Universidade do Minho
Escola de Engenharia
Departamento de Informática

Bruno Alves Martins Carvalho

Automated and Intelligent Hacking Detection System

July 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Bruno Alves Martins Carvalho

Automated and Intelligent Hacking Detection System

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
José Bacelar Almeida
Luís Sobral Silva

July 2022

ACKNOWLEDGEMENTS

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

1

ABSTRACT

The Controller Area Network (CAN) in the backbone of automotive networking, connecting many Electronic Control Units (ECUs) that control virtually every vehicle function from fuel injection to parking sensors. It possesses, however, no security functionality such as message encryption or authentication by default. Attackers can easily inject or modify packets in the network, causing vehicle malfunction and endangering the driver and passengers. There is an increasing number of ECUs in modern vehicles, primarily driven by the consumer's expectation of more features and comfort in their vehicles as well as ever-stricter government regulations on efficiency and emissions. Combined with vehicle connectivity to the exterior via Bluetooth, Wi-Fi, or cellular, this raises the risk of attacks. Traditional networks, such as Internet Protocol (IP), typically have an Intrusion Detection System (IDS) analysing traffic and signalling when an attack occurs. The system here proposed is an adaptation of the traditional IDS into the CAN bus using a One Class Support Vector Machine (OCSVM) trained with live, attack-free traffic. The system is capable of reliably detecting a variety of attacks, both known and unknown, without needing to understand payload syntax, which is largely proprietary and vehicle/model dependent. This allows it to be installed in any vehicle in a plug-and-play fashion while maintaining a large degree of accuracy with very few false positives.

KEYWORDS intrusion detection system, machine learning, controller area network, connected vehicles

RESUMO

A CAN é a principal tecnologia de comunicação interna automóvel, ligando muitas ECUs que controlam virtualmente todas as funções do veículo desde injeção de combustível até aos sensores de estacionamento. No entanto, não possui por defeito funcionalidades de segurança como cifragem ou autenticação. É possível aos atacantes facilmente injetarem ou modificarem pacotes na rede causando estragos e colocando em perigo tanto o condutor como os passageiros. Existe um número cada vez maior de ECUs nos veículos modernos, impulsionado principalmente pelas expectativas do consumidores quanto ao aumento do conforto nos seus veículos, e pelos cada vez mais exigentes regulamentos de eficiência e emissões. Isto, associada à conexão ao exterior através de tecnologias como o Bluetooth, Wi-Fi, ou redes móveis, aumenta o risco de ataques. Redes tradicionais, como a rede IP, tipicamente possuem um IDSs que analiza o tráfego e assinala a presença de um ataque. O sistema aqui proposto é uma adaptação do IDS tradicional à rede CAN utilizando uma OCSVM treinada com tráfego real e livre de ataques. O sistema é capaz de detetar com fiabilidade uma variedade de ataques, tanto conhecidos como desconhecidos, sem a necessidade de entender a sintaxe do campo de dados das mensagens, que é maioritariamente proprietária. Isto permite ao sistema ser instalado em qualquer veículo num modo *plug-and-play* enquanto mantém um elevado nível de desempenho com muito poucos falsos positivos.

PALAVRAS-CHAVE sistema de deteção de intrusão, aprendizagem máquina, controller area network, veículos conectados

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

INTRODUCTORY MATERIAL

2

INTRODUCTION

Computation and networking were first introduced in the 1970s out of the manufacturer's necessity to meet new and stringent emissions regulations (**?**). Then, the concept of connected cars appeared in Formula 1 in 1980, with the vehicle being able to transmit data to the pitlane (**?**). In 1996, General Motors introduced its OnStar roadside assistance program, where the vehicle could register an accident and automatically call the nearest emergency centre (**?**). The use of Global Positioning System (GPS) by civillians was allowed in May of 2000, which not only improved navigation but also allowed stolen vehicles to be tracked (**?**). Remote diagnostics were introduced in 2001 and, since then, we saw the inclusion of vehicle health reports and 4G Long-Term Evolution (LTE) networking. Manufacturers are currently working on adding 5G connectivity to their vehicles. This networking ability makes the modern vehicle a part of the Internet of Things (IoT). Not only are vehicles more connected than ever, but they also have much more computing power. Many modern vehicles offer at least some driving assistance functionality and make decisions to avoid accidents.

This increase in functionality means that a modern car contains about 100 million lines of software code, and is expected to have around 300 million lines of code by 2030. For comparison, a passenger plane has around 15 million (**?**). This leaves a lot of room for vulnerabilities to appear, and many attacks have been successfully demonstrated. These include not only locking and unlocking the vehicle, and information theft, but also disruption of safety-critical components such as sensors and brakes (**?**). Manufacturers have, however, failed to keep up with the cybersecurity needs of the vehicles they produce. The two main reasons for this issue are that first, automakers specialised in the manufacturing of vehicles and not in software engineering, and second, there is an increased cost to making secure software (**?**).

Here, a prototype of an IDS for CAN, which is currently the dominant networking protocol in the automotive industry, is presented. IDSs are common in traditional IP networks, but their implementation in CAN is still being explored. Most proposed IDS methodologies are very resource heavy, with complex deep learning models often being implemented (**?**). This results in lower energy efficiency, which is of major importance to both the manufacturers and the customer. The proposed system can perform blind network traffic analysis and signal when an attack is occurring while being constrained by the limited computing resources that are present in a vehicle as well as its efficiency goals.

3

BACKGROUND

3.1 MODERN VEHICLES

In recent years, vehicles have become ever smarter, providing functionality that goes beyond simple transportation of passengers and goods. They can guide us through GPS, make calls, receive over-the-air updates, inform other vehicles of their position, and even drive themselves. Having this computational capacity, being identifiable, able to transfer data over a network and connected to each other makes modern vehicles a part of the Internet of Things (**?**).

3.1.1 *Architecture*

An overview of the architecture of a modern vehicle can be seen in Figure **??**. It is common practice to have several domains connected by a central gateway ECU.

In all of these domains, ECUs are responsible for controlling their components, such as precise fuel combustion in the powertrain domain or air conditioning in the body control domain. These controllers are often based on the ARM platform, with an automotive-grade product line generally being used due to unique constraints in the automotive environment.

In 2003, the AUTOSAR was established as a "worldwide development partnership of vehicle manufacturers, suppliers, service providers and companies from the automotive electronics, semiconductor and software industry" with the objective of creating an open and standardised software architecture for ECU. The use of such a manufacturer independent standard allows for a reduction in development effort and improved software quality, as software modules can be used in vehicles of different manufacturers and components of suppliers, thereby reducing complexity and expense. AUTOSAR maintains five standards: the AUTOSAR Classic Platform, the AUTOSAR Adaptive Platform, the Foundation, AUTOSAR acceptance tests, and AUTOSAR application interface.

The AUTOSAR Classic architecture is shown in Figure **??**. It is composed of three software layers which run on a microcontroller: application, Runtime Environment (RTE), and Basic Software (BSW). The RTE handles communication between the application layer, which is mostly hardware independent, and the BSW. The BSW is then divided intro three layers, along with complex drivers: services, ECU abstraction, and microcontroller

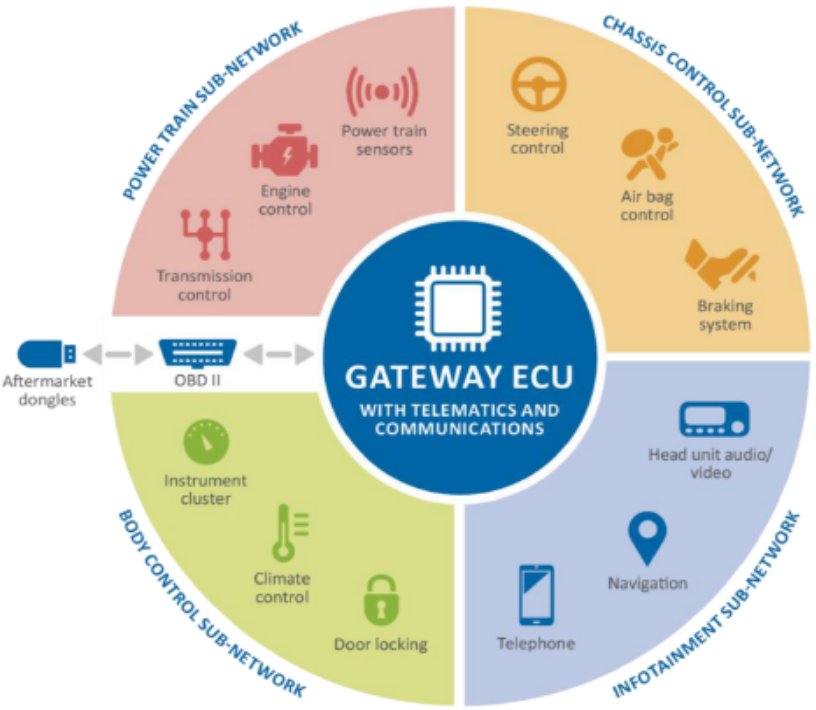


Figure 1: Main domains of a modern car (**?**)

abstraction. Services are divided further, into functional groups representing the infrastructure for system, memory and communication services.

The Virtual Function Bus (VFB) allows for communication between different Software Componentss (SWCs) and the use of BSW services, both within the individual ECU and between ECUs. Because the development of SWCs is based on the VFB, SWCs are independent of the underlying hardware, making them easier to reuse and integrate into different projects on different platforms as well as to flexibly relocate existing SWCs to ECUs during development.

New use cases, such as autonomous driving, communication with traffic infrastructure, multimedia applications, and smartphone integration, led to the creation of the AUTOSAR Adaptive Platform. This standard aims to provide support for the deployment of customer applications and an environment for applications that require a high amount of computing power (*e.g.* computer vision). It implements the AUTOSAR Runtime for Adaptive Applications, with a POSIX-based operating system at its core running (**?**) on virtualized hardware. In contrast to the AUTOSAR Classic Platform, the AUTOSAR Runtime Environment for the Adaptive Platform dynamically links services and clients during runtime. This allows applications to be developed, tested and distributed or updated independently of one another.

