core configurations.

Framework	Average inference time (µs)	
	Single core ARM Cortex A57 CPU	**Dual core ARM Cortex A57 CPU**
PLSTM [140]	681.18	644.76
RepNet [141]	19.46	21.46
CANet [138]	395.63	378.72
INDRA	80.35	72.91

We ran the frameworks 10 times for the different CPU configurations and computed the average inference time (in µs), as shown in Table 11. From the results in Table 11, it is clear that our proposed *INDRA* framework has significantly faster inference times compared to the prior works (excluding RepNet) under all three configurations. This is partly due to the lower memory footprint of our proposed IDS. As mentioned earlier, even though RepNet has a lower inference

time, it has the worst performance out of all frameworks, as shown in Figure 58. The large inference times for the better performing frameworks can impact the real-time performance of the control systems in the vehicle, and can result in catastrophic missing of deadlines. We also believe that using a dedicated deep learning accelerator (DLA) would give us significant speed up compared to the above presented configurations.

Thus, from Figure 58, and Table 10 and Table 11, it is clear that *INDRA* achieves a clear balance of having superior intrusion detection performance while maintaining low memory footprint and fast inference times, making it a powerful and lightweight IDS solution.

5.5.6. SCALABILITY RESULTS

In this subsection we present an analysis on the scalability of our proposed IDS by studying the system performance using the ECU utilization metric as a function of increasing system complexity (number of ECUs and messages).

Each ECU in our system model has a real-time utilization (U_{RT}) and an IDS utilization (U_{IDS}) from running real-time and IDS applications respectively. In this work, we primarily focus on analyzing the IDS overhead (U_{IDS}), as it is a measure of the compute efficiency of the IDS. Since the safety-critical messages monitored by the IDS are periodic in nature, the IDS can be modeled as a periodic application with period that is the same as the message period [46]. Thus, monitoring an i^{th} message m_i results in an induced IDS utilization ($U_{IDS,\, m_i}$) at an ECU, and can be computed as:

$$U_{IDS,m_i} = \left(\frac{T_{IDS}}{P_{m_i}}\right) \tag{26}$$

where, T_{IDS} and P_{mi} indicate the time taken by the IDS to process one message (inference time), and the period of the monitored message, respectively. Moreover, the sum of all IDS utilizations as a result of monitoring different messages is the overall IDS utilization at that ECU (U_{IDS}) and is given by:

$$U_{IDS} = \sum_{i=1}^{n} U_{IDS,m_i} \tag{27}$$

To evaluate the scalability of our proposed IDS, we consider six different system sizes. Moreover, we consider a pool of commonly used message periods {1, 5, 10, 15, 20, 25, 30, 45, 50, 100} (all periods in ms) in automotive systems to sample uniformly, when assigning periods to the messages in the system. These messages are evenly distributed among different ECUs and the IDS utilization is computed using (26) and (27). In this work, we assume a pessimistic scenario where all of the ECUs in the system have only a single core. This would allow us to analyze the worst case overhead of the IDS.

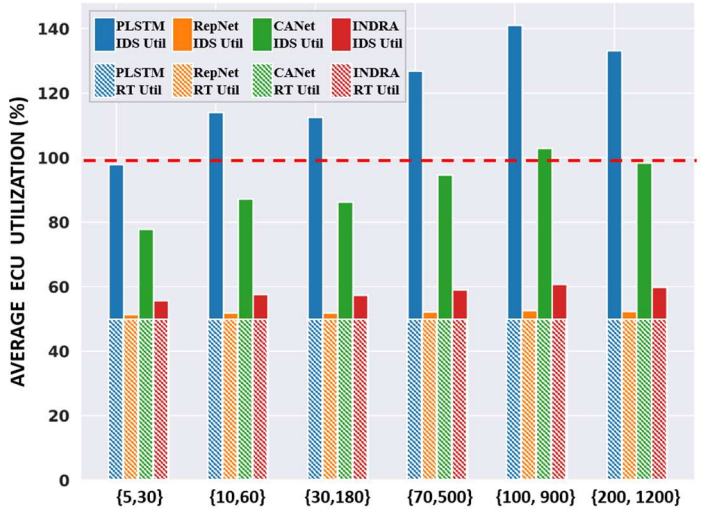


Figure 59 Scalability results of our proposed IDS for different system sizes and the prior works PLSTM [140], RepNet [141] and CANet [138].

Figure 59 illustrates the average ECU utilization under various system sizes denoted by {p, q}, where p is the number of ECUs and q is the number of messages in the system. A very pessimistic estimate of 50% real-time ECU utilization for real-time automotive applications is assumed ("RT Util", as shown in the dotted bars). The solid bars on top of the dotted bars represent the overhead incurred by the IDS executing on the ECUs, and the red horizontal dotted line represents the 100% ECU utilization mark. It is important to avoid exceeding the 100% ECU utilization under any scenario, as it could induce undesired latencies that could result in missing deadlines for time-critical automotive applications, which can be catastrophic. It is clear from the results that the prior works PLSTM and CANet incur heavy overhead on the ECUs while RepNet and our proposed *INDRA* framework have very minimal overhead that scales favorably to increasing system sizes. From the results in this section (Figure 58, Figure 59; Table 10, Table 11), it is apparent that not only does *INDRA* achieve better performance in terms of both accuracy and low false positive rate for intrusion detection than state-of-the-art prior work, but it is also lightweight and scalable.

5.6 CONCLUSION

In this chapter, we proposed a novel recurrent autoencoder based lightweight intrusion detection system called *INDRA* for distributed automotive embedded systems. We proposed a metric called the intrusion score (IS), which measures the deviation of the prediction signal from the actual input. We also presented a thorough analysis of our intrusion threshold selection process and compared our approach with the best known prior works in this area. The promising results indicate a compelling potential for utilizing our proposed approach in emerging automotive

platforms. In our future work, we plan to exploit the dependencies that exist between signals to improve the performance of our intrusion detection framework.

6. LATTE: LSTM SELF-ATTENTION BASED ANOMALY DETECTION IN EMBEDDED AUTOMOTIVE PLATFORMS

Modern vehicles are experiencing a rapid increase in the complexity of embedded systems integrated into various vehicle subsystems, due to the increased interest in autonomous driving. The aggressive competition between automakers to reach autonomy goals is further driving the complexity of Electronic Control Units (ECUs) and the communication network that connects them [146]. Additionally, recent solutions for Advanced Driver Assistance Systems (ADAS) require interactions with various external systems using a variety of communication standards such as 5G, Wi-Fi, and Vehicle-to-X (V2X) protocols [2]. The V2X communication facilitates a spectrum of connections such as vehicle-to-vehicle (V2V), vehicle-to-pedestrian (V2P), vehicle-to-infrastructure (V2I), and vehicle-to-cloud (V2C) [147]. These new solutions are transforming modern vehicles by making them more connected to the external environment. To support the increasingly sophisticated ADAS functionality and connectivity to the outside world, highly complex software is required to run on the ECUs in such vehicles, to handle highly safety-critical and time-sensitive automotive applications, e.g., pedestrian and traffic sign detection, lane changing, automatic parking, path planning, etc. This increased software and hardware complexity of the automotive electrical/electronic (E/E) architecture and increased connectivity with external systems has an important implication: it provides a large attack surface and thus gives rise to more opportunities for attackers to gain unauthorized access to the in-vehicle network and execute cyber-attacks. The complexity in emerging vehicles also leads to poor attack visibility over the network, making it hard to detect attacks that can be easily hidden within normal operational activities. Such cyber-attacks on vehicles can induce various anomalies in the network, altering

the normal behavior of the network as well as the compute system (ECU) behavior. Due to the time-sensitive and safety-critical nature of automotive applications, any minor instability in the system due to these induced anomalies could lead to a major catastrophe, e.g., delaying the perception of a pedestrian, preventing an airbag from deploying in the case of a collision, or erroneously changing lanes into oncoming traffic, due to maliciously corrupted sensor readings.

An attack via an externally-linked component or compromised ECU can manifest in several forms over the in-vehicle network. One of the most commonly observed attacks is flooding the in-vehicle network with random or specific messages which increases the overall network load and results in halting any useful activity over the network. An advanced remote attack on an ECU could involve sending a kill command to the engine during normal driving. More sophisticated attacks could involve installing malware on the ECU and using it to achieve malicious goals. Some of the recent state-of-the-art attacks have used a vehicle's infotainment system as an attack vector to launch buffer overflow and denial of service attacks [148], performed reverse engineering of keyless entry automotive systems to wirelessly lock pick the vehicle immobilizer [149], etc. Researchers in [150] foresee a much more severe attack involving potentially targeting the U. S. electric power grid by using public electric vehicle charging stations as an attack vector to infect vehicles that use these stations with malware. Many other attacks on real-world vehicles are presented in [25], [26], [90], [151]. The common aspect of these attacks is that they involve gaining unauthorized access to the in-vehicle network and modifying certain fields in the message frames, thereby tricking the receiving ECU into thinking that the malicious message is legitimate. All of these attacks can have catastrophic effects and need to be detected before they are executed. This problem will get exacerbated with the onset of connected and autonomous vehicles. Hence,

restricting external attackers via early detection of their attacks is vital to realizing secure automotive systems.

Conventional computer networks utilize protective mechanisms such as firewalls (software) and isolation units such as gateways and switches (hardware) to protect from external attacks [152]. However, persistent attackers have been coming up with advanced attacks that leverage the increased compute and communication capabilities in modern ECUs, causing the traditional protection systems to become obsolete. This raises a need for a system-level solution that can continuously monitor the vehicle network, to detect cyber-attacks. One promising solution is to deploy a software framework for anomaly detection, which involves monitoring the network for unusual activities and raising an alarm when suspicious activity is detected. This approach can be extended to detect and classify various types of attacks on the in-vehicle network. Such a framework can learn the normal system behavior at design time and monitor the network for anomalies at runtime. A traditional approach for anomaly detection uses rule-based approaches such as monitoring message frequency [133], memory heat map [130], etc., to detect known attack signatures. However, due to the increased complexity of cyber-attacks, such traditional rule-based systems fail to recognize new and complex patterns, rendering these approaches ineffective. Fortunately, recent advances in deep learning and the availability of in-vehicle network data have brought forth the possibility of using sophisticated deep learning models for anomaly detection.

In this chapter, we present a novel anomaly detection framework called *LATTE* to detect cyber-attacks in the Controller Area Network (CAN) based automotive networks. Our proposed *LATTE* framework uses sequence models in deep learning in an unsupervised setting to learn the normal system behavior. *LATTE* leverages that information at runtime to detect anomalies by observing for any deviations from the learned normal behavior. This is illustrated in Figure 60.

The plot on the top right shows the expected deviation (computed using the model that was trained at design time) vs the observed deviation. The divergence in signal values during the attack intervals (shown in red area) can be used as a metric to detect cyber-attacks as anomalies. Our proposed *LATTE* framework aims to maximize the anomaly detection accuracy, precision, and recall, while minimizing the false-positive rate.

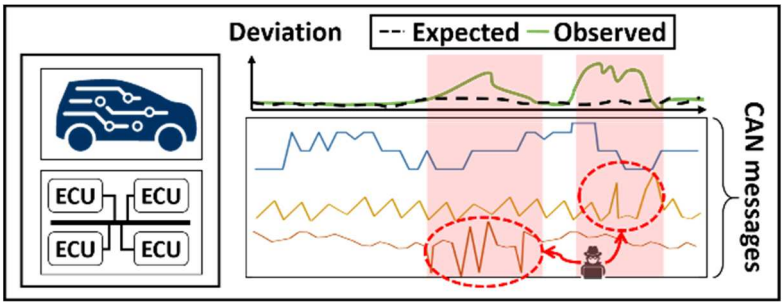


Figure 60 An example of an anomaly detection framework that monitors the network traffic and detects deviations from expected normal behavior during the attack intervals (shown in red).

Our novel contributions in this chapter can be summarized as follows:

- We propose a stacked Long-Short Term Memory (LSTM) based predictor model that integrates a novel self-attention mechanism to learn the normal automotive system behavior at design time;
- We design a one class support vector machine (OCSVM) based detector that works with the LSTM self-attention predictor model to detect different cyber-attacks at runtime;
- We present modifications to existing vehicle communication controllers that can help in realizing the proposed anomaly detection system on a real-world ECU;
- We perform a comprehensive analysis on the selection of deviation measures that quantify the deviation from the normal system behavior;

- We explore several variants of our proposed *LATTE* framework and selected the best performing one, which is then compared with the best-known prior works in the area (statistical-based, proximity-based, and ML-based works), to show *LATTE'*s effectiveness.

6.1. RELATED WORK

Several automotive attacks have been studied by researchers to discover vulnerabilities in automotive systems. Recent attacks such as [153] exploit the vulnerability in security access algorithms to deploy airbags without any actual impact. The attackers in [154] reverse engineered a telematics control unit to exploit a memory vulnerability in the firmware to circumvent the existing firewall and remotely send diagnostic messages to control an ECU. Other attacks that compromised the ADAS camera sensor were studied in [155]. All of these attacks create anomalous behavior during vehicle operation, which a good anomaly detection framework must detect.

Anomaly detection has been a popular research topic in the domain of computer networks, and several solutions have been proposed to detect cyber-attacks in large-scale computer networks [156]. Although some of these solutions are highly successful in defending computer networks against various attacks, they require high compute power. The resource-constrained nature of automotive systems makes many of these solutions hard to adapt for detecting cyber-attacks in the in-vehicle networks. In the past decade, several solutions were developed to tackle the problem of anomaly detection in automotive systems [47], [106], [121], [122], [124], [127], [128], [131], [137], [138], [140]-[142], [157]-[159]. These works can be broadly divided into two categories *(i)* heuristic-based, and *(ii)* machine learning based. Heuristic-based anomaly detection approaches typically observe for traces of known attack patterns, whereas a machine-learning-based approach

can learn the normal behavior during an offline phase and observes for any deviation from the learned normal behavior at run-time, to detect anomalies. The heuristic-based techniques can be simple and have fast detection times when compared to machine learning based techniques. However, machine learning based techniques can detect both known and unknown attacks, which is not possible with heuristic based techniques. Some of the key prior works in these categories are discussed in the rest of this section.

6.1.1. HEURISTIC BASED ANOMALY DETECTION

The authors in [121] used a language theory-based model to obtain signatures of known attacks from the vehicle's CAN bus. However, their approach fails to detect anomalous sequences when the model misses the packets transmitted during the early stages of an attack. In [122], the authors used transition matrices to detect anomalous sequences in a CAN bus-based system. This approach was able to achieve low false-positive rates for simple attacks but failed to detect realistic replay attacks. The authors in [157] proposed a Hamming-distance based model which monitors the CAN network to detect attacks. However, the model had very limited attack coverage. In [124], the authors proposed a specification-based approach and compared it with predefined attack patterns to detect anomalies. In [127], a time-frequency analysis model is used to continuously monitor CAN message frequency to detect anomalies. In [106], a heuristic-based approach is used to build a normal operating region by analyzing the messages at design time and using a message-frequency-based in-vehicle network monitoring system to detect anomalies at runtime. The authors in [128] use a clock-skew based fingerprint to detect anomalies by observing the variations in clock-skew of sender ECUs at runtime. In [131], the authors propose an anomaly detection system that monitors the entire system for change in entropy to detect anomalies. However, their approach fails to detect smaller anomalous sequences that result in minimal change in the entropy. *In a*

nutshell, heuristic-based anomaly detection systems provide low-cost and high-speed detection techniques but fail to detect complex and new attacks. Additionally, modeling every possible attack signature is practically impossible, and hence these anomaly detection approaches have a limited scope.

6.1.2. MACHINE LEARNING BASED ANOMALY DETECTION

Recent works leverage advances in machine learning to build highly efficient anomaly detection systems. A deep neural network (DNN) based approach was introduced in [137], that continuously monitors the network and observes for change in communication patterns. However, this approach is only designed and tested for a low priority system (a tire pressure monitoring system), which limits us from directly adapting this technique to safety-critical systems. In [141], the authors proposed a recurrent neural network (RNN) based intrusion detection system that attempts to learn the normal behavior of CAN messages in the in-vehicle network. A hybrid approach was proposed in [142], which utilizes both specification and RNN based systems in two stages to detect anomalies. In [140] the authors propose an LSTM based predictor model that predicts the next time step message value at a bit level and detects intrusions by observing for large deviations in prediction errors. A long short-term memory (LSTM) based multi message-id detection model was proposed in [138]. However, the model is highly complex and has a high implementation overhead when deployed on an ECU. In [47], the authors proposed a GRU-based lightweight recurrent autoencoder and a static threshold-based detection scheme to detect various attacks in the in-vehicle network. The use of static threshold values for detection limits the scheme to detecting only very simple attacks. In [158], the authors propose a deep convolutional neural network (CNN) model to detect anomalies in the vehicle's CAN network. However, the model does not consider the temporal relationships between messages, which can better predict certain

attacks. The authors in [159] proposed an LSTM framework with a hierarchical attention mechanism to reconstruct the input messages. A non-parametric kernel density estimator along with a k-nearest neighbors classifier is used to reconstruct the messages and the reconstruction error is used to detect anomalies. *Although most of these techniques attempt to increase the detection accuracy and attack coverage, none of them offers the ability to process very long sequences with relatively low memory and runtime overhead and still achieve reasonably high performance.*

In this chapter, we propose a robust deep learning model that integrates a stacked LSTM based encoder-decoder model with a self-attention mechanism, to learn normal system behavior by learning to predict the next message instance. Table 12 summarizes some of the state-of-the-art anomaly detection works and their key features and highlights the unique characteristics of our proposed *LATTE* framework. At runtime, we continuously monitor in-vehicle network messages and provide a reliable detection mechanism using a non-linear classifier. Sections 6.3 and 6.4 provide a detailed explanation of the proposed model and overall framework. In Section 6.5 we show how our model is capable of efficiently identifying a variety of attack scenarios.

Table 12 Comparison between our proposed *LATTE* framework and state-of-the-art works.

Technique	Task	Network architecture	Attention type	Detection technique	Requires labeled data?
BWMP [140]	Bit level prediction	LSTM network	-	Static threshold	Yes
RepNet [141]	Input recreation	Replicator network	-	Static threshold	No
HAbAD [159]	Input recreation	Autoencoder	Hierarchical	KDE and KNN	Yes
LATTE	Next message value prediction	Encoder-decoder	Self-attention	OCSVM	No

6.2. BACKGROUND

Solving complex problems using deep learning was made possible due to advances in computing hardware and the availability of high-quality datasets. Anomaly detection is one such problem that can leverage the power of deep learning. In an automotive system, ECUs exchange safety-critical messages periodically over the in-vehicle network. This time series exchange of data results in temporal relationships between messages, which can be exploited to detect anomalies. However, this requires a special type of neural network, called Recurrent Neural Network (RNN) to capture the temporal dependencies between messages. Unlike traditional feed-forward neural networks where the output is independent of any previous inputs, RNNs use previous sequence state information in computing the output, which makes them an ideal choice to handle time-series data.

6.2.1. RECURRENT NEURAL NETWORK (RNN)

An RNN [160] is the most basic sequence model that takes sequential data such as time-series data as the input and learns the underlying temporal relationships between data samples. An RNN block consists of an input, an output, and a hidden state that allows it to remember the learned temporal information. The input, output, and hidden state all correspond to a particular time step in the sequence. The hidden-state information can be thought of as a data point in the latent space that contains important temporal information about the inputs from previous time steps. The current stage output of an RNN is computed by taking the previous hidden-state information along with the current input. Moreover, since the backpropagation in RNNs occurs through time, the error value shrinks or grows rapidly leading to vanishing or exploding gradients. This severely restricts RNNs from learning patterns in the input data that have long-term dependencies [161].

To overcome this problem, long short-term memory (LSTM) networks [162] with additional gates and states were introduced.

6.2.2. LONG SHORT-TERM MEMORY (LSTM) NETWORK

LSTMs are enhanced RNNs that consist of a cell state, hidden state, and multiple additional gates that help in learning long-term dependencies. The cell state carries the relevant long-term dependencies throughout the processing of an input sequence, whereas the hidden state contains relevant information from the recent time steps accommodating short-term dependencies. The gates in LSTM regulate the flow of the information from the hidden state to the cell state. These combinations of gates and states give LSTM an edge over the simple RNN in remembering long-term dependencies in sequences. LSTMs have therefore replaced simple RNNs in the areas of natural language processing, time-series forecasting, and machine translation [161].

In general, LSTMs overcome many of the limitations of RNNs and provide a more than acceptable solution for the vanishing and exploding gradient descent problems. However, their performance drops significantly when handling very long sequences (e.g., with 100 or more time steps). This is mainly because the predictions of an LSTM unit at the current time step *t,* are heavily influenced by its previous hidden state and cell state at time step *t-1* as compared to the past time steps. Therefore, for a very long input sequence, the representation of the input at the first time step tends to diminish as the LSTM processes inputs at the future time steps. To overcome this limitation, we need a mechanism that can look back and identify the information that can influence future sequences. One such look-back mechanism is neural attention, which is discussed next.

