



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Lightweight Multi-Method Intrusion Detection for Automotive CAN Bus Networks

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE & ENGINEERING - INGEGNERIA INFOR-
MATICA

Author: **Edoardo Galtieri**

Student ID: 222455

Advisor: Prof. Stefano Longari

Academic Year: 2024-2025

Abstract

The Controller Area Network (CAN) bus, developed in the 1980s as a reliable and cost-effective communication protocol for automotive Electronic Control Units (ECUs), has become the standard for in-vehicle networks. While the protocol was chosen for its real-time performance and fault tolerance, it was designed without security mechanisms such as message authentication or encryption, as vehicles were considered isolated systems. Modern vehicles, however, are increasingly connected through external interfaces including Bluetooth, Wi-Fi, and cellular networks, transforming this security gap into a critical vulnerability. Recent research has demonstrated the feasibility of remote attacks capable of manipulating safety-critical systems including brakes, steering, and engine control, exposing the urgent need for practical defensive solutions. Intrusion Detection Systems (IDSs) have emerged as one of the most useful approaches for securing CAN networks. However, existing solutions face an important tradeoff: machine learning methods achieve high detection rates but require hardware incompatible with automotive ECUs, while lightweight approaches fit resource constraints but provide incomplete attack coverage. This thesis explores the design and validation of a multi-method lightweight IDS pipeline that integrates Hamming distance-based payload analysis, residual timing-based monitoring, and watchdog timer supervision. We investigate two architectural configurations—serial two-stage filtering and parallel OR-logic configuration—to determine optimal strategies for combining detection methods. The system is evaluated on the X-CANIDS benchmark dataset across different attack types and validated through embedded implementation on simulated STM32 microcontroller hardware to assess practical deployability. By comparing performance against state-of-the-art machine learning and lightweight IDS approaches, this work demonstrates whether the combination of complementary detection methods can bridge the gap between detection effectiveness and deployment feasibility.

Keywords: CAN bus security, intrusion detection systems(IDSs), automotive cybersecurity, lightweight detection, real-time systems.

Abstract in lingua italiana

Il Controller Area Network (CAN) bus, sviluppato negli anni '80 come protocollo broadcast di comunicazione affidabile ed economico per le unità di controllo elettroniche (ECU) automobilistiche, è diventato lo standard per le reti veicolari interne. Sebbene il protocollo sia stato scelto per le sue prestazioni in tempo reale e per la sua tolleranza ai guasti, è stato progettato senza meccanismi di sicurezza come autenticazione o cifratura dei messaggi, poiché i veicoli erano considerati sistemi isolati. I veicoli moderni, tuttavia, sono sempre più connessi attraverso interfacce esterne, inclusi Bluetooth, Wi-Fi e reti cellulari, che trasformano questo gap di sicurezza in una vulnerabilità critica. Recenti ricerche hanno dimostrato la fattibilità di attacchi remoti, capaci di manipolare sistemi critici per la sicurezza, inclusi freni, sterzo e controllo motore, evidenziando l'urgente necessità di soluzioni di difesa pratiche. I sistemi di rilevamento delle intrusioni (IDS) sono considerati uno degli approcci più efficaci per la protezione delle reti CAN. Tuttavia, le soluzioni esistenti presentano un compromesso fondamentale: i metodi basati su machine learning raggiungono alti tassi di rilevamento ma richiedono hardware incompatibile con le ECU all'interno delle automobili odierne, mentre gli approcci più leggeri rispettano i vincoli di risorse ma forniscono una copertura incompleta degli attacchi.

Questa tesi esplora la progettazione e la validazione di una pipeline di IDSs multi-metodo che integra analisi del payload basata sulla distanza di Hamming, monitoraggio sul timing basato sui residui e supervisione tramite watchdog timer. In questo studio vengono investigate due configurazioni architetturali: un filtro sequenziale a due stadi e una configurazione parallela con logica OR. Il sistema viene valutato sul dataset di benchmark X-CANIDS attraverso diversi tipi di attacco e validato mediante implementazione embedded su hardware microcontrollore STM32 simulato, per valutarne la fattibilità pratica di deployment. Confrontando le prestazioni con approcci IDS allo stato dell'arte, basati su machine learning e a basso costo computazionale, questo lavoro dimostra che la combinazione di metodi di rilevamento complementari possa colmare il divario tra efficacia di rilevamento e fattibilità di deployment.

Parole chiave: sicurezza CAN bus, sistemi di rilevamento intrusioni (IDS), sicurezza automobilistica, rilevamento a basso costo computazionale, sistemi real-time.

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
1 Introduction	1
1.1 Domain Context	1
1.2 Problem and Motivation	2
1.3 Thesis Structure	3
2 Background on CAN Bus	5
2.1 History of CAN Protocol	5
2.2 CAN protocol implementation	6
2.2.1 Physical Layer	6
2.2.2 Data Link Layer	7
2.3 Node Structure	7
2.4 CAN Frames and Arbitration	9
2.4.1 Data Frame Format	9
2.5 Error handling in CAN Bus	10
2.5.1 Fault Confinement in CAN Bus	11
2.6 Intrinsic Vulnerabilities	12
3 CAN Bus Security	15
3.1 Taxonomy of Attacks	15
3.1.1 Fabrication	16
3.1.2 Fuzzing	16
3.1.3 Replay	16
3.1.4 Denial-of-Service (DoS)	16

3.1.5	Masquerade	17
3.2	Defensive Measures	17
3.2.1	Intrusion Detection System (IDS)	17
3.2.2	Intrusion Prevention System (IPS)	18
3.2.3	Secure Onboard Communication (SecOC)	19
3.2.4	Network Segmentation	19
3.3	Limitations and Problems	19
4	State of the Art: IDS in Automotive	21
4.1	Cyberphysical	21
4.1.1	Overview	21
4.1.2	Representative Works	21
4.1.3	Strengths and Limitations	22
4.2	Entropy-Based	23
4.2.1	Overview	23
4.2.2	Representative Works	23
4.2.3	Strengths and Limitations	24
4.3	Graph-based	24
4.3.1	Overview	24
4.3.2	Representative Works	25
4.3.3	Strengths and Limitations	25
4.4	HW-Physical-Based	26
4.4.1	Overview	26
4.4.2	Representative Works	26
4.4.3	Strengths and Limitations	27
4.5	Message Frequency	28
4.5.1	Overview	28
4.5.2	Representative Works	28
4.5.3	Strengths and Limitations	29
4.6	Sequence-Based	29
4.6.1	Overview	29
4.6.2	Representative Works	30
4.6.3	Strengths and Limitations	30
4.7	Limitations of IDSs	31
5	Proposed Pipeline Approach	33
5.1	Motivation	33
5.2	Component Selection Methodology	34

5.3	Pipeline Architecture	36
5.3.1	Hamming Distance	36
5.3.2	ErrIDS	38
5.3.3	Watchdog Suspension Check	41
5.4	Configuration Modes: Series vs. Parallel	42
6	Experimental Setup	45
6.1	Detection Metrics	45
6.1.1	Measurement Approaches by Attack Type	46
6.1.2	The Collateral Effect	46
6.2	Testing Dataset	48
6.2.1	X-CANIDS	48
6.3	Experimental Setup	51
6.3.1	Python Simulation Environment	51
6.3.2	STM32 Environment	52
7	Testing and Results	53
7.1	Test 1: Serial Configuration	53
7.1.1	Test Description and Results	53
7.1.2	Discussion	59
7.2	Test 2: Parallel Configuration	62
7.2.1	Test Description and results	62
7.2.2	Discussion	69
7.3	Test 3: Wokwi STM32 Tests	72
7.3.1	Implementation Details	72
7.3.2	Test Methodology and Results	73
7.3.3	Discussion	75
7.4	Comparison with State-of-the-Art Systems	76
7.4.1	Performance Analysis by System	77
7.4.2	Overall Assessment	80
8	Limitations of the approach	83
8.1	Primary Limitations	83
9	Conclusions and Future Work	85
9.1	Future Directions	86

Bibliography	87
List of Figures	93
List of Tables	95
List of Acronyms	97

1 | Introduction

Modern vehicles have evolved from purely mechanical systems into sophisticated cyber-physical platforms, integrating hundreds of Electronic Control Units (ECUs) that manage critical functions ranging from engine performance and braking systems to infotainment and driver assistance features. Due to this transformation vehicles are no longer isolated machines but connected systems that communicate internally through in-vehicle networks (IVN) and externally through wireless interfaces such as Bluetooth, Wi-Fi. While this enables advanced safety features, autonomous driving capabilities, and enhanced user experiences, at the same time it exposes vehicles to cybersecurity threats that were not possible before.

1.1. Domain Context

The Controller Area Network (CAN) protocol is the standard for in-vehicle communication due to its reliability, real-time performance, and cost-effectiveness. CAN enables ECUs to exchange messages over a shared bus without requiring point-to-point wiring between every component, reducing vehicle wiring complexity and manufacturing costs. However, the protocol was designed without security as a design requirement. CAN messages lack authentication mechanisms, encryption, or sender verification. Any node on the bus can transmit messages claiming to originate from any ECU, and all nodes must trust received messages without validation. This security gap has become critical as researchers have demonstrated numerous attack vectors against modern vehicles. Koscher et al. pioneered automotive security research by showing that physical access to the OBD-II diagnostic port allows complete control over safety-critical systems including brakes, steering, and engine [19]. Subsequent work by Miller and Valasek [26], and others proved that remote exploitation is feasible through compromised telematics units, Bluetooth interfaces, and cellular connections, enabling attackers to manipulate vehicle functions without physical proximity. These demonstrations revealed that vulnerabilities in CAN protocol design create genuine safety risks for drivers and passengers.

1.2. Problem and Motivation

Automotive security has become an increasing concern over the past few years due to the increased attack surfaces given from more and more components. Since preventive mechanisms alone cannot guarantee protection, Intrusion Detection Systems (IDSs) emerged as one of the most practical approach for securing CAN networks. The studies about the context contain various IDS methodologies: e.g. voltage-based fingerprinting that identifies ECUs through electrical signal characteristics, frequency-based analysis that detects timing anomalies in periodic message transmission, and machine learning approaches that model normal traffic patterns to identify deviations.

Especially, Machine learning-based IDSs have demonstrated high detection rates, often approaching or exceeding 99% true positive rates across different attack scenarios. However, this performance comes at a cost that goes against automotive industry constraints. Deep learning models require costly and heavy hardware for inference and possibly processing latencies measured in hundreds of milliseconds. Automotive manufacturers cannot always cope with the request of dedicated security processors or external hardware for IDS deployment. Furthermore, real-time safety requirements demand that attack detection occur within milliseconds to enable countermeasures before an attacker is able to perform some actions, which is incompatible with the computational overhead of complex neural networks.

Existing lightweight IDSs operate within ECU resource constraints but cannot obtain the same performance as ML algorithms, or they do not have enough attack coverage. Rule-based systems detect specific known patterns but miss zero-day attacks. Timing-based methods identify inter-arrival anomalies but cannot detect attacks where the payload is valid. Payload-based approaches using checksums or different distances between messages catch message manipulation but fail against timing-preserving attacks or message suspension.

To the best of our knowledge, no prior work has systematically demonstrated that combining multiple lightweight detection methods—each individually limited in coverage—can achieve detection rates competitive with machine learning systems while operating within the latency, memory, and computational constraints of standard automotive ECUs. Existing research either evaluates single detection methods in isolation or combines methods without validating deployment feasibility on resource-constrained hardware. This work investigates whether a lightweight IDS pipeline can achieve comprehensive attack coverage comparable to state-of-the-art ML systems while maintaining sub-millisecond detection latency and minimal resource footprint suitable for production ECU deployment

For these reasons this thesis addresses three specific objectives: (1) demonstrating that combining complementary lightweight detection methods achieves competitive detection rates without machine learning, (2) determining the optimal architectural configuration (parallel versus serial) for maximizing attack coverage while minimizing computational overhead, and (3) validating practical deployability through performance comparison against state-of-the-art systems and embedded implementation on resource-constrained hardware representative of automotive ECUs.

1.3. Thesis Structure

The thesis is organized into nine chapters. Chapter 2 provides the background on the CAN protocol, including message structure, arbitration mechanisms, error handling, and the security vulnerabilities that enable automotive attacks. Chapter 3 describes common attack types in this context (fabrication, fuzzing, masquerade, replay, suspension) and existing defensive approaches including cryptographic solutions, network segmentation, and intrusion detection methodologies while also mentioning their limitations.

Chapter 4 presents a focused analysis of possible IDS approaches relevant to our work. For each category, we discuss representative systems, their detection principles, strengths, and limitations that motivate our pipeline approach.

Chapter 5 defines our detection pipeline architecture, explaining the selection criteria for component methods (Hamming distance and ErrIDS), the idea behind both serial and parallel configurations, and the theoretical advantages and tradeoffs of each architectural choice. This chapter establishes the hypothesis that parallel configuration should provide more coverage and better performance at the cost of accumulated false positives from both methods.

Chapters 6 and 7 are the core of the thesis. The former describes our evaluation methodology, including the X-CANIDS dataset characteristics, detection metrics (TPR, FPR, precision, F1-score), threshold determination procedures, and experimental setup for both Python-based evaluation and embedded implementation. The latter presents comprehensive results across five attack types, comparing serial versus parallel configurations and analyzing performance by attack intensity, CAN ID characteristics, and attack-specific detection challenges. We also position our work within the current automotive IDS landscape through comparison against four state-of-the-art systems evaluated on the same X-CANIDS dataset.

Chapter 8 acknowledges the limitations of our approach, including dataset-specific evaluation scope, vulnerability to adaptive adversaries who understand detection thresholds, vehicle-specific training requirements, and the gap between simulation-based validation

and production deployment. We discuss how these limitations affect our results and identify directions for strengthening the system’s real-world applicability.

Finally, Chapter 9 synthesizes the thesis contributions and explores future research directions, including multi-dataset validation and production-grade implementation addressing safety certification requirements.

2 | Background on CAN Bus

The Controller Area Network (CAN) has become the standard in modern vehicles since its introduction by Robert Bosch in 1986 and its standardization in 2012 (ISO 11898-1). This protocol was implemented with broadcast packets in mind, which means that all connected nodes receive every transmission.[24, 27] This mechanism enables efficient communication between the embedded Electronic Control Units (ECUs) in modern cars. In contemporary architectures all aspects of the car are managed by these ECUs, typically using two split segments based on performance requirements. Critical and safety messages are carried on a high-speed bus (about 500Kbps) while non critical features (such as A/C or windows) utilize a lower speed bus (about 125Kbps).[35]

2.1. History of CAN Protocol

The development of CAN began in 1983 at Robert Bosch GmbH, driven by the need for a serial bus system that could support the increasing complexity of vehicle electronics better than existing protocols. The system was officially introduced at the 1986 SAE Congress in Detroit, featuring a multi-master architecture with non-destructive bitwise arbitration to ensure high-priority message delivery [6]. Semiconductor support followed rapidly; by mid-1987, Intel had released the first dedicated CAN controller (82526), followed shortly by Philips Semiconductors with the 82C200 [5]. The protocol saw its first series production application in 1991 within the Mercedes-Benz W140 S-Class and was subsequently ratified as the ISO 11898 standard in 1993 [6].

Following its standardization, CAN adoption expanded significantly beyond the automotive sector into industrial automation, leading to the development of standardized higher-layer protocols to manage network communication. In Europe, the CAN in Automation (CiA) group released the CANopen profile in 1995, which became the standard for embedded networks in medical, maritime, and off-highway applications [5]. Simultaneously in the United States, the Open DeviceNet Vendor Association (ODVA) popularized DeviceNet, which became a dominant solution for factory automation [6]. These standardized application layers effectively made proprietary solutions obsolete, cementing CAN's

position as a versatile fieldbus across multiple industries.

Despite its maturity, the protocol has continued to evolve to meet the bandwidth requirements of software-defined vehicles. In 2012, Bosch introduced CAN Flexible Data-Rate (FD) to address the limitations of the original standard, enabling higher bit rates and data payloads of up to 64 bytes—an essential upgrade for reducing ECU software update times [6]. This evolution has continued with the development of the third-generation CAN XL protocol starting in 2018, which supports payloads up to 2048 bytes to ensure the technology remains relevant for future IP-based vehicle architectures [6].

2.2. CAN protocol implementation

2.2.1. Physical Layer

The physical layer of the CAN protocol operates on a dual-wire twisted pair interface consisting of two lines designated as CAN High (CAN_H) and CAN Low (CAN_L). Communication is achieved through differential signaling, where the logic state is determined not by the absolute voltage of a single wire, but by the voltage difference between the two lines.

This signaling method defines two logical states:

- **Recessive (Logic 1)** : This is the default idle state of the bus. Terminating resistors pull both lines to a common nominal voltage (generally 2.5V), resulting in a voltage difference near zero.
- **Dominant (Logic 0)**: To transmit data, a node actively drives the voltage of CAN_H to 3.5V and CAN_L to 1.5V. This creates a significant voltage differential which overrides the recessive state [20].

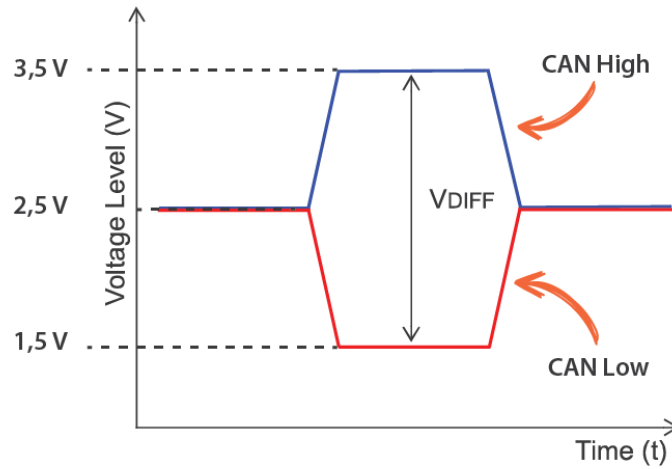


Figure 2.1: Voltage difference between CAN_H and CAN_L.

The primary advantage of this differential approach is its inherent immunity to electromagnetic interference. Since external electrical noise typically affects both wires simultaneously (a phenomenon known as common-mode noise), the voltage difference between the lines remains preserved, ensuring reliable communication even in the harsh electrical environment of a vehicle [20].

2.2.2. Data Link Layer

The Data Link Layer of the CAN Bus is the one operating on frames, arbitration, error handling and acknowledgement of messages. It ensures the frames are properly structured and correctly transmitted on the bus. All these aspects will be thoroughly analyzed in the following sections.

2.3. Node Structure

As specified in section 2 the CAN protocol was created with broadcast communication in mind, which means that every ECU receives every packet and, upon interpretation, perform some specific action.

Each ECU is made up of three main components:

- **Microcontroller Unit (MCU):** It is the central core of all the ECU operations. It serves the purpose of taking decisions for the operational integrity of the ECUs and their functioning. It executes application logic deciding which messages require which resources and what can be broadcasted to the rest of the nodes. It is also

dedicated to the control of peripherals such as sensors and actuators.

- CAN Controller:** The Controller is the component implementing the Data Link Layer(Layer 2) of the OSI model within the ECUs. It manages the functions inherent to this layer, including error handling, message arbitration, and frame formatting acting as an intermediary between the host Microcontroller and the physical Transceiver. Its primary function is the translation and encapsulation of the message from the Microcontroller into digital sequences of bits to be sent to the Transceiver through the CAN TX channel. On the contrary when a sequence comes from the Transceiver on the CAN RX, it has to be decoded into valid message format and sent to the Microcontroller. Another critical feature of the Controller is its hardware acceptance filtering, which discards irrelevant traffic to preserve and maintain useful MCU resources.
- CAN Transceiver:** The Transceiver serves as the physical interface between the ECU and the network, implementing the Physical Layer (Layer 1) of the OSI model. Its primary function is to convert the digital logic signals from the Controller into the differential voltage levels required for transmission on the shared bus. On the other hand, when a message arrives on the bus, the Transceiver has to take it and translate the physical analog voltage signal back into a sequence of bits forwarded to the controller for interpretation.