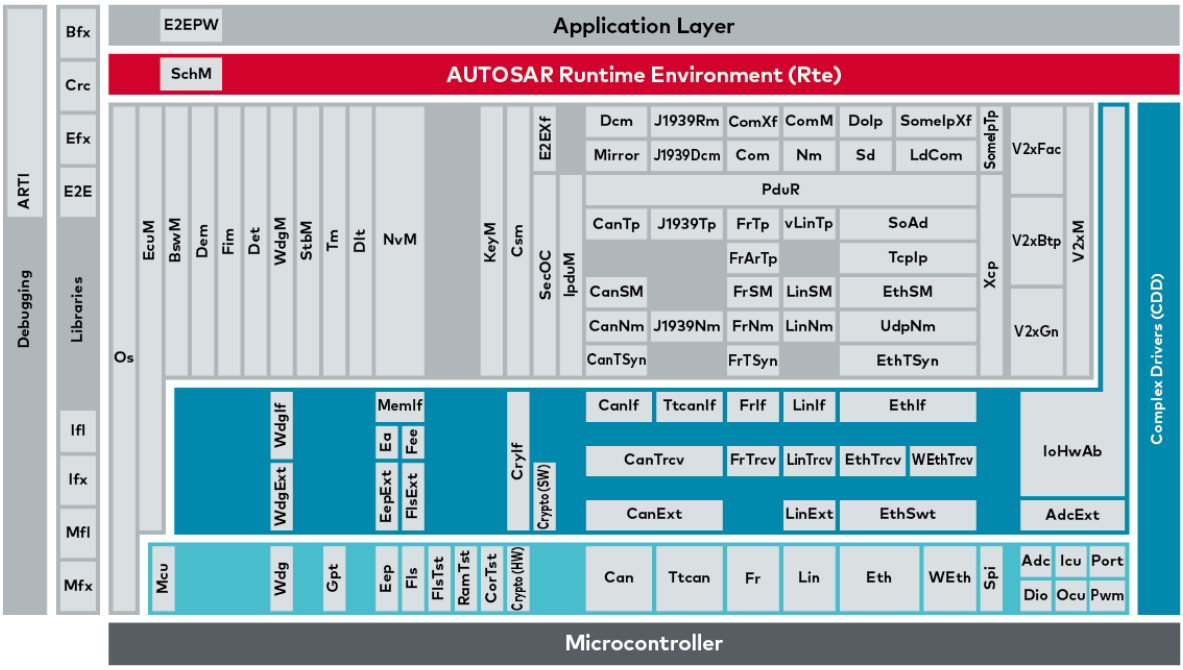


Figure 2: The AUTOSAR architecture (**?**)

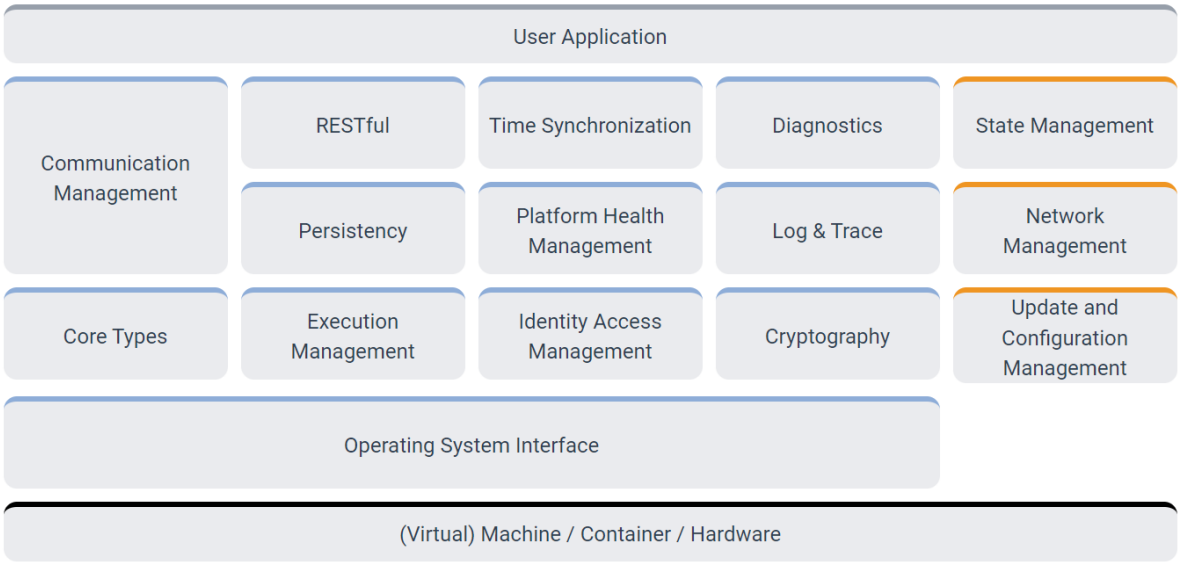


Figure 3: The AUTOSAR Adaptive Platform architecture (**?**)

3.1.2 *Applications*

A significant part of automotive innovation has been driven by software applications. These take advantage not only of the data generated by internal sensors to perform complex tasks but allow for better multimedia functionality and communication with external entities. In **?**, the authors state that automotive applications can be classified based on these 5 criteria: functional domain, influence on safety, driver involvement, connectivity, and time constraint.

Applications like the anti-lock braking system are real-time functional and safety-critical systems with no driver involvement or external connectivity elements. Another safety-critical application is adaptive cruise control, which is part of the Advanced Driver-Assistance Systems (ADAS) class of applications. However, unlike the anti-lock braking system, it requires some level of awareness from the driver. A network application like door and window control via smartphone is real-time and interacts with the vehicle's body.

Such an increase in computational complexity allows for the introduction of vulnerabilities in the software and, with applications extending multiple domains and interacting with other applications, there is an increased concern about their security. Until recently, applications were developed by the manufacturers to run directly on the embedded hardware. The usual approach was to add a new ECU for every new function to be implemented. However, this method has made the vehicle's internal network too complex for safe and efficient handling. Therefore, a platform-based approach has become of interest to both manufacturers and developers as it provides developers with an abstraction of the underlying hardware as well as making it possible to write reusable applications to run on the same standardised platform, but different hardware. This simplifies both the application development as well as its deployment. Besides, the authors in **?** suggest that an approach like that of the mobile platforms could be taken. In this case, an open Software Developer Kit (SDK) is provided to third-party developers to create and distribute their applications similarly to mobile app stores.

This increase in application diversity and functionality means that the vehicle handles much more sensitive information, especially when considering its connectivity to devices as personal as one's smartphone. This includes information like common routes and schedules, call history, or even conversations recorded with the internal microphones (otherwise used for phone conversations on virtual assistances). As such, modern vehicles are becoming a major target for attackers who seek to obtain information on the user, not only because they contain sensitive information, but also because of the lack of security elements present in the vehicle design and architecture. This is because manufacturers have always designed their vehicles with safety, but no security in mind (**?**).

3.2 ENGINEERING STANDARDS

Functional safety is already an integral part of road vehicle engineering, with the functional safety norm **?** being well established. The industry's pursuit to design and build safer connected vehicles is leading to the development of several secure software development standards for the automotive sector. One such standard is the **?** Road Vehicles - Cybersecurity Engineering. It's predecessor, **?**, already established some high-level guiding principles

such as defining a complete lifecycle process framework, and it also recommends an initial assessment of potential threats and risks that may be considered relevant.

An overview of the ISO/SAE 21434:2021 can be found on Figure **??**. Clause 4 offers some general considerations and contextualises the approach to vehicle cybersecurity taken in the standard. Clause 5 concerns the organisational structure, and describes the organisation's security objectives and how to achieve those objectives as well as responsibility and authority assignment. Clause 6 discusses cybersecurity management at the project level, detailing, among others, the proposed approach for tailoring and incorporating off-the-shelf components. Clause 7 describes cybersecurity responsibility distribution between customer and supplier. Clause 8 details the continuous development activities, such as risk assessment and vulnerability management, until the end of cybersecurity support. Clause 9 concerns the concept phase, which involves considerations of vehicle functionality as well as the cybersecurity goals for each item. Clauses 10 and 11 discuss the product development phase, including the specification of cybersecurity requirements and architectural design, as well as integration and verification activities and the validation of cybersecurity goals and claims. Clauses 12 to 14 describe the components of the post-development phase, such as production (manufacturing and assembly of an item or component), operations and maintenance (including cybersecurity incident response), and end of support and decommissioning. Finally, Clause 15 discusses threat analysis and cybersecurity risk assessment.

3.3 IN-VEHICLE NETWORKS

For a long time, the only electronic component in a vehicle was the radio. However, pushes for more environmentally friendly vehicles, as well as increased comfort, led to the addition of many more electronic modules. These can be a part of several subsystems, such as the powertrain, chassis, body, and infotainment system, each with its own set of requirements. These requirements may be an assurance of message delivery and minimum delivery times for safety-critical applications, or high bandwidth for infotainment purposes. Longevity must also be considered. In **?**, it was shown that the average car age in the European Union is 11.5 years. For comparison, the average smartphone lifespan for consumers in the United States has been documented to be around 2.75 years (**?**). Network boot-time is relevant as well, as automotive boot-time should remain under 100ms, while some smartphones take over 1 minute to boot (**?**). Power consumption must not be forgotten, as it impacts the vehicle's range. The manufacturer's desire to reduce the vehicle's complexity, weight, and cost, led to the use of a heterogeneous set of networks with different characteristics.

? lists some roles that a network might serve. These are safe/time-critical data, control data, backbone, infotainment, multi-gig data, and low cost/speed data. Below is an introduction to some of the dominant automotive network protocols, and a general overview of these networks can be seen in Table **??**.

3.3.1 *Controller Area Network*

The development of the CAN standard was started in 1983 by Uwe Kiencke. It was first introduced in February of 1986 at the Society of Automotive Engineers Congress by Robert Bosch GmbH (**?**). Following the publication of