6.2.3. ATTENTION

Attention, or neural attention is a mechanism in neural networks that can mimic the visual attention mechanism in humans [163]. A human eye can focus on certain objects or regions with higher resolution compared to their surroundings. Similarly, the attention mechanism in neural networks can allow focusing on the relevant parts of the input sequence and selectively output only the most relevant information. While sequence models such as LSTMs typically take the previous hidden state information and the input at the current time step to compute the current output, they suffer in performance when processing very long input sequences as the information from the first time step is less representative in the hidden states compared to the information from the very recent time steps. Incorporating attention mechanisms with LSTMs can overcome this problem by allowing the sequence models to capture the crucial information from any past time steps of the input sequence.

Attention mechanisms are frequently used in encoder-decoder architectures [161]. An encoder-decoder architecture mainly consists of three major components *(i)* encoder, *(ii)* latent vector, and *(iii)* decoder. The encoder converts the input sequence to a fixed-size latent representation called a latent vector. The latent vector contains all the information representing the input sequence in a latent space. The decoder takes the latent vector as input and converts it to the desired output. However, due to the latent vector's fixed-length representation of the input sequence, it fails to encapsulate all the information from a very long input sequence, thereby resulting in poor performance. To address this problem, the authors in [164] introduced an attention mechanism in sequence models that enabled encoders to build a context vector by creating customized shortcuts to parts of the inputs. This ensures that the context vector represents the crucial parts and learns the very long-term dependencies in the input sequence leading to

improved decoder outputs. In [165], the authors propose a self-attention mechanism for an LSTM encoder-decoder model that consumes all the encoder hidden states to compute the attention weights.

An illustration of generating the input to the decoder in an LSTM-based encoder-decoder model rolled out in time for 4 time steps is shown in Figure 61. The input to the LSTM at each time step is represented as (x_t) and the initial hidden vector is h_0. The colored rectangle next to each LSTM unit for every time step represents the hidden state information and the height of each color signifies the amount of information from each time step. Inside the LSTM cell at each time step, a square filled with a different color is used to represent the hidden state information of that time step. Moreover, for this example, we consider a scenario where the output at the last time step ($t=4$) has a high dependency on the input at the second time step (x_2). We can see that in Figure 61(a), the LSTM hidden state at $t=4$ largely comprises of information from the third (blue) and fourth (orange) time steps. This results in sending the decoder an incorrect representation of current time step dependency, which leads to poor results at the output of the decoder. On the contrary, in Figure 61(b), the self-attention block consumes all hidden state representations at each time step as well as the current time step ($t=4$) and generates the context vector (decoder input). It can be observed that the self-attention mechanism clearly captures the high dependency of output at $t=4$ on the output at $t=2$ (shown in the hidden state information at the output of self-attention). This can also be seen in the attention weights computed by the self-attention where the information from the second (green) time step is given high weightage compared to others. Therefore, by better representing the important parts of the input sequence in the decoder input, the self-attention mechanism is able to facilitate better decoder outputs. Also, unlike other attention mechanisms such as [166], the attention vector in self-attention aligns encoder outputs to encoder hidden states,

thereby removing the need for any feedback from previous decoder predictions. Moreover, due to the lack of a decoder feedback loop, the self-attention mechanism can quickly learn the temporal dependencies in the long input sequences. In this work, for the first time, we adapt the self-attention mechanism to a stacked LSTM based encoder-decoder network to learn the temporal relationships between messages in a CAN based automotive network.

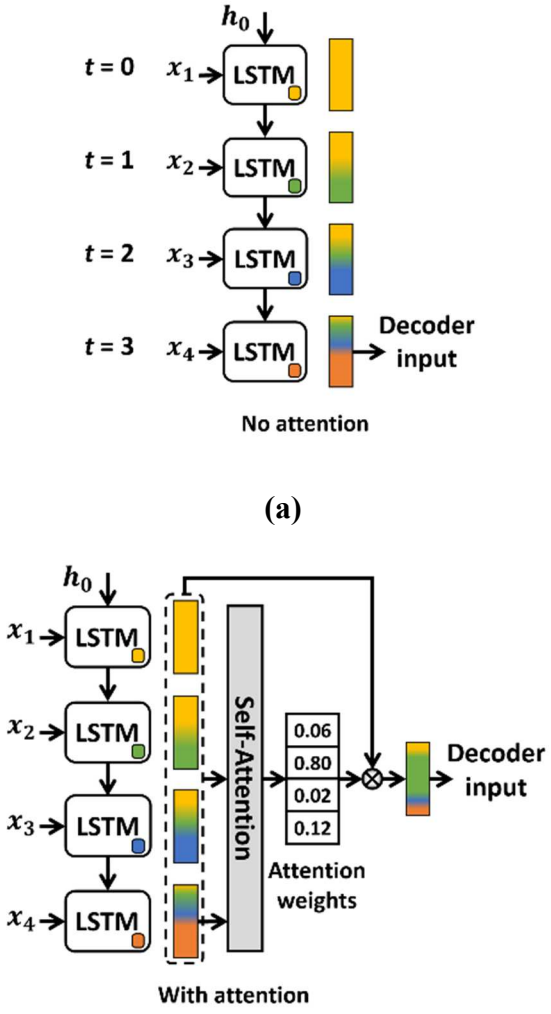


Figure 61 Comparison of input to the decoder in case of (a) no attention, (b) with attention in sequence models using LSTMs.

6.3. PROBLEM FORMULATION

6.3.1. SYSTEM OVERVIEW

In this work, we consider an automotive system that consists of multiple ECUs connected using a CAN based in-vehicle network, as shown in Figure 62. Each ECU consists of three major components: *(a)* processor, *(b)* communication controller, and *(c)* transceiver. A processor can have single or multiple cores that are used to execute real-time automotive applications. Most of these automotive applications are hard real-time and have strict timing and deadline constraints. Each application can be modeled as a set of data dependent and independent tasks mapped to different ECUs. The dependent tasks communicate by exchanging messages over the CAN network. A communication controller acts as the interface between the computation and communication realms. It facilitates the data movement from the processor to the network fabric and vice versa. Some of the important functions of a communication controller include packing of data from the processor into CAN frames, managing the transmission and reception of CAN frames, and filtering CAN messages based on the pre-programmed CAN filters (done by the original equipment manufacturer (OEM) when programming the communication controller). Lastly, a transceiver acts as an interface between the physical CAN network and the ECU, and facilitates the transmission and reception of CAN frames to and from the network respectively. In this work, we do not consider monitoring the execution within the CAN hardware IPs as it would require access to proprietary information that is only available to OEMs. We therefore assume that the proprietary CAN hardware IPs are “black boxes” and design an anomaly detection solution that does not require the complexity that comes with monitoring the internals of these IP blocks. This assumption is also consistent with all prior works on in-vehicle network anomaly detection work. However, if we were able to get access to this hardware stack and the program execution on

the CAN hardware IP, our framework can be extended to analyze CAN IPs and detect the attacks before they appear on the in-vehicle network.

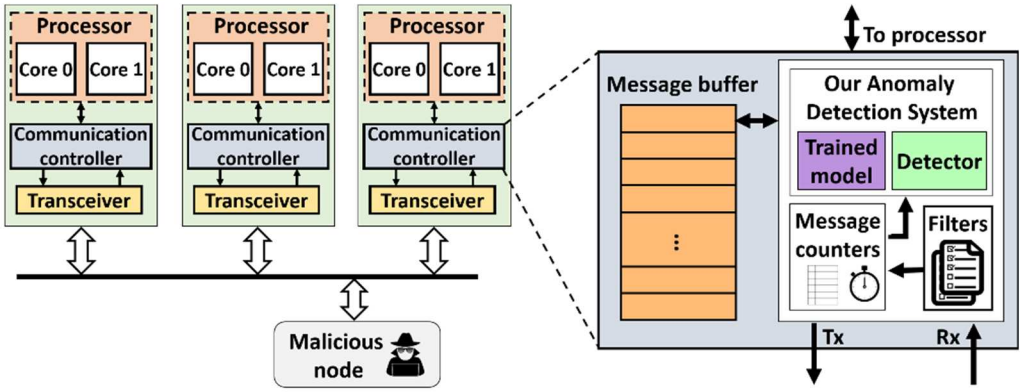


Figure 62 Overview of the system model with our proposed modifications to the communication controller.

To accommodate anomaly detection, we require modifications to existing CAN communication controllers. A traditional CAN communication controller consists of message filters that are used to filter out unwanted CAN messages and message buffers to temporarily store the messages before they are sent to the processor. This can be observed in the right region of Figure 62. We introduce message counters to this controller, which take the output of the message filters and keep a track of message frequencies. This bookkeeping helps in the observation of any abnormal message rates that may occur during a distributed denial of service (DDoS) attack (see Section 6.3.3). After confirming the message rate, the message is sent to the deployed anomaly detection system where it goes through a two-step process to determine whether the message is anomalous or not.

In the first step, our trained LSTM based attention model is used to predict the next message instance, which is then used to compute the deviation from the true message. This deviation measure is given as the input to a detector unit that uses a non-linear classifier to determine if a

given deviation measure represents a normal or an anomalous message. The details related to the models and the deviation metrics used in our framework are discussed in detail in sections 6.4.2 and 6.4.3 respectively. Messages are temporarily stored in the message buffer before they are validated and sent to the processor. If the anomaly detection system determines a particular message to be anomalous, it is discarded from the buffer and will not be sent to the processor, thereby avoiding the execution of attacker messages.

Note: Our anomaly detection system is implemented in the communication controller instead of a centralized ECU to *(i)* avoid single-point failures, *(ii)* prevent scenarios where the in-vehicle network load increases significantly due to high message injection (e.g., due to a DDoS attack, explained in Section 6.3.3), where the centralized ECU will not be able to communicate with a target ECU, and *(iii)* enable independent and immediate detection without delay compared to relying on a message from a centralized ECU. Lastly, we chose the communication controller instead of the processor to avoid jitter in real-time application execution.

6.3.2. COMMUNICATION OVERVIEW

In this work, we consider Controller Area Network (CAN) as the in-vehicle network protocol that is used for exchanging time-critical messages between ECUs. CAN is a lightweight, low-cost, event-triggered in-vehicle network protocol, and is the defacto industry standard. Several variants of CAN have been proposed over time, but the CAN standard 2.0B remains the most popular and widely used in-vehicle network protocol till today.

A CAN message consists of one or multiple signal values. Each signal contains independent information corresponding to a sensor value, actuator control, or computation output of a task on an ECU. Signals are grouped with additional information to form CAN frames. Each CAN frame

mainly consist of a header, payload, and trailer segments (Figure 63). The header consists of an 11-bit (CAN standard) or 29-bit (CAN extended) unique message identifier and a 6-bit control field. This is followed by a 64-bit payload segment and a 15-bit cyclic redundancy check (CRC) field in the trailer segment. The payload segment consists of multiple signals that are arranged in a predetermined order as per the definitions in the CAN database (.dbc) files. In addition, the CAN frame also has a 1-bit start of the frame (SOF) field at the beginning of the header, two 1-bit delimiters separating the 1-bit acknowledgment (ACK) field, and a 7-bit end of frame (EOF) field in the trailer segment.

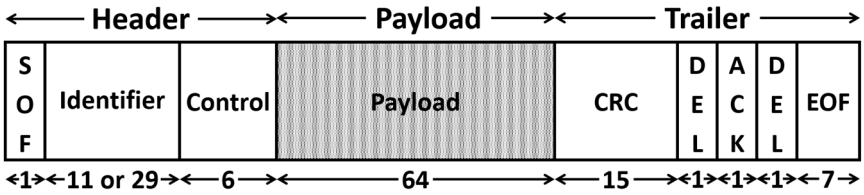


Figure 63 Controller Area Network (CAN) 2.0B communication frame.

In this work, our proposed anomaly detection framework operates on the payload segment of the CAN frame i.e., signals within each message. The main motivation for monitoring the payload field is because the attacker needs to modify the bits in the payload to launch any attack (a modification in the header or trailer segments would simply result in the frame getting invalidated at the receiving ECU). Our proposed *LATTE* framework learns the temporal dependencies between the message instances at design time by learning to predict the next message instances and observe for deviations at runtime to detect cyber-attacks. Moreover, as our framework mainly focuses on monitoring of the payload field, our technique is agnostic to the in-vehicle network protocol and can be extended to other in-vehicle network protocols such as CAN-FD, FlexRay, etc., with minimal changes. The details related to the detection of cyber-attacks using our proposed anomaly detection system are presented in sections 6.4.2 and 6.4.3.

6.3.3. THREAT MODEL

We assume that the attacker can gain access to the in-vehicle network using the most common threat vectors such as connecting to the vehicle OBD-II port, probing into the in-vehicle network, and via advanced threat vectors such as connected V2X ADAS systems, insecure infotainment systems, or by replacing a trusted ECU with a malicious ECU. We also assume that the attacker has access to the in-vehicle network parameters such as flow control, BAUD rate, parity, channel information, etc. that can be obtained by a simple CAN data logger, and can help in the transmission of malicious messages. We further assume a pessimistic situation where the attacker can access the in-vehicle network at any instance and try to send malicious messages.

Given the above assumptions, our proposed anomaly detection system tries to protect the in-vehicle network from the multiple types of cyber-attacks listed below. These attacks are modeled based on the most common and hard-to-detect attacks in the automotive domain.

1. *Constant attack*: In this attack, the attacker overwrites the signal value to a constant value for the entire duration of the attack interval. The complexity of detection of this attack depends on the change in magnitude of signal value. Intuitively, a small change in the magnitude of the signal value is harder to detect than larger changes.
2. *Continuous attack*: In this attack, the attacker tries to trick the anomaly detection system by continuously overwriting the signal value in small increments until a target value is achieved. The complexity of detecting this attack depends on the rate of change of the signal value. Larger change rates are easier to detect than smaller rates.
3. *Replay attack*: In this attack, the attacker plays back a valid message transmission from the past, tricking the anomaly detector into believing it to be a valid message. The complexity

for detecting this attack depends mainly on the frequency and sometimes on the duration of the playbacks. High-frequency replays are easier to detect compared to low-frequency replays.

4. *Dropping attack*: In this attack, the attacker disables the transmission of a message or group of messages resulting in missing or dropping of communication frames. The complexity of detecting this attack depends on the duration for which the messages are disabled. Longer durations are easier to detect due to missing message frames for a prolonged time compared to shorter durations.
5. *Distributed Denial of Service (DDoS) attack*: In this attack, the attacker floods the in-vehicle network with an arbitrary or specific message with the goal of increasing the overall bus load and rendering the bus unusable for other ECUs. This is the most common and easy to launch attack as it requires no information about the nature of the message. These attacks are fairly simple to detect even using a rule-based approach as the message frequencies are fixed and known at design time for automotive systems. Any deviation in this message rate can be used as an indicator for detecting this attack.

Problem objective: The main objective of our work is to develop a real-time anomaly detection framework that can detect various cyber-attacks in CAN-based automotive networks, that has *(i)* high detection accuracy, *(ii)* low false-positive rate, *(iii)* high precision and recall, *(iv)* large attack coverage, and *(v)* minimal implementation overhead (low memory footprint, fast runtime) for practical anomaly detection in resource-constrained ECUs.

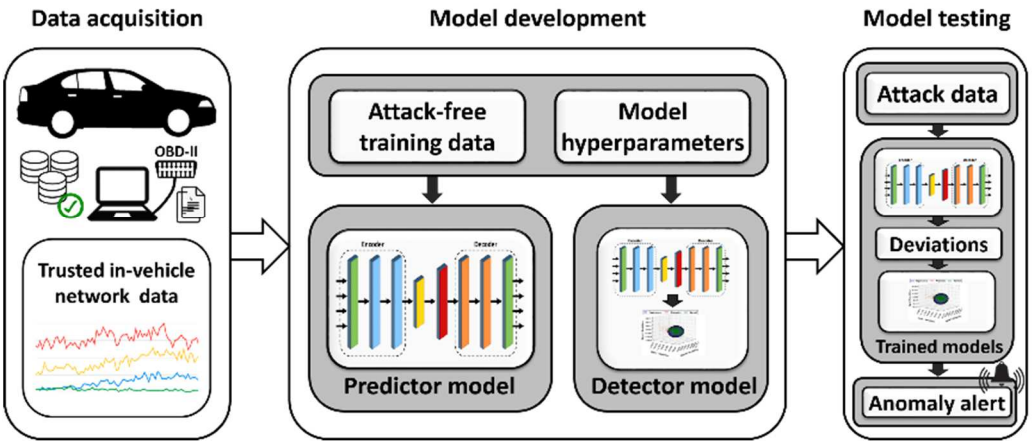


Figure 64 Overview of proposed *LATTE* framework.

6.4. PROPOSED FRAMEWORK

An overview of our proposed *LATTE* framework is shown in Figure 64. Our framework consists of a novel self-attention based LSTM deep learning model that is trained with data obtained from a data acquisition step. The data acquisition step collects trusted in-vehicle network data under a controlled environment. We then post-process and use this data to train the stacked LSTM self-attention predictor model in an unsupervised setting to learn the normal operating behavior of the system. We also developed a one class support vector machine (OCSVM) based detector model that utilizes the predictions from the LSTM predictor to detect cyber-attacks as anomalies at run-time. After training, the framework is tested by being subjected to various attacks. The details of this framework are presented in the subsequent subsections.

6.4.1. DATA ACQUISITION

This is the first step of the *LATTE* framework and involves collecting the in-vehicle network data from a trusted vehicle. It is important to ensure that the in-vehicle network and the ECUs in the vehicle are free from the attackers. This is because the presence of an attacker can result in

logging corrupt in-vehicle network data that falsely represents the normal operating conditions, leading to learning an inaccurate representation of the normal system behavior with our proposed models. Moreover, it is also crucial to cover a wide range of normal operating conditions and have the data collected over multiple intervals, to ensure high confidence in the collected data. The performance of the anomaly detection system is highly dependent on the quality of the collected data, and thus this is a crucial step. Additionally, the type of data collected depends on the functionalities or ECUs that are subjected to monitoring by the anomaly detection system. The most common access point to collect the in-vehicle network data is the OBD-II port, which gives access to the diagnostic and most commonly used messages. However, we recommend probing into the CAN network and logging the messages, as it gives unrestricted access to the in-vehicle network, unlike the OBD-II port.

After collecting the message data from the in-vehicle network, the data is prepared for pre-processing to make it easier for the training models to learn the temporal relationships between messages. The full dataset is split into groups based on the unique CAN message identifier and each group is processed independently. The data entries in the dataset are arranged as rows and columns with each row representing a single data sample corresponding to a particular timestamp and each column representing a unique feature of the message. The columns consist of the following features: *(i)* timestamp at which the message was logged, *(ii)* message identifier, *(iii)* number of signals in the message, *(iv)* individual signal values (one per column), and *(v)* a single bit representing the label of the message. The label column is 0 for non-anomalous samples and 1 for anomalous samples. The label column is set to 0 for all samples in the training and validation dataset as all the data samples are non-anomalous and collected in a trusted environment. The label column will have a value of 1 for the samples in the test dataset during the attack interval and 0

for the other cases. However, it is important to highlight that we do not use this label information while training our predictor and detector models. Moreover, for each signal type, the signal values are scaled between 0 to 1 as there can be a high variance in the signal magnitudes. Such high variance in the input data can result in very slow or unstable training. Additionally, in this work, we do not consider timestamp as a unique feature. We use the concept of time in a relative manner when training (to learn patterns in sequences) and during deployment. We are not dependent on absolute time during training and deployment. We use the dataset presented in [138] to train and evaluate our proposed *LATTE* framework. The dataset consists of both normal and attack CAN message data. Details related to the models and the training procedure are discussed in the next subsections, while the dataset is discussed in Section 6.5.1.

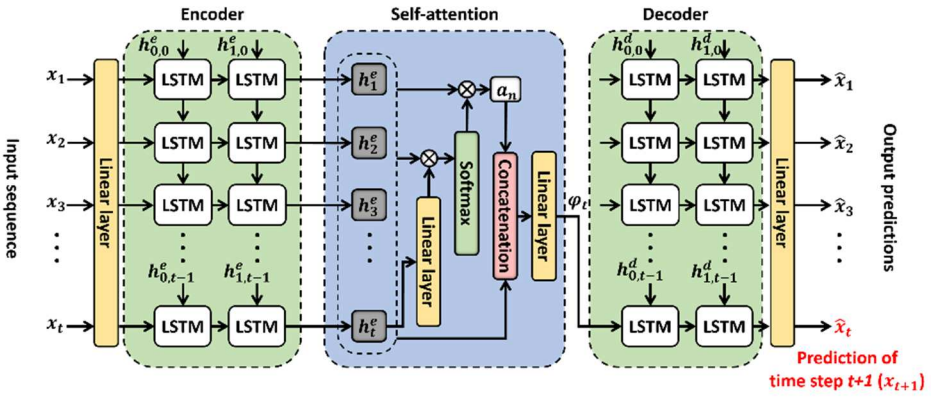


Figure 65 Our proposed predictor model for the *LATTE* anomaly detection framework showing the stacked LSTM encoder –decoder rolled out in time for *t* time steps along with the self-attention mechanism generating context vector for time step *t*. The output at time step *t* ($\hat{x}_t$) is the prediction of the input at time step *t+1* (x_{t+1}).