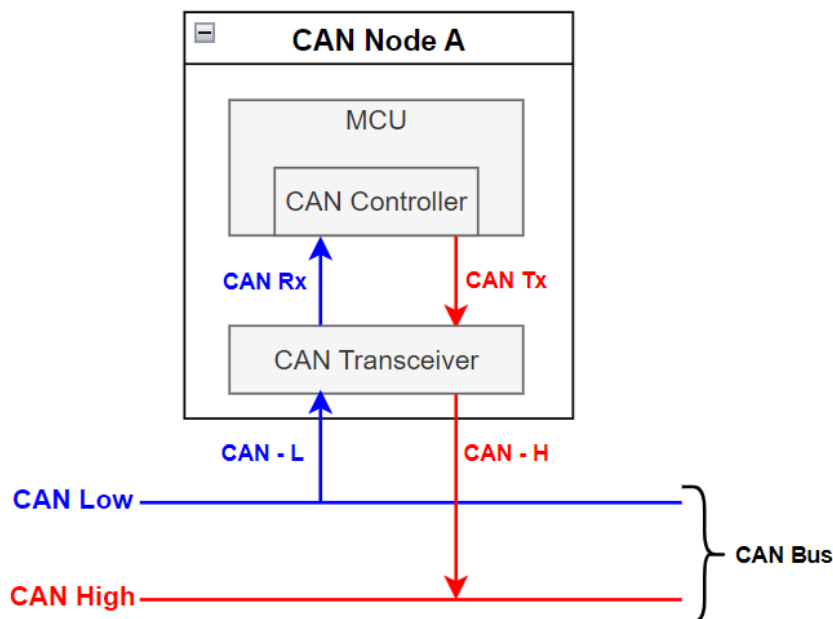


Figure 2.2: ECU's Node internal structure.

2.4. CAN Frames and Arbitration

Data transmission within the CAN protocol is organized into structured packets known as frames. The standard defines four distinct frame types:

1. **Data Frame:** The most common frame type, used to carry actual data between network nodes. Its specific structure is detailed in the following section.
2. **Remote Frame:** Used by a node to request the transmission of a specific Data Frame with the same identifier. Structurally identical to a Data Frame except that the RTR (Remote Transmission Request) bit is recessive and the Data Field is empty (zero bytes).
3. **Overload Frame:** Transmitted by a node to inject a delay between frames if it is not ready to process the next message (e.g., due to buffer overflows). It consists of an Overload Flag and an Overload Delimiter.
4. **Error Frame:** When a node detects an error in a receiving or sending packet, it informs all the other nodes of the discovered error by sending on the bus an error frame composed of six consecutive bits of the same polarity.

2.4.1. Data Frame Format

Data frames are composed of specific fields, each serving a distinct function in the transmission protocol:

- **Start of Frame (SOF):** A single dominant bit that marks the beginning of a message, allowing all nodes to synchronize their internal clock to the transmitter.
- **Arbitration Field:** This section determines the priority of the message and is divided into:
 1. **Identifier (ID):** Consisting of 11 bits (Standard Frame) or 29 bits (Extended Frame), which identifies the message content.
 2. **Remote Transmission Request (RTR):** Distinguishes a Data Frame (Dominant) from a Remote Frame (Recessive).
- **Control Field:** Composed of two reserved bits and 4 other bits to indicate Data Length Code (DLC)
- **Data Field:** The actual payload of the frame, containing the application data. It can range from 0 to 8 bytes in length.

- **Cyclic Redundancy Check (CRC):** sixteen bits that function as the verification process with which the nodes check whether a packet is valid or not. In Particular 15 bits are dedicated to the checksum while the other is reserved as delimiter of the section
- **Acknowledgement (ACK):** two bits(actual ACK and ACK delimiter) set to recessive by the sending node. Upon positive reception other ECUs will set it to dominant.
- **End of Frame (EOF):** Marks the conclusion of the data packet with a sequence of seven consecutive recessive bits.

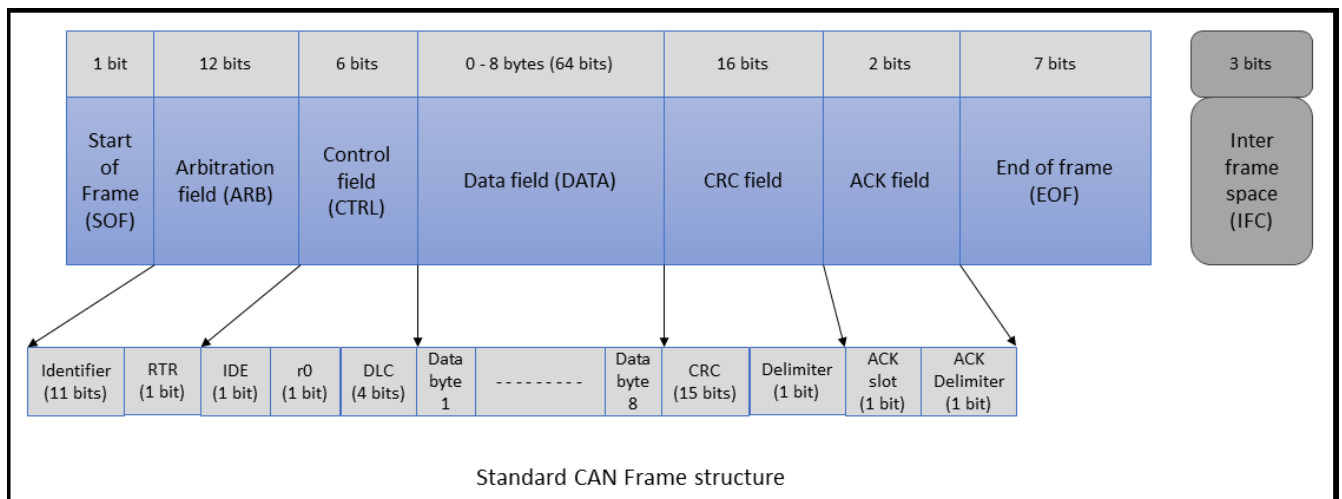


Figure 2.3: Data Frame format.

2.5. Error handling in CAN Bus

Error handling in CAN Bus is performed by all the ECUs continuously listening on the channel. When an error is detected on a frame, an Error Frame is transmitted. This frame is constructed as follows:

- **Error Flag:** six consecutive bits of the same polarity. By sending six identical bits, the node intentionally violates the bit-stuffing rule¹.
- **Error Delimiter:** It consists of eight recessive bits, providing a stall that allows the bus to return to an idle state before any node attempts to re-transmit.

¹The protocol dictates a strict "5-bit rule": whenever a transmitting node sends five consecutive bits of the same logic level (whether dominant or recessive), it must automatically insert a sixth bit of the opposite level into the stream[1]

2.5.1. Fault Confinement in CAN Bus

A fault confinement mechanism is described in the CAN protocol to prevent a malfunctioning ECU from permanently damaging the transmission of others. It is a finite state machine that track changes in value of two error counters: **Transmission Error Counter (TEC)** and **Receive Error Counter (REC)**, and change its state accordingly.

Table 2.1: Increment and decrement rules for TEC and REC

Event	TEC Change	REC Change
Successful Transmission	-1 (if > 0)	No change
Successful Reception	No change	-1 (if $1 \leq \text{REC} \leq 127$)
Transmitter Detects Error	+8	No change
Receiver Detects Error	No change	+1
Severe Receiver Error	No change	+8 (if dominant bit detected after error flag)

Error Handling Finite State Machine

1. **Error Active:** It's the default state and the start of the state machine. When TEC and REC are less than 128 the ECU is in this situation. Errors are communicated by sending 6 dominant bit on the channel normally.
2. **Error Passive:** it is the middle stage of the machine, created to prevent possible malfunctioning ECUs to disrupt the other's transmissions. When one between REC or TEC is greater than 127 the ECU switch state into the Error Passive which means errors can only be signaled by 6 recessive bits(instead of dominant). Also, each node has to wait exactly 8 bits(Suspend Transmission Time) after a transmission to try and send another message.
3. **Bus-Off:** When TEC exceeds the limit of 255, the ECU is effectively disconnected from the bus and can only re-enter after 128 occurrences of 11 consecutive recessive bits.

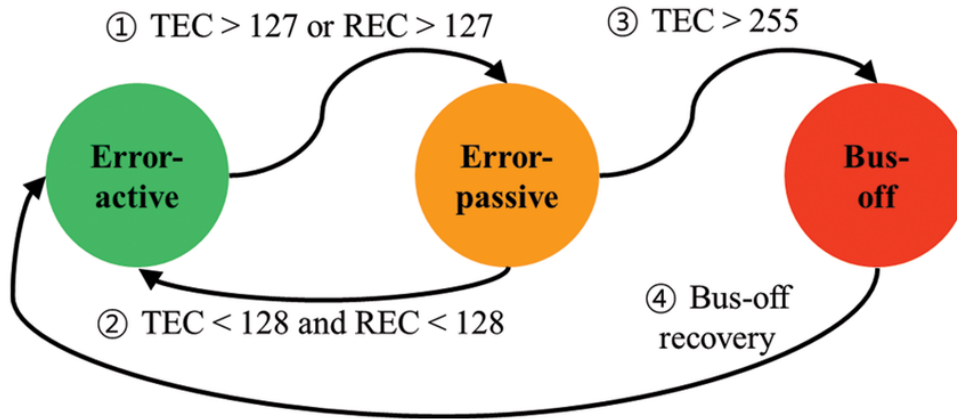


Figure 2.4: Error Handling Finite State Machine[22]

2.6. Intrinsic Vulnerabilities

The CAN protocol was originally designed for closed, trusted environments with limited amounts of nodes, where reliability and real-time performance were the priority. Consequently, security was not a primary design objective. This has resulted in several intrinsic vulnerabilities when faced with the challenge of having to manage dozens of ECUs[8].

1. **Lack of Authentication:** Since it is a broadcast protocol, any connected node can transmit and receive, allowing any unauthorized node to join the network and participate in the communication [8]. This means that the protocol cannot distinguish between a malicious node and a valid one.
2. **Lack of Encryption:** Due to the limited computational resources of ECUs, researchers could not implement complex cryptographic algorithms to provide integrity and privacy; this means that any attacker could potentially eavesdrop on communications or inject malicious data on the bus in plaintext.
3. **Lack of Network Segmentation:** The ECUs are all connected on a single shared channel that was not created with compartmentalization in mind, so an attacker could gain access to the entire network if able to gain physical access to the bus.
4. **Vulnerability to Denial of Service (DoS):** The protocol is inherently susceptible to DoS attacks due to its arbitration logic (Wired-AND). Because of how arbitration works in CAN Bus, higher priority nodes sending dominant bits (logic 0) are always allowed to speak first. If an attacker on a controlled node exploits this mechanism and continuously sends dominant bits (effectively monopolizing the bus), the other nodes are blocked from communication because of the priority mechanism which

results in the disruption of the system. [8, 9]

3 | CAN Bus Security

As mentioned in the previous chapter, CAN Bus suffers from some inherent vulnerabilities that an attacker may exploit to gain control of ECUs or send malicious messages on the channel. In this chapter we will analyze several types of attacks and the respective countermeasures to prevent or detect such attacks. Before the analysis a brief classification of potential adversary models:

- Impersonation: The attacker injects messages on the bus impersonating as the victim node, which requires access to the physical channel.
- Masquerading: The attacker disrupts the communication with a Bus-Off or targeted injection attack (these will be analyzed in the following section) and assumes the victim's identity but with another physical location. It requires the knowledge of the target ECU's parameters and possibly remote access to it.
- Compromised ECU: The attacker has full control over the victim ECU and can send valid messages on its behalf.

A well-known demonstration is a 2015 study conducted by some researchers, Miller and Valasek, that exploited the infotainment system of a Jeep Cherokee and gained access to the CAN bus, in particular to the steering and braking, which brought 1.4M vehicles to be retired from the market [26].

3.1. Taxonomy of Attacks

Attacks are the specific actions that a malicious user may perform to disrupt communication or gain control of the system. In this chapter, we analyze the main types of attacks commonly observed in In-Vehicle Networks (IVN) and discuss those used to evaluate our system (summarized in Table 3.1) as we will see in later chapters.

3.1.1. Fabrication

In message fabrication attacks the attackers inject messages with valid CAN ID and data fields onto the bus. These messages are indistinguishable from legitimate traffic except for their payload content which may contain malicious instructions that appear authentic to receiving ECUs. [16]

3.1.2. Fuzzing

Fuzzing is a technique utilized in cybersecurity discover possible system vulnerabilities. It consists of packets sent on the channel, with both ID and payload being completely randomly generated. In the automotive CAN context it is exploited as an attack vector to cause malfunctions and disruptive behavior in ECUs.

3.1.3. Replay

In message replay attacks, valid CAN packets captured while the attacker was listening on the channel are replayed exactly as recorded without any alteration. Replay attacks can disrupt normal operations or trigger unintended ECU behaviors. [16]

3.1.4. Denial-of-Service (DoS)

In Denial-of-Service (DoS) attacks, the bus is flooded with packets to prevent all legitimate nodes from communicating on the channel. DoS attacks can be either targeted or complete. In a complete DoS, the attacker continuously injects dominant bits to prevent any communication from being performed by any ECU, while in a targeted DoS, the attacker injects high-priority messages to starve specific lower-priority IDs.[11]

Suspension

A suspension attack is a specific type of targeted Denial-of-Service attacks in which a targeted ECU is prevented from communicating on the bus. One common method of performing such an attack is:

- **Bus-Off:** During a Bus-Off attack an attacker continuously injects a dominant bit during transmission while the targeted node is sending a recessive one creating bit errors. This will result in the node noticing this error on the bus and incrementing the TEC. Once the counter has reached its limit of 255, the ECU will enter the Bus-Off state, ceasing all communications, effectively removing it from the network.

[11]

3.1.5. Masquerade

Masquerade attacks are a combination of the fabrication and suspension attack types. The attacker first suspends the target ECU (e.g., via Bus-Off attack), then injects messages with the same CAN ID, effectively impersonating the legitimate sender. The forged messages contain malicious payloads designed to manipulate vehicle behavior. Advanced masquerade attacks may also emulate the target ECU's transmission timing (clock skew) to evade detection by timing-based intrusion detection systems.[14, 16]

As demonstrated by Sagong et al in 2018, masquerade attackers can also perform cloaking attacks, in which an adversary adjusts message timing and modify its clock to match the targeted ECU's clock skew to avoid detection.[29]

Table 3.1: Summary of CAN Attack Types

Attack Type	ID Validity	Payload Validity	Detection Difficulty
Fabrication	Valid	Valid (but malicious)	High
Fuzzing	Random	Random	Low
Replay	Valid (captured)	Valid (captured)	Medium
DoS	Any/Valid	Any	Low
Suspension	High-priority/Error	N/A	Medium
Masquerade	Valid (spoofed)	Valid (but malicious)	Very High

3.2. Defensive Measures

Just as attacks and vulnerabilities exists, researchers and engineers have developed numerous countermeasures to defend against such threats. Among these defensive measures we examine: (1) dedicated systems such as Intrusion Detection Systems (IDS) and Intrusion Prevention System (IPS), (2) cryptographic algorithms to ensure authentication such as SecOC and finally (3) network segmentation.

3.2.1. Intrusion Detection System (IDS)

These systems serve as a critical security layer in cybersecurity contexts. Their primary function is focused on detection rather than active prevention. They continuously monitor the network to catch and alert for any malicious activity that may happen on the bus [13].

The need for these systems came into play when researchers realized that cryptography alone cannot detect when an attacker has already compromised a legitimate ECU. IDSs can be categorized as passive or active. Passive IDSs monitor and alert administrators to detected threats, while active IDSs (also known as Intrusion Prevention Systems) automatically respond to threats by blocking or mitigating attacks. These systems function through sensors deployed either as standalone network monitors or integrated directly into individual ECUs [24].

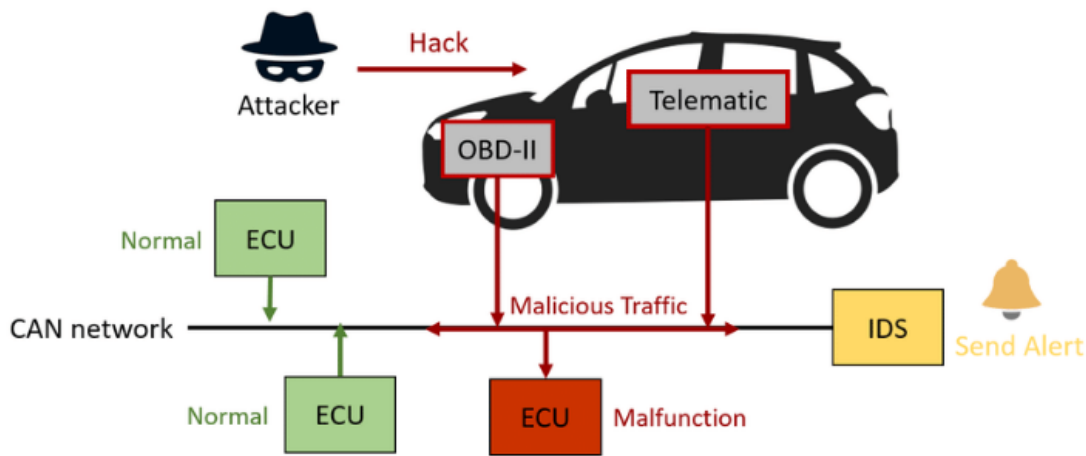


Figure 3.1: Typical IDS deployment in an automotive CAN network

[37]

3.2.2. Intrusion Prevention System (IPS)

While IDSs focus solely on detection, IPSs move beyond visibility to active intervention. IPSs are an active security control designed to neutralize threats in real-time before they can affect the vehicle's behavior. This is accomplished by invalidating or discarding identified malicious messages in real-time. Other functions include rule-based message filtering (similar to firewall functionality) and the ability to isolate compromised nodes by interacting with the physical layer through voltage manipulation.

1. **Dominant Bit Injection (Error Flagging):** As described in the previous chapter the CAN Bus protocol relies on two voltage states to communicate, recessive and dominant (logical 1 and 0 respectively). To force a malicious frame to be recognized as an error by all other nodes, the IPS injects a stream of 0s to violate the bit stuffing rule, forcing every node to recognize the error and discard it.

2. **Forced Bus-Off State:** To permanently isolate a malfunctioning or compromised ECU, the IPS can repeat the above-mentioned process for a specific target until its TEC exceeds 255, leaving the node in a bus-off state and effectively disconnecting it from the network.

3.2.3. Secure Onboard Communication (SecOC)

It is a standard defined by AUTomotive Open System ARchitecture (AUTOSAR) with the sole purpose of adding authenticity checking to messages. It is based on a shared Advanced Encryption Standard (AES) key that is known to all devices on the vehicle. An external attacker should not be able to communicate with the rest of the network without any knowledge of the AES key. This mechanism has the additional function of protecting the user in case of third-party device installations. While message authentication is not a new concept, early implementations involved proprietary algorithms that were difficult to standardize. AUTOSAR defined SecOC as an improvement over these checksum algorithms employing the AES CMAC for cryptographic message authentication. However, SecOC introduces significant trade-offs and limitations. First, the computational complexity of cryptographic algorithms may impact the real-time computation of the ECUs. Second, the limited space on the payload of the frame (8 byte) must be addressed. It is up to the implementers to choose how much of this payload capacity can be sacrificed in favor of the authentication tag [2].

3.2.4. Network Segmentation

Network segmentation is the practice of dividing a network into separate domains or zones based on their function and security needs. In vehicles, this is designed as a central gateway ECU that controls communication between these different zones, filtering messages according to security policies. The main advantage is containment. If an attacker manages to compromise one zone, such as the infotainment system, the segmentation prevents them from reaching safety-critical systems like brakes or engine control. This separation reduces the overall attack surface and is now required by automotive cybersecurity standards like ISO/SAE 21434.

3.3. Limitations and Problems

However, these defensive measures have inherent limitations that could affect their effectiveness. IDS limitations will be discussed in the next chapter. Here, we focus on the

limitations of IPS, SecOC, and network segmentation.

IPSs Limitations

For IPS, the main limitation is the safety risk posed by false positives. If an IPS incorrectly identifies a legitimate message as malicious and blocks it, this could endanger the passengers, particularly if the message was safety-critical. Another critical limitation is the reaction speed at which the IPS must operate. To prevent an attacker from taking control of the vehicle, the IPS must react within microseconds. Vendors must ensure this real-time performance to guarantee safety, any delay may cause the malicious packet to reach the ECU and be processed.[12]

SecOC Limitations

Regarding SecOC, the first limitation originates from the fact that the CAN protocol was not originally designed with any authentication in mind. Adding this property consumes significant payload space and may result in the truncation of the Message Authentication Code (MAC), which lowers security strength [25]. Another security concern is that SecOC's entire security relies on the secrecy of the cryptographic keys. If keys are not stored correctly, attackers could extract them and compromise the entire system. Finally, the cryptographic calculations have a high computational overhead that introduces latency in the computations, which may impact real-time control loops.

Network Segmentation Limitations

Regarding network segmentation, it typically relies on a central gateway that manages traffic between network zones. This introduces three main vulnerabilities. First, if an attacker compromises the gateway, they gain access to the entire network and potentially all ECUs, enabling full vehicle control. Second, if an attacker gains access to a specific network segment, they can potentially launch their attack in that section without passing through the gateway. Third and most importantly, the gateway serves as a single point of failure. It must process and filter all traffic between network segments, creating potential performance bottlenecks under high load of traffic.