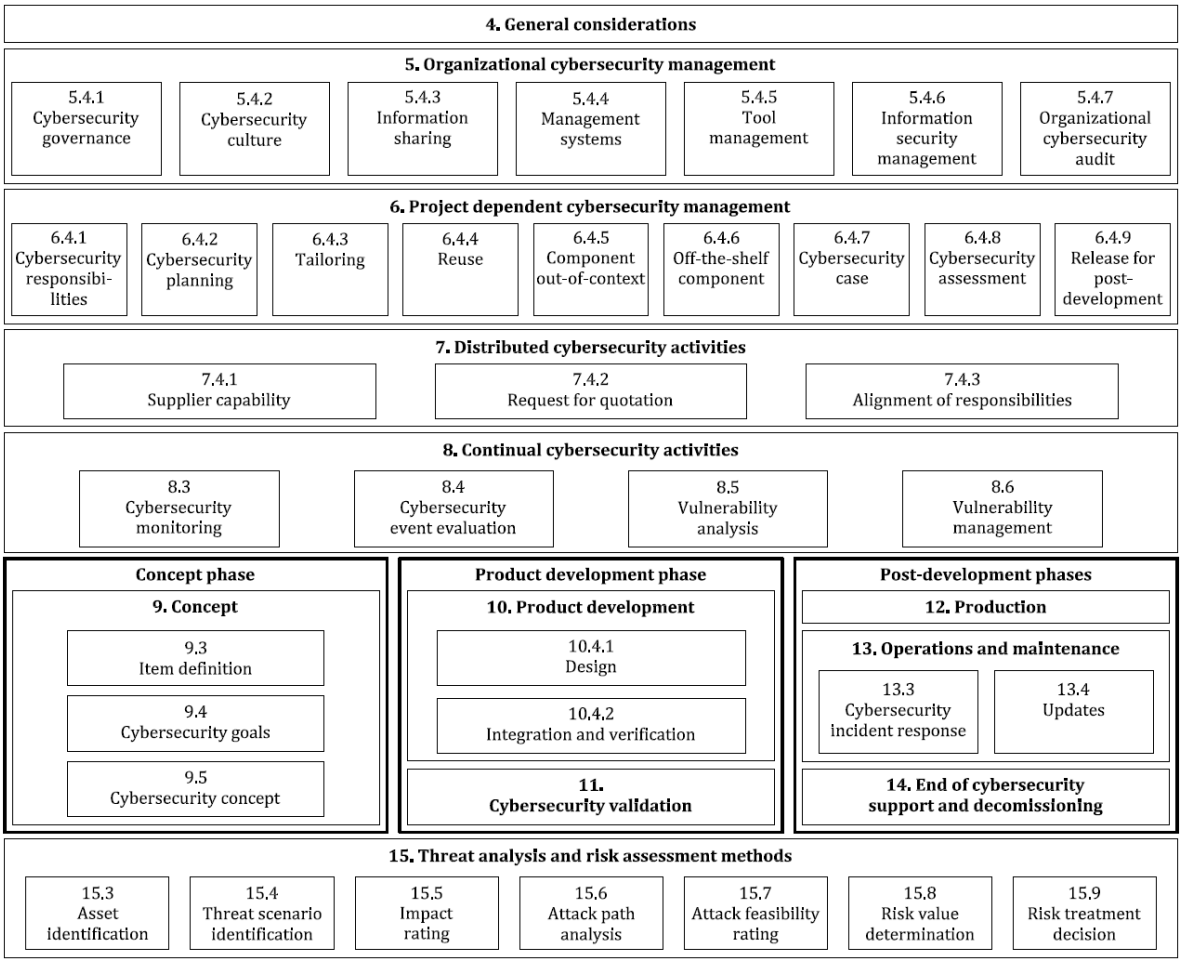


Figure 4: Content overview of the **?**

Technology	Domain	Maximum bandwidth	Collision avoidance mechanism
LIN	Low Cost/Speed	20kb/s	Time divided media access
CAN	Low Cost/Speed	125 Kb/s (Low-Speed CAN)	Non-destructive arbitration
	Backbone	1Mb/s CAN 2.0	
	Control	5 Mb/s CAN FD	
		10Mb/s CAN XL	
FlexRay	Safety/Time CriticalBackboneControl	10Mb/s	Time divided media access
MOST	Infotainment	150Mb/s	N/A
Automotive Ethernet	Safety/Time Critical	10Gb/s	N/A
	Multi-Gig		
	Infotainment		
	Backbone		
	Control		

Table 1: Characteristics of dominant automotive networks

CAN 2.0 by Bosch in 1991, the International Organization for Standardization (ISO) released the CAN standard ISO 11898. The Mercedes W140, first available in 1991, was the first vehicle with a CAN-based in-vehicle network (**?**).

CAN messages are sent in broadcast mode, which means that all ECUs in a network receive all transmissions. Devices that receive a message acknowledge their reception. Each ECU listens for messages from a subset of IDs, and each message ID must be sent by no more than one ECU (**?**). It was designed with the objective of building a low-cost, and reliable network, and is therefore used mainly used for powertrain, chassis and body electronics. Its limited bandwidth of 1Mb/s does not allow for its use in other domains, such as the infotainment system (**?**).

A deeper look into CAN is conducted in Section **??**.

3.3.2 *Local Interconnect Network*

Local Interconnect Network (LIN) is a supplement to the CAN bus, offering lower performance and reliability, but also a much lower cost. It is a product of the LIN Consortium, founded by BMW, Volkswagen Group, Audi, Volvo Cars Mercedes-Benz, Volcano Automotive Group, and Motorola, and provides cost-efficient communication in applications where the bandwidth and versatility of CAN are not required. This typically includes mechatronic nodes like the vehicle's windows, wipers, side mirrors, or seat controllers (**?**).

The first version of the specification was released in 1999, with updates soon following. LIN 2.0 was released in 2003 and the protocol was standardised in ISO 17987:2016.

Unlike CAN, a typical LIN cluster consists of a single commander node and up to 16 responders connected by a single copper wire, with communication speeds up to 20kbit/s (**?**). It is usually deployed as a CAN sub-network, with only the LIN master node being connected to the CAN-bus. All communications are initiated by the commander node, with one responder replying to a given message identifier (the commander node can also act as a responder, replying to its own messages). There is, therefore, no need to implement collision detection (**?**).

This protocol also implements a sleep and wake-up mechanism, allowing it to potentially save power. Responders can enter sleep via a command issued by the commander node or are forced into that state after more than 4 seconds of LIN inactivity. The wake-up command can be issued by any node (**?**). We can see in Figure **??** that the number of LIN nodes has recently surpassed the number of CAN nodes.

There have been recent efforts to study the security aspects of LIN, such as the one detailed in (**?**), and it is noted that due to its link with CAN, the latter's vulnerabilities are also a cause for concern for the former. (**?**) showed that a compromised LIN node can destroy as well as create "erroneous, but valid packets". The authors also suggested that the effect of an intrusion can be minimised by the use of appropriate firewalls. However, since LIN is usually used for non-safety-critical purposes, compromising the bus would most likely not be life-threatening.

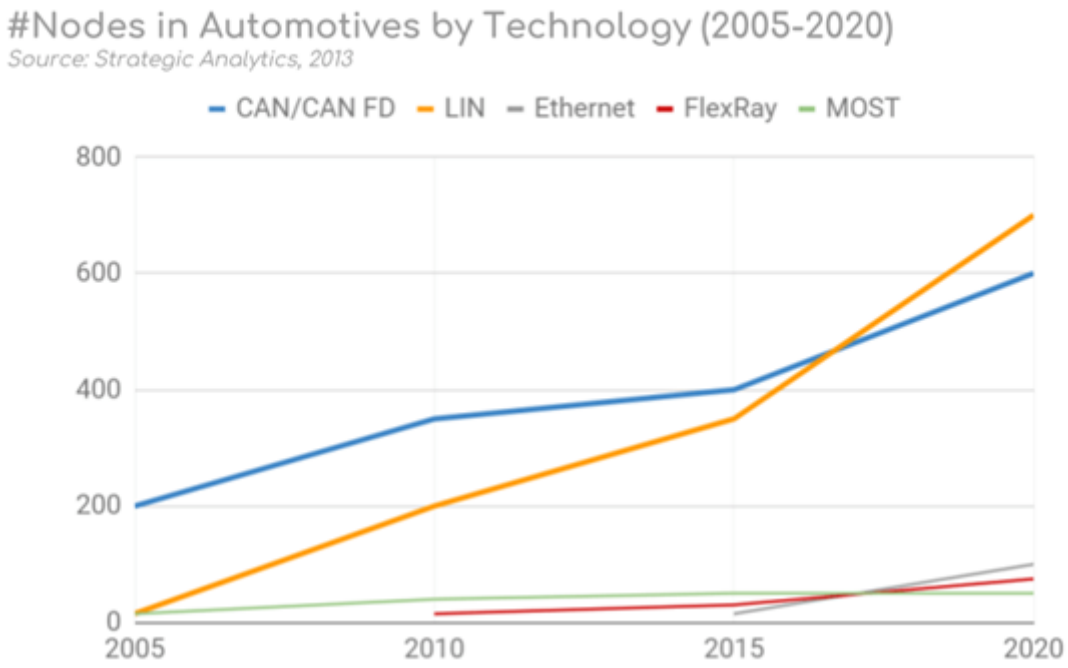


Figure 5: Number of nodes in automotive by technology (**?**)

3.3.3 *FlexRay*

The continuous search for improved safety, performance, and comfort of vehicles means that the speed and reliability of data communicated between ECUs must increase. FlexRay was developed by the FlexRay Consortium, founded in 2000 by Daimler Chrysler and BMW along with Motorola and Phillips, and disbanded in 2009. The Consortium had the goal of creating a bus protocol capable of delivering achieving high data rate, deterministic and fault-tolerant communications (**?**) as a way of overcoming CAN's limitations in these aspects. Features that require a higher throughput for safety/time-critical data like active cruise control, anti-lock braking system, or break-by-wire, thus use FlexRay as the preferred communication protocol, as it satisfies the reliability requirements for these safety-critical systems.

The FlexRay bus consists of a single twisted-pair copper cable, with ECUs connected using multi-drop technology. Its clock synchronisation is deterministic and uses time divided media access as a collision avoidance mechanism, in contrast to the arbitration mechanism used by CAN. This makes it suitable for safety/time-critical applications. It is also used for backbone and control data, similarly to CAN. However, it provides much higher bandwidth: 10Mb/s, compared to CAN's maximum of 1Mb/s, and is more expensive (**?**).

The protocol is defined in ISO 17458-1 to 17458-5.

3.3.4 *Media Oriented Systems Transport*

As the name suggests, Media Oriented Systems Transport (MOST) is primarily used for high-speed multimedia. It can use either a single twisted pair of copper cables, or plastic optical fibre (which has the advantage of being completely immune to electromagnetic disturbance), and supports 25, 50, and 150 Mb/s data transfer speeds (**?**). The MOST network is capable of managing up to 64 devices, usually configured in a ring topology. Star or double ring configurations are also possible (**?**), with the latter being relevant for safety-critical applications (**?**).

The MOST specification, described in ISO 21806, defines the physical and data link layers, as well as all the seven layers of the ISO/OSI model.

3.3.5 *Automotive Ethernet*

Automotive Ethernet is an adaptation of standard Ethernet that fulfils the requirements of the automotive environment, such as the use of a single twisted-pair wiring, which provides lower cost and better electromagnetic compatibility.

The motivation behind this adaptation is primarily the amount of full-duplex bandwidth it provides. In 2020, IEEE published the IEEE 802.3ch-2020 standard, detailing the specification for automotive Ethernet with data transfer speeds of up to 10Gb/s (10GBASE-T1). This allows fields like ADAS to grow and develop without much concern for bandwidth restrictions. Another important factor is that Ethernet is already an established technology outside of the automotive industry. This means that there is a large knowledge base to support it, especially when compared to the other protocols discussed here. Its use of the IP stack is a point in its favour as well. It is also a scalable technology, with adding new nodes to the network being a relatively easy task (**?**).

As for the network topology, Ethernet is, by nature, a point-to-point network. This means that connecting more than two nodes requires an Ethernet switch.

This technology does not aim, however, to only replace other high-bandwidth networks like MOST. The 10BASE-T1s version of the automotive Ethernet protocol is a direct competitor to CAN and FlexRay, as it describes a multi-drop bus topology network over a single twisted pair of copper wires with speeds up to 10Mb/s and time divided media access for collision avoidance (**?**). This would allow for a vehicle using only IP based technologies, simplifying the overall design of the network.

3.4 CONTROLLER AREA NETWORK

CAN is currently the dominant network protocol in automotive. As laid out in **??**, it is a broadcast network with no central bus master, which allows for data consistency across the entire system. The two lowest layers of the ISO/OSI model for CAN are defined in the ISO 11898 (**?**) and the implementation can be seen in Figure **??**. Protocols such as the CANopen protocol, supported by the *CAN in Automation* group (CiA), establish the link between the physical and data link layers and those above (**?**).