6.4.2. PREDICTOR MODEL

We designed predictor and detector models that work in tandem to detect cyber-attacks as anomalies in the in-vehicle network. The predictor model attempts to learn the normal system behavior via an unsupervised learning approach to predict the next message instance with high

accuracy at design time using the normal (non-anomalous) data. During this process, the predictor model learns the underlying distribution of the normal data and relates it to the normal system behavior. This knowledge of the learned distribution is used to make accurate predictions of the next message instances at runtime for normal messages. In the event of a cyber-attack, the message values no longer represent the learned distribution or maintain the same temporal relationships between messages, leading to large deviations between the predictions and the true (observed) messages. In this work, we employ a non-linear classifier based detector model to learn the deviation patterns that correspond to the normal messages, which is then used to detect anomalies (i.e., attacks that cause anomalous deviations) at runtime. The details related to the detector model are discussed in detail in Section 6.4.3.

Our proposed predictor model consists of a stacked LSTM based encoder-decoder architecture with the self-attention mechanism. This is illustrated in Figure 65. The first linear layer in the predictor model takes the time series CAN message data as the input and generates a 128 dimensional embedding for each input. Each input sample consists of k features where each feature represents a particular signal value within that message. The output embedding from the linear layer is passed to the stacked two-layer LSTM encoder to produce a 64-dimension encoder output ($h_1^e, h_2^e \ldots h_t^e$). The encoder output is the latent representation of the input time-series signal values that encompass the temporal relationships between messages. The self-attention block generates the context vector (φ_t) by applying the self-attention mechanism to the encoder outputs. The self-attention mechanism begins by applying a linear transformation on the encoder's current hidden state (h_t^e) and multiplies the result with the encoder output. The output from the multiplication is passed through a softmax activation to compute the attention weights. The attention weights represent the importance of each hidden state information from the earlier time

steps, at the current time step. The attention weights are scalars multiplied with the encoder outputs to compute the attention applied vector (a_n) which is then combined with the encoder output to compute the input to the decoder (context vector (φ_t)). The context vector along with the previous decoder's hidden state (h_{t-1}^{d}) is given as input to the stacked two-layer decoder, which produces a 64-dimension output that is passed to the last linear layer to obtain a k dimensional output. This k dimension output represents the signal values of the next message instance. Thus, given an input sequence $X = \{x_1, x_2, \ldots x_t\}$, our predictor model predicts the sequence $\hat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_t\}$, where the output at time step t ($\hat{x}_t$) is the prediction of the input at time step $t+1$ (x_{t+1}). The last prediction ($\hat{x}_t$) is generated by consuming the complete input sequence (X).

This predictor model is trained using non-anomalous (normal) data without any labels in an unsupervised manner. To train the model with sequences, we employ a rolling window approach. We consider a window of fixed size length (known as subsequence length) consisting of signal values over time. The window with signal values is called a subsequence and has subsequence length number of samples of signal values. Our predictor model learns the temporal dependencies that exist between the signal values within the subsequence and uses them to predict the signal values in the next subsequence (i.e., window shifted to the right by one-time step). The signal values corresponding to the last time step in the output subsequence represent the final prediction, as the model consumes the entire input subsequence to generate them. We compare this last time step in the output subsequence with the actual signal values and compute the prediction error using the mean square error (MSE) loss function. This process is repeated until the end of the training dataset. The subsequence length is a hyperparameter related to the LSTM network and is independent of the vehicle and the message data, and need to be selected before training the model. We conducted multiple experiments with different model parameters and selected the

hyperparameters that gave us the best performance results. The predictor model is trained by splitting the dataset into training (80%) and validation (20%) data without shuffling, as shuffling would destroy the existing temporal relationships between messages. During the training process, the model tries to minimize the prediction error in each iteration (a forward and backward pass) by adjusting the weights of the neurons in each layer using backpropagation through time. At the end of each training epoch, the model is validated (forward pass only) using the validation dataset to evaluate the model performance. We employ mini-batches to speed up the training process and use an early stopping mechanism to avoid overfitting. The details related to the non-anomalous dataset and the hyperparameters selected for the model are presented in Section 6.5.1.

6.4.3. DETECTOR MODEL

After training the predictor model, we train a separate classifier (detector model) that utilizes the information from the predictor to detects attacks. The anomaly detection problem can be treated as a binary classification problem as we are mainly interested in distinguishing between normal and anomalous messages. In general, as the in-vehicle network data recordings can grow in size very rapidly, labeling this data can get very expensive. Additionally, due to the nature of the frequency of attack scenarios, the number of attack samples would be quite small compared to normal samples even when the dataset is labeled. This results in having a highly imbalanced dataset that would result in poor performance when trained with a traditional binary classifier in a supervised learning setting. However, a popular non-linear classifier known as a support vector machine (SVM) can be altered to make it work with unbalanced datasets where there is only one class. Hence, in this work, we use a one class support vector machine (OCSVM) to classify the messages as anomalous or normal. The OCSVM learns the distribution of the training dataset by constructing the smallest hypersphere that contains the training data at design time and identifies

any sample outside the hypersphere as an anomaly at runtime. We train an OCSVM by using the output from the previously trained predictor model. We begin by giving the previously used normal training dataset as the input to the predictor to generate the predictions. We then compute the deviations (prediction errors) for all the training data and pass it as input to the OCSVM. The OCSVM tries to generate the smallest hypersphere that can fit most of the deviation points and uses it at runtime to detect anomalies. Figure 66 shows an example of a hypersphere generated by training an OCSVM for a message with three signals. Each axis in the figure represents the relevant signal deviation and the dark blue sphere represents the decision boundary. It can be observed that almost the entirety of training data (shown via green dots) is confined to within the blue sphere.

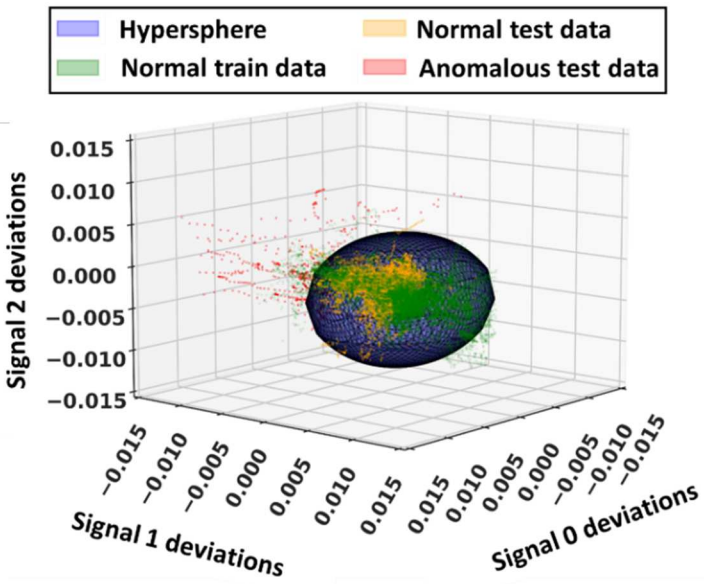


Figure 66 OCSVM decision boundary shown in the blue sphere with the green dots showing the normal samples from training data, and yellow and red dots showing the normal and anomalous samples respectively from test data.

In our work, the deviation of a message is represented as a vector where each element of the vector corresponds to the difference between the true and predicted signal value. Therefore, for a message m with k_m number of signals, the deviation vector ($\Delta_{m,t}$) computed at time step t is given

by equation (28).

$$\Delta_{m,t} = \left(\hat{S}_{i,t} - S_{i,t+1}\right) \in \mathbb{R}^2 , \forall\, i \in [1, k_m] \tag{28}$$

where $\hat{S}_{i,t}$ represent the prediction of the next true i^{th} signal value ($S_{i,t+1}$) made at time step t. We also experimented with other deviation measures that are given by equations (29), (30) and (31).

$$\Delta_{m,t}^{sum} = \sum_{i=1}^{k_m} \left|\Delta_{m,t}\right| , \forall\, i \in [1, k_m] \tag{29}$$

$$\Delta_{m,t}^{avg} = \frac{1}{k_m} \sum_{i=1}^{k_m} \left|\Delta_{m,t}\right| , \forall\, i \in [1, k_m] \tag{30}$$

$$\Delta_{m,t}^{max} = \max\left(\left|\Delta_{m,t}\right|\right) , \forall\, i \in [1, k_m] \tag{31}$$

Moreover, there can be situations where some of the signal deviations in a message can be positive while others are negative. This could potentially result in making the sum or mean of signal deviations zero or near zero, falsely representing no deviation or very small deviation. To avoid these situations, we use the absolute signal deviations to compute the deviations for the variants. *Note*: Unlike equation (28) that uses a vector of k dimensions to represent the message deviation, equations (29), (30), and (31) reduce the vector to a single value using different reduction operations. We explored these reduced deviation scores (shown in equations (29), (30), and (31)) that utilize absolute deviation values to determine the best one, as discussed in Section 6.5.2.

In summary, our predictor model predicts the normal samples with very small deviations and anomalous samples with high deviations. The OCSVM takes this predictor property into account when constructing the hypersphere. In Figure 66, the yellow dots and red dots represent the normal and anomalous samples respectively in the test dataset. It can be observed that when

the test data with anomalies is given as input to the OCSVM, it generally correctly classifies the yellow samples within the hypersphere and red samples outside the hypersphere. Thus, both predictor and detector models work collectively to detect attacks as anomalies. The details related to the testing process are described in the next subsection.

6.4.4. MODEL TESTING

In the deployment/testing step, we present a test dataset consisting of anomalous samples representing multiple attacks (outlined in Section 6.3.3) along with the normal samples to the *LATTE* framework. The normal messages have a label value of 0 and the attack messages have a label value of 1. During this step, each sample (signal values in a message) is first sent to the predictor model to predict the signal values of the next message instance, and the deviation is computed based on the true message data. This deviation vector is passed to the OCSVM detector model, to compute the position of the deviation vector in the k-dimensional space, where k represents the number of signals in the message. The message is marked as non-anomalous when the point corresponding to the deviation vector falls completely inside the learned hypersphere. Otherwise, the message is marked as anomalous and an anomaly alert is raised. This can be used to invoke an appropriate remedial action to suppress further actions from the attacker. However, the design of remedial actions and response mechanisms falls outside the scope of this chapter. The performance evaluation of our proposed *LATTE* framework under various attack scenarios is presented in detail in sections 6.5.2 and 6.5.3.

6.4.5. ANOMALY DETECTION SYSTEM DEPLOYMENT

Our proposed anomaly detection system can be deployed in a real-world vehicle in two different approaches. The first is a global monitoring or centralized approach, where a powerful

centralized ECU monitors the messages on the CAN bus and detects anomalies. The second approach involves distributing the anomaly detection task to across ECUs and only monitoring the messages that are relevant to that particular ECU (distributed monitoring). Both choices have pros and cons, but we believe that the distributed monitoring has multiple advantages over the centralized approach because of the following reasons:

- A centralized approach is prone to single-point failures, which can completely open up the system to the attacker;
- In extreme scenarios such as during a DDoS attack (explained in Section 6.3.3), the in-vehicle network can get highly congested, and the centralized system might not be able to communicate with the victim ECUs;
- If an attacker succeeds in fooling the centralized ECU, attacks can go undetected by the other ECUs, resulting in compromising the entire system; whereas with a distributed detection scheme, fooling multiple ECUs is required which is much harder, and even if an ECU is compromised, this can still be detected by the decentralized intelligence in a distributed detection;
- In a distributed detection, ECUs can stop accepting messages as soon as an anomaly is detected without waiting for a centralized system to notify them, leading to faster response;
- The computation load of detection is split among the ECUs with a distributed approach, and the monitoring can be limited to only the required messages. Thus, multiple ECUs can monitor a subset of messages independently, with lower overhead;

Many prior works, e.g., in [121] and [106], consider a distributed local detection approach for these reasons. Moreover, with automotive ECUs becoming increasingly powerful, the collocation of detection tasks with real-time automotive applications in a distributed manner

should not be a problem, provided the overhead from the detection is minimal. The light weight nature and anomaly detection performance of our proposed *LATTE* framework are discussed further in Section 6.5. Moreover, as the detector looks at the payload segment individually, it needs to keeps a track of the previous messages to detect anomalous patterns. We can cache the previous normal samples and predictions (in the case of anomalies) and use them to preserve the dependencies within the data, which can be later used in determining whether the next sample is normal or anomalous. To minimize the storage overhead, we can employ a circular buffer of size equal to the subsequence length (configured at design time). Using this approach, we can still look into the message dependencies in the past.

6.5. EXPERIMENTS

6.5.1. EXPERIMENTAL SETUP

To evaluate the effectiveness of our proposed *LATTE* framework, we first explored five variants of the same framework with different deviation criteria: *LATTE*-ST, *LATTE*-Diff, *LATTE*-Sum, *LATTE*-Avg, and *LATTE*-Max. *LATTE*-ST uses our proposed predictor model with a static threshold (ST) value to determine whether a given message is anomalous or normal based on the deviation. The other four variants use the same predictor model but different detection criteria for computing the deviations for OCSVM. *LATTE*-Diff uses the difference in signal values (equation (28)); *LATTE*-Sum and *LATTE*-Avg use a sum and mean of absolute signal deviations respectively (equations (29), and (30)); and *LATTE*-Max uses the maximum absolute signal deviation (equation (31)), as the input to the detector model.

Subsequently, we compare the best variant of our framework with four prior works: Bitwise Message Predictor (BWMP [140]), Hierarchical Attention-based Anomaly Detection (HAbAD

[159]), a variant of [159] called Stacked HAbAD (S-HAbAD [159]), and RepNet [141]. BWMP [140] trains an LSTM based neural network that aims to predict the next 64 bits of a CAN message by minimizing the bitwise prediction error using a binary cross-entropy loss function. At runtime, BWMP uses the prediction loss as a measure to detect anomalies. HAbAD [159] uses an LSTM based autoencoder model with hierarchical attention. The HAbAD model attempts to recreate the input message sequences at the output and aims to minimize reconstruction loss. Additionally, HAbAD uses supervised learning in the second step to model a detector using the combination of a non-parametric kernel density estimator (KDE) and k-nearest neighbors (KNN) algorithm to detect cyber-attacks at runtime. Lastly, S-HAbAD is a variant of HAbAD that uses stacked LSTMs as autoencoders and uses the same detection logic used by the HAbAD. The S-HAbAD variant is compared against to show the effectiveness of using stacked LSTM layers. Lastly, RepNet [141] uses simple RNNs to increase the dimensionality of input signal values and attempts to reconstruct the signal values at the output by minimizing the reconstruction error using mean squared error. At runtime, RepNet monitors for large reconstruction errors to detect anomalies. The results of all experiments are discussed in detail in subsections 6.5.2-6.5.4.

We conducted all experiments using an open-source CAN message dataset developed by ETAS and Robert Bosch GmbH [138]. The dataset consists of CAN message data for different message IDs consisting of various fields such as timestamps, message ID, and individual signal values. Additionally, the dataset consists of a training dataset with only normal data and a labeled test dataset with multiple attacks (as discussed in Section 6.3.3). The attack data in the dataset is modeled from the real world attacks that are commonly seen in automotive systems. It is important to note that we do not use any labeled data during the training or validation of our models and learn the normal system behavior in an unsupervised manner. The labeled data is given to the

models only during the testing phase and used to compute performance metrics. Moreover, the dataset consists of multiple message frequencies {15, 30, 45} ms. Since the high frequency messages pose a significant challenge to the anomaly detection system, and could result in high overhead, in this work we consider the message frequency of 15 ms for all of our experiments.

We used PyTorch 1.5 to implement all of the machine learning models including *LATTE* and its variants, and the models from the comparison works. Our proposed predictor model is trained with 80% of the available normal data and the remaining 20% is used for validation. We conducted multiple experiments with different model parameters, and selected the hyperparameters that gave us the best performance results. The training phase is repeated for 500 epochs with an early stopping mechanism that monitors the validation loss after the end of each epoch and stops if there is no improvement after 10 (patience) epochs. We used the ADAM optimizer with mean squared error (MSE) as the loss function. Additionally, we employed a rolling window approach (discussed in Section 6.4.2) with a subsequence length of 32 time steps, a batch size of 256, and a starting learning rate of 0.0001. We used the scikit-learn package to implement the OCSVM in the detector model (Section 6.4.3). We used a radial basis function (RBF) kernel with a kernel coefficient (*gamma*) equal to the reciprocal of the number of features (i.e., number of signals in the message). Moreover, to speed up OCSVM training, we set the kernel cache size to 400 MB and enabled the shrinking technique to avoid solving redundant optimizations. All the simulations are run on an AMD Ryzen 9 3900X server with an Nvidia GeForce RTX 2080Ti GPU.

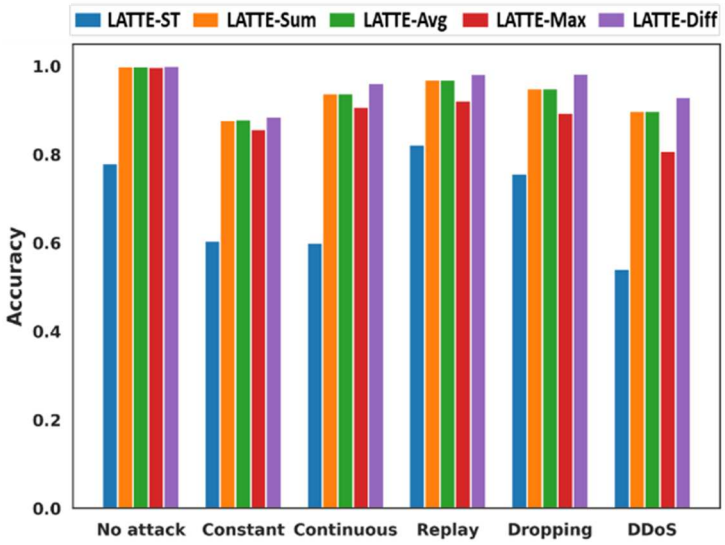
Before looking at the experimental results for various performance metrics, it is important to understand some key definitions in the context of anomaly detection. We define a true positive as the scenario when an actual attack is detected as an anomaly by the anomaly detection system and a true negative as the situation where an actual normal message is detected as normal.

Additionally, a false positive would be a false alarm where a normal message is incorrectly classified as an anomaly and a false negative would occur when an anomalous message is incorrectly classified as normal. Using the above definitions, we evaluate our proposed framework using four different metrics: *(i) Detection accuracy*: a measure of the anomaly detection system's ability to detect anomalies correctly, *(ii) False positive rate*: i.e., false alarm rate, *(iii) F1 score*: a harmonic mean of precision and recall; we use the F1-score instead of individual precision and recall values as it captures the combined effect of both precision and recall metrics, and *(iv) receiver operating characteristic (ROC) curve with area under the curve (AUC)*: a popular measure of classifier performance. A highly efficient anomaly detection system has high detection accuracy, F1 score, and ROC-AUC while having a very low false-positive rate.

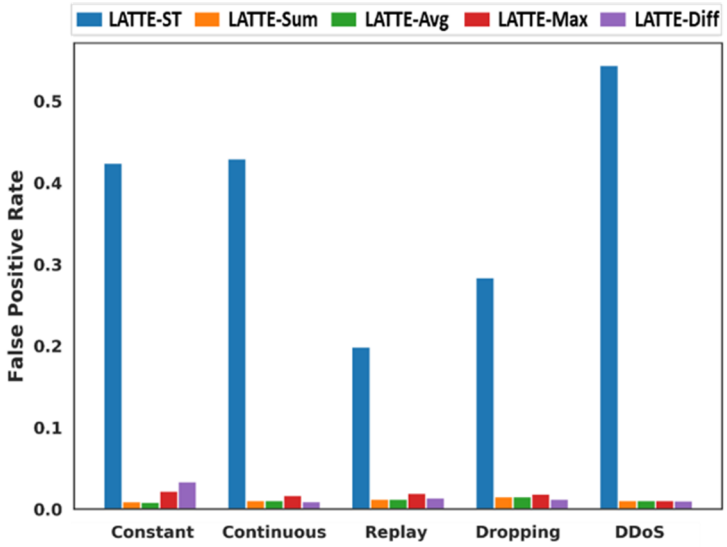
6.5.2. COMPARISON OF *LATTE* VARIANTS

In this subsection, we present the comparison results of the five variants *LATTE*-ST, *LATTE*-Sum, *LATTE*-Avg, *LATTE*-Max and *LATTE*-Diff. All the variants of *LATTE* use the trained predictor model (discussed in Section 6.4.2) to make the predictions and use OCSVM as a detector except in the case of *LATTE*-ST, which uses a fixed threshold scheme introduced in [47] to predict the given message as normal or anomalous. The main purpose of this experiment is to analyze the impact of using a non-linear classifier such as OCSVM on the model performance instead of a simple static threshold scheme (*LATTE*-ST). Additionally, with the last four variants, we aim to study the effect of different deviation criteria on the OCSVM detection performance. The deviations for any given message in *LATTE*-Diff ($\Delta_{m,t}$), *LATTE*-Sum $\left(\Delta_{m,t}^{sum}\right)$, *LATTE*-Avg $\left(\Delta_{m,t}^{avg}\right)$ and *LATTE*-Max $\left(\Delta_{m,t}^{max}\right)$ are computed using the equations (28), (29), (30) and (31) respectively.