4 | State of the Art: IDS in Automotive

Intrusion Detection Systems (IDSs) are the most common defense against attacks on in-vehicle networks. Over the years, researchers have developed many different types of IDS, each using different methods to detect malicious activity. Some analyze message patterns, others look at physical signals, and some use machine learning to identify anomalies. Each approach has its own advantages and limitations in terms of accuracy, speed, and deployment complexity.

In this chapter, we examine the main IDS categories found in automotive security research. We start with cyber-physical approaches that use vehicle dynamics to detect attacks (Section 4.1). Then we cover entropy-based methods that measure signal randomness (Section 4.2), graph-based techniques that model message relationships (Section 4.3), and hardware-based methods that use physical device characteristics (Section 4.4). We also discuss approaches based on message frequency (Section 4.5) and message sequences (Section 4.6). For each category, we present key papers and analyze their strengths and weaknesses.

4.1. Cyberphysical

4.1.1. Overview

These systems detect attacks by correlating network data with the physical physics/dynamics of the vehicle (e.g., speed, braking force). They operate at the application level to try and identify attacks.

4.1.2. Representative Works

Detection and Mitigation of Sensor and CAN Bus Attacks in Vehicle Anti-Lock Braking Systems

Kang and Shen addressed security vulnerabilities in Anti-Lock Braking Systems (ABS), particularly sensor attacks and CAN bus intrusions. Existing anomaly detection methods lacked precision and provided no mitigation once attacks were detected [17].

The authors proposed a two-stage system combining detection and mitigation. The detection stage uses vehicle state equations and a real-time friction model to predict expected wheel speed and brake force, identifying attacks through deviations from sensor measurements. The mitigation stage subtracts identified attack values and employs an Extended Kalman Filter (EKF) to reconstruct accurate vehicle states, enabling safe ABS operation during attacks [17].

Validation through CarSim simulations and physical experiments demonstrated significant accuracy improvements over existing algorithms (IBM and SBM) and successful wheel-lock prevention during active attacks [17].

4.1.3. Strengths and Limitations

Cyber-physical IDS have important strengths. They can detect sophisticated attacks that look legitimate at the message level but cause physically impossible vehicle behavior. For example, if an attacker injects fake brake commands, the system detects the inconsistency between the commanded deceleration and the actual vehicle dynamics. These methods are also naturally resistant to replay attacks because replayed messages from different driving contexts will violate physical constraints.

However, there are significant limitations. First, these systems need accurate vehicle models and reliable sensor data, which are not always available. Sensor noise and modeling errors can cause false positives. Second, the computational cost is high because the system must continuously run state estimation algorithms such as Kalman filters. This overhead may be too much for resource-constrained ECUs. Third, these methods are system-specific, a model for the braking system cannot be used for steering or engine control. Each subsystem needs its own model, requiring significant engineering effort. Finally, cyber-physical IDS don't work for non-safety-critical systems like infotainment, where physical models do not apply.

These limitations highlight the need for detection approaches that do not rely on detailed physical models and can operate uniformly across different ECUs.

4.2. Entropy-Based

4.2.1. Overview

Entropy was originally introduced by Shannon in 1948, as a measure of information content and uncertainty in communication systems. In the context of automotive IDS, entropy quantifies the predictability of CAN traffic. High entropy indicates high randomness (e.g. during fuzzing attacks), while low entropy suggests repetitive patterns (e.g., replay attacks). Entropy-based IDS methods leverage information theory to detect anomalies in CAN network traffic. These approaches measure the randomness, uncertainty, or complexity of message characteristics such as ID distribution, payload values, or timing patterns. The underlying principle is that normal vehicle operation exhibits predictable entropy patterns, while attacks introduce abnormal randomness or reduce entropy in measurable ways.

4.2.2. Representative Works

TCE-IDS: Time Interval Conditional Entropy Based Intrusion Detection System for Automotive Controller Area Networks

Yu et al. developed the Time Interval Conditional Entropy-Based Intrusion Detection System (TCE-IDS) to secure vehicles against cyberthreats such as DoS and fuzzing attacks and to overcome the limitations of traditional information entropy methods [36]. Previous approaches often relied on sliding window information entropy, which can be computationally intensive and ineffective against impersonation attacks.

Instead, TCE-IDS uses the conditional entropy of time intervals between adjacent messages to identify anomalies. By calculating entropy values based on frame IDs and data fields, the system establishes a stable reference range for normal traffic; experimental analysis revealed that legitimate messages maintain low entropy values, while malicious injections cause significant deviations [36].

The authors implemented this lightweight solution on an Arduino-based platform using real vehicle data from a Ford EcoSport, achieving a detection accuracy of over 99% against DoS, fuzzing, and impersonation attacks. Furthermore, the system demonstrated high efficiency, suitable for real-time analysis, with a detection response time of only 2 ms [36].

4.2.3. Strengths and Limitations

Entropy-based IDS are lightweight and do not require training data, making them practical for resource-constrained ECUs. They can detect both known and unknown attacks by identifying statistical deviations from normal patterns. They are also extremely flexible since entropy can be calculated on multiple features simultaneously for multi-layered detection.

However, these methods have several important limitations. Normal driving produces varying entropy patterns depending on the scenario (e.g., highway vs. city driving), potentially causing false positives. Sophisticated attackers who understand the baseline entropy could craft attacks that maintain normal statistical properties, evading detection. Selecting the right time window is also challenging: if windows are too short entropy estimates are noisy, while with windows that are too long detection delays are introduced. That may allow malicious messages to affect vehicle behavior before an alert is raised.

Additionally, low-frequency messages do not provide enough samples for reliable entropy calculations within practical time windows. Entropy methods may also struggle with stealthy attacks that inject only a few malicious messages, as these may not significantly alter the overall traffic entropy. Finally, entropy baselines established during normal operation may not generalize well across different drivers, road conditions, or vehicle loads, requiring periodic recalibration or adaptive thresholding mechanisms.

This motivates lightweight detection techniques with deterministic behavior and minimal sensitivity to parameter tuning.

4.3. Graph-based

4.3.1. Overview

Graph-based IDS model CAN network traffic as a graph structure where nodes represent CAN IDs or ECUs, and edges represent relationships between them, such as message sequences, timing dependencies, or co-occurrence patterns. The basic idea is that normal CAN traffic has a predictable graph structure, and attacks disrupt this structure in detectable ways.

For example, if messages with IDs 0x100 and 0x200 normally appear in a specific order, a graph can represent this as a directed edge from 0x100 to 0x200. If an attacker injects messages that violate this order, the graph structure changes. Similarly, if certain IDs frequently appear together in normal traffic, suspension attacks will break these co-

occurrence patterns, altering the graph.

4.3.2. Representative Works

DGIDS: Dynamic graph-based intrusion detection system for CAN

Song et al. developed the Dynamic Graph-based Intrusion Detection System (DGIDS) to address security vulnerabilities in CAN networks, particularly against DoS and injection attacks [31]. Existing graph-based methods had a major limitation—they used static analysis over fixed time windows. While they could detect that something was wrong, they could not identify which specific message ID was under attack or provide theoretical proof of their detection capabilities.

DGIDS takes a different approach by constructing graphs dynamically in real-time as messages arrive. Instead of waiting to analyze a full time window, it updates graph properties such as node degree, maximum degree, and edge distribution with every single message. The system establishes normal behavior thresholds during an offline training phase using what they identify as Base ID (BID) cycles. The authors did not just validate it experimentally—they also provided theoretical analysis using permutation and probability statistics to mathematically prove the detection probabilities [31].

The authors implemented DGIDS on an STM32F407 microcontroller and tested it on public datasets like CAR-Hacking and Ford. Results showed that DGIDS outperforms existing graph-based methods, achieving 100% recall for multiple injection attack types. The system is also extremely efficient, requiring minimal memory and achieving a detection delay of only 4.1 μ s for node degree updates [31].

4.3.3. Strengths and Limitations

Graph-based IDS capture relational information that statistical approaches miss, detecting attacks that manipulate communication patterns while maintaining correct individual messages. They identify which specific IDs are under attack through structural analysis and detect complex multi-step attacks where individual messages appear normal but relationships are abnormal. Graph structures provide natural interpretability through visualization, and dynamic approaches achieve low latency via incremental updates.

However, these methods require substantial memory for graph storage, problematic for resource-constrained ECUs on high-traffic buses. Graph operations are computationally expensive despite incremental optimizations. Parameter selection (features, thresholds) requires extensive testing. Normal driving produces varying graph structures (parking vs.

highway), requiring adaptive baselines. Methods need training and may not generalize across vehicle models. Finally, graphs detect structural anomalies but may miss payload-based attacks that preserve graph structure.

These constraints suggest the need for approaches that capture structural information without maintaining complex global models.

4.4. HW-Physical-Based

4.4.1. Overview

Hardware and physical-layer IDS detect attacks by analyzing the electrical and physical characteristics of CAN signals rather than message content. The basic idea is that every ECU has unique hardware properties that create distinctive fingerprints in the signals it transmits. Even if an attacker can forge valid messages with correct IDs and payloads, they cannot perfectly replicate the physical characteristics of the target ECU’s hardware.

These methods operate at the lowest communication level, analyzing properties like voltage levels, signal rise/fall times, and clock skew. These characteristics are determined by the ECU’s physical components—transistors, oscillators, capacitors—and are inherent to the hardware, making them very difficult to manipulate.

4.4.2. Representative Works

ClockIDS: A Real-Time Vehicle Intrusion Detection System Based on Clock Skew

Zhao et al. developed ClockIDS to overcome limitations in existing clock skew-based fingerprinting. Previous voltage-based methods were too resource-intensive, while earlier clock skew approaches failed to create unique fingerprints because results varied with message periods rather than device characteristics [38].

ClockIDS uses a period-independent fingerprinting technique. The system monitors message arrival times, estimates clock skew via Recursive Least Squares (RLS), and converts different ID periods into unified metrics. Detection employs two stages: Empirical Rule (ER) rapidly detects clock anomalies, then Dynamic Time Warping (DTW) identifies attack sources through time-series comparison [38].

Testing on commercial vehicles (Luxgen U5, Buick Regal) and BYD dashboards achieved over 98% detection accuracy and 96% source recognition against spoofing, bus-off, and

masquerade attacks. Real-time performance (1.99 ms per detection) remains stable as ECU count increases, unlike SVM approaches [38].

VALID: Voltage-Based Lightweight Intrusion Detection for the Controller Area Network

Schell and Kneib developed VALID to overcome limitations in existing voltage-based IDS methods. Previous systems like VoltageIDS [10] and Scission [18] effectively fingerprinted ECUs through electrical characteristics but required expensive external ADCs and complex calculations, making them impractical for cost-sensitive automotive micro-controllers [30].

VALID uses ECUs' built-in ADCs to capture 1-2 voltage samples during "zero bit" transmissions. The system applies signal alignment to handle environmental variations and employs RLS to build lightweight regression models per ECU. Detection uses CUMulative SUM (CUSUM) to track cumulative deviations between observed and predicted voltages, triggering alerts when deviations exceed thresholds established during training [30].

Testing on real vehicle data (Fiat 500 and Porsche Panamera) with 10,000 simulated impersonation attacks achieved 99.54% detection accuracy, 0.002% false positive rate, and 100% attack source identification. The system requires 75% less processing power than other voltage-based methods and detects attacks after an average of 13 malicious messages, enabling practical real-time deployment on resource-constrained ECUs [30].

4.4.3. Strengths and Limitations

Hardware and physical-layer IDS detect sophisticated attacks like frequency masquerading with clock skew emulation by validating physical devices rather than message content. Physical characteristics are extremely difficult to forge, requiring specialized hardware. These methods provide device authentication, identifying attack sources, and work independently of CAN protocol details.

However, voltage-based methods traditionally required expensive external ADCs, and newer built-in approaches still demand higher sampling rates. Physical characteristics are environmentally sensitive (temperature affects clock skew, aging changes voltage, cables degrade), necessitating recalibration. Signal processing overhead exceeds simple filtering. Manufacturing tolerances require per-vehicle calibration rather than shared fingerprints. Finally, recent research has shown that sophisticated attackers with detailed knowledge of physical-layer detection might craft partial fingerprint mimicry attacks, though this

remains extremely difficult and requires specialized equipment.

As a result, there is a need for solutions that do not depend on physical-layer fingerprinting.

4.5. Message Frequency

4.5.1. Overview

Message frequency-based IDS detect attacks by monitoring how often messages with specific CAN IDs appear on the bus. The basic principle is: most in-vehicle messages are transmitted periodically at fixed rates (e.g., every 10ms, 20ms, or 100ms). Attacks that inject additional messages or suspend legitimate transmissions create detectable changes in these frequency patterns.

These methods work by counting how many times each message ID appears within time windows (e.g., 1 second or 5 seconds). During normal operation, each ID should appear with a consistent frequency. For example, a message sent every 100ms should appear about 10 times per second. Injection attacks increase this frequency, while suspension or DoS attacks decrease it. By establishing baseline frequencies during normal operation, the system detects when current frequencies deviate significantly.

4.5.2. Representative Works

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

Olufowobi et al. proposed using the CUSUM algorithm to detect abrupt changes in CAN message arrival frequencies. The key insight is that injection attacks increase message frequency for specific IDs, while DoS attacks decrease or eliminate them. Unlike simple threshold-based counting, CUSUM formulates detection as a sequential change-point problem, identifying when message frequency transitions from normal to anomalous with low detection delay, by tracking cumulative deviations from baseline patterns established during training [28].

Validation on real-world CAN datasets with DoS, fuzzing, and spoofing attacks demonstrated effective detection with low latency—average delays of 0.032 seconds for gear spoofing and 0.113 seconds for RPM spoofing. With properly tuned parameters, CUSUM-based frequency monitoring achieves high detection accuracy with minimal false alarm

rates [28].

4.5.3. Strengths and Limitations

Message frequency-based IDS are extremely lightweight, requiring only message counters and basic arithmetic operations, making them ideal for resource-constrained automotive ECUs. They effectively detect common attack types— injection attacks that increase message frequency, and DoS or suspension attacks that decrease it. Detection is fast (typically within 1-5 seconds), results are easily interpretable, and no specialized hardware or extensive training data is required. These methods work independently of payload content and CAN protocol details.

However, frequency-based methods cannot detect sophisticated attacks that preserve normal transmission frequencies, such as frequency masquerading or low-rate replay attacks. They are sensitive to legitimate frequency variations caused by driving mode changes or event-triggered transmissions, potentially causing false positives. The time window size creates a fundamental trade-off—short windows enable fast detection but produce noisy estimates, while long windows are stable but introduce detection delays. Additionally, these methods provide no information about message content, cannot identify attack sources, struggle with low-frequency messages, and cannot distinguish between different attack types that produce similar frequency changes.

This indicates that frequency-based detection alone is insufficient and should be complemented by additional, orthogonal detection mechanisms.

4.6. Sequence-Based

4.6.1. Overview

Sequence-based IDS detect attacks by analyzing the temporal ordering of CAN messages rather than their individual content or frequency. Unlike graph-based methods that focus on structural relationships, sequence-based approaches examine the specific order in which messages appear. During normal vehicle operation, message sequences exhibit repeatable and predictable patterns, as certain IDs consistently appear before or after others due to ECU scheduling and functional dependencies.

By monitoring deviations from expected message sequences, these approaches are particularly effective at detecting context-dependent attacks such as replay or masquerade attacks, where injected messages are indeed valid but occur at inconsistent points within

the communication breaking its logical ordering.

4.6.2. Representative Works

Analysis of ID Sequences Similarity Using DTW in Intrusion Detection for CAN Bus

Sun et al. developed SIDuDTW to address limitations in periodicity-based IDS approaches. While previous methods monitored message transmission timing and broadcast intervals to detect attacks, they failed to detect replay and suppression attacks that maintain correct periodicity but violate logical message sequences.[33]

SIDuDTW reconstructs ID values to stabilize transmission patterns and groups them into logical "waves" using a fixed key value strategy based on a periodic Baseline ID (BID). The system employs DTW to calculate similarity between incoming traffic ("Window wave") and a continuously updated normal profile ("Criterion wave"), reconstructing ID sequences that respect the pseudo-periodicity of CAN traffic. Parameters including update probability and wave length are optimized via Simulated Annealing (SA) for robustness against environmental noise.[33]

Evaluation on the Car-Hacking dataset and a real vehicle (XiaoPeng P5) demonstrated over 95% accuracy across DoS, injection, replay, and suppression attacks. SIDuDTW significantly outperformed existing methods maintaining F1-scores above 90% for all categories. The system achieves real-time performance with approximately 4ms similarity calculation overhead.

4.6.3. Strengths and Limitations

Sequence-based IDS detect attacks that maintain correct timing but violate message ordering. This catches attacks that simple frequency or timing methods miss, since attackers can replicate message rates but struggle to perfectly match sequence patterns. An additional advantage is the fact that deviations can be traced back to specific violated sequences, helping to understand the nature and origin of the attack.

However, sequence-based methods require extensive training data to capture legitimate message orderings, including rare scenarios. Event-driven messages do not follow a fixed ordering and may appear differently across driving modes, which complicates sequence modeling and can lead to false alarms when strict matching is applied. Additionally, the number of possible message sequences grows rapidly with the number of monitored IDs, leading to higher memory and computational requirements. Finally, these approaches

are ineffective against attacks that preserve valid message ordering while manipulating payload content or timing characteristics.

Overall, sequence-based IDS are effective at identifying context-dependent attacks but have limitations under unpredictable scenarios and resource constraints. This motivates combining sequence-based analysis with complementary detection techniques to achieve broader and more reliable coverage.

4.7. Limitations of IDSs

Automotive IDS face several limitations that must be considered during development and deployment:

- **Limited resources:** All ECUs within a vehicle have severe constraints regarding memory, computational power, and bandwidth. Typical ECUs have clock speeds of 80-400 MHz, flash memory of 2-16 MB, and RAM of 128 KB to 2 MB. These constraints limit the complexity of IDS algorithms that can be deployed on ECUs.[24]
- **Real-time constraint:** the CAN messages are exchanged between ECUs in real-time, which means that queuing or delaying packets for analysis may be too risky and cause violations in automotive environments. IDSs must process messages within microseconds to avoid affecting the vehicle's operation.[24]
- **Size, Weight, and Cost:** Adding IDS capabilities may require extra hardware or modifications to existing ECUs, increasing both vehicle weight and manufacturing costs.[24]
- **False Positive Rate:** High false positive rates can lead to legitimate warnings being ignored. In safety-critical systems, even a 0.1% false positive rate may be too heavy.
- **Attack Coverage:** No single IDS can detect all attack types. Each approach has blind spots—for example, timing-based IDS may miss payload-based attacks, while payload-based IDS may miss frequency or suspension attacks. Thus the need for a comprehensive method to broaden the spectrum of attacks.

These limitations reveal a fundamental gap in existing work: the need for IDS solutions that are simultaneously lightweight, comprehensive, and accurate. Current approaches optimize for one or two of these properties at the expense of others. High-performance ML systems achieve excellent detection but require external hardware, while lightweight frequency-based systems deploy easily but miss entire attack categories. This gap mo-

tivated our research to combine multiple lightweight detection principles in a parallel pipeline that addresses resource constraints while maintaining high detection rates and near-zero false positive rates across diverse attack types.

5 | Proposed Pipeline Approach

In this chapter, we will discuss the IDS pipeline that we wanna prove to be better than heavy ML systems

While there are systems that use hybrid approaches we felt there was the need for such an approach to be lightweight and fast while still having a broad surface of attacks and a low false positive rate

5.1. Motivation

Machine learning algorithms dominate contemporary intrusion detection landscape, achieving near-perfect detection rates (99%+) in diverse attack scenarios. However, this performance comes at a cost for resource constrained environments such as CAN Bus: ML-based IDS solutions require costly and powerful HW to address their needs. Standard automotive Electronic Control Units (ECUs)—typically mid-range microcontrollers with 256 KB to a few MB of RAM, operating under strict real-time and cost constraints—cannot support these resource-intensive approaches. ML-based detection deployment often requires dedicated security processors or external hardware modules, increasing both manufacturing cost and system complexity while introducing additional attack surfaces through expanded hardware interfaces. In contrast, existing lightweight IDS approaches operate within ECU resource constraints using simple heuristics or statistical methods, but suffer from incomplete attack coverage and possibly poor detection performances compared to ML systems.

No prior work has systematically demonstrated that combining multiple lightweight detection methods—each individually limited—can achieve both comprehensive attack coverage and ECU-deployable resource requirements competitive with ML-based systems. Existing research generally evaluates single detection methods, limiting coverage to specific attack categories. This gap motivates our research: can a multi-method lightweight IDS achieve detection rates comparable to state-of-the-art ML systems while operating within the strict latency, memory, and computational constraints of standard automotive ECUs?