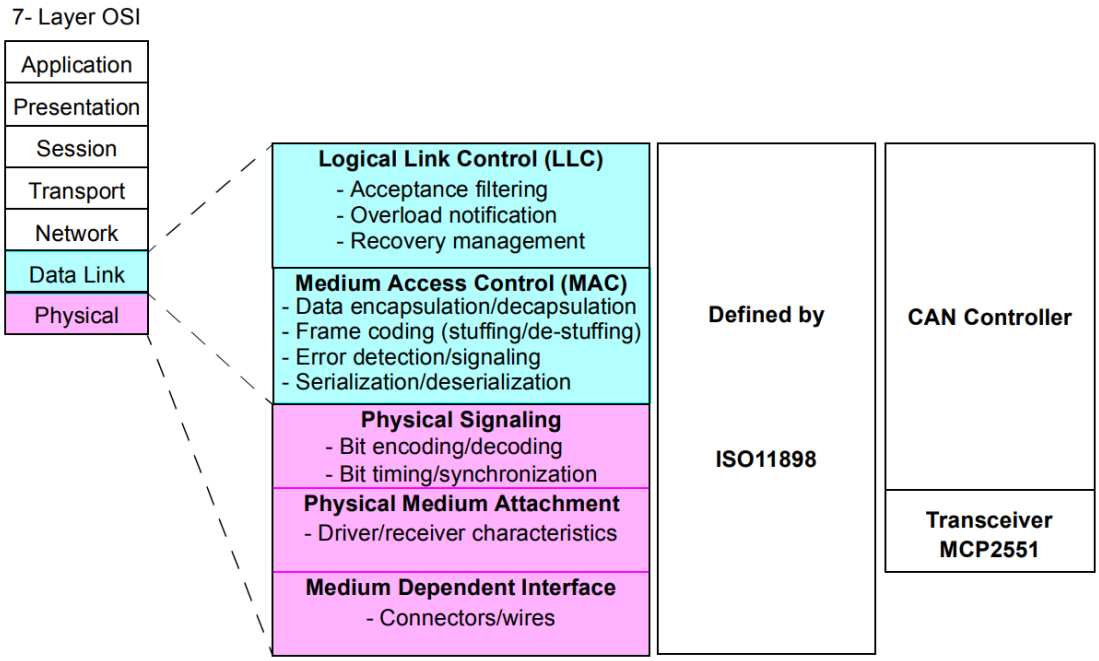


Figure 6: CAN and the OSI model (**?**)

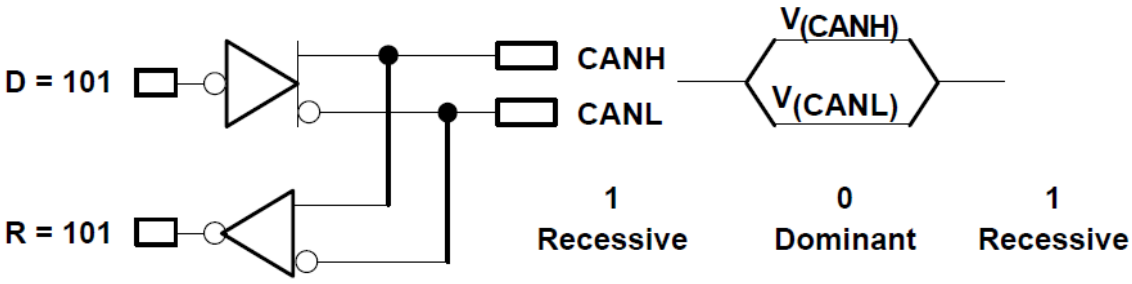


Figure 7: The inverted logic of a CAN bus (**?**)

The CAN bus differs from a conventional bus because it inverts the logic state between the bus, driver input and receiver output. In CAN, a logic high is associated with a 0 (the dominant bit), and a logic low with a 1 (the recessive bit) (**?**). An illustration of this mechanism can be seen in Figure **??**.

CAN uses Carrier-Sense Multiple-Access (CSMA), meaning that each node must wait for bus inactivity before transmitting, with Collision Detection and Arbitration on Message Priority (CP+AMP), meaning that collisions are resolved through bit-wise arbitration using the message's ID field. Messages with lower ID have a higher priority, as a smaller identifier will hold the dominant state on the bus for longer (**?**). The bus can, therefore, be thought of as acting as an AND gate, as explained by **?**.

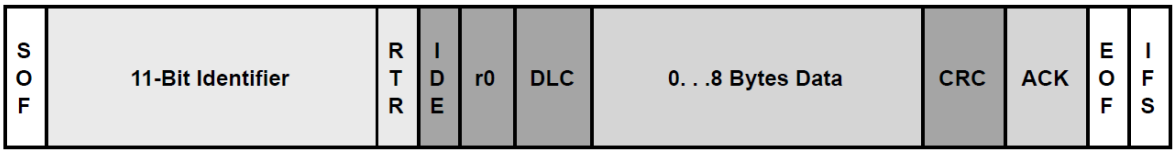
There are four message types in CAN: the data frame, the remote frame, the error frame, and the overload frame. Error-free messages have the last bit of the EOF set as recessive, and a dominant bit in this field causes the message to be re-transmitted.

The data frame is the most common message type and is composed of seven fields: SOF, arbitration, control, data, CRC, ACK, and EOF. The SOF field consists of a single dominant bit, and the arbitration decides which of the nodes attempting to transmit is allowed to do so. The RTR field distinguished between data (dominant bit) and remote (recessive bit) frames. The control field indicates the amount of data being sent in the data field. The remote frame is used to request a data frame from another node. If a node broadcasts a remote frame containing a particular identifier, the node responsible for this identifier immediately generates a data frame as a response. The error frame is used to notify errors in frame transmission, and the overload frame signals that the controller has not finished processing the current message, and delays the start of the next message (**?**). CAN is available in three generations: Classical CAN, CAN FD, and CAN XL. These are described below.

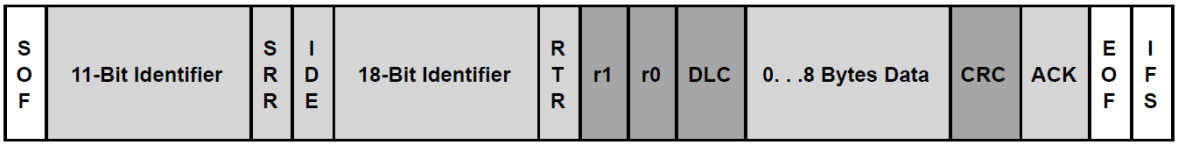
3.4.1 *Classical CAN*

There are three versions of what can be called "classical" CAN: low-speed CAN, CAN 2.0A, and CAN 2.0B. As the name suggests, low-speed CAN has limited bandwidth, at 125Kb/s. However, it is fault-tolerant, as it can operate even when one of the two wires fails. This version of the protocol is standardised in ISO 11898-3. CAN 2.0 versions A and B allow for speeds up to 1Mb/s, with the main difference between the two versions being that the former uses an 11-bit identifier (standard), and the latter a 29-bit identifier (extended). The bit fields for standard (low-speed CAN and CAN 2.0A) and extended (CAN 2.0B) versions can be seen of Figures **??** and **??**, respectively. The meaning of the bit fields are:

- **SOF (Start Of Frame)**: Single dominant bit that marks the start of a message.
- **Identifier**: Establishes the priority of a message (lower ID means higher priority).
- **RTR (Remote Transmission Request)**: When this bit is dominant, information is required from the node specified in the identifier.
- **SRR (Substitute Remote Request)**: Replaces the RTR bit as a placeholder in the extended format.
- **IDE (Identifier Extension)**: When this bit is dominant, a standard CAN identifier is being transmitted.
- **r0**: Reserved bit.
- **r1**: Reserved bit.
- **DLC (Data Length Code)**: 4-bit field that contains the number of bytes of data being transmitted.
- **Data**: Up to 64 bits may be transmitted.
- **CRC (Cyclic Redundancy Check)**: 16-bit field containing the checksum of the transmitted data for error detection.
- **ACK (Acknowledge)**: Every accurate message received has this recessive bit overwritten by a dominant bit.



(a) Standard CAN: 11-bit identifier



(b) Extended CAN: 29-bit identifier

Figure 8: Classical CAN frame structure (**?**)

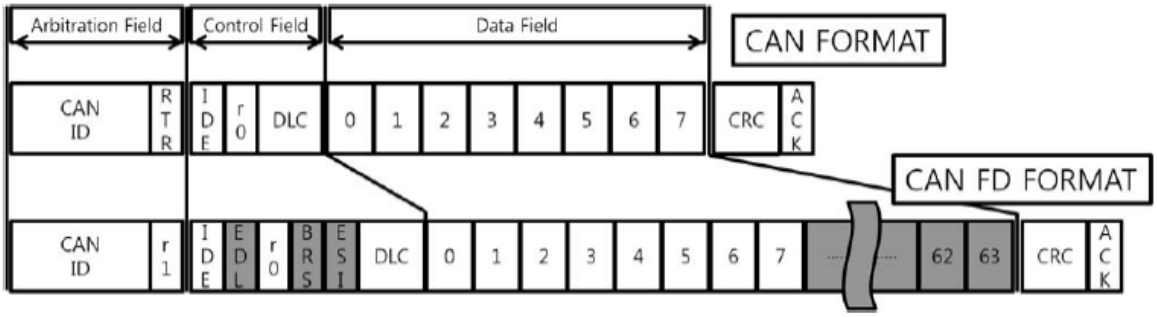


Figure 9: Comparison of CAN and CAN FD data frames (**?**)

- **EOF (End-Of-Frame)**: 7-bit field that signals the end of a CAN frame, while also disabling bit-stuffing.
- **IFS (Inter-Frame Space)**: 7-bit field that allows the controller enough time to move the received frame into the message buffer.

3.4.2 *CAN FD*

Increasing demand for more bandwidth led to the development of CAN FD (standing for Flexible Data-Rate), which was started in 2011 by Robert Bosch GmbH. The fact that this design is based on CAN 2.0 provides several advantages. Costs are similar, and it has a small impact on current software and applications. The physical layer and topologies can also be maintained (**?**). It is primarily used in high-performance vehicle ECUs and is established as an international standard in ISO 11898-2:2015.

The main difference between CAN 2.0 and CAN FD is the flexible data rate that CAN FD offers. This means that it allows for the data frame to be transmitted at different speeds, depending on the characteristics of the network. Development was faced with two main challenges: avoiding increased message delays caused by placing more bytes of data in a single frame, and maintaining CAN's practical wiring (**?**). The solution to this was the introduction of a new frame structure, which can be seen in Figure **??**.