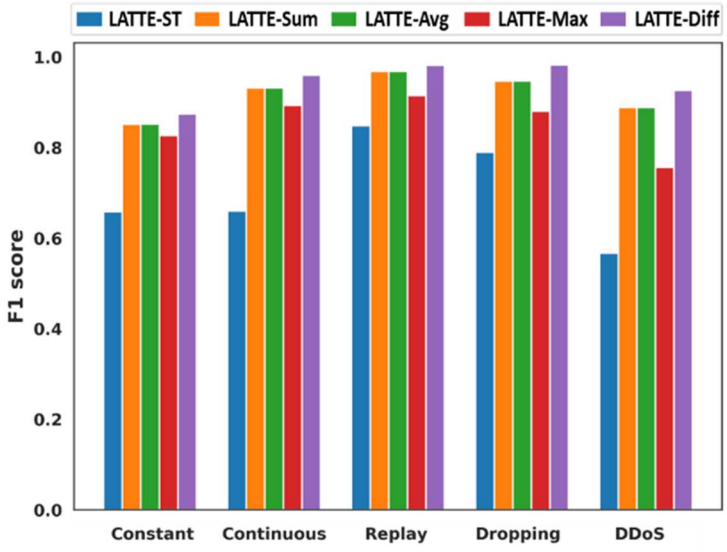
Figure 67(a)-(c) shows the detection accuracy, false-positive rate, and F1 score respectively for the five different variants of *LATTE* under five different attack scenarios discussed in Section 6.3.3. The 'No attack' case involves testing the model with new non-anomalous data that the model has not seen before. Firstly, from Figure 67(a)-(c) it is clear that the OCSVM based detection models clearly outperform the static threshold models (*LATTE*-ST). This is mainly because of their ability to process complex attack patterns and generate non-linear decision boundaries that can distinguish better between normal and anomalous data. Moreover, it can be seen that *LATTE*-Diff outperforms all the OCSVM based models in detection accuracy, false-positive rate, and F1 score. Lastly, in Figure 67(d), we present the ROC curves and the corresponding AUC values in the brackets next to each legend. Out of the various attacks, we show results for continuous attacks, as it is the most challenging attack to detect. This is because during this attack, the attacker constantly tries to fool the anomaly detection system into thinking that the signal values in the messages are legitimate. This requires careful monitoring and the ability to learn complex patterns to differentiate between normal and anomalous samples.



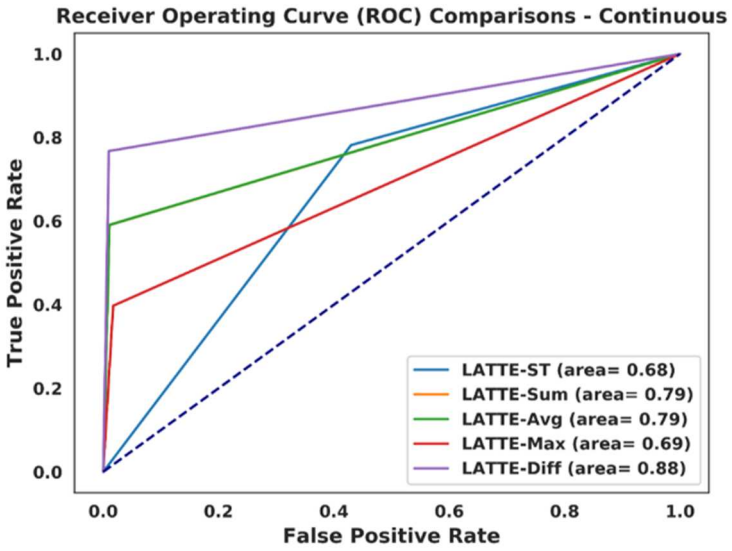
(a)


LATTE-ST
LATTE-Sum
LATTE-Avg
LATTE-Max
LATTE-Diff
False Positive Rate
0.0
0.1
0.2
0.3
0.4
0.5
Constant
Continuous
Replay
Dropping
DDoS

(b)


LATTE-ST
LATTE-Sum
LATTE-Avg
LATTE-Max
LATTE-Diff
F1 score
0.0
0.2
0.4
0.6
0.8
1.0
Constant
Continuous
Replay
Dropping
DDoS

(c)



(d)

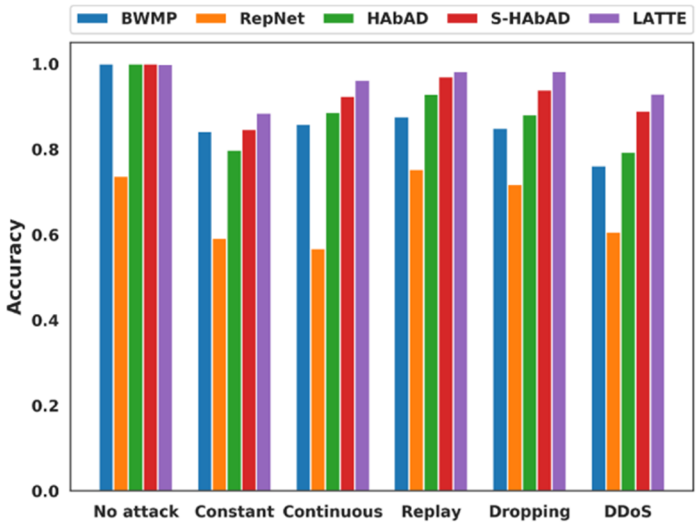
Figure 67 Comparison of (a) detection accuracy, (b) false-positive rates, (c) F1 score of *LATTE* variants under different attack scenarios, and (d) ROC curve with AUC for continuous attack.

On average, across all attacks, LATTE-Diff was able to achieve an average of 13.36% improvement in accuracy, 11.34% improvement in F1 score, 17.86 % improvement in AUC and 47.9% reduction in false positive rate, and up to 42% improvement in accuracy, 32.6% improvement in F1 score, 29.4% improvement in AUC and 95% decrement in false positive rate, compared to the other variants. Therefore, we selected *LATTE*-Diff as our candidate model for subsequent experiments where we present comparisons with the state-of-the-art anomaly detectors. Henceforth, we refer to *LATTE*-Diff as *LATTE.*

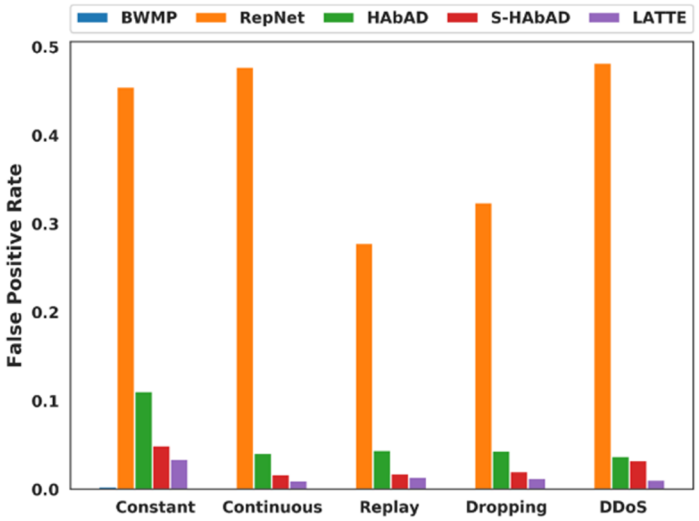
6.5.3. COMPARISON WITH PRIOR WORKS

We compared our *LATTE* framework with BWMP [140], HAbAD [159], a variant of HAbAD called S-HAbAD [159], and RepNet [141]. Figure 68(a)-(c) show the detection accuracy, false-positive rate, and F1 score respectively for these frameworks under different attack scenarios.

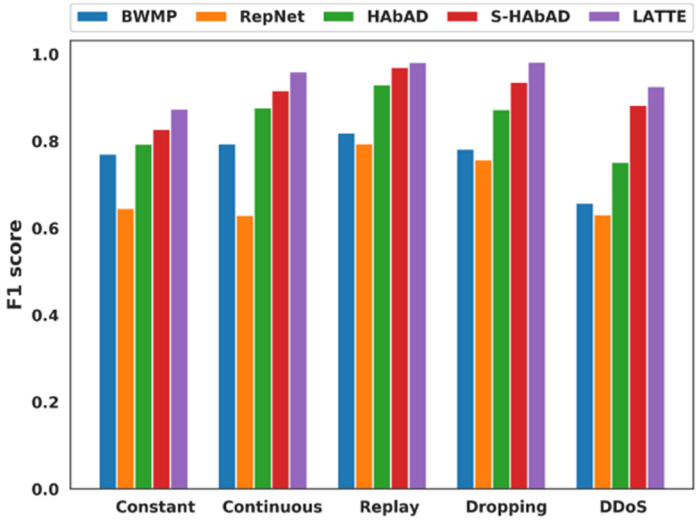
It can be observed that *LATTE* outperforms all the prior works in terms of detection accuracy, false-positive rate, and F1 score. Moreover, from Figure 68(d), we can see that *LATTE* outperforms all comparison works in terms of ROC-AUC under challenging continuous attack. This is mainly due to three factors. Firstly, the stacked LSTM encoder-decoder structure provides adequate depth to the model to learn complex time-series patterns. This can be seen when comparing HAbAD with S-HAbAD, as the latter differs only in terms of stacked LSTM layers in comparison to the former. Second, the self-attention mechanism helps *LATTE* in learning message sequences that have very long-term dependencies. Lastly, the use of powerful OCSVMs as non-linear classifiers helps in constructing a highly efficient classifier. These factors together resulted in the superior performance of *LATTE* compared to all the comparison works. On average, across all attacks, *LATTE* was able to achieve an average of 18.94% improvement in accuracy, 19.5% improvement in F1 score, 37% improvement in AUC and 79% reduction in false positive rate, and up to 47.8% improvement in accuracy, 37.5% improvement in F1 score, 76% improvement in AUC and 95% reduction in false positive rate.



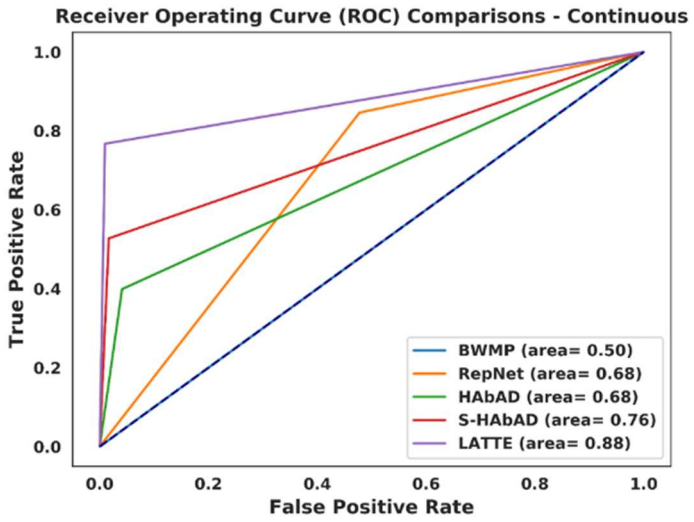
(a)


BWMP
RepNet
HAbAD
S-HAbAD
LATTE
False Positive Rate
0.5
0.4
0.3
0.2
0.1
0.0
Constant
Continuous
Replay
Dropping
DDoS

(b)


BWMP
RepNet
HAbAD
S-HAbAD
LATTE
F1 score
1.0
0.8
0.6
0.4
0.2
0.0
Constant
Continuous
Replay
Dropping
DDoS

(c)



(d)

Figure 68 Comparison of (a) accuracy, (b) false-positive rates, (c) F1 score of *LATTE* and the comparison works under different attack scenarios, and (d) ROC curve with AUC for continuous attack.

To highlight the effectiveness of our proposed *LATTE* framework, we further compared *LATTE* with statistical and proximity based techniques. We selected Bollinger bands (a popular statistical technique used in the finance domain) as the candidate for a statistical technique to detect anomalies in time series data. Bollinger bands generate envelopes that are two standard deviation levels above and below the moving average. In this work, we considered two different moving average based variants of the approach: *(i)* simple moving average (SMA), and *(ii)* exponential weighted moving average (EWMA) similar to [167]. We also compared *LATTE* against a local outlier factor (LOF) [168] based anomaly detection technique, which is a popular proximity-based anomaly detection technique. The LOF algorithm measures the local deviation of each point in the dataset with respect to the neighbors (given by KNN) to detect anomalies. The F1 score results for SMA based Bollinger bands (SMA-BB), EWMA based Bollinger bands (EWMA-BB), LOF, and

LATTE under different attack scenarios are shown in Figure 69. It can be seen that *LATTE* outperforms both statistical and proximity-based anomaly detection techniques under different attack scenarios. This is mainly because the complex patterns in CAN message data are hard to capture using statistical and proximity-based techniques. On the other hand, the LSTM based predictor model in our proposed *LATTE* framework learns these complex patterns and is thus able to more efficiently detect complex attacks.

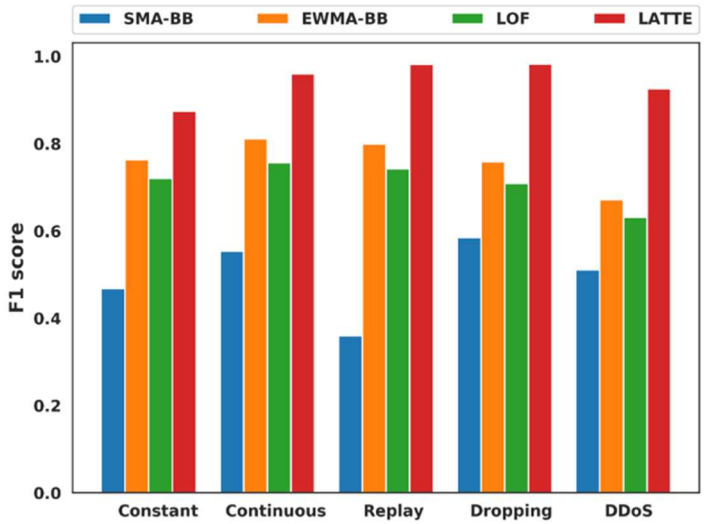


Figure 69 Comparison of F1 score for SMA-BB [167], EWMA-BB [167], LOF [168], and *LATTE* under different attack scenarios.

6.5.4. OVERHEAD ANALYSIS

In this subsection, we present an overhead analysis of our *LATTE* framework. We quantify the overhead of our *LATTE* framework and the comparison works using memory footprint, the number of model parameters, and the inference time metrics. We profiled each framework on a dual core ARM Cortex- A57 CPU on an NVidia Jetson TX2 board (shown in Figure 70), which has similar specifications to that of a real-world ECU. We repeated the inference time experiment 10 times and computed the average inference time. Moreover, in this work, we consider a total

buffer size of 2.25 KB. This accounts for the storage of 32 CAN message payloads (0.25 KB assuming a worst case max payload of 8 Bytes) that represent the subsequence length number of past messages, and storage of 16 CAN message frames (2 KB assuming the CAN extended protocol and a worst case max payload of 8 Bytes) that is used by the transceiver. In this work, we only introduce the additional 0.25 KB storage as the 2 KB transceiver buffer space is already available in the traditional CAN communication controller interfaces. We consider a 2 KB transceiver buffer, as it is the most commonly used size in many real-world automotive ECUs such as Woodward SECM 112, and dSpace MicroAutoBox. Additionally, we computed the area overhead of the 0.25 KB buffer using CACTI tool [169] by modeling the buffer as a scratchpad cache using 32 nm technology node. Our additional 0.25 KB buffer resulted in a minimal area overhead of around 581.25 μm^2. From Table 13, we can observe that our *LATTE* framework has minimal overhead compared to both attention-based prior works (HAbAD and S-HAbAD) and the non-attention based work (BWMP except RepNet). The high runtime and memory overhead in HAbAD and S-HAbAD is associated with the use of KNNs. KNN does not generalize the data in advance, but rather scans through each training data sample to make a prediction. This makes it very slow and consume high memory overhead (due to the requirement of having training data available at runtime). It needs to be noted that, even though RepNet has the lowest memory and runtime overhead, it fails to capture the complex attack patterns due to the smaller model size and the lack of ability of simple RNNs to learn long-term dependencies, leading to poor performance (as shown in Figure 68).

Table 13 Overhead of *LATTE*, BWMP [140], HAbAD [159], S-HAbAD [159], RepNet [141]

Framework	Memory footprint (KB)	#Model parameters (x10^3)	Average inference time (µs)
BWMP [140]	13,147	3435	644.76
HAbAD [159]	4558	64	685.05
S-HAbAD [159]	5600	325	976.65
LATTE	1439	331	193.90
RepNet [141]	5	0.8	68.75

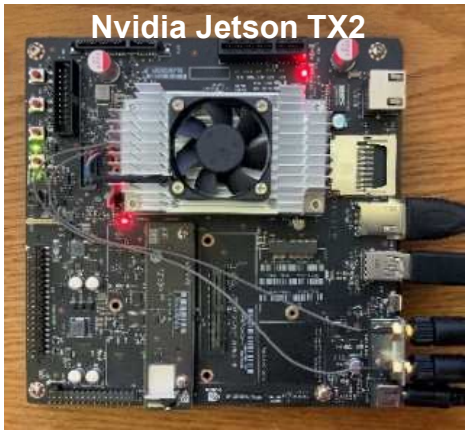


Figure 70 Nvidia Jetson TX2 development board

Assuming a distributed anomaly detection implementation, we factor in this additional latency into our real-time constraints for message transmission (i.e., a constant time overhead). But since the latency overhead (shown in Table 13) is very minimal, we envision that our proposed *LATTE* framework will have a minimal change in the timing constraints when compared to the prior works. Moreover, the deadline constraints for some of the fastest (i.e., most stringent) safety-critical applications are around 10 ms, which is much higher than our overhead that is around 193 µs. Hence, the additional latency due to our anomaly detection should not violate any safety-critical deadlines. In summary, from Figure 68 and the results in Table 13, we can clearly observe that *LATTE* achieves superior performance compared to all of the comparison works across diverse attack scenarios, while maintaining relatively low memory and runtime overhead.

6.6. CONCLUSION

In this chapter, we proposed a novel stacked LSTM with self-attention framework called *LATTE* that learns the normal system behavior by learning to predict the next message instance under normal operating conditions. We presented a one class support vector (OCSVM) based detector model to detect cyber-attacks by monitoring the message deviations from the normal behavior. We presented a detailed analysis by comparing our proposed model with multiple variants of our model and the best-known prior works in this area. Our *LATTE* framework surpasses all the variants and the best-known prior works under different attack scenarios while having a relatively low memory and runtime overhead. As a part of future work, we will explore extending our framework to detect malfunctions such as blockages, deadlocks, and faults in addition to detecting malicious behavior on the in-vehicle network.

7. TENET: TEMPORAL CNN WITH ATTENTION FOR ANOMALY DETECTION IN AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

Today's vehicles are becoming increasingly autonomous and connected, to achieve improved safety and fuel efficiency goals. To support this evolution, new technologies such as advanced driver assistance systems (*ADAS* [170]), vehicle-to-vehicle (*V2V*), 5G vehicle-to-infrastructure (*5G V2I*), etc. have emerged [2]. These advances have led to an increase in the complexity of electronic control units (*ECUs*) and the in-vehicle network that connects them. As a result, vehicles today represent distributed cyber-physical systems (CPS) of immense complexity. The ever-growing connectivity to external systems in such vehicles is introducing new challenges, related to the increasing vulnerability of these vehicles to a variety of cyber-attacks [46].

Attackers can use various access points (known as an attack surface) in a vehicle, e.g., Bluetooth, telematic systems, and OBD-II ports, to gain unauthorized access to the in-vehicle network. After gaining access to the network, an attacker can inject malicious messages to try and take control of the vehicle. Recent automotive attacks on the in-vehicle network include manipulating speedometer and indicator signals [26], unlocking doors [171], manipulating the fuel level indicator [171], etc. These types of attacks confuse the driver but are not fatal. More complex machine learning-based attacks can cause incorrect traffic sign recognition in a vehicle's camera-connected ECU [172]. In [153], researchers analyzed vulnerabilities in airbag systems and remotely deployed the airbags in a vehicle. These types of attacks can be catastrophic and potentially fatal.

With the increasing complexity of vehicular CPS, the attack surface is only going to increase, paving the way for more complex and novel attacks in the future. Thus, there is an urgent need for an advanced attack detection solution that can actively monitor the in-vehicle network and detect complex cyber-attacks. One of the many approaches to achieve this goal is by using an anomaly detection solution (ADS). An ADS can be a hardware or software-based system that continuously monitors the in-vehicle network to detect attacks without any human supervision. Many state-of-the-art ADS use machine learning algorithms to detect cyber-attacks due to large availability of vehicle network data and more computationally capable ECUs today. At a very high level, the machine learning model in an ADS tries to learn the normal operating behavior of the vehicle system during design and test time. This learned knowledge of the normal system behavior is then used at runtime to continuously monitor for any anomalous behavior, to detect attacks. The major advantage of this approach is that it can detect both known and unknown attacks. Due to its high attack coverage and ability to detect complex attack patterns, we focus on (and make new contributions to) machine learning based ADS for cyber-attack detection in vehicles.