This thesis addresses three specific objectives:

1. Demonstrate that combining complementary lightweight detection methods—Hamming distance payload analysis, ErrIDS timing monitoring, and watchdog-based suspension detection—achieves competitive detection rates across diverse attack types without relying on computationally expensive machine learning algorithms or GPU acceleration.
2. Determine the optimal architectural configuration for integrating multiple detection methods by comparing serial two-stage filtering against parallel OR-logic configuration, evaluating the tradeoffs between computational efficiency and attack coverage to identify which architecture maximizes detection effectiveness while maintaining real-time performance.
3. Validate the practical deployability of the proposed system through comprehensive performance comparison against existing state-of-the-art IDS methods (both ML-based and lightweight) on the X-CANIDS benchmark dataset, and verify resource requirements through embedded implementation on STM32 microcontroller hardware representative of automotive ECU platforms.

These objectives fill the gap between high-performing but hardware-intensive ML systems and resource-efficient but coverage-limited lightweight approaches. By demonstrating that strategic combination of simple detection methods can achieve high overall detection rate with sub-millisecond latency on mid-range microcontroller hardware, this work provides a possibility for practical automotive intrusion detection that meets both the security requirements of modern connected vehicles and the strict cost, power, and real-time constraints of automotive manufacturing.

5.2. Component Selection Methodology

We started from a pool of 4,141 papers related to security and intrusion detection in automotive networks. To extract relevant studies, we developed a Python-based filtering system structured in three stages.

Stage 1 - Publication Quality: The first stage filtered papers by source title and publisher to ensure research quality and reproducibility. Table 5.1 shows the filtering keywords used.

Papers had to contain at least one source title keyword AND be published by one of the recognized publishers to pass this stage. This ensured we only considered peer-reviewed

Filter Type	Keywords (case-insensitive)
Source Title	Transactions, Proceedings, Conference Lecture Notes in Computer Science Symposium, USENIX
Publisher	Elsevier, Springer Institute of Electrical and Electronics Engineers (IEEE) Association for Computing Machinery (ACM)

Table 5.1: Stage 1 filtering keywords for publication quality

research with rigorous review processes and reproducible methodologies.

Stage 2 - Domain Relevance:

The second stage filtered by domain relevance to ensure papers addressed automotive security specifically. Table 5.2 shows the filtering keywords used.

Filter Type	Keywords (case-insensitive)
Automotive Domain	Automotive, Vehicular, Vehicle Transportation, Cars
Security Context	Security, Secure, Cyber Resilience Cyber-Physical Systems, CPS
Related Domains	Smart Cities Internet of Things Journal

Table 5.2: Stage 2 filtering keywords for domain relevance

Papers had to contain at least one automotive domain keyword AND at least one security context keyword in their source title to pass this stage. This excluded generic network security approaches not adapted for automotive constraints.

Stage 3 - Research Focus: Finally the third stage filtered for practical intrusion detection research. Table 5.3 shows the filtering criteria.

Filter Type	Keywords (case-insensitive)
Include (Title must contain)	detection, IDS, identification, intrusion
Exclude (Title must NOT contain)	survey, review

Table 5.3: Stage 3 research focus filtering keywords

Papers had to contain at least one inclusion keyword AND none of the exclusion keywords in their title to proceed. This ensured focus on practical detection methods rather than literature summaries or purely theoretical analysis.

After the three-stage filtering process, we had 109 potentially relevant papers. To identify methods suitable for deployment on resource-constrained automotive ECUs, we applied four criteria: (1) exclude machine learning approaches due to their computational overhead and training challenges, (2) require lightweight implementation without external hardware, (3) ensure broad attack surface coverage across multiple attack types, and (4) demonstrate high detection performance with low false positive rates. Applying these criteria reduced the pool to 20 candidate systems. We then analyzed each candidate in depth, ranking them based on computational efficiency, detection accuracy, and attack surface. This analysis led to the selection of two complementary methods that together maximize attack detection while maintaining lightweight operation: Hamming distance-based payload analysis and ErrIDS timing analysis, described in detail in Section 5.3.

5.3. Pipeline Architecture

The detection pipeline combines two complementary methods: Hamming distance-based payload analysis and ErrIDS timing analysis. This multi-layered approach enables detection of both content-based attacks (fabrication, fuzzing, masquerade, injection) and timing-based attacks (replay, suspension). The following sections describe each method's implementation and integration into the pipeline.

5.3.1. Hamming Distance

Marchetti, Stabili and Colajanni proposed a method to detect malicious CAN messages by computing the Hamming distance¹ between consecutive payloads of the same message ID [32]. The proposed algorithm was designed and implemented exactly for low-end ECUs with low computational complexity and small memory footprints [32].

Implementation and Evaluation

Starting from the definition of Hamming distance shown in equation 5.1:

$$\mathcal{H}_d(x, y) = \sum_{i=1}^k |x_i - y_i| \quad (5.1)$$

¹The Hamming Distance between two strings of equal length is the number of positions at which the corresponding symbols differ. It is a metric that only counts substitutions and assumes that the two strings being compared are of the same length. It was introduced by Richard Hamming in 1950 and it is used in several different environments such as cryptography or information theory [4, 32]

the authors define a variation of the classic distance by evaluating the bitwise XOR between payloads of CAN messages with the same ID:

$$\mathcal{H}_d(p_t, p_{t+1}) = \sum_{i=1}^n p_t^i \oplus p_{t+1}^i \quad (5.2)$$

where p_t and p_{t+1} are consecutive payloads of the same CAN ID, and n is the payload length in bits (typically 64 for 8-byte messages) [32].

During training, the system observes normal CAN traffic and establishes baseline Hamming distance ranges for each message ID. For each ID, the Hamming distance is computed between consecutive messages, and the minimum and maximum observed distances define the normal range:

$$\mathcal{R}_{ID} = [\min(\mathcal{H}_d), \max(\mathcal{H}_d)] \quad (5.3)$$

This training process uses attack-free CAN traffic to ensure baselines represent legitimate behavior taking messages from 20% of the collected data. The other 80% is used for validation of the method to check the zero false alarms raised. The core assumption of the algorithm is that Hamming distances between two consecutive normal payloads will fall inside the range calculated during training, while injected or malicious attacks will disrupt these ranges going outside their boundaries.

To prove the effectiveness of the system, the authors decided to implement a fuzzing attack and a replay attack (attacks are described in Chapter 3).

During live detection phase the previously created model is used on the collected data to determine possible attacks based on the Hamming distance of two consecutive payloads with same message ID. If the distance that is being checked is outside the boundaries of the model, a flag is raised [32].

Analysis of the Hamming distance ranges revealed that message IDs naturally cluster into three categories based on payload variability [32]:

- **NoRange:** Hamming range = 0. Consecutive payloads that are identical. These are constant messages like vehicle identification or static configuration data.
- **SmallRange:** Hamming range $< \sigma$. Payloads exhibit limited variation, typical of gradually changing sensor values or control signals.
- **MidRange:** Hamming range $\geq \sigma$. Payloads show significant variation, often con-

taining composite signals or rapidly changing values.

The threshold value $\sigma = 6$ was determined empirically by sorting IDs by their Hamming range and identifying a natural gap in the distribution. This threshold groups IDs with similar detection characteristics, as evidenced by consistent detection rates within each category [32].

Detection performance varies significantly across these categories. NoRange and SmallRange IDs achieve high detection rates because any deviation from their narrow normal ranges is clearly anomalous. However, MidRange IDs suffer from higher false negative rates—when attackers inject random payloads, there is a higher probability that the malicious payload’s Hamming distance falls within the wide normal range, making the attack indistinguishable from legitimate traffic [32]. An insight is that while MidRange IDs are hard to protect against Fuzzing (random injection), they actually yielded better detection results for Replay attacks (10-20%) compared to SmallRange/NoRange (<2% or 0%).

Strengths and limitations

The Hamming distance method offers several advantages. Its computational simplicity (single XOR operation per message) and minimal memory requirements (344 Bytes) make it suitable for resource-constrained ECUs. The algorithm has $O(n)$ complexity where n is the number of payload bits (maximum 64 for standard CAN). Payload-based analysis enables effective detection of injection, fabrication, and fuzzing attacks that frequency or entropy-based systems might miss [32].

However, the method has significant limitations. It struggles with replay attacks, particularly for IDs with very low payload variability where replayed messages appear identical to normal traffic. The method is completely ineffective against suspension and frequency modulation attacks since it only analyzes payload content and ignores timing characteristics. This motivates the need for a complementary timing-based detection system to achieve comprehensive attack coverage.

5.3.2. ErrIDS

Lee *et al.* proposed ErrIDS (Error-based Intrusion Detection System), a lightweight timing-based IDS designed to detect vehicular intrusions including sophisticated frequency masquerading attacks that can emulate the clock skew of legitimate ECUs, without requiring special hardware equipment [21].

ErrIDS exploits a previously overlooked feature in CAN timing analysis: the residual² between nominal and actual inter-arrival times. While previous periodicity-based IDS approaches treated these residuals as noise to be eliminated through averaging, Lee *et al.* discovered that residual patterns contain meaningful information about transmission behavior.

The key point of their work lies in understanding CAN arbitration delays. When a transmission loses arbitration or experiences delays in the CAN controller's transmission buffer, the next message in the sequence typically arrives earlier than expected to compensate, creating a predictable pattern: a large positive residual (delayed message) is consistently followed by a small or negative residual (early message). This oscillating pattern occurs because the MCU periodically requests transmissions from the CAN controller regardless of whether previous requests are still pending. The authors validated this hypothesis by comparing CAN buses with single ECUs (no arbitration delays, constant intervals) versus multiple ECUs (frequent delays, oscillating residuals).

Residual Calculation. For each CAN ID, the nominal transmission period t is determined from benign training data and rounded to standard frequencies (10ms, 20ms, 100ms, 200ms, 1000ms). The i -th residual is calculated as:

$$r_i = at_i - t \quad (5.4)$$

where t is the nominal period and at_i is the actual measured timestamp interval between consecutive messages i and $i - 1$. The cumulative residual over k messages is:

$$R_{cum}[k] = \sum_{i=1}^k r_i \quad (5.5)$$

Intrusion Detection. Under normal conditions, $R_{cum}[k]$ oscillates within bounded limits as delayed and early messages cancel each other out. To eliminate linear drift caused by hardware clock variations, the authors apply a rotation transformation to the cumulative residual, yielding a stationary oscillating signal. Attack detection relies on analyzing the gradient (first derivative) of $R_{cum}[k]$:

$$\frac{dR_{cum}[k]}{dk} = R_{cum}[k] - R_{cum}[k - 1] = r_k \quad (5.6)$$

²The residual is defined as the difference between the actual timestamp interval and the nominal period, where the timestamp interval is the time elapsed between two consecutive transmissions of the same CAN ID.

When an attack occurs, the cumulative residual exhibits sudden shifts:

- **Message injection:** Timestamp intervals shrink, producing large negative gradients exceeding the lower threshold.
- **Suspension:** Missing messages create large positive gradients exceeding the upper threshold.
- **Frequency masquerading:** Even with perfect clock skew emulation, time synchronization errors at attack initiation cause detectable discontinuities in $R_{cum}[k]$.

Threshold Determination. Detection thresholds are determined per CAN ID by assuming gradients follow a normal distribution during benign operation. Thresholds are computed as:

$$thr_U = K \times \sigma + \mu \quad (5.7)$$

$$thr_L = -K \times \sigma + \mu \quad (5.8)$$

where K is a constant (set to 5 or 7 to achieve $FPR < 0.0001\%$), and μ and σ are the mean and standard deviation of gradient values for each ID during training. Critically, these thresholds can be shared across vehicles of the same make and model because residual patterns depend on CAN bus configuration (number of ECUs, message priorities) rather than individual device characteristics like clock skew.

Implementation Note: The original ErrIDS paper addresses periodic-and-on-event (PE) messages by manually identifying them through payload analysis and excluding event-triggered transmissions from cumulative residual calculation. This filtering requires either access to DBC files (which specify transmission types) or manual traffic analysis to distinguish time-triggered from event-triggered instances of the same message ID. Our implementation does not perform this PE filtering, treating all messages as purely periodic. This simplification enables deployment without proprietary DBC specifications but results in elevated false positive rates on PE messages (044h, 5A0h, 5B0h) where event-driven transmissions violate ErrIDS’s periodicity assumptions.

Strengths and Limitations

ErrIDS represents a significant advancement in lightweight automotive intrusion detection by addressing critical weaknesses in previous timing-based approaches while maintaining practical deployability. The method’s primary strength lies in its ability to detect sophisticated frequency masquerading attacks even when adversaries emulate the clock skew of legitimate ECUs—a capability that distinguishes it from clock-based fingerprint-

ing methods which were shown to be vulnerable to such techniques. By focusing on cumulative residual patterns rather than device-inherent characteristics like clock skew, ErrIDS achieves a practical advantage: thresholds determined on one vehicle can be shared with other vehicles of the same make and model. This cross-vehicle property improves scalability for automotive manufacturers, who would otherwise need to calibrate IDS parameters individually for millions of vehicles. The method requires no special hardware beyond standard CAN timestamp avoiding the cost and complexity of voltage-based fingerprinting approaches that demand additional sensing equipment and high sampling rates. Computational overhead remains minimal, consisting only of basic arithmetic operations (subtraction, summation, threshold comparison) suitable for resource-constrained ECUs.

However, several limitations must be acknowledged. The method fundamentally depends on periodic message transmission, making it ineffective for event-driven or aperiodic CAN traffic. This limitation manifests as elevated false positive rates for periodic-and-on-event (PE) messages during vehicle motion, where driver actions trigger additional transmissions that disrupt expected residual patterns. While the authors address this by manually identifying PE messages and excluding event-triggered instances from cumulative residual calculation, this approach requires CAN database (DBC) knowledge typically unavailable in practice. ErrIDS focuses exclusively on timing analysis and cannot detect payload-manipulating attacks that preserve correct inter-arrival times, such as fuzzing attacks with legitimate periodicity or data field modifications that maintain message frequency.

5.3.3. Watchdog Suspension Check

While the Hamming distance and ErrIDS methods detect payload and timing anomalies respectively, neither is able to detect suspension attacks where legitimate messages are suppressed or delayed in time. To address this limitation, we introduced an additional watchdog-based monitoring mechanism.

The approach is: for each monitored message ID, we maintain a timer that resets upon message reception. If the expected message does not arrive within a timeout threshold (calculated by multiplying the nominal transmission period by a fixed value), the watchdog triggers a suspension alert. This guarantees that missing or delayed messages are detected within a certain time, instead of going unnoticed by payload or gradient-based analysis.

The watchdog timeout threshold for each ID is determined during training by testing for reasonable values and looking at the performances. Implementation details, memory optimization strategies, and performance evaluation are presented in Chapter 6.

5.4. Configuration Modes: Series vs. Parallel

We considered two approaches while creating the pipeline with the Hamming distance, ErrIDS, and watchdog monitoring mechanisms: series and parallel configurations. Each one offers different advantages and disadvantages between detection latency, computational efficiency, and implementation complexity.

- **Series Configuration:** In the series configuration, systems execute sequentially, one after the other, to reduce computational load. Our pipeline operates as follows:
 1. **Hamming Distance (First Stage):** Each message is first evaluated using Hamming distance analysis. Three outcomes are possible:
 - If $\mathcal{H}_d > \theta_{high}$: Immediately flag as anomalous (definite attack)
 - If $\theta_{low} < \mathcal{H}_d \leq \theta_{high}$: Pass to ErrIDS for further analysis (suspicious)
 - If $\mathcal{H}_d \leq \theta_{low}$: Accept as normal (skip ErrIDS)
 2. **ErrIDS (Second Stage):** Only messages flagged as suspicious by Hamming distance are evaluated by timing analysis.
 3. **Watchdog (Continuous):** Suspension monitoring runs independently, triggering alerts when expected messages fail to arrive.

The series approach reduces average computational load by skipping ErrIDS for clearly normal or clearly anomalous messages. However, it increases latency for suspicious messages and requires careful threshold calculations to avoid filtering out attacks that only ErrIDS could detect.

- **Parallel Configuration:**

In the parallel architecture, all detection methods execute simultaneously:

1. **Hamming Distance:** Analyzes payload Hamming distances.
2. **ErrIDS:** Analyzes timing gradients.
3. **Watchdog:** Monitors message arrival times.

Results are combined using logical OR: an anomaly flag from any method triggers an alert. This approach offers the advantage of having low latency, maximum coverage and simpler logic since there is no need for thresholds calculations outside the ones derived by the methods. The trade-off is that parallel processing has higher computational overhead since all methods execute for every message.

For resource-constrained automotive ECUs we decided to opt for parallel implementation. Despite higher computational overhead, it provides lower detection latency (critical for real-time systems), maximum attack coverage (important for safety), and simpler implementation (reduces development and validation effort). Performance evaluation and measured resource usage are presented in Chapter 6.

6 | Experimental Setup

This chapter describes the metrics used to evaluate the detection pipeline's performance and explains how they are calculated. Due to the fundamental differences between injection and suspension attacks and how they are performed on the tested dataset, we employed different measurement approaches: packet-based metrics for injection attacks and ID-based metrics for suspension attacks.

We will also focus on the dataset used to test our approach and on the setup we had before going deep into the implementation and testing into Chapter 7

All our main tests are performed in Python 3 on a system with Windows 10. CPU: Intel® Core™ i5-7400 CPU @ 3.00GHz RAM: 24 GB with 2133MHz.

6.1. Detection Metrics

All detection systems are evaluated using standard metrics for detection:

- **True Positive (TP):** Attack packets correctly flagged as malicious
- **False Negative (FN):** Attack packets incorrectly classified as benign
- **False Positive (FP):** Benign packets incorrectly flagged as attacks
- **True Negative (TN):** Benign packets correctly classified as benign

We use these metrics to derive the performance of our system.

- **True Positive Rate (TPR/Recall):** The True Positive Rate measures the proportion of attack packets successfully detected. $TPR = \frac{TP}{TP+FN}$. A TPR of 100% means all attacks were detected.
- **False Positive Rate (FPR):** The False Positive Rate measures the proportion of benign traffic incorrectly flagged as attacks. $FPR = \frac{FP}{FP+TN}$. Low FPR in safety-critical systems usually stands below 0.1%.
- **Precision:** Precision measures the proportion of raised alerts that are true attacks.

$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$. High precision means that when the system raises an alert, it is likely a genuine attack. This metric complements FPR by focusing on alert reliability.

- **F1 Score:** The F1 score provides a balanced metric combining precision and recall. $\text{F1} = 2 \times \frac{\text{Precision} \times \text{TPR}}{\text{Precision} + \text{TPR}}$. F1 score approaching 1.0 indicates excellent balanced performance.

6.1.1. Measurement Approaches by Attack Type

Due to fundamental differences in how injection and suspension attacks manifest, we employ different measurement methodologies.

Packet-Based Metrics (Injection Attacks): For injection attacks—including fabrication, fuzzing, masquerade, and replay—we use packet-based metrics. Each malicious packet injected by the attacker is evaluated individually: if the packet triggers detection, it is counted as a True Positive (TP); if it evades detection, it is a False Negative (FN). Benign packets follow similar logic: if a benign packet triggers an alert, it is counted as a False Positive (FP); if it passes normally without triggering detection, it is a True Negative (TN). This granular packet-level approach is appropriate for attacks that target individual packets by modifying their content or transmission timing.

ID-Based Metrics (Suspension Attacks): Suspension attacks operate differently from injection attacks—they suppress legitimate message transmissions rather than injecting malicious content. For these attacks, we use ID-based detection metrics. Each test file contains one suspended message ID (the attack target). We evaluate detection success ($\text{TPR} = 1.0$) when two conditions are met: the watchdog timer triggers for the suspended ID, and the detection occurs within the attack interval specified in the dataset. If the watchdog never triggers or triggers outside the attack window, $\text{TPR} = 0.0$ for that file. The final suspension TPR is the proportion of test files where the attack was successfully detected. False positives are measured at the ID level: if the watchdog triggers for a non-suspended ID, it is counted as a false positive. The FPR is calculated as:

$$\text{FPR}_{\text{suspension}} = \frac{\text{Number of benign IDs flagged}}{\text{Total benign IDs in system}} \quad (6.1)$$

6.1.2. The Collateral Effect

During testing we observed a high rate of FP that initially appeared to indicate poor system performance. However, detailed analysis revealed that the majority of these false

positives were direct consequences of the attacks themselves rather than algorithmic errors. So we decided to classify them as **collateral false positives**, meaning benign packets flagged as a consequence of the attack and not detection failures.

Both detection methods exhibit this behaviour through different mechanisms:

Hamming Distance: When an attack packet is processed, its payload becomes the baseline for the next comparison. This corrupts the baseline, the next benign packet is compared against the attack payload rather than a legitimate payload, producing an Hamming distance that falls out of range for that ID. That triggers detection even though the benign packet itself is normal.

ErrIDS Timing: When attacks flood the CAN bus, legitimate messages go through queuing delays. Benign packets may arrive outside their expected timing windows and trigger detection not because they are malicious, but because the attack disrupted normal bus timing.