Figure 10: Illustration of the CAN FD flexible data-rate mechanism (**?**)

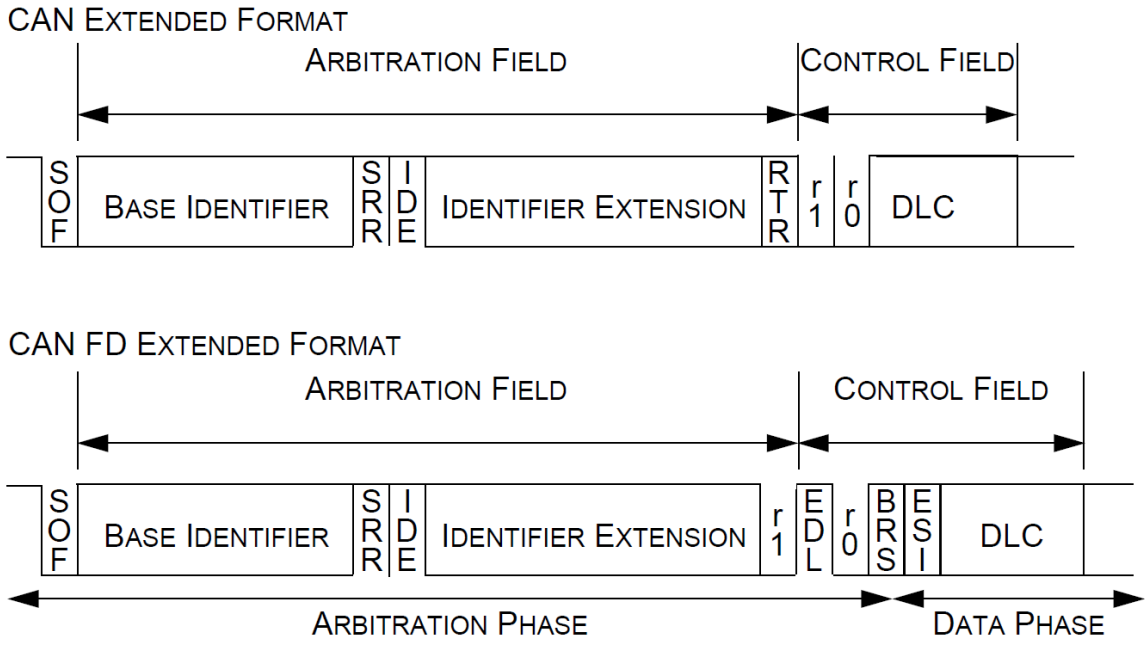


Figure 11: Comparison of extended CAN and CAN FD data frames (**?**)

In the arbitration field, we can see that the RTR bit is not present in the CAN FD frame. This is because there are no remote frames in CAN FD.

Three new bits were added to the control field: extended data length (EDL), bit rate switch (BRS), and error state indicator (ESI). The EDL bit is used to distinguish between the classical CAN (bit is dominant) and the CAN FD (bit is recessive) frame formats. If the BRS bit is dominant, it signals that the data frame is sent at the arbitration rate (i.e. up to 1Mb/s). Otherwise, the remaining part of the data frame is sent at a higher speed (around 5Mb/s). The ESI bit is used for identifying an error in the CAN FD node.

The CRC has also been updated from 15 bits in classical CAN to 17 or 21 bits in CAN FD. Lastly, the payload portion of the frame stops at the ACK bit, which means that this is the end of the potentially higher bit rate. An illustration of this can be seen in Figure **??**.

There exists also an extended version of CAN FD, shown in Figure **??**. It functions much like the classical extended CAN, allowing for more IDs to be used.

A major challenge in securing CAN has been the limited speed and payload size, as it makes it difficult to implement typical cryptographic schemes, such as authentication, to defend against attacks. CAN FD brings some relief in this matter, and new techniques have been proposed, like the ones in (**?**) and (**?**).

3.4.3 *CAN XL*

Fueled by much of the same objectives as CAN FD, as well as the rise in popularity of automotive Ethernet, CAN XL presents itself as the third generation of CAN, capable of providing speeds above 12Mb/s in the frame's data phase (the arbitration phase of the frame maintains its top bit-rate of 1Mb/s). This puts both 10BASE-T1s Ethernet and CAN XL at the same level in terms of bit rate.

In contrast to 10BASE-T1s Ethernet, CAN XL offers a wider range of topologies to choose from, meaning that not all CAN XL networking topologies can be exactly replaced with Ethernet, as the latter has a stricter bus topology. An example of this can be seen in Figure **??**. The increase in data field length for CAN XL also allows for transparent Ethernet frame tunnelling, enabling the use of IP communication (**?**). Another advantage this protocol has over automotive Ethernet is the fact that it is much easier to upgrade existing CAN networks to CAN XL than it is to transition to Ethernet and the IP stack, which is more complex and requires more expensive controllers (**?**). Seen as though CAN XL can be mixed with CAN FD, it also allows manufacturers to optimise cost by using the cheaper technology where higher bandwidth is not required (**?**).

The CAN XL frame structure can be seen in Figure **??**.

In Classical CAN and CAN FD, the CAN-ID field (11 bit or 29 bit) is used for both arbitration and addressing purposes. In CAN XL these functions are separated. The CAN XL protocol separates the priority functions (11-bit Priority ID) and the addressing (32-bit AF). The 11-bit Priority Field provides the uniquely assigned priority of the CAN XL DF. The 32-bit AF (Acceptance Field) is included in the 64-bit hardware acceptance filter of the CAN XL controller. It may contain node address or content-indicating information.

The CAN XL DF includes two CRC (cyclic redundancy check) fields: the 13-bit PCRC (Preamble CRC) in Control Field and the 32-bit FCRC (Frame CRC) in CRC Field. The CRCs are cascaded, which means the FCRC protects the whole frame, including the PCRC. Both CRCs can detect any five randomly distributed bit-errors, which corresponds to a Hamming distance of 6 (**?**). New in the control field are also the possibilities to indicate the payload type, much like the EtherType field in Ethernet, and a virtual CAN ID, allowing the separation of the CAN network into virtual networks and comparable to the VLAN ID in Ethernet (**?**).

CAN XL, therefore, presents itself as an incremental upgrade step for existing CAN and CAN FD networks, while also closing the gap in transmission speed between CAN and automotive Ethernet. It aims to be a reliable alternative to 10BASE-T1s, with a simpler protocol stack and cheaper controllers. The fact that it allows for Ethernet frame tunnelling may also open the doors to joint single-based communication in the lower levels of the network (CAN) and signal-based communication at the higher end (10BASE-T1s) (**?**).

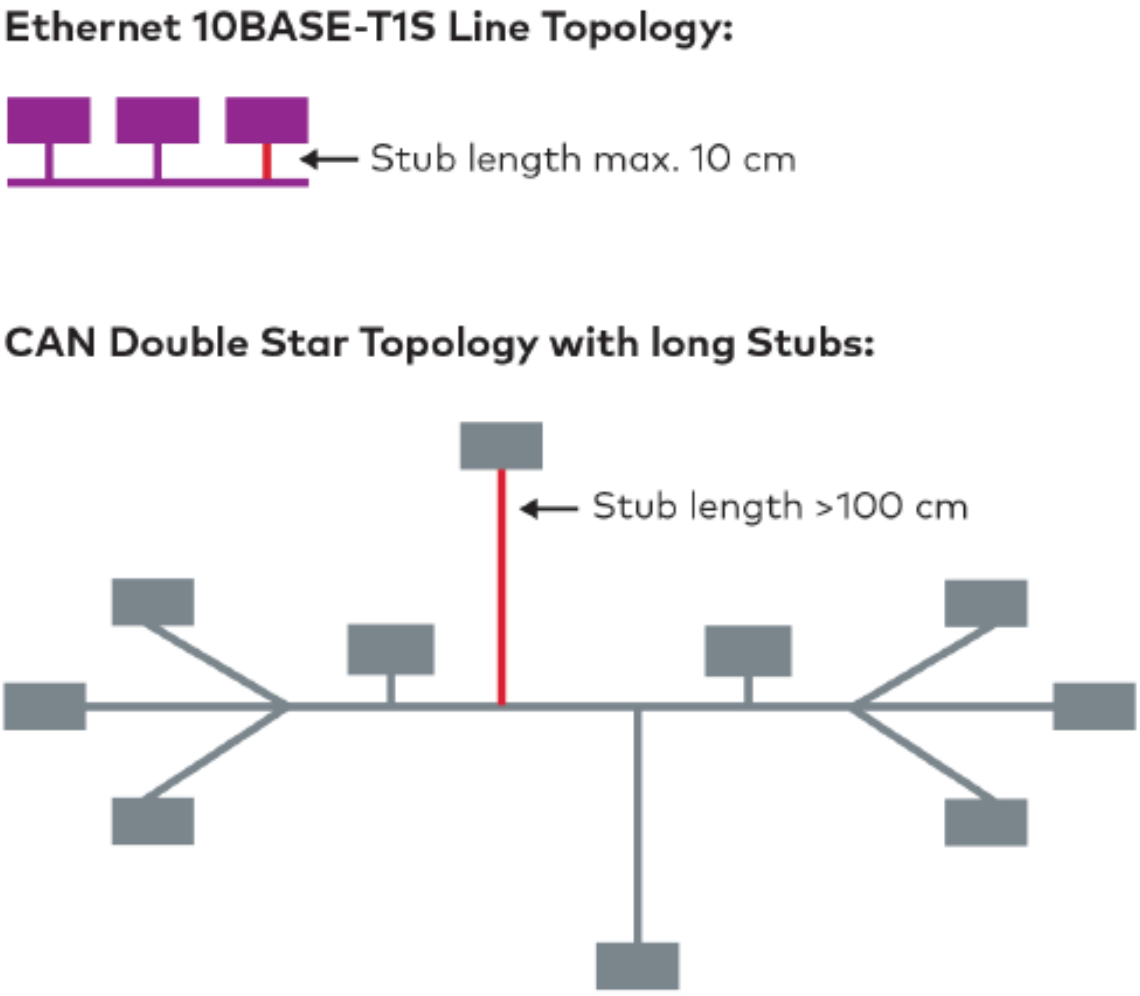


Figure 12: Comparison between 10BASE-T1s and CAN XL topology requirements (**?**)

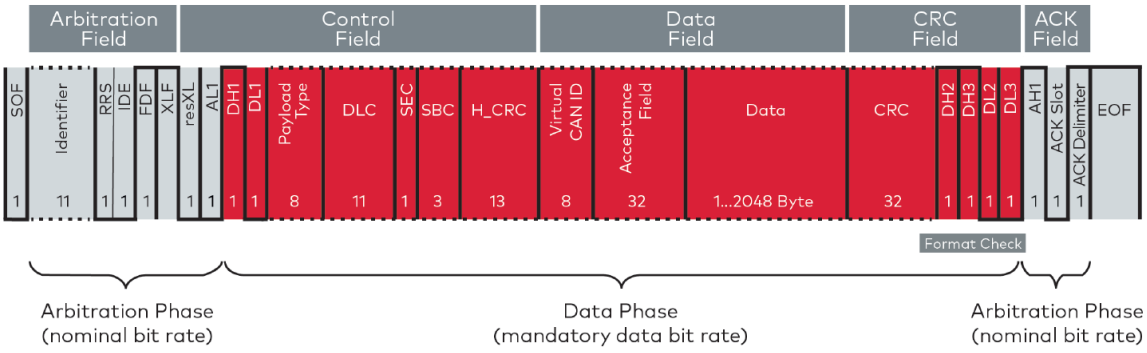


Figure 13: Current status of development of the CAN XL frame (**?**)

3.5 AUTOMOTIVE CYBERSECURITY

3.5.1 *Vulnerabilities*

CAN

The reason CAN is so dominant in the automotive networking space is because of its simplicity. This has, however, become an exploitable property with the exterior connectivity that did not exist when the protocol was developed.