In this chapter, we propose a novel ADS framework called *TENET* to actively monitor the in-vehicle network and observe for any deviation from the normal behavior to detect cyber-attacks. *TENET* attempts to increase the detection accuracy, receiver operating characteristic (ROC) curve with area under the curve (AUC), Mathews correlation coefficient (MCC) metrics, and minimize false negative rate (FNR) with minimal overhead. Our novel contributions can be summarized as follows:

- We present a temporal convolutional neural attention (TCNA) architecture to learn very-long term temporal dependencies between messages in the in-vehicle network;

- We introduce a metric called divergence score to quantify the deviation from expected behavior;
- We adapt a decision tree-based classifier to detect a variety of attacks at runtime using the proposed metric;
- We compare our *TENET* framework with multiple state-of-the-art ADS frameworks to demonstrate its effectiveness.

7.1. RELATED WORK

Several researchers have proposed solutions to detect in-vehicle network attacks. These solutions can be classified as either signature-based or anomaly-based. In this section, we discuss these solutions in detail and present a distinction between the existing works and our proposed *TENET* framework.

The authors in [173] proposed a language-theory based model to derive attack signatures. However, their technique fails to detect attack packets at the initial stages of the attack. In [174], [175] message frequency-based techniques were proposed to detect attacks. A transition matrix-based ADS was proposed in [122] to detect attacks on the controller area network (CAN). However, the approach could not detect complex attacks, such as replay attacks. An entropy-based ADS was presented in [131], [176] to detect in-vehicle network attacks. However, these techniques fail to detect small variations in the entropy and modifications in CAN message data. In [157], the Hamming distance between messages was used to detect attacks. However, this approach incurs a high computational overhead. In [177], ECUs were fingerprinted using their voltage measurements during message transmission and reception. However, this method cannot detect attacks at the application layer. *In general, signature-based ADS approaches can detect attacks in the network*

with high accuracy and low false-positive rate. However, obtaining all possible attack signatures and frequently updating them is impractical. Moreover, none of these works provide a holistic solution to detect unknown and complex attack patterns.

In contrast, anomaly-based solutions attempt to learn the normal system behavior and observe for any abnormal behavior in the network to detect both known and unknown attacks. In [137], the authors used deep neural networks (DNNs) to extract the low dimensional features of transmitted packets and differentiate between normal and attack-injected packets. The authors in [141] used a recurrent neural network (RNN) to learn the normal behavior of the network and used that information to detect attacks at runtime. Several other works, such as [138], [139], [178], and [140], have proposed long short-term memory (LSTM) based ADS to learn the relationship between messages traversing the in-vehicle networks. However, these models were not tested on complex attack patterns and impose high overheads on the ECU. The authors in [48] proposed an LSTM based encoder-decoder architecture with an attention mechanism as an ADS for in-vehicle network. The attention was used to enhance the encoder's context vector to provide the decoder with quality inputs. A gated recurrent unit (GRU) based autoencoder ADS was proposed to learn the normal system behavior in [47]. However, a static threshold approach was used to classify messages, which is unable to capture non-linear behaviors. In [159], an LSTM based encoder-decoder ADS was proposed with attention to reconstruct input messages. A kernel density estimator (KDE) and k-nearest neighbors (KNN) were further used to detect anomalies. But the approach incurs a high overhead on the ECU. An approach that combined an LSTM with a convolutional neural network (CNN) was proposed in [179] to learn the dependencies between messages in a CAN network. However, the model was trained on a labeled dataset in a supervised manner; due to the large volume of in-vehicle CAN message data, labeling the data is impractical.

All these works suggest that sequence models with LSTMs and GRUs are popular for detecting attacks on vehicles. *However, the increased vehicular CPS complexity today has resulted in very long-term dependencies between messages exchanged between ECUs that cannot be effectively captured using LSTMs and GRUs.* This is because the current time step output of LSTMs and GRUs is heavily influenced by recent time steps compared to time steps in the distant past, which makes it hard to capture very long-term dependencies. Processing very long sequences also exacerbates the computational and memory overhead of LSTMs and GRUs.

In summary, none of the existing ADS provides a holistic approach that can efficiently learn very long-term dependencies between in-vehicle network messages with a low memory and computational overhead, and also accurately detect a multitude of simple and complex attacks on the vehicle. Our proposed *TENET* ADS uses a novel TCNA (temporal CNN with attention) model to overcome these shortcomings of state-of-the-art ADS. The next section describes *TENET* in detail. The comparative performance analysis results are presented in Section 7.3.

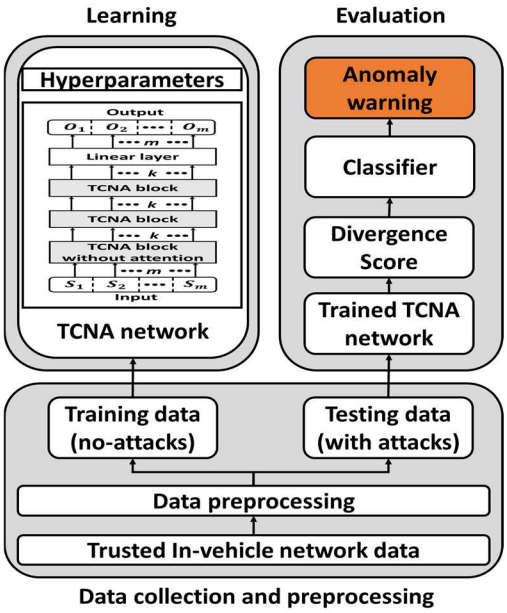


Figure 71 Overview of the different phases of the TENET framework.

7.2. TENET FRAMEWORK: OVERVIEW

The *TENET* framework consists of three phases: *(i)* data collection and preprocessing, *(ii)* learning, and *(iii)* evaluation. The first phase involves collecting in-vehicle network data from a trusted vehicle and preprocessing the collected data. In the learning phase (*offline*), the preprocessed data is used to train a Temporal Convolutional Neural attention (TCNA) network in an unsupervised manner to learn the normal behavior of the system. In the evaluation phase (*online*), the trained TCNA network is deployed and used to calculate a divergence score (DS), which is then used by a decision tree-based classifier to detect attacks. The overview of our proposed *TENET* framework is shown in Figure 71.

7.2.1. DATA COLLECTION AND PREPROCESSING

This first phase of the *TENET* framework involves collecting in-vehicle network data from a trusted vehicle and over a variety of normal operating conditions. Otherwise, the model may learn an incorrect representation of the normal operation of the vehicle. In this work, we recommended splicing into the vehicle network and directly logging the messages using a standard logger such as Vector GL 1000 [180], as this allows one to record any message traversing the network.

After data collection, the data is prepared for pre-processing to facilitate easy and efficient training of the machine learning models. Every vehicle message in typical vehicle network protocols (CAN, FlexRay [45], etc) has a unique identifier (ID) and each message in the dataset is grouped by this unique ID and processed independently. Each message has the following attributes (columns): *(i)* unique timestamp corresponding to the log time of the message (used for relative

ordering of messages), *(ii)* message ID, *(iii)* number of signals in the message, *(iv)* individual signal values in the message (which together constitute the message payload), and *(v)* label of the message ('0' for no-attack and '1' for attack). Due to the possibility of high variance in message signal values, all signal values of each signal type are scaled between 0 and 1. The learning phase and evaluation phases in *TENET* use training and testing data, respectively. The label values of all samples in the training dataset are set to 0 to represent no-attack data. The test data has a label value of 1 for attack samples and a label value of 0 for no-attack samples. Furthermore, the original training data is split into training (85%) and validation (15%) sets. Details of the training procedure and the model architecture are discussed in the next subsections.

7.2.2. MODEL LEARNING

In this subsection, we describe our proposed TCNA network architecture and the training procedure we employed for it. *TENET* uses this TCNA network to learn the normal system behavior of the in-vehicle network in an *unsupervised* manner. The proposed TCNA model takes the sequence of signal values in a message as the input and uses CNNs to predict the signal values of the next message instance, by trying to learn the underlying probability distribution of the normal data.

An early adaptation of CNNs for sequence modeling tasks was presented in [181], where a convolution-based time-delay neural network (TDNN) was proposed for phoneme recognition. To capture very-long term dependencies, traditional CNNs need to employ a very deep network of CNN layers with large filters. Consequently, this increases the number of convolutional operations incurring a high computational overhead. Thus, adapting CNNs directly to sequence modeling tasks in resource constrained automotive systems is not a feasible solution. However, recent

advances have enabled the use of CNNs to capture very-long term dependencies with the help of dilated causal convolution (DCC) layers [182]. The *dilation factor* of each DCC layer dictates the number of input samples to be skipped by that layer. The total number of samples influencing the output at a particular time step is called the *receptive field*. Using a larger dilation factor enables an output to represent a wider range of inputs, which helps to learn very-long term dependencies. Unlike RNNs/LSTMs/GRUs, CNNs do not have to wait for the previous time step output to process the input at the current time step. Thus, CNNs can process input sequences in parallel, making them more computationally efficient during both training and testing. Due to these promising properties, we adapt dilated CNNs for learning dependencies between in-vehicle messages in our TCNA model. We custom designed our TCNA network to consist of three *TCNA blocks*. A TCNA block consists of an attention block and a TCN residual block (TRB), as shown in Figure 72(a). The input to the first TCNA block is a time series of message data with n signal values as features. This partial sequence from the complete time-series dataset, that is given as the input to the model every time, is called a subsequence. The TRB is inspired by [182] and employs two DCC layers, two weight normalization layers, and two ReLU layers stacked together, as shown in Figure 72(b). This residual architecture helps to efficiently backpropagate gradients and encourages the reuse of learned features. We enhanced this TRB from [182] by: *(i)* adding an attention layer (discussed later); *(ii)* removing dropout layers to avoid thinning the network and provide our attention block with non-sparse inputs; and *(iii)* avoiding zero-padding the input time-series by computing the length of the subsequence using (32):

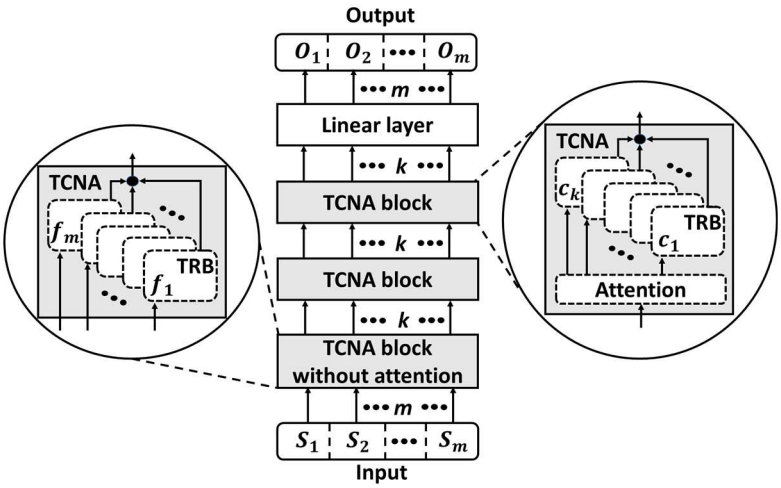
$$R = (k - 1) * 2^l \tag{32}$$

where R is the subsequence length, k is the kernel size, and l is the number of DCC layers in the network. This was done because we found that padding zeros to an input sequence distorts the sequential nature of in-vehicle time series data.

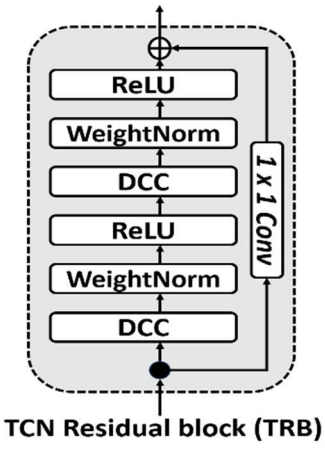
The first TCNA block does not contain an attention block, and the inputs are directly fed to the TRB as shown in Figure 72(a), where $\{f_1, f_2, \ldots, f_m\}$ represent multiple channels of the first TRB block, m is equal to the number of features of the inputs, $\{c_1, c_2, \ldots, c_k\}$ represent multiple channels of the remaining TRBs, and k is the number of channels of TRB inputs. The first DCC layer inside the TRB processes each feature of the input sequence as a separate *channel* as shown in Figure 72(a). A 1-D dilated causal convolution operation is performed using a kernel of size two and the number of filters is three times the input features *(m)* in each DCC layer. The input and normalized outputs are passed through a ReLU activation function. This process is repeated one more time inside the TRB. A convolution layer with a filter size of 1×1 is added to make the dimensions of the outputs from the last ReLU activation and the input of the TRB consistent with each other. Each DCC layer in the TRB learns temporal relationships between messages by applying filters to its inputs and output dimensions are the same except for the first TRB. The output from the DCC layer is weight normalized for fast convergence and to avoid explosion of weight values. The weight updating filter weight values.

Our TCNA block also contains an attention block. Attention mechanisms enable deep neural networks to focus on the important aspects of the input sequences when producing outputs [183]. We devised a scaled dot product attention mechanism and modeled the attention as a mapping of three vectors, namely query *(Q)*, key *(K)*, and value *(V)*. A weight vector is computed by comparing the similarities between the Q and K vectors, and a dot product between the weight vector and the V vector generates the output attention weights. As this attention mechanism does not use the

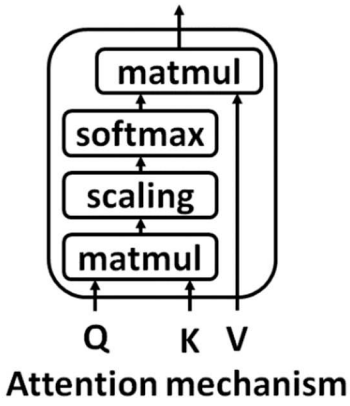
previous output information when generating the attention weights, it is a *self-attention* mechanism. In the context of our proposed TCNA network, self-attention mechanisms can help in identifying important feature maps and enhance the quality of intermediate inputs received by the DCC layers. This further assists with efficient learning of the very-long term dependencies between messages in an in-vehicle network.



(a)



(b)



(c)

Figure 72 (a) TCNA network architecture with the internal structure of the TCNA block, (b) TCN residual block showing the various layers of transformation and, (c) the attention mechanism.

The output feature maps of the TRB are given as the input to this attention block, shown in Figure 72(c). The attention block repeats its inputs to obtain the Q, K, and V vectors. A scalar-dot product is performed between Q and the transpose of key (K^T) to calculate the similarities between each Q and K vectors. The resultant dot product is scaled by a factor of $1/\sqrt{d_k}$ and passed through a *softmax* layer to calculate attention weights as shown in (33):

$$Attention(Q, K, V) = softmax\left({QK^T}/{\sqrt{d_k}} \right).V \tag{33}$$

where the term d_k represents the dimension of the K vector. The attention weights represent the importance of each feature map of the previous DCC layer. The attention weights are then scalar multiplied with V to produce the output of the attention block. Thus, the attention block uses a self-attention mechanism to improve the quality of feature maps that will be received by the subsequent TRBs. Ultimately, as shown in Figure 72(a), an input sequence flows through the entire TCNA network and is fed to the final linear layer which produces an output of m dimensions. The m-dimensional output represents the predicted signal values.

The TCNA network is trained using a rolling window approach. Each window consists of signal values corresponding to the current subsequence. Our TCNA network learns the temporal dependencies between messages inside a subsequence and tries to predict the signal values of the subsequence that are shifted by one time step to the right. We employ a mean squared error (MSE) loss function to compute the prediction error between signal values of the last time step in the predicted subsequence and the last time step of the input subsequence. The error is back propagated to update the weights for the filters. This process is repeated for each subsequence until the end of the training data, which constitutes one epoch. We train the model for multiple epochs and employ

a mini-batch training approach to speed up the training. At the end of each epoch, the model is evaluated using the unseen validation data. An early stopping mechanism is used to prevent model overfitting. The details of the model hyperparameters are discussed later, in Section 7.3.

7.2.3. MODEL TESTING

7.2.3.1. ATTACK MODEL

Here we present details of the various attack scenarios considered in this study. Our *TENET* framework attempts to detect the following complex and most widely seen attack scenarios in the in-vehicle network:

1) Plateau attack: This is an attack scenario where the attacker sets a constant value for a signal or multiple signals over the attack interval. It is hard to detect this attack especially when the set constant value is close to the true signal value.

2) Suppress attack: In this attack, the attacker tries to suppress a signal value by either disabling the ECU or deactivating the communication controller, effectively resulting in no message being transmitted. It is hard to detect short bursts of these attacks as they could be confused for a missing or delayed message.

3) Continuous attack: This attack represents a scenario where the attacker gradually overwrites the true signal value. The attacker then eventually will achieve the target value without triggering most ADS frameworks. These attacks are hard to detect and require an advanced ADS.

4) Playback attack: This attack involves the attacker using the previously observed sequences of signal values and trying to replay them again at a later time to trick the ADS. If the ADS is not trained to understand patterns in the sequence of transmitted messages, it will fail to detect these types of advanced attacks.

7.2.3.2. EVALUATION PHASE

We use the trained TCNA network in conjunction with a detection classifier to detect attacks on vehicles at runtime. The high frequency of messages in the in-vehicle network requires a detection classifier that is lightweight and can classify messages quickly with high detection accuracy and minimal overhead. Hence, we use the well-studied categorical variable decision tree-based classifier to detect between normal and attack samples (binary classification) due to their simpler nature, speed, and precise classification capabilities.

A decision tree starts with a single node (*root* node), which branches into possible outcomes. Each of those outcomes leads to additional nodes called *branch nodes*. Each branch node branches off into other possibilities and ends in a *leaf node* giving it a treelike structure. During training, the decision creates the tree structure by determining the set of rules in each branch node based on its input. During testing, the decision tree takes the input and traverses the tree structure until it reaches a leaf node. The evaluation phase begins by splitting the test data with attacks into two parts: *(i)* calibration data, and *(ii)* evaluation data. In the first part, only the calibration data is fed to the trained TCNA network to generate the predicted sequences. We then compute a divergence score (DS) for each signal in every message using (34):

$$DS_i^m(t) = \left(\hat{S}_i^m(t) - S_i^m(t+1)\right) \ \forall \ i \ \in [\,1, N_m], m \ \in \ [\,1, M] \tag{34}$$

where m represents the m^{th} message sample and M represents the total number of message samples, i represents the i^{th} signal of the m^{th} message sample and N_m represents the total number of signals in the m^{th} message, t represents the current time step, $\hat{S}_i^m(t)$ represents the i^{th} predicted signal value of the m^{th} message at time step t, and $S_i^m(t+1)$ represents the true i^{th} signal value of the m^{th} message sample at time step $t + 1$.

The DS is higher during an attack as the TCNA model is trained on the no-attack data and fails to predict the signal values correctly in the event of an attack. This sensitive nature of the DS to attacks makes it a good candidate for the input to our detection classifier. Moreover, the group of signal level DS for each message sample is stacked together to obtain a DS vector. We train the decision tree classifier using this DS vector as input to learn the distribution of both no-attack samples and attack samples. We use the unseen evaluation data (that has both attack and no-attack samples) to evaluate the performance of *TENET*.

7.3. EXPERIMENTAL SETUP

To evaluate the effectiveness of the *TENET* framework, we conducted various experiments. We compared *TENET* against four state-of-the-art prior works on ADS: RN [141], INDRA [47], LATTE [48] and HAbAD [159]. Together, these approaches reflect a wide range of sequence modeling architectures. RN [141] uses RNNs to increase the dimensionality of input signal values and reconstruct the input signal at the output by minimizing MSE. The trained RN model scans continuously for large reconstruction errors at runtime to detect anomalies over in-vehicle networks. INDRA [47] uses a GRU-based autoencoder that reconstructs input sequences at the output by reducing the MSE reconstruction loss. At runtime, INDRA utilizes a pre-computed static threshold to flag anomalous messages. LATTE [48] uses stacked LSTMs and self-attention mechanisms to build a predictor model, which is trained at design time to learn the normal operation of the system. At runtime, LATTE uses a one class support vector machine (OCSVM) based detector to detect attacks. HAbAD [159] uses an LSTM based autoencoder model with attention to detect anomalies in real-time embedded systems. HAbAD uses a supervised learning detector that combines a kernel density estimator (KDE) and k-nearest neighbors (KNN) algorithm

to detect anomalies. The comparisons of *TENET* with the above-mentioned ADS are presented in subsections 7.3.2 and 7.3.3.

We adopted an open-source CAN message dataset developed by ETAS and Robert Bosch GmbH [138] to train our model, and the comparison works. The dataset consists of multiple CAN messages with different number of signals that were modeled after real-world vehicular network information. Moreover, the dataset has a distinct training set that has normal CAN messages and a labeled testing dataset for different types of attacks. For training and validation, we used the training dataset from [138] without any attack scenarios in an unsupervised manner. We tested our proposed *TENET* framework, and all comparison works by modeling various real-world attacks (discussed in Section 7.2.3.1) using the test dataset in [138]. Note that *TENET* can be easily adapted to other in-vehicle network protocols such as Flexray [45] and Ethernet, as it relies only on the message payload information. However, the lack of any openly available datasets for these protocols prevents us from showing results on them.