We distinguish between two types of false positives:

- **Algorithmic FP:** Detection errors on genuinely normal traffic (true false positives)
- **Collateral FP:** Benign packets delayed by attacks, triggering timing violations (explained false positives)

Collateral Mitigation Strategy

To mitigate this effect and prevent collateral damage from inflating FPR measurements, we implement a temporal cooldown mechanism: The timestamp of the most recent attack packet is tracked and saved. Whenever this attack occurs, a cooldown window of 500 ms is created. During this cooldown, each benign packet that is flagged as an attacks is classified as "collateral" rather than FP. After this cooldown expires, every benign packet that triggers an alert is flagged as an actual algorithmic FP. This approach recognizes that timing disturbances caused by attacks are expected system behavior, not detection failures. The 500 ms window was chosen empirically based on observed CAN bus recovery time after attack bursts.

Effective True Negative Calculation

Standard FPR calculation assumes all correctly classified benign packets are True Negatives. However, collateral packets represent a special case, as they trigger alerts, but these alerts correctly identify attacks perpetrated on the bus. To reflect this, we compute an

adjusted FPR:

$$\text{Effective TN} = \text{TN}_{\text{active}} + \text{Collateral} \quad (6.2)$$

$$\text{FPR}_{\text{clean}} = \frac{\text{FP}_{\text{algorithmic}}}{\text{FP}_{\text{algorithmic}} + \text{Effective TN}} \quad (6.3)$$

This adjusted metric distinguishes between algorithmic limitations (true FP) and correct detection of attack effects (collateral), providing an honest assessment of system performance under normal operating conditions.

6.2. Testing Dataset

We evaluated the pipeline on the X-CANIDS dataset, which provides comprehensive coverage of real-world CAN traffic and diverse attack scenarios for our research purposes.

6.2.1. X-CANIDS

X-CANIDS was created by the Hacking and Countermeasure Research Lab (HCRL) as part of their intrusion detection research [14]. The dataset is now publicly available and contains both raw CAN messages and deserialized signals collected from a real commercial vehicle, facilitating reproducible research in automotive cybersecurity. The dataset contains CAN messages captured from a 2017 Hyundai LF Sonata e-VGT. Data collection used a Kvaser Memorator Professional HS/HS interface (with built-in clock providing $1\mu\text{s}$ timestamp resolution) connected to the vehicle's OBD-II port [14].

Dataset Structure

The benign part of the dataset is made of six driving dumps + one idling dump. Each dump consists of 62-64 arbitration IDs. Since the dataset was collected from a real commercial vehicle, it contains more numerous arbitration IDs than other public datasets that were made synthetically [14].

The benign portion of the dataset consists of seven capture sessions: six driving dumps (vehicle in motion under various conditions), one idling dump (vehicle stationary with engine running).

Each dump contains 62-64 unique arbitration IDs. Since the dataset was collected from a real commercial vehicle, it includes more arbitration IDs than synthetic datasets, providing

realistic diversity in message types and transmission patterns [14].

Data Format

Table 6.1 shows the structure of the CAN message data.

Field	Type	Description
timestamp	Index	Monotonic timestamp with microsecond resolution. Messages sorted chronologically. Example: 0 days 00:00:00.026248 indicates 26.248ms from capture start.
arbitration_id	Column	CAN Arbitration ID in range 0–2047 (000h–7FFh). Identifies message type and priority.
dlc	Column	Data Length Code (0–8 bytes). Specifies payload length.
data	Column	Raw payload bytes. Represented as Python bytes object (e.g., b'\x00\x08\x0c \x04').
label	Column	Benign (0) or attack (1). Benign dumps (Dump 1–7) contain only label 0. Attack dumps mark injected packets as 1.

Table 6.1: X-CANIDS data format and field descriptions

Each row represents one CAN message. The timestamp index is unique—only one message can be transmitted on the CAN bus at any given time due to the bus arbitration protocol [14].

Example data showing three consecutive CAN messages from Dump 1:

timestamp	arb_id	dlc	data	label
00:00:00.026248	902	8	b'n\x84e\x04...'	0
00:00:00.026416	356	4	b'\x00\x08...'	0
00:00:00.028296	1427	6	b'\x00\x01...'	0

Attack Dataset

The intrusion dataset was created by injecting attacks into Dump 6 (vehicle in motion) during the time window 480s–1440s (16-minute attack period).

The attack surface comprises 126 distinct attack files:

- **17 Fuzzing attacks:** Varying injection rates to test detection under different attack intensities
- **35 Fabrication attacks:** Injecting well-crafted CAN messages with a specific AID and payload
- **35 Masquerade attacks:** Injecting messages using legitimate IDs with malicious payloads
- **35 Suspension attacks:** Suppressing transmission of specific message IDs
- **4 Replay attacks:** Retransmitting previously captured legitimate messages

Attack descriptions and threat models are detailed in Chapter 3.

Table 6.2 summarizes the attack dataset composition.

Attack Type	Files	Benign Packets	Attack Packets
Fuzzing	17	4.2M	9K – 1.8M
Fabrication	35	4.2M	1K – 100K
Masquerade	35	4.2M	1K – 100K
Suspension	35	4.2M	N/A (suppression)
Replay	4	4.2M	1.9M

Table 6.2: Attack dataset statistics per file

Each attack file contains approximately 4.2 million benign packets (baseline traffic from Dump 6), with attack packets overlaid during the 480s–1440s window. The number of attack packets varies:

- **Fuzzing:** 9,000 – 1,800,000 packets (depends on injection rate)
- **Fabrication/Masquerade:** 1,000 – 100,000 packets (depends on target ID and injection pattern)
- **Replay:** 1,900,000 packets per file
- **Suspension:** No injected packets (attack = absence of expected messages)

In attack files, the `label` column is set to 1 for malicious packets and 0 for legitimate traffic. Suspension attack files contain only label 0 since the attack suppresses transmissions rather than injecting packets.

Data Processing

For this research, we focus exclusively on raw CAN messages (`timestamp`, `arbitration_id`, `dlc`, `data`, `label`). Signal deserialization, while available in X-CANIDS, is not required for our lightweight detection approach, which operates directly on CAN frame metadata and payloads without requiring DBC files or signal-level interpretation.

6.3. Experimental Setup

This section describes the experimental environment used to validate the proposed intrusion detection pipeline. We performed a two-phase evaluation strategy: (1) algorithm development and validation in Python using the complete X-CANIDS dataset, and (2) implementation on a resource-constrained microcontroller to calculate detection times and memory request.

6.3.1. Python Simulation Environment

Algorithm development and testing were conducted in Python using Visual Studio Code with Jupyter Notebook integration (.ipynb format).

Table 6.3 lists the Python libraries used and their purposes in the detection pipeline.

Library	Version	Purpose
pandas	2.x	DataFrame operations for CAN message processing, time series analysis
pyarrow	Latest	Efficient Parquet file reading for large datasets (3M+ messages per file)
numpy	Latest	Numerical operations for Hamming distance calculations, gradient computations
collections	Standard	<code>deque</code> for sliding window implementation, <code>defaultdict</code> for ID-based state tracking
matplotlib	Latest	Performance visualization, ROC curves, metric plots
sklearn	Latest	Confusion matrix generation, metric validation
os	Standard	File system operations for batch dataset processing

Table 6.3: Python dependencies and their roles in the detection pipeline

6.3.2. STM32 Environment

To validate the system’s lightweight characteristics and real-time detection capabilities, we implemented the detection pipeline on an STM32 microcontroller—a platform representative of automotive ECU hardware constraints. The target platform was an STM32 Nucleo C031C6 with the specifications shown in Table 6.4.

Specification	Value
MCU	STM32C031C6 (ARM Cortex-M0+)
Clock Speed	48 MHz
Flash Memory	32 KB
SRAM	12 KB
Architecture	64-bit

Table 6.4: STM32 Nucleo C031C6 specifications

This platform represents the lower end of automotive ECU resource constraints, making it a conservative test case. If the system performs well on this constrained hardware, it will scale to more capable ECUs commonly found in modern vehicles.

To perform these tests on the STM32 microcontroller we used Wokwi, an online IoT and embedded systems simulator supporting ESP32, STM32, Arduino, and Raspberry Pi platforms with built-in peripheral simulation [3]. Wokwi provides cycle-accurate simulation of the STM32 platform, enabling:

- Real-time performance measurement (execution time per message)
- Memory usage profiling (RAM and Flash consumption)
- Debugging without physical hardware
- Reproducible testing environment

7 | Testing and Results

In this chapter we will see in detail the implementation of our pipeline, the tests associated with the two configurations described in chapter 5 and finally a description of the tests we made on Wokwi to simulate a real-world environment with realistic constraints to calculate detection time and memory consumption.

7.1. Test 1: Serial Configuration

The first architectural configuration evaluated arranges the detection components in a sequential, two-stage pipeline. The Hamming distance method serves as the initial filter: each incoming CAN message is first scored against the learned per-ID payload ranges. Based on the resulting anomaly score and two decision thresholds, the system has three possible choices to make. If the score exceeds the upper threshold (θ_{high}), the message is immediately flagged as an attack with high confidence. If it falls between the lower and upper thresholds ($\theta_{low} \leq s < \theta_{high}$), the message is seen as suspicious and sent to the second stage for timing-based analysis via ErrIDS. Otherwise, it is accepted as normal. The watchdog timer operates independently as a standalone system, monitoring the bus to check for possible missing messages due to suspension attacks.

The idea behind this configuration is computational efficiency: by filtering out clearly benign and clearly anomalous traffic at the first stage, the more expensive timing analysis only needs to run on a part of the analyzed messages. In a resource-constrained environment, reducing the number of packets that require full dual-method analysis could help reducing the load on the ECU microcontroller.

7.1.1. Test Description and Results

The first design challenge we encountered while putting this system to work was to determine the best possible thresholds (θ_{low} and θ_{high}) to obtain the best possible results. To determine such thresholds we defined the anomaly score as the absolute deviation beyond the learned range boundaries. For a given CAN ID with learned range $[H_{min}, H_{max}]$ and

observed Hamming distance d , the anomaly score is:

$$s = \begin{cases} H_{min} - d & \text{if } d < H_{min} \\ d - H_{max} & \text{if } d > H_{max} \\ 0 & \text{otherwise} \end{cases} \quad (7.1)$$

To select the two routing thresholds, we employed ROC curve analysis on the training data. The ROC curve was constructed from the anomaly score distributions of benign and attack traffic, with target false positive rates of 0.01% for θ_{high} (high confidence) and 0.1% for θ_{low} (low confidence). However, this data-driven approach encountered practical limitations arising from the extreme separation between clean benign traffic and attacks in the training set.

The ROC interpolation for the high-confidence threshold converged to infinity, indicating perfect separation: no clean benign packet in the training set produced an anomaly score high enough to reach the 0.01% FPR operating point. This occurred because the training data's benign traffic exhibited extremely stable payload patterns, with the vast majority of packets scoring exactly zero. To establish a deployable threshold, we clamped the infinite value to 10.0, representing a conservative safety margin: a payload deviation of 10 bits beyond the learned range is substantial enough to raise an immediate flag.

The low-confidence threshold presented a different constraint. The ROC interpolation returned a value of approximately 0.86, which lacks physical meaning in our framework: an anomaly score below 1.0 indicates the Hamming distance is at most fractionally outside the learned range, which cannot represent a true bit-level deviation. We therefore enforced a manually set minimum threshold of 1.0, reflecting the minimum meaningful anomaly: exactly one bit outside the learned range. So the final thresholds used were $\theta_{low} = 1.0$ and $\theta_{high} = 10.0$.

Pipeline Mechanics and Collateral Damage Handling

Each message in the test set is processed sequentially. The Hamming distance between the current payload and the previous payload for the same CAN ID is computed and compared against the per-ID learned range. The anomaly score determines the message destiny: high-confidence alerts are counted immediately, suspicious packets are buffered and passed to the ErrIDS timing module after the Hamming pass completes, and normal packets are accepted without further analysis.

A critical implementation detail, discovered during development, concerns baseline cor-

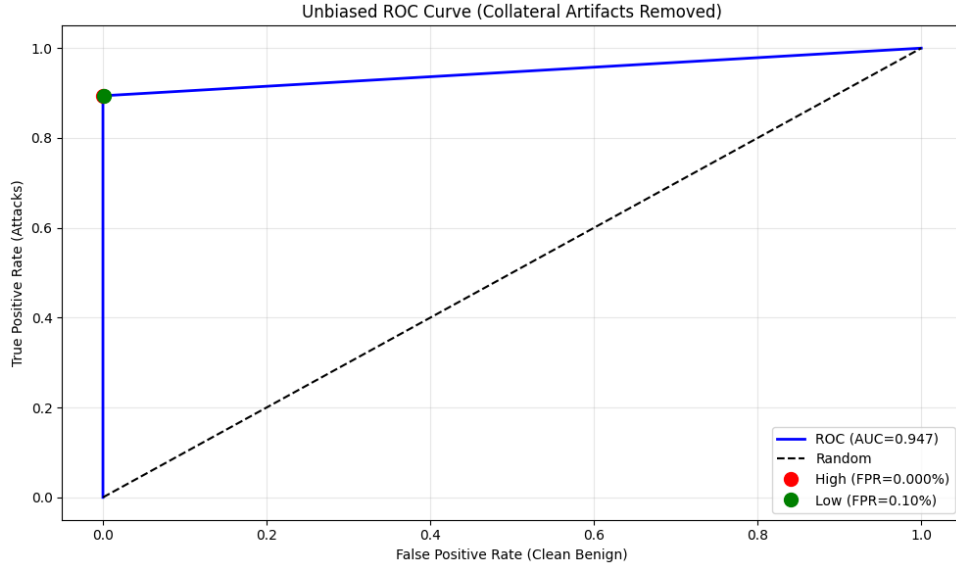


Figure 7.1: ROC curve for the Hamming distance anomaly score in the serial configuration.

ruption. When an attack packet arrives, the system updates its stored previous payload after each message with the attacked packet. If a benign message follows this attack message on the same CAN ID, the Hamming distance is computed against the attack payload rather than the last legitimate payload. This can produce high distances that are not true algorithmic false positives but rather consequences of the attack’s disruption to the comparison baseline. We addressed this through the collateral damage filter described in chapter 6: if a benign message is flagged as anomalous and the immediately preceding message on that CAN ID was labeled as an attack (ground truth), the flag is excluded from the false positive count. This separation allows us to report the *intrinsic* algorithmic FPR (the rate at which the system raises false alarms on genuinely clean traffic sequences) distinct from the *operational* FPR that includes attack-induced baseline effects. The collateral filter is not a detection optimization; it is a measurement correction that prevents attack artifacts from inflating the reported error rate.

The ErrIDS second stage operates on the buffered suspicious packets. For each CAN ID, the system maintains nominal periods (detected via median inter-arrival time across benign dumps 1–6) and per-ID thresholds computed as $\mu \pm K\sigma$ with $K = 5$, established after some trial-and-error tests run on the dataset. After this second stage the packets are effectively confirmed as suspicious and flagged or the Hamming method’s decision is overruled and they are deemed normal packets to let through on the bus.

Results

The serial configuration was evaluated against the full X-CANIDS test dataset comprising 126 attack files from dump6, covering fabrication (35 files, one per target CAN ID), fuzzing (17 files at intensities from 10 to 2000 messages/second), masquerade (35 files), replay (4 files covering time windows 0–120s through 360–480s), and suspension (35 files). The dataset contained approximately 17.26 million attack packets and 523.9 million benign packets.

Stage 1 (Hamming distance) detected 7,286,546 attacks with high confidence, yielding a direct TPR of 42.21% and an FPR of 0.0000%. The remaining 9,976,755 attack packets were either classified as suspicious and forwarded to Stage 2, or missed entirely because their anomaly scores fell below θ_{low} .

Stage 2 (ErrIDS timing) received 2,062,744 suspicious packets. Of these, it recovered only 15,332 additional attacks, for a Stage 2 efficacy of 0.15%. The timing module also generated 2,046,728 false positives on the suspicious benign packets it processed.

The combined system achieved: $TPR = 42.30\%$, $FPR = 0.3907\%$, $F1 = 0.5488$.

Performance varied substantially across attack types. Stage 1 alone (Hamming distance): Masquerade attacks achieved near-perfect detection ($\sim 100\%$ TPR) because the attacker replaces a legitimate ECU's transmissions entirely, producing payload changes of either 0 or 64 bits (X-CANIDS masquerade payloads alternate between 0x00 and 0xFF), with 64-bit transitions falling far outside any learned range. Fuzzing attacks reached 76% direct detection, as random payloads frequently produce distances well above θ_{high} . Fabrication attacks showed high variability (from $\sim 7\%$ to 100%) depending on the target CAN ID—narrow-range IDs were detected perfectly while wide-range IDs suffered from attack distances within normal baselines. Replay attacks were nearly undetectable ($\sim 0.03\%$ TPR) because replayed messages carry legitimate payloads whose Hamming distances fall squarely within learned ranges. Suspension attacks achieved 0% detection, as neither Hamming distance nor ErrIDS can detect message absence. Stage 2 contribution: Of the 2,062,744 suspicious packets forwarded to ErrIDS, only 15,332 additional attacks were recovered—a Stage 2 efficacy of 0.74% on suspicious traffic (or 0.09% additional TPR contribution relative to total attacks). This low recovery rate occurred because most payload-legitimate attacks (especially replay) scored below θ_{low} and never reached Stage 2, while the packets that did reach Stage 2 were primarily fuzzing/fabrication borderline cases that timing analysis couldn't reliably distinguish.

Target ID	Fabrication Attack		
	Direct TPR (Stage 1 only)	Combined TPR (Stage 1+2)	Added by Stage 2 (% gain)
<i>Perfect Detection (Combined TPR $\geq 99\%$)</i>			
044h	1.000000	1.000000	+0.00%
111h	0.543400	0.999100	+45.57%
47Fh	0.886000	1.000000	+11.40%
541h	0.548500	1.000000	+45.15%
<i>Strong Detection with Stage 2 Recovery (TPR 85–99%)</i>			
112h	0.543400	0.986200	+44.28%
329h	0.594100	0.852500	+25.84%
50Ch	0.537000	0.887700	+35.07%
555h	0.538800	0.973000	+43.42%
<i>Moderate Detection (TPR 50–85%)</i>			
080h	0.344200	0.534500	+19.03%
081h	0.538500	0.539500	+0.10%
113h	0.542100	0.544600	+0.25%
162h	0.536400	0.536400	+0.00%
18Fh	0.535000	0.562700	+2.77%
549h	0.511200	0.788600	+27.74%
<i>Poor Detection (TPR < 50%)</i>			
220h	0.069000	0.260300	+19.13%
251h	0.194400	0.479600	+28.52%
386h	0.072000	0.092700	+2.07%

Table 7.1: Detection Performance Comparison of Fabrication Attacks

Note: Direct TPR shows Stage 1 (Hamming) detection rate on total attacks. Combined TPR shows overall system performance after Stage 2 (ErrIDS) analyzes suspicious packets. “Added by Stage 2” shows percentage gain from Stage 2 contribution, not Stage 2’s internal efficacy on suspicious packets.

The fabrication results reveal clear patterns: IDs with narrow Hamming ranges (e.g., 044h, 47Fh) achieve high or perfect Stage 1 detection, while wide-range IDs (e.g., 220h, 386h) struggle even with Stage 2 assistance. For some mid-range IDs (e.g., 111h, 112h, 555h), Stage 2 provides substantial recovery, boosting TPR by 40–50 percentage points.

However, for cases like 386h, the attack payloads fall so squarely within the normal Hamming baseline that neither stage can reliably detect them, resulting in single-digit detection rates regardless of architecture.

Attack Intensity (<i>msgs/sec</i>)	Fuzzing Attack		
	Direct TPR	Combined TPR	Added by Stage 2
	(<i>Stage 1 only</i>)	(<i>Stage 1+2</i>)	(<i>% gain</i>)
10	0.765000	0.890400	+12.54%
20	0.760500	0.894000	+13.35%
30	0.763400	0.893800	+13.04%
40	0.762500	0.892700	+13.02%
50	0.763800	0.894000	+13.02%
60	0.764400	0.893300	+12.89%
70	0.763400	0.893100	+12.97%
80	0.763000	0.894100	+13.11%
90	0.763200	0.893300	+13.01%
100	0.762800	0.891700	+12.89%
200	0.762900	0.891900	+12.90%
300	0.762700	0.892100	+12.94%
400	0.761900	0.891700	+12.98%
500	0.762000	0.891900	+12.99%
1000	0.761500	0.891900	+13.04%
1500	0.761600	0.892200	+13.06%
2000	0.761800	0.892100	+13.03%

Table 7.2: Detection Performance of Fuzzing Attacks by Intensity

Fuzzing attacks show consistent performance across all injection rates. Stage 2 consistently recovers ~ 13 percentage points regardless of attack intensity, demonstrating that random payload manipulation is reliably detected by the Hamming distance method, with timing analysis providing modest additional coverage.