Because messages are broadcast to the entire network, an attacker with access to one part of the network, like an ODB port, can eavesdrop on all traffic as well as send messages to all nodes. When eavesdropping, messages can be easily interpreted because there is no encryption in the protocol. There is also no authentication methods that allow nodes to know the source of the received frame, meaning that nodes cannot assert if a received frame was sent by a legitimate node or a bad actor. The ID-based arbitration scheme also facilitates Denial of Service attacks, as flooding the system with high-priority messages would cause all others to back off (**?**).

Wireless sensor networks

Because of the wireless nature of the communication channel, messages can easily captured and replayed. Other problems with these communication protocols include the lack of proper cryptographic protection, the use of obscure code in the name of security, bad programming practices, and insufficient hardware protection against side channel attacks (**?**).

? documented out several vulnerabilities of a Tire Pressure Monitoring System (TPMS), showing that its messages are both unencrypted and easily spoofed. They also showed that in-vehicle systems do not perform any input validation on incoming messages, and that it is highly possible to track vehicles through it.

V2X communication

Connecting a compromised device to a vehicle can result in attacks to in-vehicle systems. It's very common to connect a smartphone to a vehicle, either via Bluetooth or USB. Since smartphones can be infected with malware, in-vehicle applications must be protected it. **?**, for example, trusts all contents from registered smartphones, which may be infected (**?**). Other equipment for V2V and V2I communication can also be exploited via a bad implementation or configuration.

3.5.2 *Common attack patterns*

The authors of **?** surveyed previously published research on attacks against autonomous vehicles, although some attacks are also effective against non-autonomous vehicles. They classify attack research into three categories: attacks on the automotive control system, attacks on the autonomous driving system components, more specifically through mobile apps, and attacks on Vehicle-to-Everything (V2X) communication technologies.

Automotive control systems

One type of automotive control system attack targets the vehicle's ECUs. Taking advantage of network vulnerabilities, research shows that it is possible to load firmware into ECUs that control components such as the door-lock and telematics, primarily through fuzzing (**?**). The vulnerabilities described by **?** showed that it is possible for an attacker to remotely display a false tire pressure warning or disable the TPMS ECU, as well as vehicle tracking from 10-40 metres. More recently, it was shown that ECUs present in Tesla vehicles could be controlled remotely by sending arbitrary CAN packets, which led Tesla to introduce code-signing protection in an Over-The-Air (OTA) update (**?**). **?** showed that, even remotely, and attacker can reprogram ECUs, control display and sound systems, the instrument cluster, body components, interfere with the engine operation, lock and disable brakes, prevent the car from turning on or off, etc. They showed that, even from telematics ECUs, this type of attack has the same effects as directly injecting messages into CAN buses.

In-vehicle network attacks aim to disrupt communication between vehicle components. In (**?**), the authors showed that the CAN bus is vulnerable to selective DoS attacks, which cannot be detected at the frame level. Sniffing and replay attacks have also been demonstrated, as per **?**. **?** also explored possible attacks on the FlexRay protocol. They showed that, because there is no encryption, it is possible to read all data sent to the bus, as well as insert unauthenticated messages. A successful attack causes serious disruption of the vehicle control system and may harm drivers and passengers.

Passive key-less entry and start systems used in modern cars are a major attack target. It has been shown that some systems are vulnerable to relay attacks by duplicating the existing signal (**?**). Through the analysis of cryptographic vulnerabilities, **?** demonstrated how private keys could be recovered from the Hitag2 transponder in order to bypass the key's cryptographic authentication in as little as 6 minutes. **?** also showed how to recover cryptographic keys and clone a VW group remote via eavesdropping on a single signal, as well as another Hitag2 attack able to clone the remote in just a few minutes.

External communication technologies

A comprehensive survey of attacks on Vehicular Ad Hoc Networks (VANETs) **?** showed that these are vulnerable to the same attacks as a regular network, such as DoS, man in the middle, message suppression, or spoofing.

The integration of Bluetooth has become standard, and it brings with it a new set of vulnerabilities. In **?**, it was shown that manufacturers still include the ability to use legacy pairing, which led the authors to identify vulnerabilities. Other research has shown the execution of attacks using Bluetooth as a vulnerable component, as shown in **?** with BMW cars.

Mobile apps

The increase in connectivity between the driver's vehicle and his smartphone led to an expansion in the vehicle's attack surface. A malicious smartphone app could be used to gain unauthorised access to a vehicle without needing to be in its proximity (i.e., using long-range wireless networks). The authors of **?** exploited CAN vulnerabilities in the process of demonstrating this scenario. Vulnerabilities in the MirrorLink protocol have also

allowed attackers to send CAN packets through a smartphone, opening the possibility for a variety of exploits. The attack method present in **?** took advantage of a heap overflow. A vulnerability in Blue Link, a vehicle control application from Hyundai Motors, allowed the recovery of the driver's username, password, and PIN. An attacker could also exploit this vulnerability to remotely unlock the vehicle (**?**). With the smartphone being relatively vulnerable to attacks, and with manufacturers providing more functionality to it's connection to the vehicle, it is expected that these type of applications become a major target (**?**).

3.5.3 *Proposed solutions*

The development of protection mechanisms faces multiple constraints resulting from the limited computing resources available to each ECU as well as the time-sensitive nature of their operations. There is also a need to assert retro-compatibility with currently used embedded technologies, and, when considering external entities, interoperability between their security mechanisms. Two distinct vehicles, for example, should not be prevented from communicating because their protocols are incompatible (**?**).

There is an effort by European projects such as Secure Vehicular Communication (SeVeCom) (**?**), Preparing Secure Vehicle-to-X Communication Systems (PRESERVE) (**?**), or E-safety Vehicle Intrusion Protected Applications (EVITA) (**?**), to design secure vehicle communication architectures.

For VANETs, the usage of public-key infrastructure has also been proposed, to assert the authenticity of the messages, as well as the use of pseudo-anonymity to preserve privacy. For preventing DoS attacks, some routing protocols have also been studied (**?**).

Regarding internal protection, there have been several attempts to leverage cryptography in securing the ECU and the CAN bus. An authentication protocol, such as the one proposed in (**?**) or (**?**), would prevent most injection attacks. Another approach, taken by the EVITA project, consists in the implementation of a dedicated hardware module for cryptographic operations (**?**).

IDS has traditionally been used in network security, more specifically in desktop IT. (**?**) is one of the first to mention its usage in automotive IT as a "promising supplemental measure". They also discuss a way to optimally communicate a warning to the driver.

3.6 MACHINE LEARNING

As stated in **??**, the use of machine learning in an IDS allows it to adapt and detect previously unseen attacks. Such an approach has two phases: training and testing. First, attributes and classes from the training data must be identified. These must be reduced to a subset of those most relevant for classification, discarding unnecessary ones (*i.e.*, dimensionality reduction). Then, the resulting dataset is used to train the model, with the trained model being used to classify unknown data. The training data can be unlabelled, which creates an unsupervised learning problem, partially labelled, for semi-supervised learning, or fully labelled, making the learning phase supervised (**?**).

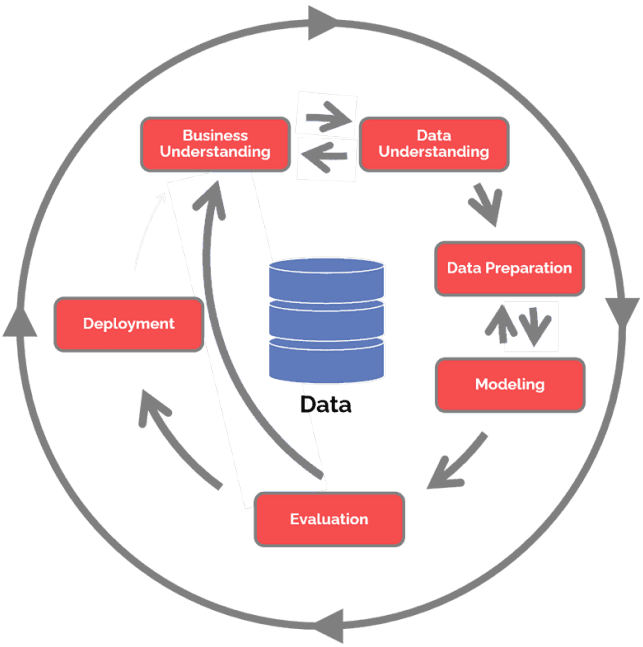


Figure 14: CRISP-DM diagram (**?**)

In (**?**), the CRoss Industry Standard Process for Data Mining (CRISP-DM) was presented as a six-phase model describing the data science life cycle in a industry, tool, and application neutral way. A representation of this can be see in Figure **??**.

The model's six phases are:

- Business understanding, where the project's requirements are defined.
- Data understanding, where data is collected and examined.
- Data preparation, where the collected data is processed to produce the final dataset.
- Modeling, where machine learning methods are applied to produce the model.
- Evaluation, to verify whether the produced model is able to verify the requirements established in the first phase.
- Deployment, with the model being implemented.

In most applications, machine learning models are trained and then used for a long time without any modifications. In the security space, however, new threats emerge frequently. This means that models must be updated often in order to learn to detect new and emerging attack, as well as in cases where the network traffic suffers modifications in its usual pattern. Training times are, therefore, important. The model's ability to learn incrementally is also valuable, as training from scratch every time new data appears would be very time consuming.

Network traffic also generates a huge amount of unlabelled data, and labelling it is a laborious task. Quickly acquiring novel attack data can also be difficult, but is necessary to update the model as soon as possible (**?**).

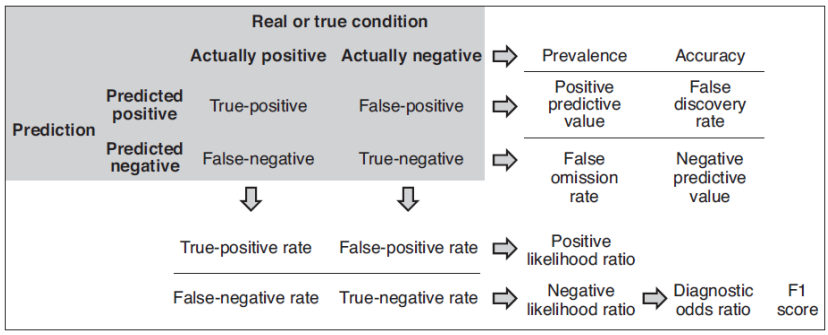


Figure 15: Components of a confusion matrix (**?**)

Deep learning is a subset of machine learning where the neural networks have more than one hidden layer. While machine learning algorithms make use of labelled and structured data to make predictions, deep learning algorithms automate feature extraction and can ingest and process unstructured data (**?**).

3.6.1 *Performance metrics*

There are several metrics used to classify a model's performance. To access if the model will generalise to an independent dataset, cross validation techniques are commonly used. Cross validation consists of a re-sampling method that iterates several times over a given dataset, sampling distinct training and test sets each time. This is particularly useful if the amount of available data is small.