We used PyTorch 1.8 to model and train various machine learning models including *TENET*, and the comparison works. Our framework uses 85% of data for training and the remaining 15% for validation. We trained *TENET* for 200 epochs with an early stopping mechanism that constantly monitors the validation loss after each epoch. If no improvement in validation loss is observed in the past 10 (patience) epochs, training is terminated. We used MSE to compute the prediction error and the ADAM optimizer with a learning rate of 1e-4. We employed a rolling window approach (discussed in Section 7.2.2) with a batch size of 256, and a subsequence length of 64. We used scikit-learn to implement the decision tree classifier, with the *gini* criterion, and *best* splitter to detect anomalies based on the divergence score. Before discussing the results, we define performance metrics that we used in the context of ADS. We classify a message as a true positive

(TP) only if the model detects a true attack as an anomaly, and a true negative *(TN)* is when a normal message is detected as a no-attack message. When the model detects a normal message as an anomalous message it is defined as a false positive *(FP)*, whereas an actual anomalous message which is not detected is a false negative *(FN)*. Using these definitions, we evaluate the ADS based on four different performance metrics:

(i) Detection Accuracy: quantifies the ability of the ADS to detect an anomaly correctly, as calculated using (35):

$$Detection\ accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{35}$$

(ii) Receiver Operating Characteristic (ROC) curve with area under the curve (AUC): which measures the ADS performance as the area under the curve in a plot between the true positive rate (TPR) and false positive rate (FPR). The TPR and FPR are computed using (36) and (37) respectively.

$$TPR = \frac{TP}{TP+FN} \tag{36}$$

$$FPR = \frac{FP}{FP+TN} \tag{37}$$

(iii) False Negative Rate (FNR): which quantifies the probability that a *TP* will be missed by the ADS (lower is better). It is calculated using (38).

$$FNR = \frac{FN}{FN+TP} \tag{38}$$

(iv) Mathews Correlation Coefficient (MCC): which provides an accurate evaluation of the ADS performance while working with imbalanced datasets, and is defined in (39).

$$MCC = \frac{(TP*TN) - (FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (39)$$

Another metric that is sometimes used is the F-1 score, which is the harmonic mean of precision and recall. As both precision and recall do not include the true negatives in their computation, the F-1 score metric fails to represent the true performance of the classifier. Unlike the F-1 score metric, the MCC metric that we consider includes all the cells of the confusion matrix, thus providing a much more accurate evaluation of the frameworks.

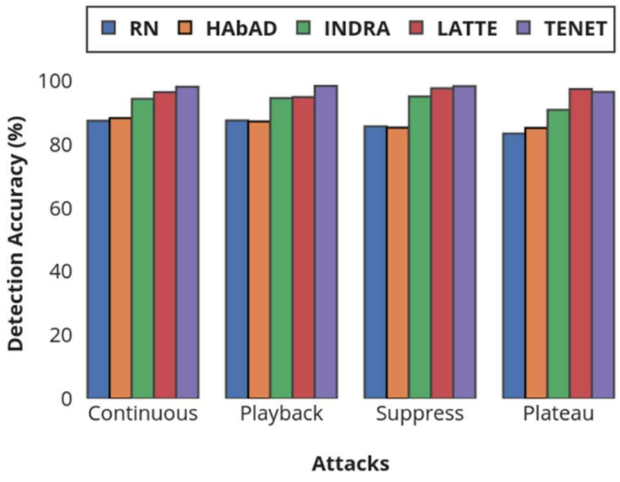
7.3.1. RECEPTIVE FIELD LENGTH SENSITIVITY ANALYSIS

In the first experiment, we compare the performance of our TCNA architecture with four different receptive field lengths while the remaining hyperparameters are unchanged. We conduct this analysis to evaluate whether very long receptive lengths can help with a better understanding of normal system behavior. All the variants are evaluated based on their performance on two training metrics: average training loss and average validation loss, and the best model is selected for further comparisons. The average training loss value represents the average loss between the predicted behavior and observed behavior of each iteration in the training data. In contrast, the average validation loss represents the average loss between the predicted behavior and the observed behavior of each iteration in the validation data.

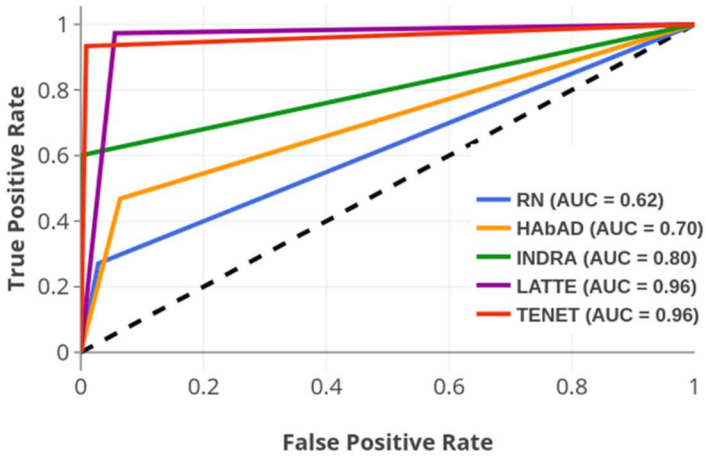
Table 14 TCNA variants with different receptive field lengths.

Metrics	**Receptive field lengths**			
	16	**32**	**64**	**128**
Average training loss	4.1e-4	3e-4	**2.5e-4**	6.8e-4
Average validation loss	5.5e-4	4.3e-4	**2.9e-4**	9.3e-4

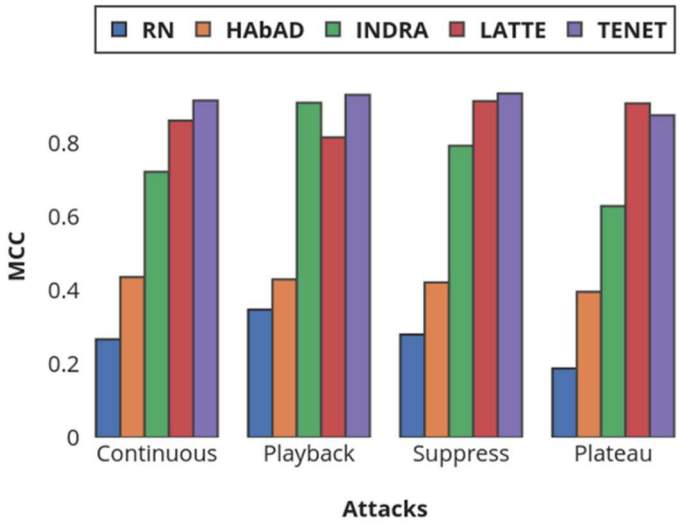
Table 14 shows the average training and validation loss of the four variants of TCNA. We can observe that a receptive length of 64 has the lowest average training and validation loss. Therefore, we select 64 as our receptive field value, which is twice the maximum receptive field length presented in one of the comparison works (sequence length of 32 in [159]). This long receptive field length enables us to more effectively learn very long-term dependencies in the input time series data and allows us to better understand the normal vehicle operating behavior.



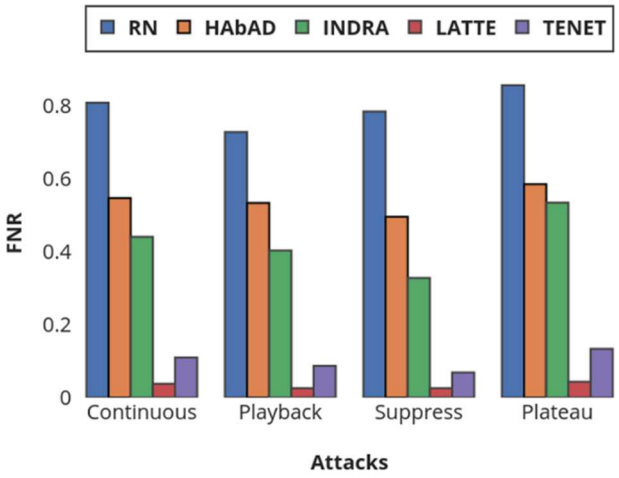
(a)



(b)



(c)



(d)

Figure 73 Comparison of (a) detection accuracy, (b) ROC-AUC for playback attack, (c) MCC, and (d) FNR for *TENET* and ADSs from prior works.

7.3.2. PRIOR WORK COMPARISON

In this subsection, we compare our *TENET* framework with the state-of-the-art ADS works RN [141], INDRA [47], LATTE [48] and HAbAD [159]. The results of the comparison on the metrics discussed in the previous section are as shown in Figure 73. From Figure 73(a)-(d), it is

evident that *TENET* outperforms all comparison works in detection accuracy, ROC-AUC and MCC metrics and achieves the second-best performance in FNR under various attack scenarios. Table 15 summarizes the average relative percentage improvement of *TENET* over the comparison works under all attack scenarios. (*Note*: A negative value in Table 15 indicates percentage degradation in performance). We can see that our proposed *TENET* framework achieves an improvement of up to 69.47% in FNR, 64.30% in MCC, 37.25% in ROC-AUC, and 9.48% in detection accuracy under all attack scenarios.

Table 15 Relative percentage improvement of *TENET* vs. other ADS.

Prior ADS Works	Detection accuracy	ROC-AUC	MCC	FNR
LATTE [48]	1.26	0.00	3.95	-6.63
INDRA [47]	3.32	17.25	19.14	32.70
HABAD [159]	9.07	26.50	49.26	44.05
RN [141]	9.48	37.25	64.30	69.47

In summary, our *TENET* framework with a customized TCNA network outperforms all prior recurrent architectures with and without attention, due to its ability to capture very-long term dependencies in time-series data. Moreover, the attention mechanism within the TCNA improves the quality of the outputs of the TRB, which enables efficient learning of very-long term dependencies. Thus, our TCNA network with the decision tree classifier represents a formidable anomaly detection framework.

7.3.3. MEMORY OVERHEAD AND LATENCY ANALYSIS

Lastly, we compare the number of trainable parameters, the memory footprint, and inference time of the *TENET* framework, and the comparison ADS works to evaluate their memory and computational overheads. Table 16 shows the memory footprint, model parameters, and average

inference latency of *TENET* and the other ADS. It is important to consider the memory and latency overhead of ADS models because automotive ECUs are resource constrained and it is crucial to have an ADS that does not interfere with the normal operation of safety-critical automotive applications. All results are obtained for deployment on an NVIDIA Jetson TX2 with dual-core ARM Cortex-A57 CPUs, which has specifications similar to real-world ECUs.

Table 16 Memory, model size, and inference latency analysis.

ADS framework	Memory footprint (KB)	Model parameters (x 10^3)	Inference time (μs)
RN [141]	**7.2**	**1.3**	412.50
INDRA [47]	453.8	112.9	482.10
LATTE [48]	1300	332.03	258.53
HAbAD [159]	261.63	64.48	1370.10
TENET	59.62	6.06	**250.24**

It can be observed that *TENET* has the second lowest number of model parameters and memory footprint over all the other comparison works except RN [141]. Even though RN has the least number of model parameters and memory footprint, it fails to effectively capture the temporal dependencies between messages, resulting in very poor performance, as can be seen in Figure 73 (a)-(d). Compared to INDRA, LATTE, and HAbAD, *TENET* achieves an average reduction of 86.49% in memory footprint and 94.46% in the number of trainable model parameters. *TENET* is able to achieve high performance with significantly fewer trainable parameters because of the fewer filters used by each DCC layer in the TCNA network. This is achieved using the attention block in TCNA which improves the quality of the outputs of each TRB thus eliminating the need for more filters. Moreover, *TENET* also has the lowest inference time with an average of 43.09% reduction against all comparison works. *TENET* is able to achieve faster inferencing because, unlike recurrent architectures, *TENET* employs CNNs to process multiple subsequences in parallel,

which helps reduce the inference time. Thus, *TENET* is able to achieve superior performance across various attack scenarios in automotive platforms with minimal memory and computational overhead.

7.4. CONCLUSION

In this chapter, we have proposed a novel anomaly detection framework called *TENET* for automotive cyber-physical systems based on Temporal Convolutional Neural Attention (TCNA) networks. We also proposed a metric called the divergence score (DS), which measures the deviation of the predicted signal value from the actual signal value. We compared our framework with the best-known prior works that employ a variety of sequence model architectures for anomaly detection. Compared to the best performing prior work, *TENET* achieves an improvement of up to 9.48% in detection accuracy, 37.25% in ROC-AUC, 64.30% in MCC, and 69.47% in FNR metrics with 98.17% fewer model parameters, 95.41% decrease in memory footprint, and 81.73% lower inference time. Given the proliferation of connected vehicles with large attack surfaces on the roads today, the promising results in this work highlight a compelling potential for deploying *TENET* to achieve fast, low-footprint, and accurate anomaly detection in emerging automotive platforms.

8. CONCLUSION AND FUTURE WORK RECOMMENDATIONS

8.1. RESEARCH CONCLUSION

In this dissertation, we addressed significant design challenges related to the reliability, security, and real-time performance of automotive cyber-physical systems. Our proposed *ROSETTA* framework enhances the reliability of automotive cyber-physical systems by integrating jitter-aware message scheduling approaches and techniques that aid in preserving signal integrity while meeting timing and deadline constraints. In addition, *ROSETTA* enhances the security of automotive cyber-physical systems by incorporating intelligent key management and schedule optimization techniques. These techniques have minimal overhead and ensure that the ECU loads stay below 100% to avoid any undesirable overhead on the real-time automotive applications and prevent them from missing deadlines. Additionally, *ROSETTA* uses various deep learning based intrusion detection systems to detect cyber-attacks in complex modern automotive systems. Lastly, *ROSETTA* utilizes a variety of runtime management frameworks to handle real-time perturbations and support design time generated solutions while meeting real-time performance and deadline requirements. The experimental results of our proposed *ROSETTA* framework confirm the benefits of the holistic solution that jointly improves the reliability, security, and real-time performance of automotive system designs.

JAMS-SG is the first component of *ROSETTA* framework that utilizes a fast hybrid heuristic combining simulated annealing (SA) and greedy randomized adaptive search procedure (GRASP) to jointly explore jitter-aware frame packing and design time schedule synthesis for FlexRay protocol based automotive systems. Unlike prior works, we consider random jitter (which is hard to handle compared to deterministic jitter) and realistic stochastic jitter models to accurately model

the impacts of jitter on automotive systems. We propose an intelligent runtime scheduler that opportunistically packs jitter-affected time-triggered messages and high-priority event-triggered messages using a multi-level feedback queue (MLFQ). We introduce a partial message transmission scheme using our runtime scheduler and a custom addressing and segmentation scheme within the payload segment of the FlexRay frame. Lastly, our proposed framework demonstrated high robustness to varying levels of jitter by achieving an average reduction of around 16.63%, 16.31%, and 17.72% in average response times under low, medium, and high jitter conditions respectively, with no deadline misses compared to the best-known prior works in this area. Moreover, *JAMS-SG* proved to be highly scalable with no deadline misses and outperforms the best-known prior works under various system sizes in the presence of jitter.

We then proposed a priority-based multi-level monitoring approach to preserve signal integrity of high-criticality torque-related signals in automotive systems. Our proposed approach uses handshake mechanisms and performance counters to monitor the signals and uses a state machine based model to track the maturity of the faults. Moreover, our proposed technique also initiates different remedial actions in the form of fail-safe vehicle modes known as limp modes based on the severity of the fault. Lastly, a comprehensive analysis of our proposed technique is presented using real automotive hardware of a prototype hybrid electric vehicle in a hardware-in-the-loop test setup. Our experimental results demonstrate that we can detect level-1 and level-2 critical faults under 100 ms and initiate a remedial action under one second with less than 1% (insubstantial) overhead on the CAN bus.

Our next contribution is a holistic security-aware design framework called *SEDAN* that aims to improve the security of time-triggered automotive systems without violating any security, ECU utilization, and deadline constraints. We introduce a novel methodology to derive security

requirements for messages in the system using automotive signal integrity level (ASIL), a risk classification scheme defined in ISO 26262. The messages and the derived security requirements are set up for joint exploration and synthesis of message schedules and security characteristics (e.g., key size) using GRASP while ensuring that no ECU utilization exceeds 100%. At runtime, *SEDAN* employs a session key-based approach using station-to-station key agreement protocol using the elliptic curve cryptography (ECC) operations. Moreover, *SEDAN* uses an authenticated encryption and decryption scheme to simultaneously achieve confidentiality and authenticity to the message data. We benchmarked various cryptographic algorithms such as advanced encryption standard (AES), Rivest–Shamir–Adleman (RSA), and ECC on a real-world automotive ECU to model the runtime behavior of encryption/decryption operations. Lastly, *SEDAN* demonstrated an average reduction of around 68% in average response time for messages under all application loads without any deadline misses compared to the best-known prior works in this area.

Next, we proposed a novel deep learning based intrusion detection framework called *INDRA* that utilizes a gated recurrent unit (GRU) based recurrent autoencoder network to learn the normal system behavior at design time. At runtime, *INDRA* uses a static threshold approach that uses the proposed intrusion score (IS) metric, which quantifies the deviation from learned normal behavior. We present a comprehensive analysis on the selection of the threshold for IS and compare our proposed *INDRA* framework with various state-of-the-art intrusion detection works. Our proposed *INDRA* framework outperformed all the comparison works and achieved an average improvement of around 18.1% in detection accuracy and 37.4% reduction in false positive rate when evaluated under different attack scenarios. Lastly, benchmarking our proposed *INDRA* framework on a real-world automotive ECU and the scalability analysis conducted using the benchmarking results has demonstrated the lightweight nature and scalability of our proposed *INDRA* framework.

We subsequently proposed a deep learning based anomaly detection framework called *LATTE* that uses stacked long short-term memory networks (LSTMs) organized in an encoder-decoder configuration and integrates novel self-attention mechanisms to predict the next message value at design time. The self-attention mechanism enhanced the ability of the model to focus on the important hidden state information from the past. At runtime, *LATTE* monitors the prediction error (deviation) for each message sample and uses one class support vector machine (OCSVM) based non-linear classifier to detect anomalies as cyber-attacks. We presented a comprehensive analysis for the selection of the deviation measure. To evaluate the effectiveness of our proposed *LATTE* framework, we compared it with various machine learning, statistical, and proximity-based anomaly detection techniques under different attack scenarios. On average, across all attacks, *LATTE* achieved an average of 18.94% improvement in detection accuracy, 19.5% improvement in F1 score, 37% improvement in receiver operating characteristic curve with area under the curve (ROC-AUC), and 79% reduction in false-positive rate. Moreover, the overhead analysis results obtained by implementing our proposed *LATTE* framework on a real-world automotive ECU indicates the lightweight nature and fast execution time of the proposed approach.

Lastly, we proposed a temporal CNN with neural attention (TCNA) based anomaly detection framework called *TENET* to learn very long-term temporal dependencies between messages to better understand the normal operating conditions of the system at design time. During runtime, *TENET* uses a decision tree based fast classifier and the proposed divergence score (DS) to quantify the deviation from the normal behavior to detect anomalies as cyber-attacks. Compared to the best performing prior works, *TENET* achieved up to 9.48% improvement in detection accuracy, 64.30% improvement in Matthews correlation coefficient (MCC), 37.25% improvement in ROC-AUC, and 69.47% reduction in false-negative rate. Moreover, the implementation of

TENET on a real-world automotive ECU indicates the low memory footprint, a fewer number of model parameters, and fast inference time of our proposed approach.

8.2. RECOMMENDATIONS FOR FUTURE WORK

Modern automotive systems are rapidly evolving to realize the goals of autonomous driving. This increased the adoption of a suite of sensors, complex AI algorithms, a diverse set of new communication standards, and advanced vehicle control. As a result, numerous new challenges have emerged that impact the reliability, security, and real-time performance of automotive cyber-physical systems. We envision the following as possible directions for future work.

- *Resilience to adversarial attacks*: Modern deep learning algorithms have shown superior performance in the area of sensing and perception that are crucial for various ADAS applications. However, these algorithms are vulnerable to carefully crafted adversarial attacks. In [30], researchers generated various robust visual adversarial perturbations to a stop sign that resulted in it being misidentified as a 45 mph speed limit sign. Other researchers were able to blind a Mobileye C2-270 camera and demonstrate jamming, spoofing, and relay attacks on an Ibeo LUX3 LiDAR sensor [155]. More recent attacks include tricking the lane change system of a Tesla Model S with bright stickers on the road by Tencent Keen security lab in 2019 [31] and object removal attacks on LiDAR sensors in 2021 [32]. Moreover, recent model inversion attacks [185] that try to reconstruct training data from the model parameters are gaining popularity. Such attacks pose a significant threat to the proprietary data of the automakers that are used to train the deep learning models. Moreover, with newer and scalable learning approaches for large deep learning algorithms, such as with federated learning in data center

environments, the need for creating new approaches for tamper-proof deep learning algorithms and building resilience to adversarial attacks becomes even more imperative.

- *Advanced deep learning models for intrusion detection*: Several deep learning based models that encompass monitoring and attack detection at an in-vehicle network level (such as in Chapters 5, 6, and 7) and vehicular ad hoc network (VANET) level (such as [184], [186]-[188]) have been studied in the literature. However, due to the rollout of increasingly connected vehicles, we envision that Black-hole DDoS attacks [189] (where communication between vehicles is blocked) and Sybil attacks [190] (where a vehicle operates with multiple identities) will become increasingly common. Such attacks will result in confusing the existing deep learning algorithms, potentially causing failure across vehicle subsystems. Moreover, sequence models such as RNNs, LSTMs, and GRUs operate sequentially on the input data, which limits their ability to achieve faster inference times, especially on resource-constrained ECUs. The state-of-the-art advanced deep learning models such as Transformers [183], graph neural nets [191], customized natural language processing (NLP) models [192], and generative adversarial networks (GANs) [193] that were proven to perform well in their respective fields can be adapted to automotive systems.