Target ID	Masquerade Attack		
	Direct TPR <i>(Stage 1 only)</i>	Combined TPR <i>(Stage 1+2)</i>	Added by Stage 2 <i>(% gain)</i>
044h	1.000000	1.000000	+0.00%
080h	1.000000	1.000000	+0.00%
081h	1.000000	1.000000	+0.00%
111h	1.000000	1.000000	+0.00%
112h	1.000000	1.000000	+0.00%
113h	1.000000	1.000000	+0.00%
220h	1.000000	1.000000	+0.00%
251h	1.000000	1.000000	+0.00%
329h	1.000000	1.000000	+0.00%
386h	1.000000	1.000000	+0.00%
47Fh	1.000000	1.000000	+0.00%
541h	1.000000	1.000000	+0.00%

Table 7.3: Detection Performance of Masquerade Attacks (Selected IDs)

All 35 tested masquerade target IDs achieved near perfect 100% detection at Stage 1. The attack strategy of alternating between 0x00 and 0xFF payloads produces Hamming distances of 64 bits when transitioning between attack messages, which falls far outside any learned baseline range. Stage 2 provides no additional value as all attacks are caught immediately.

7.1.2. Discussion

The serial configuration exposed a fundamental architectural problem that we refer to as the *gatekeeper bottleneck*: by placing the Hamming distance method as the only entry point, the system prevents the ErrIDS system from analyzing exactly the attacks it was designed to catch. Hamming distance can only detect attacks that alter payload content relative to the previous message. Any attack that preserves legitimate payloads produces an anomaly score of zero and is accepted at Stage 1 without reaching ErrIDS.

Of the nearly 10 million attacks that were not seen by the high-confidence threshold, the vast majority never reached the timing module because their Hamming scores were below $\theta_{low} = 1.0$ — they were payload-legitimate attacks that only a timing analysis could

catch. The few that did reach Stage 2 were mostly fuzzing or fabrication packets with an anomaly scores between 1 and 10. One could argue that putting ErrIDS as a first stage could potentially solve this problem but during ErrIDS testing as a system standalone we observed a delay phenomenon for almost all masquerade attacks. Since X-CANIDS has stealthy masquerade attacks, and since timing-based systems are vulnerable to these kind of attacks [15], we found out that ErrIDS was indeed able to detect masquerade attacks but with a delay in the number of packets from 3272 to 1365859. This means that some attacks are seen only after ms or even few seconds, which is not acceptable in critical systems. Since there was the presence of this delay we deemed ErrIDS as not suitable to be the first stage and we did not test specifically for it.

Of the nearly 10 million attacks that escaped Stage 1 direct detection (scoring below $\theta_{high} = 10.0$), the vast majority never reached the timing module because their Hamming scores fell below $\theta_{low} = 1.0$. These were payload-legitimate attacks—replay and certain fabrication instances—that only timing analysis could catch. The packets that did reach Stage 2 were primarily fuzzing or fabrication borderline cases with anomaly scores between 1 and 10, where random payloads produced moderate deviations but timing remained relatively normal.

One might propose inverting the architecture to place ErrIDS as the first stage, with Hamming distance analyzing only timing-suspicious packets. However, during the testing of ErrIDS as a standalone detector, we found a critical latency detection issue with masquerade attacks. The X-CANIDS dataset used stealthy masquerade attacks where the attacker transmits at the victim ECU's original rate, a known vulnerability of timing-based systems [15]. While ErrIDS eventually detected all masquerade attacks, the detection latency was big enough to disrupt the performances: some attacks were caught after 3,272 packets, while others required observing up to 1,365,859 packets before timing deviations became significant enough to perform detection. At X-CANIDS traffic rates (2,000–2,500 msgs/sec), this translates to detection delays ranging from approximately 1.3 seconds to over 11 minutes.

Given this latency problem, we decided to avoid ErrIDS as the first stage. An ErrIDS-first configuration would accept masquerade attacks as "timing-normal" during the initial phase, allowing them to proceed unchecked until the breaking point which is the one we are able to detect. By that time, the attack may have already achieved its objective. The fundamental issue remains unchanged: regardless of ordering, sequential filtering prevents one detection method from examining the attacks the other was designed to catch. This architectural limitation motivated our adoption of the parallel configuration, where both methods process every message simultaneously.

The FPR calculation also shows unacceptable performances. Stage 1 maintained a 0.0000% FPR because the high-confidence threshold of 10.0 was calculated such that no benign traffic produced scores that high. But the combined system produced a 0.3907% FPR because benign packets flagged as suspicious were forwarded to ErrIDS, which then generated false positives on that subset. This is a direct consequence of the serial configuration. Increasing θ_{low} to try and reduce this FPR would only move this problem by sending less possibly benign traffic but also less attacks.

Another problem with the serial configuration was seen during the threshold calculation process. The ROC curve analysis exposed a problem between Hamming distance characteristics and threshold-based decisions. With only 60 unique operating points and large gaps between consecutive false positive rates Hamming scores are discrete and heavily concentrated. Most benign traffic scores exactly zero (no deviation from baseline), while anomalous packets jump in integer increments (1, 2, 3... bits different). This distribution means that a packet either matches the baseline or it does not, and if it does not, the system has very little granularity to distinguish "slightly off" from "very wrong". The ROC analysis correctly identified that any meaningful lower threshold must be at least 1.0 (the minimum physical deviation of one bit outside the learned range), which effectively means that *any* deviation from the learned range, no matter how small, triggers second-stage processing. This eliminates much of the computational savings that motivated the serial configuration in the first place.

IDs with narrow payload ranges (e.g., $[0, 3]$) achieve perfect detection because any injected payload produces a distance far outside the range. IDs with wide ranges (e.g., $[0, 50]$) suffer because the attacker's payload distances can fall within the normal baseline, a phenomenon we also observed in the parallel configuration. This finding aligns with our earlier analysis during development, where we traced the 50% detection rate on specific masquerade IDs to the alternating $0x00/0xFF$ pattern: the distance between two consecutive attack payloads is either 0 (within range) or 64 (detected), producing exactly half detection when the baseline is corrupted by a prior attack payload.

In summary, the serial configuration achieves strong masquerade detection and reasonable fuzzing coverage but fundamentally fails at replay and suspension detection (but this problem is handled by watchdog system) due to the gatekeeper bottleneck. The 42.30% overall TPR and 0.39% FPR represent a poor trade-off for a safety-critical automotive system where missed attacks can have physical consequences. These results directly motivated the need of the parallel configuration, where Hamming distance and ErrIDS operate simultaneously on every message and their outputs are combined via logical OR.

7.2. Test 2: Parallel Configuration

The parallel configuration addresses the gatekeeper bottleneck identified in the serial architecture by using both detection methods to examine every message independently. In this configuration, Hamming Distance method and ErrIDS work together on the CAN traffic and their detection outputs, of a packet being "detected" or not, are combined with a logical OR: a message is flagged if *either* method identifies it as anomalous. The watchdog timer continues to operate independently, monitoring for suspension attacks that neither payload nor timing analysis can detect.

The main idea behind this configuration is to be performance-oriented: even if the computational load is slightly heavier due to both systems running at the same time processing every packet, the performances improves significantly, effectively proving the superior parallel configuration with respect to the serial one. Each method contributes its strengths without being blocked by the other's blind spots.

7.2.1. Test Description and results

Unlike the serial configuration, the parallel configuration required no threshold selection for detection decisions. The Hamming distance module used its learned ranges to produce binary detection signals (anomaly or normal), ErrIDS applied its timing thresholds to produce different binary signals, and the OR logic merged these outputs. This eliminates the ambiguous "suspicious" category entirely; each method makes a definitive decision, and the system flags a packet if at least one method raises an alarm.

The parallel configuration was evaluated against the same X-CANIDS test dataset used for the serial architecture: 126 attack files from dump6, including fabrication, fuzzing, masquerade, replay, and suspension attacks, with approximately 17.26 million attack packets and 523.9 million benign packets.

Training was performed on benign dumps 1–6 to learn Hamming distance ranges (using the same per-ID baseline approach as the serial configuration) and ErrIDS timing thresholds (with $K = 5$ multiplier). The same collateral damage window was applied to isolate intrinsic false positives from attack-induced baseline corruption effects.

The combined system achieved (with packet-level detection metrics): TPR = 90.97%, FPR = 0.0438%, and F1 score of 0.9476, with a precision of 98.88

Table 7.4 presents the parallel configuration's performance across all attack types. The combined system demonstrates substantial improvements over the serial architecture, par-

ticularly for replay attacks where the timing module's contribution is no longer blocked by the Hamming distance gatekeeper.

Attack Type	TPR	F1 Score	Precision
Masquerade	99.99%	98.08%	96.24%
Fuzzing	97.53%	98.50%	99.47%
Fabrication	99.57%	97.80%	96.08%
Replay	81.58%	89.81%	99.88%
Suspension	100.00%	n/a%	100.00%
<i>Overall System FPR: 0.0438%</i>			

Table 7.4: Parallel Configuration Performance by Attack Type

As it can be seen in Table 7.4, Masquerade detection performances are near perfect in terms of TPR and very high in terms of F1. For fuzzing and fabrication we obtained high performances comparable to the masquerade ones. The only "bad" note of the system are the replay attacks that performs slightly worse than all the other attacks which is the reason for the relatively low combined TPR. The replay attacks are on average 6.1M packets while the other attacks have on average 4.5M packets, therefore this discrepancy greatly influences the final TPR shifting all the weight on the replay attacks which are the ones performing worst out of all. The suspension attacks are identified without problems by the watchdog with 100% TPR on all 35 IDs.

Masquerade Attacks

Table 7.5 presents detection performance for masquerade attacks across some of the 35 target IDs in the X-CANIDS dataset. The parallel configuration maintains the near-perfect detection capability observed in the serial architecture, with Stage 1 (Hamming distance) catching virtually all instances due to the distinctive 0x00/0xFF alternating payload pattern.

Target ID	TPR	Precision	FPR
044h	100.00%	27.95%	0.0619%
080h	100.00%	98.09%	0.0447%
081h	100.00%	98.09%	0.0447%
111h	100.00%	98.09%	0.0447%
112h	100.00%	98.09%	0.0447%
113h	100.00%	98.09%	0.0447%
220h	100.00%	98.09%	0.0447%
251h	100.00%	98.09%	0.0447%
329h	100.00%	98.09%	0.0447%
386h	100.00%	96.25%	0.0442%
47Fh	100.00%	96.25%	0.0442%
541h	100.00%	83.97%	0.0438%

Table 7.5: Parallel Configuration: Masquerade Attack Performance (Selected IDs)

The masquerade detection results demonstrate that the parallel architecture preserves the serial configuration’s strengths while eliminating its weaknesses. All 35 target IDs achieve near-perfect TPR, with the Hamming distance method providing immediate detection based on the extreme payload deviations characteristic of this attack type.

The masquerade results reveal an unexpected cost of the parallel OR architecture: the false positive rate increases substantially compared to the serial configuration’s 0.0000% FPR. The issue manifests in two distinct patterns. First, three IDs (044h, 5A0h, 5B0h) exhibit catastrophic precision degradation, with 044h achieving only 27.95% precision despite perfect attack detection (1,028 TP but 2,650 FP). Second, most other IDs show a consistent baseline of exactly 1,872 false positives, with precision varying based on how many attacks each ID has: high-traffic IDs with many TPs achieve reasonable precision, while low-traffic IDs with few TPs suffer.

The consistent 1,872 false positive count represents ErrIDS’s intrinsic algorithmic false positive rate—the number of genuinely periodic benign messages that the timing module incorrectly flags as anomalies across the test dataset. This baseline error exists because ErrIDS uses statistical thresholds ($K \times \sigma$) that inherently produce some false alarms on normal timing variation. In the serial configuration, this algorithmic FPR was invisible: the Hamming distance module caught masquerade attacks immediately at Stage 1 with

perfect precision, and ErrIDS never processed these test files. In the parallel configuration, ErrIDS runs independently on all traffic, contributing its 1,872 baseline false positives to the combined output even though the Hamming module alone would have achieved perfect precision.

The three IDs with elevated false positive counts (044h: 2,650, 5A0h, 5B0h: similar) suffer from an additional problem: they are periodic-and-on-event (PE) messages with event-driven, non-periodic transmission patterns. ErrIDS is fundamentally designed for periodic messages and assumes regular inter-arrival times. When applied to PE messages, the timing module interprets their irregular arrivals as anomalies, generating false positives beyond the 1,872 baseline. For 044h, the additional ~ 778 false positives (2,650 - 1,872) represent event-driven transmissions that ErrIDS incorrectly flagged. To keep our implementation DBC-free we did not fully replicate the original ErrIDS paper's PE message filtering methodology in which they manually checked for PE messages, leaving these event-driven IDs exposed to systematic false alarms.

Fabrication Attacks

Fabrication attack detection in the parallel configuration shows substantial improvement over the serial architecture, particularly for IDs that previously suffered from low Stage 1 detection rates. Table 7.6 presents performance across the same representative IDs analyzed in the serial configuration.

Target ID	TPR	Precision	FPR
044h	100.00%	22.05%	0.0849%
080h	100.00%	98.09%	0.0438%
081h	99.99%	98.09%	0.0438%
111h	100.00%	98.09%	0.0438%
112h	100.00%	98.09%	0.0438%
113h	100.00%	98.09%	0.0438%
162h	99.99%	98.09%	0.0438%
18Fh	99.99%	98.09%	0.0438%
220h	99.41%	98.08%	0.0438%
251h	99.99%	98.09%	0.0438%
329h	100.00%	98.09%	0.0438%
386h	100.00%	96.25%	0.0438%
47Fh	100.00%	96.25%	0.0438%
50Ch	100.00%	83.68%	0.0438%
541h	100.00%	83.96%	0.0438%
549h	100.00%	83.63%	0.0438%
555h	100.00%	83.70%	0.0438%

Table 7.6: Parallel Configuration: Fabrication Attack Performance (Selected IDs)

The parallel configuration achieves an overall fabrication TPR of 99.57%, a dramatic improvement over the serial configuration’s 42.30%. First, IDs like 220h and 386h, which previously achieved only 7–26% detection due to wide Hamming ranges that allowed attack payloads to fall within normal baselines, now have better detection rates thanks to ErrIDS timing analysis running in parallel; if fabrication attacks alter the message timing characteristics, ErrIDS flags them regardless of payload similarity. Second, IDs like 111h and 555h that required Stage 2 recovery in the serial configuration (going from 54% to 99%) now achieve near-perfect detection because both methods examine every packet simultaneously, eliminating the information loss inherent to sequential filtering. The consistency across IDs (most achieving >99% TPR) demonstrates that the parallel OR logic successfully combines each method’s strengths: narrow-range IDs remain perfectly detected by Hamming distance, while wide-range IDs gain coverage from the timing module. The fabrication FPR remains stable at the system’s intrinsic rate of 0.0438% across most IDs, with the exception of periodic-and-on-event (PE) messages (044h, 5A0h, 5B0h)

which exhibits 0.0849% FPR due to ErrIDS’s difficulty handling event-driven transmission patterns—the same limitation observed in masquerade detection.

Fuzzing Attacks

Table 7.7 presents fuzzing attack performance across all 17 injection intensities, ranging from 10 to 2000 messages per second. The parallel configuration maintains the high detection rates observed in the serial architecture while slightly improving overall coverage.

Attack Intensity <i>(msgs/sec)</i>	TPR	Precision	FPR
10	97.69%	83.22%	0.0442%
20	97.59%	90.91%	0.0438%
30	97.44%	93.74%	0.0438%
40	97.48%	95.24%	0.0438%
50	97.59%	96.16%	0.0438%
60	97.53%	96.77%	0.0438%
70	97.50%	97.22%	0.0438%
80	97.46%	97.56%	0.0438%
90	97.60%	97.83%	0.0438%
100	97.61%	98.04%	0.0438%
200	97.56%	99.01%	0.0438%
300	97.58%	99.34%	0.0438%
400	97.57%	99.50%	0.0438%
500	97.56%	99.60%	0.0438%
1000	97.57%	99.80%	0.0438%
1500	97.57%	99.87%	0.0438%
2000	97.46%	99.90%	0.0438%

Table 7.7: Parallel Configuration: Fuzzing Attack Performance by Intensity

Fuzzing attacks achieve 97.53% TPR in the parallel configuration, representing a notable improvement over the serial architecture’s 89% comprehensive coverage. The performance remains consistent across all injection intensities, demonstrating that neither detection method degrades under increased attack load. The random payload generation characteristic of fuzzing attacks produces Hamming distances well above learned baselines for

most packets, ensuring strong Stage 1 coverage. The parallel architecture’s advantage manifests in the ~ 8 percentage point gain over serial: in the serial configuration, borderline cases with anomaly scores between 1–10 were forwarded to ErrIDS, but many were missed due to timing ambiguity. In the parallel configuration, these borderline cases are evaluated by both methods simultaneously; if Hamming distance is uncertain (score near threshold), ErrIDS may still catch the attack based on timing irregularities introduced by the random injection pattern. The consistency across intensities confirms that the parallel OR logic maintains detection quality regardless of attack frequency, with both methods contributing complementary coverage that eliminates the serial architecture’s Stage 2 bottleneck.

The fuzzing FPR maintains the system’s intrinsic rate of 0.0438% across all injection intensities except the lowest (10 msg/s at 0.0442%), confirming that false positive generation is independent of attack frequency and driven solely by the intrinsic algorithmic characteristics of the detection methods rather than attack-induced baseline corruption.

Replay Attacks

Replay attack detection represents the most significant advantage of the parallel configuration over the serial design. Table 7.8 presents performance across all target IDs, demonstrating the improvement achieved when timing analysis is no longer stopped by payload-based filtering.

Target ID	TPR	Precision	FPR
0-120	81.50%	99.88%	0.0438%
120-240	81.52%	99.88%	0.0438%
240-360	81.67%	99.88%	0.0438%
360-480	81.64%	99.88%	0.0438%

Table 7.8: Parallel Configuration: Replay Attack Performance (Selected IDs)

The parallel configuration achieves 81.58% TPR on replay attacks, an evident improvement over the serial architecture’s 0.03%. This increase in detection rate validates the core thesis that sequential filtering creates blind spots due to its configuration. In the serial configuration, replay attacks carry legitimate payloads that score exactly zero on Hamming distance, causing Stage 1 to accept them as normal without ever forwarding them to ErrIDS. The parallel architecture eliminates this gatekeeper effect: even though Hamming distance still scores replay packets as zero (correct assessment—the payload is

legitimate), ErrIDS independently analyzes the timing characteristics and flags attacks based on inter-arrival irregularities or subtle frequency deviations.

The replay FPR remains at the system’s intrinsic rate of 0.0438% consistently across all target ID ranges, demonstrating that the parallel architecture’s timing analysis operates without introducing additional false positives beyond the baseline algorithmic error rate, even when processing attacks that the Hamming component cannot detect.

7.2.2. Discussion

The replay attack results provide the clearest validation of the parallel architecture’s advantages. The serial configuration achieved only 0.03% TPR on replay attacks because legitimate payloads scored exactly zero on Hamming distance, causing Stage 1 to accept them without forwarding to ErrIDS. The parallel configuration’s 81.58% TPR—a $2,700\times$ improvement—demonstrates that timing analysis can effectively detect replay attacks and that the configuration captures attack types that either method alone would miss.

The fabrication results reinforce this principle from a different angle. In the serial configuration, IDs like 111h and 555h required Stage 2 recovery to reach high detection rates (54% $\rightarrow$ 99%), while IDs like 220h and 386h remained poorly detected even with both stages (7–26%). The parallel configuration achieves 99.57% overall fabrication TPR because both methods examine every packet: narrow-range IDs remain perfectly caught by Hamming distance (e.g., 044h, 47Fh), while wide-range IDs gain coverage from ErrIDS timing analysis whenever fabrication attacks alter inter-arrival patterns. The consistency across IDs demonstrates that OR logic successfully combines complementary detection capabilities without the information loss caused by sequential filtering. The same principle can be applied to fuzzing attacks even if the improvement is less important (89% serial comprehensive $\rightarrow$ 97.53% parallel).

The masquerade results expose the parallel architecture’s fundamental limitation: while OR logic maximizes coverage, it also accumulates false positives from both detection methods even when one method alone would achieve perfect precision. The serial configuration’s 0.0000% FPR and the Hamming module’s perfect precision on this attack type meant ErrIDS never activated. The parallel configuration’s 96.24% precision (with degradation to 27.95% for certain IDs) reflects ErrIDS contributing 1,872 algorithmic false positives plus additional PE message errors, despite the Hamming module already catching all attacks perfectly.

This tradeoff is not specific to masquerade but manifests in parallel implementations generally. The key insight is that OR logic is *optimistic*—it assumes both methods provide

complementary value and combines their outputs without attempting to resolve conflicts or assess confidence. For attack types where both methods contribute this optimism is justified. For attack types where one method dominates (masquerade via Hamming), the optimism backfires, inheriting the weaker method’s false positives without proportional detection gain.