For binary classification tasks, the confusion matrix provides several useful performance metrics by laying out true and false positives, as well as true and false negatives in a 2x2 table. Many relevant metrics can be inferred from these values, as shown in Figure **??**. The true-positive rate, also known as recall or sensitivity, provides the rate at which the model detects a truly positive case, while the false-negative rate is the proportion of positive cases missed. The false-positive rate is the probability of the model classifying a negative case as positive, and the true-negative gives the rate at which the model correctly identifies negative cases. Prevalence is the proportion of the dataset that is actually positive, while accuracy tells us how many correct predictions (both positive and negative) the model made in proportion to the total dataset. The positive predictive value metric, or precision, represents the likelihood that a positive prediction is actually positive, with the false discovery rate being its complement. The negative predictive value and false omission rate are analogous. The positive and negative likelihood ratios tell the probability that a classification is actually positive or negative is the model classifies it at such. The diagnostic odds ratio provides a way to measure the model's performance without, unlike accuracy, being affected by prevalence. The F1 score is a function of both precision and recall to give an overall indication of the classifier's performance.

Another widely used metric for evaluating performance in binary classification tasks is the area under the receiver operating characteristic curve (AUROC). The curve is a graphical plot that illustrates the true-positive versus false-positive rates as the discrimination threshold is varied. A variation in the discrimination threshold usually means a gain in one metric, but a loss in the other, *e.g.* a higher threshold value will decrease the

Dataset	Realistic Traffic	Label data	IoT traces	Zero-day attacks	Full packet captured	Year
DARPA 98	✓	✓	✖	✖	✓	1998
KDDCUP 99	✓	✓	✖	✖	✓	1999
CAIDA	✓	✖	✖	✖	✖	2007
NSL-KDD	✓	✓	✖	✖	✓	2009
ISCX 2012	✓	✓	✖	✖	✓	2012
ADFA-WD	✓	✓	✖	✓	✓	2014
ADFA-LD	✓	✓	✖	✓	✓	2014
CICIDS2017	✓	✓	✖	✓	✓	2017
Bot-IoT	✓	✓	✓	✓	✓	2018

Table 2: Comparison of network traffic datasets (**?**)

classifier's recall, but increase its precision, and vice-versa. Good performing classifiers will have a large area under the plotted curve.

For regression problems, the mean squared error (MSE) is generally used as a way to evaluate how good the model's predictions are (*i.e.*, how close they are to the true value). The MSE is calculated as the mean or average of the squared differences between predicted and expected target values in a dataset, where a lower value is preferred. The rooted mean square error (RMSE) is an extension of the MSE where the units of the metric are the same as the units of the target value (and not its square). The mean absolute error, unlike MSE and RMSE, does not punish large error values as harshly as the other two metrics, as its value increases linearly with the error size, and not quadratically. The coefficient of determination (R^2) indicates how much of the observed variations in the data are explained by the model, in which case a higher value is better (**??**).

3.6.2 *Datasets*

Both the size and quality of the dataset are very important to achieve good model performance. For networking purposes, there are different types of data, with the most common being packet capture data and NetFlow.

Packet capture data consists of data captured by the networking interface, at the packet level. NetFlow is a router feature introduced by Cisco, giving it the ability to collect IP network traffic that goes through the interface, identifying packet flow. NetFlow version 5 defines a network flow as a unidirectional sequence of packets that share the exact same seven packet attributes: ingress interface, source IP address, destination IP address, IP protocol, source port, destination port, and IP type of service (**?**). Table **??** shows a comparison between several public datasets.

A majority of the effort in compiling networking datasets has been placed in the real of IP networks. In the automotive domain, there is a clear lack of such comprehensive datasets, especially when looking for labelled datasets containing attacks. Some datasets commonly used in IDS research for CAN are **????**.

3.6.3 *Machine learning techniques*

Supervised learning

DECISION TREES A decision tree is a flowchart-like structure with three basic components: a decision node, containing the test attribute, a branch, containing a possible decision based on the attribute, and a leaf, identifying the class that the instance belongs to. The best known algorithms for constructing decision trees are the ID3 and the C4.5 algorithms (**??**).

NAÏVE BAYES This technique applies Bayes' principle with assumptions that the input features are independent. It is an optimal classifier if the features are conditionally independent, although this is rarely true. One of its biggest advantages is that training can be completed in linear time (**?**). The Hidden Naïve Bayes model is more sophisticated and can be applied to highly dimensional IDS tasks, with interrelated attributes and high-speed networks (**?**).

EVOLUTIONARY COMPUTATION An evolutionary computation algorithm is based on the *survival of the fittest* principle. Starting with a randomly generated population, a fitness value is computed for each individual from the population, revealing how good it is at solving a given problem. Fitter individuals are copied into the next generation, and the process continues. The two most widely used methods are Genetic Algorithms (GA) and Genetic Programming (GP). In GA, individuals are represented as a series of bits, to which selection and reproduction operators are applied favouring fitter solutions. In GP, individuals are represented as programs with operators (*e.g.* plus, minus, and, or) and programming blocks (*e.g.* if, while), and crossover and mutation operations are more complex than those in GA.

ARTIFICIAL NEURAL NETWORK The mechanisms behind an artificial neural network are similar to those of a brain. Input data is fed to a first layer of neurons, where a computation is performed and the output is fed to the next layer of neurons. The result is the output of the last layer. The model learns by looking at the error between the output and the desired outcome, and adjusting the computations, typically through backpropagation methods. For IDSs, the frequency of certain attacks in the dataset impacts how well it will be recognised, making it difficult for the model to recognise less frequent attacks (**?**).

FUZZY LOGIC Unlike in conventional boolean logic, where there are only true or false values, fuzzy logic allows an instance to partially belong to multiple classes at the same time, allowing for degrees of uncertainty. This makes it a good classifier for an IDS, where the border between normal and abnormal behaviour is not well defined and, therefore, reducing the rate of false alarms (**?**).

SUPPORT VECTOR MACHINES This classifier works by finding a hyperplane in the feature space that separates two classes so that the distance between the hyperplane and the closest data points is maximised.

This is a favourable method in situations where the number of features is much greater than the number of data points (**?**).

HIDDEN MARKOV MODEL A Markov process is a stochastic model describing a sequence of states where the probability of transitioning to any particular state is dependent solely on the current state and time elapsed (**?**). An Hidden Markov model is a statistical Markov model where the modelled system is assumed to be a Markov process with unseen data, where its states represent unobservable conditions (**?**).

K-NEAREST NEIGHBOURS The idea behind this method is to classify a point based on the k nearest samples, where k is the number of samples being considered. Majority voting is used to determine the classification, which can be a pitfall if the class distribution is skewed (**?**).

RECURRENT NEURAL NETWORKS A Recurrent Neural Network (RNN) is a feed-forward deep learning neural network in which each unit bases its decision making process on its current input and the output of the previous input. It is widely used in natural language processing and speech recognition (**?**). Although it can only handle limited length sequences and may suffer from short-term memory if the input length is too long, RNN variants like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been developed to solve these issues (**?**).

DEEP NEURAL NETWORK A Deep Neural Network (DNN) is a deep learning structure composed of an input layer, many hidden layers, and an output layer. Increasing the number of hidden layers improves the model's abstraction level, and its used to model complex nonlinear functions **?**.

DEEP BELIEF NETWORK A Deep Belief Network (DBN) as a deep learning model composed of stacked Restricted Boltzmann Machines (RBM) (a two-layer model with data flowing in both directions) in layers followed by a softmax classification layer. In a DBN, connections exist between the layers but not between units within each layer **?**.

CONVOLUTIONAL NEURAL NETWORK Convolutional Neural Networks (CNNs) have three main types of layers: convolutional layer, pooling layer, and fully-connected layer. The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the network increases in its complexity, identifying greater portions of the input data. Earlier layers focus on simple features, and data progresses through the layers of the CNN, it starts to recognise larger patterns **?**.

Unsupervised learning

CLUSTERING This is an unsupervised pattern discovery approach which groups unlabelled data based on their similarities or differences. Clustering algorithms can be exclusive, overlapping, hierarchical, and probabilistic

(**?**). In exclusive clustering, a data point can exist be a part of no more than one cluster. K-means clustering is an example of an exclusive clustering algorithm, where the aim is to separate data objects into k clusters, where each observation belongs to the cluster with the nearest mean. On the other hand, overlapping clustering allows a data point to be a part of multiple clusters at once (*e.g.* fuzzy k-means). Hierarchical clustering seeks to create a hierarchy of clusters, and the approach can be categorised in two ways: agglomerative, which is a bottom up approach where sub-clusters are grouped together as one moves up through the hierarchy, and divisive, where the largest cluster in diameter is separated into sub-clusters (**?**). Probabilistic clustering, data points are clustered based on the likelihood that they belong to a particular distribution, with the Gaussian Mixture model being commonly used (**?**).

ASSOCIATION RULES An association rule describes a relationship among different attributes: *if (A and B) then C*, with metrics revealing how often a given relationship occurs in the data. Fuzzy association rules extend this functionality into categorical or quantitative data, instead of boolean only. These take the form of *if (X is A) then (Y is B)* (**?**).

Statistical inference

INDUCTIVE LEARNING This method attempts to find patterns and regularities in the observations, and then makes general conclusions or theories about the data in the form of rules.

3.6.4 *Support Vector Machines*

As stated in **??**, Support Vector Machines (SVMs) are a supervised machine learning model for non-probabilistic binary classification (although methods such as Platt scaling allow for it's use in a probabilistic classification setting (**?**)) through the construction of one or several hyperplanes that best separate data into distinct categories. This classifier works by mapping the input space into a higher-dimensional feature space, where separation between classes becomes easier, and finding a hyperplane that separates two classes so that the distance between the hyperplane and the closest data points is maximised (**?**). New data is then classified based on their positioning relative to those hyperplanes. An example of this can be seen in Figure **??**.

Mapping the input space into the feature space is done through an approach called the "kernel trick". This allows the algorithm to operate in the feature space without the need to map each data point into a higher dimension, which is computationally expensive. Instead, a kernel method is more efficient by using only the dot product of all pairs of data to achieve this same effect. Popular kernels include the linear, polynomial, and Gaussian (or radial basis function) kernels:

- Linear: $K(x, y) = x^T y$
- Polynomial: $K(x, y) = (x^T y + c)^d$, where $c >= 0$ is a parameter that trades off the influence of higher-order against lower-order terms in the polynomial, and d is the polynomial degree.