- *Intelligent scheduler design using machine learning*: The era of intelligent transportation systems (ITS) is driven by VANETs using advanced communication modes such as vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I). Today, VANETs play a crucial role in building a safe and reliable connected vehicular ecosystem by relying on data gathered from vehicles and roadside units (RSUs). Moreover, this communication is highly time-sensitive

and must be reliably conveyed to several vehicles and RSUs. This brings up the challenge of designing an intelligent scheduler that can effectively handle communication from various vehicles and RSUs. In [194], the authors proposed a reinforcement learning based scheduler for V2V communication and show its effectiveness in avoiding collisions and its ability to reuse existing resources better than the state-of-the-art schedulers. In [195], the authors proposed a deep Q-network based energy-efficient V2I scheduler that satisfies all the safety and quality-of-service (QoS) concerns. These works show promising potential for using advanced machine learning approaches to tackle the problem of designing intelligent schedulers that can be deployed in emerging automotive systems. Similar approaches can be adapted to develop intelligent schedulers for scheduling in-vehicle network traffic.

- *Resource management challenges in electric vehicles*: Battery-based electric vehicles (EVs) have become increasingly popular due to growing concerns about minimizing greenhouse gas emissions. Unlike traditional internal combustion engine (ICE) vehicles that rely on gasoline, EVs only rely on the onboard electrical energy stored in the batteries. Many modern EVs use lithium-ion batteries as the primary energy storage system (ESS), which has around $1/100^{th}$ of the energy density of gasoline, making the onboard electrical energy extremely valuable. This creates a unique challenge with energy management and requires the functionalities of the EVs to be highly energy-efficient, as they can have a direct impact on the vehicle's overall range and performance. Thus, the strategies that are developed for non-electric vehicles cannot be directly adopted in EVs, which presents an exciting opportunity to explore the problem of energy-aware resource management in EVs. Many ESS management techniques are proactive in nature and require accurate estimation of the current and future approximation of the state

of the system [196]. Thus, building accurate predictive models for estimating metrics such as vehicle velocity and energy consumption of different subsystems is highly crucial for designing an efficient ESS management technique. State-of-the-art deep learning models are an excellent choice as they have proven to be highly efficient when presented with sufficient quality data, and many vehicles facilitate the recording of this data. Some of the recent works such as, [196] and [197], focus on predicting the vehicle velocity and integrate it into the vehicle's energy management strategy to make intelligent decisions. Moreover, efforts need to be focused on developing energy-aware security mechanisms to deal with various security vulnerabilities due to the increased number of electronic components and attack vectors. Additionally, the fuel economy of the EVs should also be a crucial component of EV resource management as it plays a key role in determining the range of the vehicle. Various fuel economy metrics such as kWh/mile (kilowatt hours per mile) and miles per gallon gasoline equivalent (mpgge) can be considered to analyze and compare the energy efficiencies across different vehicle types.

- *Intelligent vehicle control*: The integration of powerful multi-core ECUs that can run various complex applications has enabled intelligent vehicle control in modern vehicles. These control algorithms heavily rely on heterogeneous sensor data, communication from many external systems (V2X and 5G), and advanced computation methods (deep learning models). Many researchers have studied the adoption of intelligent vehicle control for achieving high energy efficiency and fuel economy. An intelligent engine on/off control strategy based on the vehicle velocity prediction using a neural network is proposed in [198] to improve the fuel economy of a prototype hybrid electric vehicle. In [199], a V2V based vehicle speed prediction

methodology was introduced to determine the optimal operation of the engine to increase the fuel economy. Although these techniques achieve improvements in fuel economy, they can quickly lead to poor performance when the prediction methodologies fail due to runtime disruptions such as incorrect sensor data, computation and communication uncertainties (such as jitter), and missing data. Moreover, as these control algorithms rely on information from external sources to control various safety-critical systems, they need to be protected from cyber-attacks. Thus, there is a need to enable robust and secure intelligent vehicle control, which is critical in enabling future autonomous vehicles.

- *Exploring the impact of deep learning subsystems*: Due to the large-scale integration of various deep learning models into different vehicular subsystems for realizing applications such as intrusion detection systems (IDS), intelligent vehicle control, and perception applications, it is essential to study and understand the implications of these systems on the overall performance and safety of the vehicle. Furthermore, these systems will be operating co-operatively in a pipeline manner to achieve various goals such as ESS management, secure communication, and vehicle autonomy. This becomes increasingly critical as these systems actively exchange information, which can sometimes lead to issues such as unintended error propagation, failure of a critical subsystem due to a dependent subsystem failure (cascading failures), and new complex vulnerability chains. Moreover, deep learning models lack robustness and are highly susceptible to small input perturbations [200]. Therefore, it is essential to study the performance implications of cascading these subsystems and develop new methodologies that improve the resiliency to runtime uncertainties and handle worst-case scenarios such as missing data and failure of dependent subsystems. These issues are crucial to address as they

can have detrimental effects on the latencies of automotive applications and messages, resulting in missed deadlines and catastrophic failures of safety-critical systems.

- *Reliable and secure multi-sensor fusion*: Modern vehicles are equipped with a variety of sensors to perceive the environment. They often combine the information from multiple homogeneous or heterogeneous sensors to find the single best estimate of the state of the environment, and this process is referred to as sensor fusion. Sensor fusion helps sensors complement each other's limitations and offers greater leverage to the system compared to a system with individual sensors. This results in high precision, extended spatial and temporal coverage, and improved resolution, which are crucial in safety-critical automotive systems. However, these sensor fusion algorithms are highly sensitive to the quality of the data from various sensors, which can be susceptible to various runtime perturbations and uncertainties due to the harsh operating conditions of the vehicle, such as high operating temperatures, vibrations, electromagnetic interference (EMI), radio frequency interference (RFI), aging sensors, and from various environmental sources. Moreover, works such as [30] have demonstrated that these sensors can be hijacked and tricked to achieve malicious goals. Thus, there is a need for the design of reliable and secure multi-sensor fusion algorithms that are resilient to uncertainties and robust to variations in sensor data to ensure the safety of present semi-autonomous and future autonomous vehicles. Various aspects of resource management such as scheduling of sensor fusion tasks and messages, designing of lightweight algorithms to minimize ECU overhead, robustness to faulty vehicular control, ensuring sensor data integrity, and managing security and real-time requirements are crucial elements that need to be explored when designing reliable and robust sensor fusion techniques. Moreover, it is

crucial to ensure optimal placement and configuration of heterogeneous sensors in the vehicle to achieve an extended coverage which aids in the sensor fusion. Thus, it is vital to consider works such as [170] as the baseline for achieving optimal placement and configuration.

BIBLIOGRAPHY

[1] H. Kellerman, G. Nemeth, J. Kostelezky, K. Barbehön, F. El-Dwaik, and L. Hochmuth, "BMW 7 Series architecture," ATZextra, November 2008.

[2] V. K. Kukkala, S. Pasricha, and T. Bradley, "Advanced Driver-Assistance Systems: A path toward autonomous vehicles," in IEEE Consumer Electronics Magazine, Vol. 7, Iss. 5, September 2018.

[3] SAE International, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles." [Online]. Available: https://www.sae.org/standards /content/j3016_202104/, 2021.

[4] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "Roadmap for Cybersecurity in Autonomous Vehicles," in IEEE Consumer Electronics Magazine (CEM), 2022.

[5] D. Jiang and L. Delgrossi, "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments," in Proc. of IEEE Vehicular Technology Conference (VTC) Spring, 2008.

[6] J. B. Kenney, "Dedicated Short-Range Communications (DSRC) Standards in the United States," in Proc. of the IEEE, Vol. 99, No. 7, July 2011.

[7] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in Advances in Neural Information Processing Systems (NIPS), 2015.

[9] Tesla Autopilot. [Online]. Available: https://www.tesla.com/autopilot.

[10] H. Thapliyal, S. P. Mohanty, and S. J. Prowell, "Emerging Paradigms in Vehicular Cybersecurity," in IEEE Consumer Electronics Magazine (CEM), Vol. 8, No. 6, Nov. 2019.

[11] ABESE Team, "Future advances in body electronics," in NXP White paper. [Online]. Available: https://www.nxp.com/docs/en/white-paper/BODYDELECTRWP.pdf

[12] Automotive IQ, "In-Car Network Architecture 2020". [Online]. Available: https://www.automotive-iq.com/electrics-electronics/infographics/in-car-network-architecture-2020.

[13] "Upstream Security's 2021 Global Automotive Cybersecurity Report," [Online]. Available: https://upstream.auto/2021report, 2021.

[14] ISO 26262: Road Vehicles- Functional Safety, ISO Standard, 2011.

[15] "CAN Specifications version 2.0," Robert Bosch gmbh, 1991.

[16] "FlexRay Communications System Protocol Specification, ver.3.0.1." [Online]. Available: http://www.flexray.com

[17] "SAE AS6802: Time-Triggered Ethernet," SAE Standard, 2016.

[18] "IEEE 802.1 Time-Sensitive Networking Task Group". [Online]. Available: www.ieee802.org.

[19] G. C. DiDomenico, J. Bair, V. K. Kukkala, J. Tunnell, M. Peyfuss, M. Kraus, J. Ax, J. Lazarri, M. Munin, C. Cooke, and E. Christensen, "Colorado State University EcoCAR 3 Final Technical Report," in SAE World Congress Experience (WCX), April 2019.

[20] K. Schmidt and E.G. Schmidt, “Message Scheduling for the FlexRay Protocol: The Static Segment”, in IEEE Transactions on Vehicular Technology (TVT), Vol. 58, Iss. 5, 2009.

[21] K. Schmidt and E.G. Schmidt, “Optimal Message Scheduling for the Static Segment of FlexRay”, in Proc. of IEEE Vehicular Technology Conference (VTC) Fall, 2010.

[22] Semiconductor engineering, “Chasing Reliability in Automotive Electronics.” [Online]. Available: https://semiengineering.com/chasing-reliability-in-automotive-electronics, 2019.

[23] R. Baumann, “Soft errors in advanced computer systems,” in IEEE Design & Test of Computers, Vol. 22, No. 3, 2005.

[24] D. Kraak, M. Taouil, S. Hamdioui, P. Weckx, F. Catthoor, A. Chatterjee, A. Singh, H. J. Wunderlich, and N. Karimi, “Device aging: A reliability and security concern,” in Proc. of IEEE European Test Symposium (ETS), 2018.

[25] 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 Proc. of IEEE Symposium on Security and Privacy (SP), 2010.

[26] C. Valasek and C. Miller, “Remote Exploitation of an Unaltered Passenger Vehicle,” in Black Hat USA, 2015.

[27] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, “Lock it and still lose it-on the (in) security of automotive remote keyless entry systems,” in Proc. of USENIX, 2016.

[28] L. Wouters, E. Marin, T. Ashur, B. Gierlichs, and B. Preneel, “Fast, Furious and Insecure: Passive Keyless Entry and Start Systems in Modern Supercars,” in IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), 2019.

[29] L. Wouters, B. Gierlichs, and B. Preneel, “My other car is your car: compromising the Tesla Model X keyless entry system,” in IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), 2021.

[30] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, “Robust physical-world attacks on deep learning visual classification,” in Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[31] Tencent Keen Security Lab, “Experimental Security Research of Tesla Autopilot,” 2019.

[32] Z. Hau, K. T. Co, S. Demetriou, and E. C. Lupu, “Object removal attacks on lidar-based 3d object detectors,” in arXiv preprint, 2021.

[33] R.P Weinmann and B. Schmotzle, “TBONE – A zero-click exploit for Tesla MCUs,” White paper, ComSecuris, 2020.

[34] “Kia Motors America suffers ransomware attack, $20 million ransom,” [Online]. Available: https://www.bleepingcomputer.com/news/security/kia-motors-america-suffers-ansomware-attack-20-million-ransom/

[35] J. Burke, J. McDonald, and T. Austin, “Architectural support for fast symmetric-key cryptography,” in Proc. of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2000.

[36] G. J. Simmons, “Symmetric and Asymmetric Encryption,” in ACM Computing Surveys, Vol. 11, No. 4, 1979.

[37] C. W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli, “Security-aware mapping for TDMA-based real-time distributed systems,” in Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2014.

[38] C. W. Lin, B. Zheng, Q. Zhu, and A. Sangiovanni-Vincentelli, “Security-aware design methodology and optimization for automotive systems,” in ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 21, No. 1, 2015.

[39] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, “Lightweight authentication for secure automotive networks,” in Proc. of IEEE/ACM Design, Automation & Test in Europe & Exhibition (DATE), 2015.

[40] P. Mundhenk, A. Paverd, A. Mrowca, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, “Security in Automotive Networks: Lightweight Authentication and Authorization,” in ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 22, No. 2, 2017.

[41] T. Xie and X. Qin, “Improving security for periodic tasks in embedded systems through scheduling,” in ACM Transactions on Embedded Computing Systems (TECS), Vol. 6, No. 3, 2007.

[42] V. K. Kukkala, T. Bradley, and S. Pasricha, “Priority-based Multi-level Monitoring of Signal Integrity in a Distributed Powertrain Control System,” in Proc. of IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling, July 2015.

[43] V. K. Kukkala, T. Bradley, and S. Pasricha, “Uncertainty Analysis and Propagation for an Auxiliary Power Module,” in Proc. of IEEE Transportation Electrification Conference (TEC), June 2017.

[44] V. K. Kukkala, S. Pasricha, and T. Bradley, “JAMS: Jitter-Aware Message Scheduling for FlexRay Automotive Networks,” in Proc. of IEEE/ACM International Symposium on Network-on-Chip (NOCS), October 2017.

[45] V. K. Kukkala, S. Pasricha, and T. Bradley, "JAMS-SG: A Framework for Jitter-Aware Message Scheduling for Time-Triggered Automotive Networks," in ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 24, Iss. 6, September 2019.

[46] V. Kukkala, S. Pasricha, and T. Bradley, "SEDAN: Security-Aware Design of Time-Critical Automotive Networks," in IEEE Transaction on Vehicular Technology (TVT), Vol. 69, Iss. 8, August 2020.

[47] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "INDRA: Intrusion Detection using Recurrent Autoencoders in Automotive Embedded Systems," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 39, Iss. 11, November 2020.

[48] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "LATTE: LSTM Self-Attention based Anomaly Detection in Embedded Automotive Platforms," in ACM Transactions on Embedded Computing Systems (TECS), Vol. 20, No. 5s, Article 67, August 2021.

[49] S. V. Thiruloga, V. K. Kukkala, and S. Pasricha, "TENET: Temporal CNN with Attention for Anomaly Detection in Automotive Cyber-Physical Systems," in Proc. of IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC), January 2022.

[50] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period Optimization for Hard Real-time Distributed Automotive Systems", in Proc. of IEEE/ACM Design Automation Conference (DAC), 2007.

[51] SAE, "Automotive Engineering International," July 2016. [Online]. Available: https://www.sae.org/publications/magazines/content/16autd07/

[52] I. R. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, “Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised”, in Real-Time Systems, Vol. 35, No. 3, 2007.

[53] R. Saket and N. Navet, “Frame Packing Algorithms for Automotive Applications”, in Journal of Embedded Computing (JEC), Vol. 2, No. 1, 2006.

[54] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, “Recent Advances and Trends in On-board Embedded and Networked Automotive Systems”, in IEEE Transactions on Industrial Informatics (TII), Vol. 15, Iss. 2, 2019.

[55] Audi A5 Driver assistance systems [Online]. Available: https://www.audi-mediacenter.com/en/the-new-audi-a5-and-audi-s5-coupe-6269/driver-assistance-systems-6281

[56] M. Fleiss, T. M. Müller, M. Nilsson, and J. Carlsson, “Volvo Powertrain Integration into Complete Vehicle”, in ATZ worldwide, Vol. 118, Iss. 3, March 2016.

[57] B. Tanasa, U.D. Bordoloi, P. Eles, and Z. Peng, “Reliability-Aware Frame Packing for the Static Segment of FlexRay”, in Proc. of IEEE/ACM International Conference on Embedded Software (EMSOFT), 2011.

[58] M. Kang, K. Park, and M.K. Jeong, “Frame Packing for Minimizing the Bandwidth Consumption of the FlexRay Static Segment”, in IEEE Transactions on Industrial Electronics (TIE), Vol. 60, No. 9, 2013.

[59] S. Ding, R. Huang, R. Kurachi, and G. Zeng, “A Genetic Algorithm for Minimizing Bandwidth Utilization by Packing CAN-FD Frame”, in Proc. of IEEE International Conference on Embedded Software and Systems (ICESS), 2016.

[60] H. Zeng, W. Zheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli, “Scheduling the FlexRay Bus Using Optimization Techniques”, in Proc. of IEEE/ACM Design Automation Conference (DAC), 2009.

[61] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, “Flexray Schedule Optimization of the Static Segment”, in Proc. of IEEE/ACM international conference on Hardware/software codesign and system synthesis (CODES+ISSS), 2009.

[62] H.Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli, “Schedule Optimization of Time-Triggered Systems Communicating Over the FlexRay Static Segment”, in IEEE Transactions on Industrial Informatics (TII), Vol. 7, No. 1 , 2011.

[63] M. Grenier, L. Havet, and N. Navet, “Configuring the communication on FlexRay- the case of the static segment”, in Proc. of European Congress on Embedded Real Time Software (ERTS), 2008.

[64] Z. Sun, H. Li, M. Yao, and N. Li, “Scheduling Optimization Techniques for FlexRay Using Constraint-Programming”, in Proc. of IEEE/ACM International Conference on Green Computing and Communications and International Conference on Cyber, Physical and Social Computing, 2010.

[65] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty, “Modular Scheduling of Distributed Heterogeneous Time-Triggered Automotive Systems”, in Proc. of IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), 2012.

[66] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, “Time-triggered Implementations of Mixed-Criticality Automotive Software”, in Proc. of IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012.

[67] P. Mundhenk, F. Sagstetter, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, “Policy-based Message Scheduling Using FlexRay”, in Proc. of IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014.

[68] B. Tanasa, U.D. Bordoloi, P. Eles, and Z. Peng, “Scheduling for Fault-Tolerant Communication on the Static Segment of FlexRay”, in Proc. of IEEE Real-Time Systems Symposium (RTSS), 2010.

[69] R. Lange, F. Vasques, P. Portugal, and R.S. de Oliveira, “Guaranteeing Real-Time Message Deadlines in The FlexRay Static Segment Using a On-line Scheduling Approach”, in Proc. of IEEE International Workshop on Factory Communication Systems (WFCS), 2014.

[70] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, “Task-and Network-level Schedule Co-Synthesis of Ethernet-based Time-triggered Systems”, in Proc. of IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), 2014.

[71] F. Sagstetter, S. Andalam, P. Waszecki, M. Lukasiewycz, H. Stähle, S. Chakraborty, and A. Knoll, “Schedule Integration Framework for Time-Triggered Automotive Architectures”, in Proc. of IEEE/ACM Design Automation Conference (DAC), 2014.

[72] S. S. Craciunas and R. S. Oliver, “Combined task-and network-level scheduling for distributed time-triggered systems”, in Real-Time Systems, Vol. 52, No. 2, 2016.

[73] A. Novak, P. Sucha, and Z. Hanzalek, “Efficient Algorithm for Jitter Minimization in Time-Triggered Periodic Mixed-Criticality Message Scheduling Problem”, in Proc. of ACM International Conference on Real-Time Networks and Systems (RTNS), 2016.

[74] A. Minaeva, B. Akesson, Z. Hanzálek, and D. Dasari, “Time-triggered Co-Scheduling of Computation and Communication with Jitter Requirements”, in IEEE Transactions on Computers (TC), 2018.

[75] L. Maldonado, W. Chang, D. Roy, A. Annaswamy, D. Goswami, and S. Chakraborty, “Exploiting System Dynamics for Resource-Efficient Automotive CPS Design”, in Proc. of IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019.

[76] D. Roy, W. Chang, S. K. Mitter, and S. Chakraborty, “Tighter Dimensioning of Heterogeneous Multi-Resource Autonomous CPS with Control Performance Guarantees”, in Proc. of IEEE/ACM Design Automation Conference (DAC), 2019.

[77] T. J. Yamaguchi, K. Ichiyama, H. X. Hou, and M. Ishida, “A Robust Method for Identifying a Deterministic Jitter Model in a Total Jitter Distribution”, in Proc. of IEEE International Test Conference (ITC), 2009.

[78] T. Witkowski, P. Antczak, and A. Antczak, “Solving the Flexible Open-Job Shop Scheduling Problem with GRASP and Simulated Annealing”, in Proc. of IEEE International Conference on Artificial Intelligence and Computational Intelligence (AICI), 2010.

[79] L. Liu, H. Mu, and J. Yang, “Simulated annealing based GRASP for Pareto-optimal dissimilar paths problem”, in Soft Computing (SC), Vol. 21, No. 18, 2016.