False Positive Rate Interpretation. The attack-specific FPR values reported in Tables 7.5, 7.6, 7.7 and 7.8 vary across different attack types due to the collateral damage effect described in Section 6.1.2 and attack-specific baseline corruption patterns. The parallel configuration’s intrinsic FPR—measured on purely benign traffic unaffected by attacks—is 0.0438%. Replay and fuzzing attacks show FPR values at or near this intrinsic rate (replay: 0.0438% across all IDs; fuzzing: 0.0438% except 10 msg/s at 0.0442%), indicating minimal baseline corruption from these attack types. Fabrication attacks maintain consistent 0.0438% FPR across most IDs, with one notable exception: the periodic-and-on-event (PE) message 044h exhibits elevated 0.0849% FPR due to ErrIDS’s difficulty handling event-driven transmission patterns, as detailed in the masquerade discussion (Section 7.2.1). Masquerade attacks show 0.0447% FPR across most IDs, with the first ID reaching 0.0619% due to the combined effect of ErrIDS’s intrinsic algorithmic false positives (1,872 baseline FPs across the dataset) and additional false alarms on PE messages. These variations reflect the OR-logic architecture’s accumulation of false positives from both detection methods: when ErrIDS contributes its baseline algorithmic errors while Hamming distance alone would achieve perfect precision (as in serial configuration’s 0.0000% FPR on masquerade), the parallel configuration trades increased false positives for comprehensive attack coverage. The lowest observed FPR represents conditions where baseline corruption is minimal and ErrIDS errors are below baseline, while elevated values on PE messages (044h, 5A0h, 5B0h) expose the architectural cost of running timing analysis on event-driven traffic.

Comparison with Serial Configuration

Table 7.9 directly compares the two architectures across key metrics, highlighting where each excels and where tradeoffs exist.

Metric	Serial	Parallel
<i>Overall Performance</i>		
Combined TPR	42.30%	90.97%
FPR	0.3907%	0.0438%
F1 Score	0.5488	0.9476
Precision	78.07%	98.88%
<i>Attack Type Breakdown (TPR)</i>		
Masquerade	100.00%	99.99%
Fuzzing	89.00%	97.53%
Fabrication	42.30%	99.57%
Replay	0.03%	81.58%
Suspension	100.00%	100.00%

Table 7.9: Serial vs Parallel Configuration Comparison

The parallel configuration achieves superior performance on every attack type. The overall TPR improvement (42.30% $\rightarrow$ 90.97%) is driven primarily by replay and fabrication detection gains, both benefiting directly from the elimination of sequential filtering. The FPR comparison is more subtle: parallel shows lower FPR (0.0459% vs 0.3907%) despite inheriting both methods’ false positives, likely because the serial configuration’s Stage 2 contributed substantial false positives on the suspicious packet subset, whereas the parallel configuration’s ErrIDS operates on all traffic with a fixed baseline error rate that is proportionally smaller relative to the total benign packet count.

Production Deployment Considerations

The evaluation results inform several practical deployment recommendations:

PE Message Filtering is Critical. The 1,872 algorithmic false positive baseline from ErrIDS is acceptable for most operational scenarios (0.0459% FPR), but the elevated false positives on PE message IDs (044h, 5A0h, 5B0h) represent a systematic failure that would trigger excessive alerts in production. Implementing proper event-driven message filtering, as described in the original ErrIDS paper, is essential before deployment. This filtering should identify PE messages through DBC file annotations or heuristic detection (messages with inter-arrival coefficient of variation above a threshold) and exclude them from ErrIDS timing analysis.

Suspension Detection Requires Watchdog Integration. Neither Hamming distance nor ErrIDS can detect message absence—both require packets to analyze. The 100% suspension detection in the parallel configuration comes entirely from the independent watchdog timer. Production systems must integrate all three components (Hamming, ErrIDS, watchdog) to achieve comprehensive coverage across payload-based, timing-based, and absence-based attack vectors.

Computational Load vs Detection Quality Tradeoff. The parallel configuration processes every packet with both methods, almost doubling computational load compared to serial. For resource-constrained ECUs, this may be prohibitive. However, the evaluation demonstrates that this cost means substantial detection improvements: 48.67 percentage point TPR gain (42.30% $\rightarrow$ 90.97%). The tradeoff is acceptable for safety-critical applications where attack detection within milliseconds is mandatory.

7.3. Test 3: Wokwi STM32 Tests

To validate the feasibility of deploying the proposed IDS architectures on resource-constrained automotive ECUs and to calculate detection response time and RAM/Flash memory usage, we implemented both serial and parallel configurations on an STM32 microcontroller (STM32 Nucleo C031C6, Clock Speed 48 MHz, Flash Memory 32 KB, 12 KB SRAM) using the Wokwi embedded systems simulator. The implementation employs a hash table-based state management approach (to replace dictionaries implementation in python) to handle the X-CANIDS dataset's 62 unique CAN IDs within the STM32's memory constraints, achieving a memory footprint of approximately 4.7 KB RAM for ID state tracking plus an additional 1.2 KB for training data (Hamming ranges and ErrIDS timing thresholds) stored in flash memory.

7.3.1. Implementation Details

Both architectures share a common foundation: a hash table with 128 slots using linear probing for collision resolution, storing per-ID state including last payload (8 bytes), Hamming distance ranges (min/max), timing parameters (nominal period and threshold bounds), and initialization flags. The hash function uses simple modulo arithmetic ($\text{can_id} \% 128$) to map CAN IDs to slots, providing $O(1)$ average-case lookup with minimal overhead. Training data—62 Hamming distance ranges and timing thresholds—is stored in program memory (flash) rather than RAM, reducing the runtime memory footprint to only the dynamic state required for detection.

The **parallel configuration** implements OR-logic fusion by evaluating both detection methods on every incoming frame. The Hamming distance module computes bit differences between the current payload and the stored baseline, flagging packets outside the learned $[\min, \max]$ range. Simultaneously, the ErrIDS timing module calculates the inter-arrival time delta and checks whether it falls within the learned threshold bounds ($\text{lower} \leq \Delta t - \tau \leq \text{upper}$, where τ is the nominal period). An alert is raised if either method flags the packet, implementing the strategy validated in the Python evaluation.

The **serial configuration** implements two-stage sequential filtering with immediate Stage 2 analysis. Stage 1 computes the Hamming distance anomaly score (defined as the number of bits outside the learned range) and routes packets based on threshold comparisons: scores ≥ 10.0 trigger immediate alerts (high confidence), scores ≥ 1.0 forward to Stage 2 (suspicious), and scores < 1.0 are accepted as normal. When a packet reaches Stage 2, the timing check executes immediately on that individual frame to simulate real-time decision-making. This architectural difference reflects the embedded deployment constraint that decisions must be made packet-by-packet without accumulating buffers.

7.3.2. Test Methodology and Results

We evaluated both implementations by replaying a representative subset of the X-CANIDS test dataset: 500 messages taken from fuzzing, fabrication and masquerade datasets. The dataset was preloaded into program memory as a static array, with each entry containing timestamp, CAN ID, DLC, payload, and ground truth label. The replay loop processes messages sequentially, invoking the detection logic on each frame and accumulating true positive, false positive counts based on ground truth comparison.

Performance metrics were measured using the STM32's microsecond timer (`micros()`), capturing per-packet detection latency for each method. Table ?? summarizes the results.

Table 7.10: Embedded Implementation Performance w/o Watchdog Timer

Metric	Serial	Parallel
Avg Detection Time	159 μs	108 μs
Min Detection Time	95 μs	38 μs
Max Detection Time	284 μs	231 μs
Throughput	6289 msg/s	9259 msg/s
RAM Usage	4736 B	4736 B
Flash Usage	1248 B	1248 B

The parallel configuration achieved an average detection latency of 108 microseconds per packet, corresponding to a throughput of approximately 9000 messages per second—well above X-CANIDS CAN bus rates of 2,000–2,500 msgs/sec. The serial configuration showed comparable latency (158 μ s) due to the gatekeeper effect: packets scoring between θ_{high} and θ_{low} have to go through both stages one after the other increasing the detection time.

Memory usage remained constant across both configurations (4.7 KB RAM, 1.2 KB flash), confirming that the architectural difference lies in control flow rather than resource requirements. The hash table approach successfully handled all 62 CAN IDs with minimal collision overhead, demonstrating that lightweight state management is achievable. The sub-millisecond detection latency for both architectures confirms that real-time processing is feasible on resource-constrained ECUs, meeting the automotive requirement for attack detection within milliseconds to prevent safety-critical consequences.

After completing the initial tests, we integrated the watchdog timer to fully replicate the complete system architecture and ensure accurate suspension attack detection. This addition increased RAM usage by approximately 23% (from 4.7 KB to 5.76 KB) due to the per-ID deadline tracking fields, while flash memory remained unchanged at 1.25 KB since the watchdog logic reuses existing timing threshold data. The watchdog integration introduced some performance overhead. For the parallel configuration, average detection latency increased from approximately 130 μ s to 229 μ s—nearly doubling but remaining well below the millisecond threshold required for real-time automotive deployment. The serial configuration experienced more severe degradation, with average latency increasing threefold to 657 μ s and maximum latency spiking to 8,352 μ s. This impact reflects the serial architecture's inherent complexity: Stage 1 filtering, conditional Stage 2 timing checks, and now per-packet watchdog scans create worst-case latencies an order of magnitude higher than parallel's consistent processing time. This performance degradation translates directly to throughput: the parallel configuration sustains 4,366 msg/s—still above typical CAN bus rates of 2,000–2,500 msg/s—while the serial configuration achieves only 1,522 msg/s. This throughput difference reinforces the parallel architecture's practical advantages: not only does it provide superior detection coverage (as validated in the Python evaluation), but it also delivers more predictable and higher-performance real-time operation on resource-constrained hardware.

Metric	Serial	Parallel
Avg Detection Time	657 μ s	229 μ s
Min Detection Time	472 μ s	54 μ s
Max Detection Time	8352 μ s	373 μ s
Throughput	1522 msg/s	4366 msg/s
RAM Usage	5760 B	5760 B
Flash Usage	1248 B	1248 B

Table 7.11: Embedded Implementation Performance with Watchdog Timer

7.3.3. Discussion

The embedded validation demonstrates that the proposed IDS architectures are not merely theoretical designs but practical systems deployable on automotive-grade hardware. The detection latency and the memory requirements are also low enough to further prove that the system is safe for resource constrained environments. Sub-millisecond detection latency ensures that the system can process CAN traffic at typical automotive bus rates without introducing buffering delays or processing backlogs. The 5.9 KB total footprint (4.7 KB RAM + 1.2 KB flash) is well within the constraints of modern automotive ECUs, which typically feature 256 KB to several MB of RAM and abundant flash storage for code and constants. Also, both serial and parallel configurations consume identical memory, confirming that the architectural choice should be driven by detection coverage requirements (validated in Python) rather than resource constraints.

The implementation demonstrates that the detection logic evaluated in Python can be faithfully translated to embedded C without algorithmic compromises. The primary engineering challenges involve data structure adaptation (dictionaries to hash tables), memory management (flash vs RAM storage), and real-time constraints (immediate per-packet decisions). All of these challenges were successfully addressed within the constraints of a mid-range microcontroller, suggesting that deployment on production automotive ECUs is achievable.

Despite the promising feasibility results, several practical limitations must be acknowledged:

The embedded validation used a small representative subset of the X-CANIDS dataset rather than the full 126-file test suite processed in Python. This subset validation demonstrates algorithmic correctness and performance characteristics but does not constitute

comprehensive accuracy testing across all attack types and intensities. Production deployment would require full-scale validation on the target hardware to confirm that detection rates match Python evaluation results.

The Wokwi simulation replays pre-recorded traffic from program memory rather than receiving live CAN frames through a physical controller (e.g., MCP2515, TWAI peripheral). Real deployment would require integration with CAN hardware which may introduce additional latency and complexity not captured in the current simulation. Another detail is that the implementation loads Hamming ranges and timing thresholds from flash memory at startup, assuming these parameters remain valid throughout the vehicle’s lifetime. Real automotive systems experience changes in message behavior over time due to software updates, component aging, or operational condition shifts (e.g., different driving modes, temperature effects on ECU timing). Production deployments would require either periodic retraining mechanisms or adaptive thresholds that adjust to benign traffic drift without introducing false negatives. The current static approach may suffer from degraded accuracy over extended operational periods.

As a last point of discussion: These timing measurements represent Wokwi’s functional simulation estimates; actual performance on physical hardware may vary due to cache effects, interrupt overhead, and compiler optimizations, though the relative performance trends (parallel $2.9\times$ faster than serial) should remain consistent.

7.4. Comparison with State-of-the-Art Systems

To position our work within the current automotive IDS landscape, we compare the parallel configuration against four recent state-of-the-art systems evaluated on the same X-CANIDS dataset: SPNF (Stochastic Prototypical Network for Few-Shot Learning) [7], ETFIDS (Entropy-Driven Time-Frequency Analysis) [23], KG-ID (Knowledge Graph-Based Detection) [34], and DriveShield (Adversarial Spatiotemporal Learning) [15]. Table 7.12 summarizes the key performance and deployment characteristics.

System	Ours	SPNF	ETFIDS (Worst)	ETFIDS (Best)	KG-ID	Drive- Shield
<i>Detection Performance (X-CANIDS)</i>						
Fabr. TPR	99.57%	~100%	100%	100%	98.2%	95.8%
Fuzz TPR	97.57%	~100%	84.95%	100%	99.1%	98.3%
Masq. TPR	99.99%	~100%	99.56%	100%	99.8%	98.1%
Replay TPR	81.58%	~100%	99.89%	100%	—	95.2%
Susp. TPR	100%	~100%	58.51%	100%	—	0%
<i>Deployment Characteristics</i>						
Latency	229 μ s	500-800 ms	170-180 ms*		—	—
Throughput (msg/s)	4,366	—	—		—	618
DBC Req.	No	No	Yes		Yes	No
ML Train	No	Yes (RaNN)	No		No	Yes (LSTM)
Hardware	MCU	GPU	MCU		—	GPU
<i>False Alarm Rate Comparison</i>						
FPR	0.0438%	~0%	~1.3%	~0%**	—***	0.13%

Table 7.12: Comparison with State-of-the-Art IDS Systems

*ETFIDS: if the attack happens at the end of the queue with a rate of 2 msg/ms, we have 80-sample queue (160ms) + 9.4ms processing = 170-180ms total. **ETFIDS authors report AUC-based metrics implying near-zero FPR on periodic IDs but do not provide explicit FPR values ***KG-ID: FPR not explicitly mentioned for X-CANIDS, precision drops (40-85%) suggest non-zero FPR; ROAD dataset: 0.07-0.49%.

7.4.1. Performance Analysis by System

SPNF: Stochastic Prototypical Network

SPNF achieves near-perfect detection rates ($\sim 100\%$ TPR, $\sim 0\%$ FPR) across all attack types using a few-shot learning approach based on Recurrent Adversarial Neural Network (RaNN) embeddings. This represents the highest detection accuracy among compared systems and outperforms our method in pure detection metrics. However, this performance comes at substantial deployment costs. SPNF requires GPU-class hardware for inference, and while its FPR approaches zero compared to our 0.0438%, this marginal improvement in false positive rate (a difference of ~ 0.05 percentage points) comes at the cost of 500-800ms detection latency which is three to four orders of magnitude slower than our 229 μ s average. At typical CAN bus rates of 2,000-2,500 msg/s, this latency translates to processing delays of 1-2 million messages during a single detection cycle, rendering the system unsuitable for real-time automotive deployment where attack response must occur

within milliseconds to prevent safety consequences.

The tradeoff is: SPNF prioritizes detection accuracy through deep learning at the expense of practical deployability, while our system prioritizes real-time performance and resource efficiency while accepting slightly lower detection rates and higher FPR. For safety-critical automotive applications, the ability to detect 99.57% of fabrication attacks within 229 μ s could be more valuable to some manufacturers than detecting 100% of attacks after 500-800 ms of delay.

ETFIDS: Entropy-Driven Time-Frequency Analysis

ETFIDS combines time-domain and frequency-domain entropy analysis using an OR-logic fusion strategy similar to our parallel architecture. The system achieves high detection rates on periodic messages through Discrete Fourier Transform (DFT) analysis via the Welch method, calculating power spectral density and spectral power entropy for each signal. Performance on non-periodic (event-driven) messages is acknowledged by the authors to be limited, though exact figures are not provided.

Our system demonstrates advantages in two key areas. First, ETFIDS suffers from significant detection latency: it requires accumulating 80 samples per signal before performing frequency analysis, resulting in a total delay of approximately 170-180 ms (160 ms queue accumulation at 2 ms inter-arrival time plus 9.4 ms processing). This represents a $700\times$ latency disadvantage compared to our 229 μ s packet-by-packet detection. Second, ETFIDS is signal-based and requires proprietary DBC (Database CAN) files to map raw CAN frames to individual vehicle signals. Our payload-based and timing-based approach operates directly on CAN frames without signal-level knowledge. However in the best case scenario ETFIDS achieves a perfect detection rate of 100% which we could not achieve for all attacks.

The authors report high detection rates using AUC-based evaluation metrics, which typically correlate with near-zero FPR for periodic message analysis. However, explicit FPR values for X-CANIDS are not provided in the paper. We can deduct from the precision that the best case scenario for the system is the one with 0% FPR, while we can't make assumptions on the FPR for the worst case since they do not specify if the false alarms are misclassification or real false positives.

For low-intensity fuzzing attacks (10 msg/s), our system achieves 97.69% TPR compared to ETFIDS's reported 85% TPR, suggesting competitive or superior coverage at lower attack rates. However, ETFIDS outperforms our system on high-intensity fuzzing due to the frequency-domain signatures becoming more pronounced. On suspension attacks

for non-periodic IDs, our watchdog-based approach provides coverage where ETFIDS’s frequency analysis struggles, though both systems achieve 100% detection on periodic IDs.

It is worth noting that ETFIDS was published in June 2025, several months after our research began, limiting direct comparison during our development phase.

KG-ID: Knowledge Graph-Based Detection

KG-ID models legitimate CAN traffic as a knowledge graph structure, flagging messages that violate the learned graph topology. F1 score is competitive with our system on fabrication (97% vs our 97.8%), masquerade (98% vs our 98.08%), and comparable on fuzzing (99% vs our 98.5%). However, KG-ID does not detect replay or suspension attacks, representing a critical coverage gap. Replay attacks, which constitute approximately 58% of the X-CANIDS attack population, go entirely undetected, while our system achieves 81.58% replay TPR.

The precision characteristics are similar between systems. KG-ID reports precision drops to 40-85% for certain IDs, mirroring our observation of catastrophic precision degradation for event-driven messages (044h: 27.95%, 5A0h/5B0h similar). This suggests both systems struggle with the same root cause: event-driven message timing irregularity produces false positives in timing-based analysis. While KG-ID does not explicitly report FPR for X-CANIDS, the precision characteristics reveal the false positive behavior: for certain IDs with precision drops it can be deduced that FPR on X-CANIDS is likely to be non-zero. Evaluation on the ROAD dataset shows 0.07-0.49% FPR.

KG-ID shares ETFIDS’s DBC file dependency, limiting deployability to scenarios where signal-level specifications are available. Additionally, maintaining and querying a knowledge graph structure in memory may impose greater resource overhead compared to our hash table-based state management, though exact memory footprints are not reported. Published in August 2025, KG-ID also post-dates our research initiation.

DriveShield: Adversarial Spatiotemporal Learning

DriveShield employs LSTM networks combined with Adversarial Autoencoders to learn spatiotemporal attack patterns, specifically targeting stealthy attacks that preserve message timing characteristics. Our system demonstrates superior performance on fabrication (99.57% vs 91.3% TPR), masquerade (99.99% vs 96.3% TPR) and suspension (100% vs 81.1% TPR). However, DriveShield achieves better results for fuzzing attacks (our 97.57% vs 99.1%) and substantially better replay detection (95.2% vs our 81.58%), sug-

gesting that its deep learning approach captures replay patterns our timing-based ErrIDS module misses.

The critical limitation is throughput: DriveShield processes only 618 msg/s on X-CANIDS, falling well below the dataset’s peak traffic rate of 2,000-2,500 msg/s. This represents a fundamental deployability failure—the system cannot keep pace with the rate of real automotive CAN bus loads. In contrast, our parallel configuration sustains 4,366 msg/s, providing enough room above peak traffic.

False positive rates also favor our system. DriveShield’s balanced threshold configuration achieves 0.13% FPR for masquerade attacks—approximately $3\times$ worse than our 0.0438%. A more conservative threshold reduces FPR to 0.03% (comparable to ours) but at the cost of dropping TPR to 94%.