[80] P. Kleberger, T. Olovsson, and E. Jonsson, “Security Aspects of the In-Vehicle Network in the Connected Car,” in Proc. of IEEE Intelligent Vehicles Symposium (IV), 2011.

[81] K. Tindell, H. Hansson, and A.J Wellings, “Analysing Real-Time Communications: Controller Area Network (CAN),” in Proc. of IEEE Real-Time Systems Symposium (RTSS), 1994.

[82] D. K. Nilsson, U. E Larson, and E. Jonsson, “Efficient In-Vehicle Delayed Data Authentication Based on Compound Message Authentication Codes,” in Proc. of IEEE Vehicular Technology Conference (VTC) Fall, 2008.

[83] P. Sundaram and J. D’Ambrosio, “Controller Integrity in Automotive Failsafe System Architectures,” in SAE World Congress & Exhibition (WCX), 2006.

[84] H. Buur, W. R. Cawthorne, T. W. Haines, J. J. Park, and L. G. Wozniak, “Method and apparatus for monitoring software and signal integrity in a distributed control module system for a powertrain system,” in US Patent 8428816 B2, 2013.

[85] M. K. Mandal, “Multimedia Signals and Systems,” in Springer Science & Business Media, 2012.

[86] Woodward SECM 112. [Online]. Available: mcs.woodward.com/support/wiki/index.php?title=SECM112, 2015.

[87] dSPACE Simulator Mid-Size: Standardized, off- the- shelf HIL simulator. Available from: http://www.dspace.com/en/pub/home/products/hw/simulator_hardware/dspace_simulator_mid_size.cfm, June 2015.

[88] I. Studnia, V. Nicomette, E.Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi, “Survey of security threats and protection mechanisms in embedded automotive systems,” in Proc. of IEEE Dependable Systems and Networks Workshop, 2013.

[89] 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 Proc. of USENIX Security Symposium, 2011.

[90] V. Izosimov, A. Asvestopoulos, O. Blomkvist, and M. Törngren, “Security-aware development of cyber-physical systems illustrated with automotive case study,” in Proc. of IEEE/ACM Design, Automation & Test in Europe & Exhibition (DATE), 2016.

[91] R. Zalman and A. Mayer, “A secure but still safe and low cost automotive communication technique,” in Proc. of IEEE/ACM Design Automation Conference (DAC), 2014.

[92] C. W. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli, “Security-aware mapping for CAN-based real-time distributed automotive systems,” in Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2013.

[93] C. W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli, “Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems,” in IEEE Embedded Systems Letter, Vol. 7, No. 1, 2015.

[94] G. Han, H. Zeng, Y. Li, and W. Dou, “SAFE: Security Aware FlexRay Scheduling Engine,” in Proc. of IEEE/ACM Design, Automation & Test in Europe & Exhibition (DATE), 2014.

[95] A. Hazem and H. A. Fahmy, “LCAP- A Lightweight CAN Authentication Protocol for Scheduling In-Vehicle Networks,” in Proc. of ESCAR Embedded Security in Cars Conference, 2012.

[96] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, “PRESENT: An Ultra-lightweight block cipher,” in Proc. of Springer International Workshop on Cryptographic Hardware and Embedded Systems, 2007.

[97] M. L. Chavez, C. H. Rosete, and F. R. Henriquez, “Achieving Confidentiality Security Service for CAN,” in Proc. of IEEE International Conference on Electronics, Communications and Computers (CONIELECOMP), 2005.

[98] M. Yoshikawa, K. Sugioka, Y. Nozaki, and K. Asahi, “Secure In-vehicle Systems against Trojan Attacks,” in Proc. of IEEE/ACIS International Conference on Computers and Information Science (ICIS), 2015.

[99] M. Lukasiewycz, P. Mundhenk, and S. Steinhorst, “Security-aware obfuscated priority assignment for automotive CAN platforms,” in ACM Transactions on Design Automation on Electrical Systems (TODAES), Vol. 21, No. 2, 2016.

[100] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, “The TESLA broadcast authentication protocol,” in RSA Cryptobytes, Vol. 5, No. 2, 2005.

[101] S. Shreejith and S. A. Fahmy, “Security aware network controllers for next generation automotive embedded systems,” in Proc. of IEEE/ACM Design Automation Conference (DAC), 2015.

[102] C. W. Lin and H. Yu, “Coexistence of safety and security in next-generation ethernet-based automotive networks,” in Proc. of IEEE/ACM Design Automation Conference (DAC), 2016.

[103] B. Zheng, P. Deng, R. Anguluri, Q. Zhu, and F. Pasqualetti, “Cross-layer codesign for secure cyber-physical systems,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 35, No. 5, 2016.

[104] M. Lin, L. Xu, L. T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, “Static security optimization for real-time systems,” in IEEE Transaction on Industrial Informatics (TII), Vol. 5, No. 1, 2009.

[105] H. Liang, M. Jagielski, B. Zheng, C. W. Lin, E. Kang, S. Shiraishi, C. Nita-Rotaru, and Q. Zhu, "Network and system level security in connected vehicle applications," in Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018.

[106] P. Waszecki, P. Mundhenk, S. Steinhorst, M. Lukasiewycz, R. Karri, and S. Chakraborty, "Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 36, No. 11, 2017.

[107] M. Wu, H. Zeng, C. Wang, and H. Yu, "Safety guard: Runtime enforcement for safety-critical cyber-physical systems," in Proc. of IEEE/ACM Design Automation Conference (DAC), 2017.

[108] R. G. Dutta, X. Guo, T. Zhang, K. Kwiat, C. Kamhoua, L. Njilla, and Y. Jin, "Estimation of safe sensor measurements of autonomous system under attack," in Proc. of IEEE/ACM Design Automation Conference (DAC), 2017.

[109] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," in Journal of Global Optimization, Vol. 6, No. 2, 1995.

[110] B. O'Higgins, W. Diffie, L. Strawczynski, and R. De Hoog, "Encryption and ISDN- A Natural Fit," in Proc. of IEEE International Switching Symposium (ISS), 1987.

[111] W. Diffie and M. Hellman, "New directions in cryptography," in IEEE Transactions on Information Theory (TIT), Vol. 22, No. 6, 1976.

[112] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in IEEE Transactions on Information Theory (TIT), Vol. 31, No. 4, 1985.

[113] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, and B. VanderSloot, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in Proc. of ACM SIGSAC Conference on Computer and Communications Security (CCS), 2015.

[114] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in Proc. of IEEE/ACM International Workshop on Hardware/Software Codesign (CODES/CASHE), 1998.

[115] OpenSSL: Cryptography and SSL/TLS toolkit [Online]. Available: http://www.openssl.org/

[116] NXP MPC5775K [Online]. Available: www.nxp.com/docs/en/data-sheet/MPC5775KDS.pdf

[117] NXP i.MX 6 [Online]. Available: www.nxp.com/docs/en/fact-sheet/IMX6SRSFS.pdf

[118] E. Barker and Q. Dang, "Recommendation for Key Management: Application-Specific Key Management Guidance," in NIST special publication 800-57 Part 3, Revision 1, 2015.

[119] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography," in NIST special publication 800-56A Revision 3, 2018.

[120] National Security Agency, "The Case for Elliptic Curve Cryptography." [Online]. Available: www.nsa.gov/business/programs/elliptic_curve.shtml

[121] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi, "A language-based intrusion detection approach for automotive embedded networks" in International Journal on Embedded Systems (IJES), Vol. 10, No. 8, 2018.

[122] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in Proc. of IEEE Intelligent Vehicle Symposium (IV), 2017.

[123] T. Hoppe, S. Kiltz, and J. Dittmann, “Security threats to automotive CAN networks-practical examples and selected short-term countermeasures,” in Reliability Engineering & System Safety, Vol. 96, No. 1, 2011.

[124] U. E. Larson, D. K. Nilsson, and E. Jonsson, “An approach to specification-based attack detection for in-vehicle networks,” in Proc. of IEEE Intelligent Vehicles Symposium (IV), 2008.

[125] M. Aldwairi, A. M. Abu-Dalo, and M. Jarrah, “Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework,” in EURASIP Journal on Information Security, No. 1, 2017.

[126] E. W. Myers, “An O(ND) difference algorithm and its variations,” in Algorithmica,1986.

[127] T. Hoppe, S. Kiltz, and J. Dittmann, “Applying intrusion detection to automotive IT-early insights and remaining challenges,” in Journal of Information Assurance and Security (JIAS), Vol. 4, No. 6, 2009.

[128] K. T. Cho and K. G. Shin, “Fingerprinting electronic control units for vehicle intrusion detection,” in Proc. of USENIX, 2016.

[129] X. Ying, S. U. Sagong, A. Clark, L. Bushnell, and R. Poovendran, “Shape of the Cloak: Formal analysis of clock skew-based intrusion detection system in controller area networks.” In IEEE Transactions on Information Forensics and Security (TIFS), Vol. 14, No. 9, 2019.

[130] M. K. Yoon, S. Mohan, J. Choi, and L. Sha, “Memory heat map: Anomaly detection in real-time embedded systems using memory behavior,” in Proc. of IEEE/ACM/EDAC Design Automation Conference (DAC), 2015.

[131] M. Müter, and N. Asaj, “Entropy-based anomaly detection for in-vehicle networks,” in Proc. of IEEE Intelligent Vehicles Symposium (IV), 2011.

[132] M. Müter, A. Groll, and F. C. Freiling, “A structured approach to anomaly detection for in-vehicle networks,” in Proc. of IEEE International Conference on Intelligent and Advanced System (ICIAS), 2010.

[133] A. Taylor, N. Japkowicz, and S. Leblanc, “Frequency-based anomaly detection for the automotive CAN bus,” in Proc. of World Congress on Industrial Control Systems Security (WCICSS), 2015.

[134] F. Martinelli, F. Mercaldo, V. Nardone, and A. Santone, “Car hacking identification through fuzzy logic algorithms,” in Proc. of IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2017.

[135] T. P. Vuong, G. Loukas, and D. Gan, “Performance evaluation of cyber-physical intrusion detection on a robotic vehicle,” in Proc. of IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015.

[136] M. Levi, Y. Allouche, and A. Kontorovich, “Advanced analytics for connected car cybersecurity,” in Proc. of IEEE Vehicular Technology Conference (VTC) Spring, 2018.

[137] M. J. Kang and J.W. Kang, “A novel intrusion detection method using deep neural network for in-vehicle network security,” in IEEE Proc. of Vehicular Technology Conference (VTC) Spring, 2016.

[138] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data," in IEEE Access, 2020.

[139] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon, and D. Gan, "Cloud-based cyber-physical intrusion detection for vehicles using deep learning," in IEEE Access 2018.

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

[141] M. Weber, G. Wolf, E. Sax, and B. Zimmer, "Online Detection of Anomalies in Vehicle Signals using Replicator Neural Networks," in Proc. of ESCAR USA, 2018.

[142] M. Weber, S. Klug, E. Sax, and B. Zimmer, "Embedded hybrid anomaly detection for automotive can communication," in Embedded Real Time Software and Systems (ERTS), 2018.

[143] J. Schmidhuber, "Habilitation thesis: System modeling and optimization," 1993.

[144] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in IEEE Press, 2001.

[145] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in arXiv preprint, arXiv:1406.1078, 2014.

[146] Renub Research, "Self Driving Car Market Global Forecast by Levels, Hardware, Software, and Companies," in Research and Markets – Market Research Reports, 2018.

[147] M. Hasan, S. Mohan, T. Shimizu, and H. Lu, “Securing Vehicle-to-Everything (V2X) Communication Platforms,” in IEEE Transactions on Intelligent Vehicles (TIV), Vol 5, No. 4, 2020.

[148] P. Braeckel, “Feeling Bluetooth: From a Security Perspective,” in Advances in Computers 2011.

[149] R. Verdult, F.D. Garcia, and B. Ege, “Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer,” in Proc. of USENIX, 2013.

[150] S. Acharya, Y. Dvorkin, and R. Karri, “Public Plug-in Electric Vehicles + Grid Data: Is a New Cyberattack Vector Viable?” in IEEE Transactions on Smart Grid (TSG), Vol. 11, No. 6, 2020.

[151] A. Francillon, B. Danev, and S. Capkun, “Relay attacks on passive keyless entry and start systems in modern cars,” in Proc. of Network and Distributed System Security Symposium (NDSS), 2011.

[152] R. Dastres and M. Soori, “A Review in Recent Development of Network Threats and Security Measures,” in International Journal of Information Sciences and Computer Engineering, 2021.

[153] J. Dürrwang, J. Braun, M. Rumez, R. Kriesten, and A. Pretschner, “Enhancement of Automotive Penetration Testing with Threat Analyses Results,” in SAE International Journal of Transportation Cybersecurity and Privacy, 2018.

[154] Keen Lab, “Experimental Security Assessment of BMW Cars: A Summary Report,” [Online]. Available: https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Assessment_of_BMW_Cars_by_KeenLab.pdf, 2017.

[155] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, “Remote Attacks on Automated Vehicles Sensors: Experiments on Camera and LiDAR,” in Black Hat Europe, 2015.

[156] J. Raiyn, “A survey of Cyber Attack Detection Strategies,” in International Journal of Security and Its Applications (IJSIA), Vol. 8, No. 1, 2014.

[157] D. Stabili, M. Marchetti, and M. Colajanni, “Detecting attacks to internal vehicle networks through hamming distance,” in Proc. of IEEE International Annual Conference (AEIT), 2017.

[158] H.M. Song, J. Woo, and H.K. Kim, “In-vehicle network intrusion detection using deep convolutional neural network,” in Vehicular Communications, 2020.

[159] M. O. Ezeme, Q. H. Mahmoud, and A. Azim, “Hierarchical Attention-Based Anomaly Detection Model for Embedded Operating Systems,” in Proc. of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2018.

[160] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” in IEEE Transactions on Neural Networks (TNN), Vol. 5, No. 2, 1994.

[161] S. Elsworth and S. Güttel, “Time Series Forecasting Using LSTM Networks: A Symbolic Approach,” [Online]. Available: https://arxiv.org/abs/2003.05672 2020.

[162] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” in Neural Computation, Vol. 9, No. 8, 1997.

[163] E. Sood, S. Tannert, D. Frassinelli, A. Bulling, and N. T. Vu, “Interpreting Attention Models with Human Visual Attention in Machine Reading Comprehension,” [Online]. Available: https://arxiv.org/abs/2010.06396, 2020.

[164] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” [Online]. Available: https://arxiv.org/abs/1409.0473, 2016.

[165] R. Jing, “A Self-attention Based LSTM Network for Text Classification,” in Journal of Physics: Conference Series (JPCS), Vol. 1207, No. 1, 2019.

[166] M. T. Luong, H. Pham, and C. D. Manning, “Effective Approaches to Attention-based Neural Machine Translation,” [Online]. Available: https://arxiv.org/abs/1508.04025, 2015.

[167] S. Vergura, “Bollinger Bands Based on Exponential Moving Average for Statistical Monitoring of Multi-Array Photovoltaic Systems,” in Energies, 2020.

[168] M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander, “LOF: identifying density-based local outliers,” in Proc. of ACM ACM SIGMOD International Conference on Management of Data (MOD), 2000.

[169] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “CACTI 6.0: A tool to model large caches,” HP laboratories, [Online]. Available: https://github.com/HewlettPackard/cacti, 2014.

[170] J. Dey, W. Taylor, and S. Pasricha, “VESPA: Optimizing Heterogeneous Sensor Placement and Orientation for Autonomous Vehicles,” in IEEE Consumer Electronics Magazine (CEM), Vol. 10, No. 2, 2021.

[171] C. Valasek and C. Miller, “Adventures in Automotive Networks and Control Units,” [Online]. Available: https://ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf, 2013.

[172] C. Sitawarin, A. N. Bhagoji, A. Mosenia, M. Chiang, and P. Mittal, “DARTS: Deceiving Autonomous Cars with Toxic Signs,” [Online]. Available: https://arxiv.org/abs/1802.06430.

[173] 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 Proc. of IEEE International Conference on Information Networking (ICOIN), 2016.

[174] M. Gmiden, M. H. Gmiden, and H. Trabelsi, “An intrusion detection method for securing in-vehicle CAN bus,” in Proc. of IEEE International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), 2016.

[175] H. Lee, S. H. Jeong, and H. K. Kim, “OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame,” in Proc. of IEEE Annual Conference on Privacy, Security and Trust (PST), 2017.

[176] W. Wu, Y. Huang, R. Kurachi, G. Zeng, G. Xie, R. Li, and K. Li, “Sliding window optimized information entropy analysis method for intrusion detection on in-vehicle networks,” in IEEE Access, 2018.

[177] K. T. Cho and K. G. Shin, “Viden: Attacker identification on in-vehicle networks,” in Proc. of ACM SIGSAC Conference on Computer and Communications Security (CCS), 2017.

[178] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, “LSTM-based intrusion detection system for in-vehicle can bus communications,” in IEEE Access, 2020.

[179] S. Tariq, S. Lee, and S. S. Woo, “CANTransfer: Transfer learning based intrusion detection on a controller area network using convolutional lstm network,” in Proc. of ACM Symposium on Applied Computing (SAC), 2020.

[180] Vector GL1000. [Online]. Available: https://assets.vector.com/cms/content/products/gl_logger/Docs/ GL1000_Manual_EN.pdf.

[181] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme recognition using time-delay neural networks,” in IEEE Transactions on Acoustics, Speech, and Signal Processing (TASSP), Vol. 37, No. 3, 1989.

[182] S. Bai, J. Z. Kolter, and V. Koltun, “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling,” [Online]. Available: https://arxiv.org/abs/1803.01271, 2018.

[183] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” in Proc. Neural Information Processing Systems (NIPS), 2017.

[184] L. Nie, Z. Ning, X. Wang, X. Hu, Y. Li, and J. Cheng, “Datadriven intrusion detection for intelligent Internet of vehicles: A deep convolutional neural network-based method,” in IEEE Transactions on Network Science and Engineering (TNSE), Vol. 7, No. 4, April 2020.

[185] S. Chen, M. Kahla, R. Jia, and G. J. Qi, “Knowledge-Enriched Distributional Model Inversion Attacks,” in Proc. of IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[186] M. Aloqaily, S. Otoum, I. Al Ridhawi, and Y. Jararweh, “An intrusion detection system for connected vehicles in smart cities,” in Ad Hoc Networks 90, 2019.

[187] S. Boddupalli, A. S. Rao, and S. Ray, “Resilient Cooperative Adaptive Cruise Control for Autonomous Vehicles Using Machine Learning,” in arXiv preprint, 2021.

[188] J. Ashraf, A. D. Bakhshi, N. Moustafa, H. Khurshid, A. Javed, and A. Beheshti, “Novel deep learning-enabled lstm autoencoder architecture for discovering anomalous events from

intelligent transportation systems," in IEEE Transactions on Intelligent Transportation Systems (TITS), 2020.

[189] Z. Hassan, A. Mehmood, C. Maple, M. A. Khan, and A. Aldegheishem, "Intelligent Detection of Black Hole Attacks for Secure Communication in Autonomous and Connected Vehicles," in IEEE Access, Vol. 8, 2020.

[190] S. Chang, Y. Qi, H. Zhu, J. Zhao, and X. Shen, "Footprint: detecting Sybil attacks in urban vehicular networks," in IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 23, No. 6, 2011.

[191] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," in arXiv:1812.04202, 2018.

[192] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log Anomaly Detection via BERT," in arXiv:2103.04475, 2021.

[193] A. Geiger, D. Liu, S. Alnegheimish, A. Cuesta-Infante, and K. Veeramachaneni, "TadGAN: Time series anomaly detection using generative adversarial networks," in IEEE International Conference on Big Data (Big Data), 2020.

[194] T. Sahin, R. Khalili, M. Boban, and A. Wolisz, "VRLS: A Unified Reinforcement Learning Scheduler for Vehicle-to-Vehicle Communications," in Proc. of IEEE Connected and Automated Vehicles Symposium (CAVS), 2019.

[195] R. Atallah, C. Assi, and M. Khabbaz, "Deep reinforcement learning-based scheduling for roadside communication networks," in Proc. of IEEE International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2017.

[196] C. Zhai, F. Luo and Y. Liu, “A Novel Predictive Energy Management Strategy for Electric Vehicles Based on Velocity Prediction,” in IEEE Transactions on Vehicular Technology, Vol. 69, No. 11, November 2020.

[197] K. Liu, Z. Asher, X. Gong, M. Huang, and I Kolmanovsky, “Vehicle Velocity Prediction and Energy Management Strategy Part 1: Deterministic and Stochastic Vehicle Velocity Prediction Using Machine Learning,” in SAE World Congress Experience (WCX), 2019.

[198] D. Baker, Z. Asher, and T. Bradley, “Investigation of vehicle speed prediction from neural network fit of real world driving data for improved engine on/off control of the EcoCAR3 hybrid Camaro,” SAE Technical Paper, 2017.

[199] D. Baker, Z. D. Asher, and T. Bradley, “V2V communication based real-world velocity predictions for improved HEV fuel economy,” SAE Technical Paper, 2018.

[200] S. A. Seshia, A. Desai, T. Dreossi, D.J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue, “Formal specification for deep neural networks,” in Proc. of Springer International Symposium on Automated Technology for Verification and Analysis, 2018.