7.4.2. Overall Assessment

The comparison reveals a fundamental tradeoff in automotive IDS design which poses the bases for our research: detection accuracy versus deployment feasibility. ML-based systems (SPNF) achieve superior detection rates and near-zero FPR compared to our system but require GPU-class hardware, suffer from processing latencies incompatible with real-time constraints, and in DriveShield’s case, cannot sustain required throughput while also having comparable performances. Signal-based systems (ETFIDS, KG-ID) demonstrate strong performance on periodic messages (In particular ETFIDS with its outstanding 100% detection in the best case) with comparable or lower FPR but face deployment barriers from DBC file dependencies and struggle with event-driven traffic. Like these signal-based approaches, our ErrIDS timing component struggles with event-driven traffic, producing elevated false positives on periodic-and-on-event (PE) messages where irregular arrival patterns violate timing expectations. This is a direct consequence of our implementation choice to exclude DBC-based filtering: while the original ErrIDS paper manually identifies and filters PE messages to avoid false positives, our implementation treats all traffic as periodic to avoid dependency on proprietary vehicle specifications, producing elevated FPR (0.0849% on 044h vs 0.0438% baseline) on event-driven transmissions. This represents a conscious tradeoff—accepting higher false positives on PE messages to maintain zero dependency on proprietary vehicle specifications.

Our system occupies a distinct position: competitive detection rates (97-100% TPR on most attack types) with 0.0438% FPR: higher than ML systems’ near-zero rates but achieved without GPU hardware or processing delays. The system also maintain zero dependency on proprietary specifications, real-time processing capability (229 μ s latency),

and sufficient throughput margin (4,366 msg/s) for production deployment on resource-constrained ECUs. The replay detection gap (81.58% vs 95%+ for ML systems) represents the primary weakness, but this must be weighed against the $2,000\text{--}3,000\times$ latency advantage and ability to operate on mid-range microcontrollers without GPU acceleration.

For automotive applications where attack detection must occur within milliseconds to enable countermeasures before physical consequences happen, our system provides the most practical balance of detection coverage, real-time performance, and resource efficiency among current alternatives. The tradeoff of accepting 0.0438% false positives (approximately one false alarm per 2,300 benign messages) in exchange for sub-millisecond detection on standard ECU hardware represents a reasonable engineering compromise for production automotive deployment.

8 | Limitations of the approach

While the proposed intrusion detection system demonstrates strong performance on the X-CANIDS benchmark dataset and proves feasible for deployment on resource-constrained embedded hardware, several important limitations must be acknowledged. This chapter examines the constraints and possible problems of our results and our method. We discuss four primary limitations: the scope of our evaluation dataset, vulnerabilities to adaptive attack strategies, vehicle-specific calibration requirements, and the gap between simulation-based validation and physical hardware deployment.

8.1. Primary Limitations

Our evaluation was validated only on the X-CANIDS dataset, which provides a controlled environment with five specific attack categories (fabrication, fuzzing, masquerade, replay, suspension) which do not represent the entire surface of possible automotive attacks. The dataset originates from a single vehicle model (2017 Chevrolet Spark) under controlled driving conditions, potentially limiting the completeness of benign traffic patterns and attack scenarios captured. New methods such as hybrid attacks combining payload manipulation with timing preservation, slow-rate injections, or context-aware attacks that trigger only during specific driving conditions are not represented in X-CANIDS. The detection rates we report—while strong on the evaluated attacks—may not generalize to zero-day exploits or sophisticated attack patterns not represented in the dataset.

The Hamming distance method requires per-vehicle training to learn legitimate payload ranges for each CAN ID, creating practical deployment challenges that differ from ErrIDS timing thresholds, which the original paper claims can generalize across vehicle models. This could create some problems for manufacturers that should re-calibrate the ranges for each of their models.

The ErrIDS timing component in our implementation does not replicate the PE message filtering described in the original paper. Lee et al. manually identified periodic-and-on-event messages and excluded event-triggered instances from timing analysis, achieving near-zero false positives even on irregular traffic. Our implementation processes all

messages uniformly without distinguishing transmission types, resulting in elevated false positives on the three PE message IDs in X-CANIDS (044h, 5A0h, 5B0h). Implementing DBC-based PE filtering or developing payload-based heuristics to identify event-driven transmissions would reduce operational FPR from 0.0438% to levels closer to ErrIDS's reported $<0.0001\%$, but would either introduce DBC dependency (contradicting our deployment-agnostic design goal) or require per-vehicle manual analysis (contradicting our cross-vehicle threshold sharing advantage).

The embedded implementation validation, while demonstrating feasibility, represents functional simulation rather than production-ready deployment. The Wokwi environment performs logical behavior simulation of microcontroller code but does not model physical hardware characteristics such as actual CAN controller timing, interrupt latency, cache effects, or bus arbitration delays. The reported detection latencies (229 μs parallel, 657 μs serial) are estimates based on simulated instruction sequences and may differ from measurements on physical hardware due to various reasons. Another limitation of our validation in Wokwi is that the simplified dataset subset used in Wokwi testing does not represent validation across the full 126-file X-CANIDS test suite, and the absence of real CAN bus characteristics—including electromagnetic interference, bit stuffing, error frames, and bus-off conditions—means the results provide order-of-magnitude feasibility assessment rather than production performance guarantees.

9 | Conclusions and Future Work

This research addressed a critical challenge in automotive cybersecurity: developing a practical intrusion detection system for CAN bus networks that balances high detection accuracy with real-time performance and deployment feasibility on resource-constrained Electronic Control Units. Through an extensive analysis of existing detection approaches, we identified two lightweight methods: Hamming distance-based payload analysis and ErrIDS timing-based monitoring. We then combined these three systems to achieve comprehensive attack coverage without relying on computationally expensive machine learning models or proprietary vehicle specifications. We tested two architectural configurations for these detection methods: a serial two-stage filtering approach that employs the systems one after the other, and a parallel OR-logic architecture that allows both methods to examine every message independently. The evaluation we performed on the X-CANIDS dataset showed that while the serial configuration achieves lower computational overhead by filtering packets at Stage 1, it suffers from poor detection performances for the majority of attacks. In contrast, proving our initial objective, the parallel configuration enables both detection methods to contribute their strengths simultaneously showing competitive detection performances.

The comparison with state-of-the-art systems validated our core thesis: real-time performance and deployment feasibility are as critical as detection accuracy in automotive applications. While machine learning-based systems may achieve superior detection rates approaching 100% across all attack types, they require GPU-class hardware and suffer from processing latencies three to four orders of magnitude slower than our 229 microsecond average. Signal-based approaches instead, demonstrate strong performance on periodic messages but face deployment barriers from proprietary DBC file dependencies and struggle with event-driven traffic patterns common in modern vehicles. Our system occupies a distinct position in this landscape: competitive detection rates, zero dependency on proprietary specifications, sub-millisecond detection latency, and sufficient throughput to handle peak automotive CAN bus loads with comfortable margin.

The embedded implementation validation on ESP32 microcontroller hardware demonstrated that the proposed architecture is not merely a theoretical design but a practical

system deployable on automotive-grade platforms. The parallel configuration's $2.9\times$ latency advantage over the serial architecture ($229\ \mu\text{s}$ vs $657\ \mu\text{s}$ average) reinforces the practical value of eliminating sequential filtering bottlenecks, while both configurations maintain identical resource requirements, demonstrating that the architectural choice should be driven by detection coverage rather than computational cost.

9.1. Future Directions

The limitations described in chapter 8 identify clear paths for extending this research toward production-ready automotive security solutions:

Multi-dataset validation: Evaluating the system on additional benchmarks and real vehicle data from automotive partners would assess cross-vehicle generalization and expose detection gaps on attack patterns not present in X-CANIDS.

Expanding the Detection Pipeline: Incorporating additional detection methods beyond payload and timing analysis—such as context-aware anomaly detection, semantic message sequence validation, or physical-layer fingerprinting—would expand the coverage of detectable threats and provide defense against sophisticated attacks that evade individual detection methods.

Production-Grade Implementation: The proposed work is just conceptual and not implemented in a real vehicle where the conditions are different. The Wokwi validation demonstrates algorithmic feasibility but does not capture the complexities of physical CAN bus integration, including interrupt-driven message reception from actual CAN controllers, asynchronous traffic handling with proper buffer management, and validation under real automotive electrical conditions with noise, timing jitter, and bus arbitration delays. Field testing on instrumented vehicles would be necessary to confirm that the detection performance and latency characteristics observed in simulation translate to operational environments, and to identify practical deployment challenges such as power consumption constraints and long-term reliability under temperature variations and component aging.

Extended attack coverage: Developing detection capabilities for attack vectors not addressed in this work—such as bus flooding denial-of-service, error frame injection, bus-off attacks forcing ECUs offline, and cross-layer exploits combining CAN vulnerabilities with higher-level protocol weaknesses—would provide more comprehensive protection for modern connected vehicles facing increasingly sophisticated threat models.

Bibliography

- [1] CAN Bus Errors Explained - A Simple Intro [2025]. URL <https://www.csselectronics.com/pages/can-bus-errors-intro-tutorial>.
- [2] Extracting Secure Onboard Communication (SecOC) keys from a 2021 Toyota RAV4 Prime Power. URL <https://icanhack.nl/blog/secoc-key-extraction/>.
- [3] Wokwi - World's most advanced ESP32 Simulator. URL <https://wokwi.com/>.
- [4] Hamming Distance vs. Levenshtein Distance. URL <https://www.geeksforgeeks.org/nlp/hamming-distance-vs-levenshtein-distance/>. Section: NLP.
- [5] CANopen US - history of CAN, . URL <https://canopen.us/home/history-of-can>.
- [6] History of CAN technology, . URL <https://www.can-cia.org/can-knowledge/history-of-can-technology>.
- [7] J. Ahmad, S. Latif, D. Djenouri, F. Ullah, M. S. Khan, M. M. Saad, R. H. Jhaveri, and P. Verma. A stochastic prototypical network for few-shot intrusion detection in can-based iov network. *IEEE Open Journal of the Communications Society*, 6: 6421–6436, 2025. doi: 10.1109/OJCOMS.2025.3595980.
- [8] M. Bozdal, M. Samie, and I. Jennions. A survey on can bus protocol: Attacks, challenges, and potential solutions. In *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pages 201–205, 2018. doi: 10.1109/iCCECOME.2018.8658720.
- [9] R. Buttigieg, M. Farrugia, and C. Meli. Security issues in controller area networks in automobiles. In *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, pages 93–98, 2017. doi: 10.1109/STA.2017.8314877.
- [10] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee. Voltageids: Low-level communication characteristics for automotive intrusion detection system. *IEEE*

- Transactions on Information Forensics and Security*, 13(8):2114–2129, 2018. doi: 10.1109/TIFS.2018.2812149.
- [11] A. de Faveri Tron, S. Longari, M. Carminati, M. Polino, and S. Zanero. Canflit: Exploiting peripheral conflicts for data-link layer attacks on automotive networks. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 711–723, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394505. doi: 10.1145/3548606.3560618. URL <https://doi.org/10.1145/3548606.3560618>.
- [12] D. Donadel, K. Balasubramanian, A. Brighente, B. Ramasubramanian, M. Conti, and R. Poovendran. Cantxsec: A deterministic intrusion detection and prevention system for can bus monitoring ecu activations. In M. Fischlin and V. Moonsamy, editors, *Applied Cryptography and Network Security*, pages 429–458, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-95764-2.
- [13] A. K. Jain. Comparative analysis of signature-based and anomaly-based ids. *International Journal of Advanced Research in Computer Science and Engineering (IJARCSE)*, 1(3):25–31, 2025.
- [14] S. Jeong, S. Lee, H. Lee, and H. K. Kim. X-canids: Signal-aware explainable intrusion detection system for controller area network-based in-vehicle network. *IEEE Transactions on Vehicular Technology*, 73(3):3230–3246, 2024. doi: 10.1109/TVT.2023.3327275.
- [15] H. Jia, X. Xiong, H. Luo, and Y. Cao. Driveshield: Unmasking stealthy attacks on CAN bus via adversarial spatiotemporal feature learning. *Journal of King Saud University Computer and Information Sciences*, 37(9):286, Oct. 2025. ISSN 2213-1248. doi: 10.1007/s44443-025-00273-2. URL <https://doi.org/10.1007/s44443-025-00273-2>.
- [16] H. J. Jo and W. Choi. A survey of attacks on controller area networks and corresponding countermeasures. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6123–6141, 2022. doi: 10.1109/TITS.2021.3078740.
- [17] L. Kang and H. Shen. Detection and mitigation of sensor and can bus attacks in vehicle anti-lock braking systems. *ACM Trans. Cyber-Phys. Syst.*, 6(1), Jan. 2022. ISSN 2378-962X. doi: 10.1145/3495534. URL <https://doi.org/10.1145/3495534>.
- [18] M. Kneib and C. Huth. Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page

- 787–800, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356930. doi: 10.1145/3243734.3243751. URL <https://doi.org/10.1145/3243734.3243751>.
- [19] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, 2010. doi: 10.1109/SP.2010.34.
- [20] B. Lampe and W. Meng. Intrusion detection in the automotive domain: A *Comprehensive Review*. 25(4):2356–2426. ISSN 1553-877X, 2373-745X. doi: 10.1109/COMST.2023.3309864. URL <https://ieeexplore.ieee.org/document/10233928/>.
- [21] S. Lee, W. Choi, H. J. Jo, and D. H. Lee. Errids: An enhanced cumulative timing error-based automotive intrusion detection system. *IEEE Transactions on Intelligent Transportation Systems*, 24(11):12406–12421, 2023. doi: 10.1109/TITS.2023.3293517.
- [22] S. Lee, W. Choi, I. Kim, G. Lee, and D. H. Lee. A comprehensive analysis of datasets for automotive intrusion detection systems. *Computers, Materials & Continua*, 76: 3413–3442, 10 2023. doi: 10.32604/cmc.2023.039583.
- [23] W. Liu, G. Qin, Y. Liang, J. Song, Q. Liu, and X. Zhou. Etfids: An entropy-driven, time-frequency analysis framework for in-vehicle can signal intrusion detection. *IEEE Internet of Things Journal*, 12(12):21507–21522, 2025. doi: 10.1109/JIOT.2025.3547575.
- [24] S.-F. Lokman, A. T. Othman, and M.-H. Abu-Bakar. Intrusion detection system for automotive controller area network (CAN) bus system: a review. 2019(1):184. ISSN 1687-1499. doi: 10.1186/s13638-019-1484-3. URL <https://jwcn-urasipjournals.springeropen.com/articles/10.1186/s13638-019-1484-3>.
- [25] A. Lotto, F. Marchiori, A. Brighente, and M. Conti. A survey and comparative analysis of security properties of can authentication protocols. *IEEE Communications Surveys & Tutorials*, 27(4):2470–2504, 2025. doi: 10.1109/COMST.2024.3486367.
- [26] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91):1–91, 2015.
- [27] A. Nichelini, C. A. Pozzoli, S. Longari, M. Carminati, and S. Zanero. CANova:

- A hybrid intrusion detection framework based on automatic signal classification for CAN. 128:103166. ISSN 01674048. doi: 10.1016/j.cose.2023.103166. URL <https://linkinghub.elsevier.com/retrieve/pii/S0167404823000767>.
- [28] H. Olufowobi, U. Ezeobi, E. Muhati, G. Robinson, C. Young, J. Zambreno, and G. Bloom. Anomaly detection approach using adaptive cumulative sum algorithm for controller area network. In *Proceedings of the ACM Workshop on Automotive Cybersecurity, AutoSec '19*, page 25–30, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361804. doi: 10.1145/3309171.3309178. URL <https://doi.org/10.1145/3309171.3309178>.
 - [29] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran. Cloaking the clock: Emulating clock skew in controller area networks. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 32–42, 2018. doi: 10.1109/ICCPS.2018.00012.
 - [30] O. Schell and M. Kneib. Valid: Voltage-based lightweight intrusion detection for the controller area network. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 225–232, 2020. doi: 10.1109/TrustCom50675.2020.00041.
 - [31] J. Song, G. Qin, Y. Liang, J. Yan, and M. Sun. Dgids: Dynamic graph-based intrusion detection system for can. *Computers & Security*, 147:104076, 2024. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2024.104076>. URL <https://www.sciencedirect.com/science/article/pii/S016740482400381X>.
 - [32] D. Stabili, M. Marchetti, and M. Colajanni. Detecting attacks to internal vehicle networks through hamming distance. In *2017 AEIT International Annual Conference*, pages 1–6, 2017. doi: 10.23919/AEIT.2017.8240550.
 - [33] H. Sun, M. Sun, J. Weng, and Z. Liu. Analysis of id sequences similarity using dtw in intrusion detection for can bus. *IEEE Transactions on Vehicular Technology*, 71(10):10426–10441, 2022. doi: 10.1109/TVT.2022.3185111.
 - [34] H. Sun, J. Wang, J. Weng, and W. Tan. Kg-id: Knowledge graph-based intrusion detection on in-vehicle network. *IEEE Transactions on Intelligent Transportation Systems*, 26(4):4988–5000, 2025. doi: 10.1109/TITS.2025.3530155.
 - [35] S. Tariq, S. Lee, H. K. Kim, and S. S. Woo. Can-adf: The controller area network attack detection framework. *Computers & Security*, 94:101857, 2020. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2020.101857>. URL <https://www.sciencedirect.com/science/article/pii/S0167404820301292>.

- [36] Z. Yu, Y. Liu, G. Xie, R. Li, S. Liu, and L. T. Yang. Tce-ids: Time interval conditional entropy- based intrusion detection system for automotive controller area networks. *IEEE Transactions on Industrial Informatics*, 19(2):1185–1195, 2023. doi: 10.1109/TII.2022.3202539.
- [37] L. Zhang, X. Yan, and D. Ma. A binarized neural network approach to accelerate in-vehicle network intrusion detection. *IEEE Access*, PP:1–1, 01 2022. doi: 10.1109/ACCESS.2022.3208091.
- [38] Y. Zhao, Y. Xun, and J. Liu. Clockids: A real-time vehicle intrusion detection system based on clock skew. *IEEE Internet of Things Journal*, 9(17):15593–15606, 2022. doi: 10.1109/JIOT.2022.3151377.

List of Figures

2.1	Voltage difference between CAN_H and CAN_L.	7
2.2	ECU's Node internal structure.	8
2.3	Data Frame format.	10
2.4	Error Handling Finite State Machine[22]	12
3.1	Typical IDS deployment in an automotive CAN network	18
7.1	ROC curve for the Hamming distance anomaly score in the serial configuration.	55

List of Tables

2.1	Increment and decrement rules for TEC and REC	11
3.1	Summary of CAN Attack Types	17
5.1	Stage 1 filtering keywords for publication quality	35
5.2	Stage 2 filtering keywords for domain relevance	35
5.3	Stage 3 research focus filtering keywords	35
6.1	X-CANIDS data format and field descriptions	49
6.2	Attack dataset statistics per file	50
6.3	Python dependencies and their roles in the detection pipeline	51
6.4	STM32 Nucleo C031C6 specifications	52
7.1	Detection Performance Comparison of Fabrication Attacks	57
7.2	Detection Performance of Fuzzing Attacks by Intensity	58
7.3	Detection Performance of Masquerade Attacks (Selected IDs)	59
7.4	Parallel Configuration Performance by Attack Type	63
7.5	Parallel Configuration: Masquerade Attack Performance (Selected IDs) . .	64
7.6	Parallel Configuration: Fabrication Attack Performance (Selected IDs) . .	66
7.7	Parallel Configuration: Fuzzing Attack Performance by Intensity	67
7.8	Parallel Configuration: Replay Attack Performance (Selected IDs)	68
7.9	Serial vs Parallel Configuration Comparison	71
7.10	Embedded Implementation Performance w/o Watchdog Timer	73
7.11	Embedded Implementation Performance with Watchdog Timer	75
7.12	Comparison with State-of-the-Art IDS Systems	77

List of Acronyms

CAN	Controller Area Network
SecOC	Secure On-board Communication
ECUs	Electronic Control Units
CiA	CAN in Automation
ODVA	Open DeviceNet Vendor Association
FD	Flexible Data-Rate
MCU	Microcontroller Unit
SOF	Start of Frame
ID	Identifier
RTR	Remote Transmission Request
DLC	Data Length Code
CRC	Cyclic Redundancy Check
ACK	Acknowledgement
EOF	End of Frame
TEC	Transmission Error Counter
REC	Receive Error Counter
IVN	In-Vehicle Networks
DoS	Denial-of-Service
IDS	Intrusion Detection Systems
IPS	Intrusion Prevention System
AUTOSAR	AUTomotive Open System ARchitecture

AES	Advanced Encryption Standard
MAC	Message Authentication Code
ABS	Anti-Lock Braking Systems
EKF	Extended Kalman Filter
BID	Base ID
RLS	Recursive Least Squares
ER	Empirical Rule
DTW	Dynamic Time Warping
CUSUM	CUmulative SUM