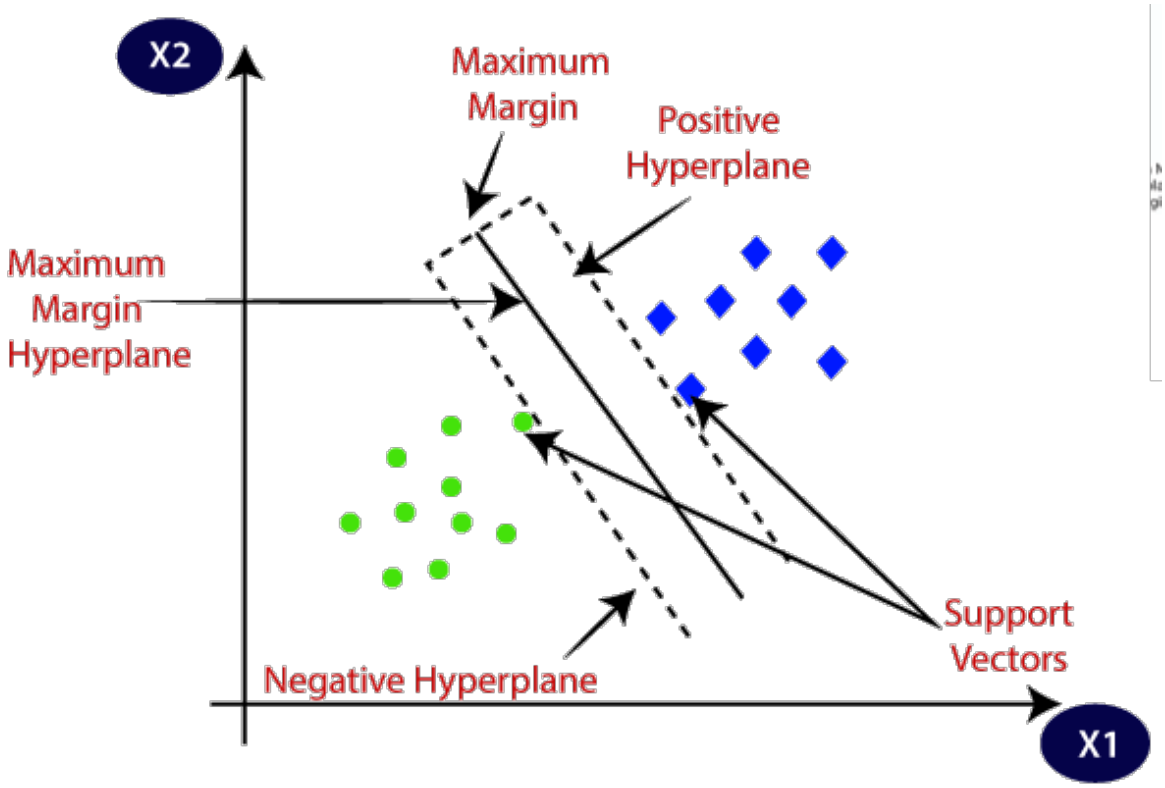


Figure 16: A hyperplane separating two classes (**?**)

- Radial basis function: $exp(-\frac{\|x-y\|^2}{2\sigma^2})$, where σ is a free parameter.
- Sigmoid (hyperbolic tangent): $K(x,y) = tanh(\alpha x^T y + c)$, where α is the slope and c a constant.

The Gaussian kernel consistently reports good results in practical situations, but is computationally more expensive than the linear or polynomial alternatives (**?**).

? then introduced an "extension to the support vector algorithm" adequate for novelty detection through unsupervised learning. Here, data is not classified as belonging to one of two classes, but as belonging to one class or everything else (hence the name). It has become one of the most popular methods for anomaly detection. Training is done on using only positive (normal) data, which is advantageous since anomalous data can be difficult to obtain when comparing to the large amount of normal data that can be easily obtained from most systems. This algorithm has been successfully used in a variety of domains, such as **???**.

The OCSVM algorithm attempts to create a decision boundary that provides the maximum separation between the training data points and the origin, with only a fraction of data points being allowed to reside outside of the decision boundary. To do this, the origin is treated as the only member of the non-target class.

The primary objective function of the OCSVM is

$$\min_{e,\xi,\rho} \frac{\|\omega\|^2}{2} - \rho + \frac{1}{\nu n} \sum_{i=1}^{n} \xi i$$
$$\text{subject to: } w^T \phi(x_i) \geq \rho - \xi_i \wedge \xi_i \geq 0$$

where ξ_i is the slack variable that allows certain points to reside outside the boundary (in order to avoid over-fitting), ω is the vector perpendicular to the decision boundary, $\phi(\cdot)$ is the nonlinear transformation that maps samples to the high-dimensional feature space, ρ is the distance to the origin, n is the training set size, and ν is the regularisation parameter which describes the trade-off between over-fitting and generalisation by setting an upper-bound on the fraction of outliers and a lower-bound on the number of support vectors.

Using Lagrange multipliers and a kernel function for dot product calculations, to avoid the computation of the nonlinear mapping, the decision function becomes

$$\min_{\alpha} \left\{ \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x, y) \right\}$$
$$\text{subject to: } 0 \leq \alpha_i \leq \frac{1}{\nu n} \wedge \sum_i \alpha_i = 1$$

which, after solving the dual problem and obtaining a set of model weights α_i, provides the decision function for any test vector z_*:

$$f(z_*) = \text{sgn} \left(\sum_i \alpha_i K(x, z_*) - \rho \right)$$

3.6.5 *Time complexity*

The amount of time it takes to run an algorithm dictates whether it can be trained in an online fashion or not. Algorithms that take a long time to process data can easily be overwhelmed by a streaming input, potentially bottlenecking network throughput.

Time complexity is typically expressed using big O notation (*e.g.* $O(n)$, $O(n\ log\ n)$, $O(n^2)$), where n is the size in units of bits needed to represent the input. (**?**) state that, as a rule of thumb, the $O(n)$ and $O(n\ log\ n)$ algorithms are considered to be linear time and are usable for online approaches. $O(n^2)$ is considered as acceptable for most practices. $O(n^3)$ and higher are considered to be much slower algorithms and used for offline approaches. Table **??** shows a comparison between different algorithms, assuming n to be much larger than m. Most machine learning methods have linear time complexity on the testing phase, and can, therefore, be used online.

Algorithm	Typical Time Complexity	Streaming Capable	Comments
Artificial Neural Network	O($emnk$)	low	e: number of epochs k: number of neurons
Association Rules	»O(n^2)	low	
Bayesian Network	»O(mn)	high	
Clustering, k-means	O($kmni$)	high	i: number of iterations until threshold is reached k: number of clusters
Clustering, hiearchical	O(n^3)	low	
Clustering, DBSCAN	O($n \log n$)	high	
Decision Trees	O(mn^2)	medium	
GA	O($gkmn$)	medium	g: number of generations k: population size
Nearest Neighbour k-NN	O($n \log k$)	high	k: number of neighbours

Table 3: Complexity comparison of machine learning algorithms during training (**?**)

Algorithm	Typical Time Complexity	Streaming Capable	Comments
ANN	O(emnk)	low	Jain et al. [107] e: number of epochs k: number of neurons
Association Rules	>> O(n^3)	low	Agrawal et al. [108]
Bayesian Network	>> O(mn)	high	Jensen [41]
Clustering, k-means	O(kmni)	high	Jain and Dubes [46] i: number of iterations until threshold is reached k: number of clusters
Clustering, hierarchical	O(n^3)	low	Jain and Dubes [46]
Clustering, DBSCAN	O(n log n)	high	Ester et al. [109]
Decision Trees	O(mn^2)	medium	Quinlan [54]
GA	O(gkmn)	medium	Oliveto et al. [110] g: number of generations k: population size
Naïve Bayes	O(mn)	high	Witten and Frank [89]
Nearest Neighbor k-NN	O(n log k)	high	Witten and Frank [89] k: number of neighbors
HMM	O(nc^2)	medium	Forney [111] c: number of states (categories)
Random Forest	O(Mmn log n)	medium	Witten and Frank [89] M: number of trees
Sequence Mining	>> O(n^3)	low	Agrawal and Srikant [92]
SVMs	O(n^2)	medium	Burges [112]

3.6.6 *Anomaly detection*

Novelty detection, also known as semi-supervised anomaly detection (**?**), is the task of recognising data in the test set that differs in some respect to the data available during training. It is particularly useful when there is a large amount of "normal" data, but insufficient "abnormal" data. This is common in settings where measurements of a normal operating state are inexpensive to obtain, but generating a large amount of data in abnormal states is both difficult and costly. The goal is to train a model with what is considered to be "normal" data and then use that model to detect anomalies, *i.e.* data not present in the training set. The techniques used should have a high detection rate, but with few false alarms, be scalable, and good time and space complexity. **?** classified anomaly detection techniques into five categories: probabilistic, distance-based, reconstruction-based, domain-based, and information-theoretic.

Probabilistic methods

This approach is based on estimating the generative probability density function of the data, which is then thresholded in order to determine if a test sample originated from the same distribution or not. The estimation of the underlying data density from multivariate training data falls under two categories: parametric and non-parametric.

Parametric statistics assumes that the distribution of the population is known and is based on a fixed set of parameters (**?**). The Gaussian distribution is commonly used for continuous variables, but more complex distributions may use mixture models such as Gaussian Mixture Model (GMM). State-space models, often used in time-series data, assume that there are unobserved variables or parameters (states) that describe the evolution of the underlying system. Common state-space models used for novelty detection are the Hidden Markov Model (HMM) and the Kalman filter.

Non-parametric statistics makes no assumptions about the population's distribution or it's parameters. The use of kernel density estimators is an approach in which the probability density function is estimated using a large number of kernels distributed over the data space (*e.g.* the Gaussian kernel).

Probabilistic methods are a mathematically well-grounded, effective, and transparent way to identify novel data as long as an accurate estimate of the data's probability density function has been obtained. However, this performance is limited when the training set is small. Non parametric approaches are usually preferred because prior knowledge of the data's distribution parameters is typically unavailable.

Distance-based methods

These methods determine the similarity between two observations based on distance metrics. Most methods are k-nearest neighbour- or clustering-based.

The k-nearest neighbours is a supervised algorithm that assumes that normal data points are close to each other, while anomalies are located far from these points. The Euclidean distance is a popular choice for distance measurement, and local density, in which the average of the k nearest neighbours is considered, determines if a point is an anomaly or not.

The k-means clustering algorithm, a unsupervised algorithm, is a clustering-based method for identifying abnormalities. Here, data points from the training set are divided into clusters and classifying data points based on the cluster into which they are inserted.

Both k-nearest neighbours and clustering algorithms rely on the existence of a suitable distance metric to detect anomalies. Although they are very effective at this task, they do not scale very well to high dimensional datasets.

Reconstruction-based methods

Reconstruction methods attempt to model the underlying training data and classify the testing data based on it's reconstruction error. Both neural networks and subspace-based methods can be trained this way.

Domain-based methods

Domain-based methods attempt to define a boundary based on the training data, and testing samples are labelled based on their position relative to the boundary. SVMs, for example, generate a hyperplane that maximises the minimum distance to the closest point of either of the two classes. As for novelty detection, SVMs have been used in one of two ways: through the one-class approach, or through the support vector data description approach.

The aim of the one-class approach, introduced by **?**, is to, though a kernel, separate all the data points from the origin and maximise the distance from this hyperplane to the origin. This does, however, require that a hyper-parameter corresponding to the percentage of data in the training set that is allowed to fall outside of this boundary.

The support vector data description method, proposed by **?** gives the training data a spherically shaped boundary with the minimum volume that encloses all, or most, of the data.

Although they are commonly used for anomaly detection, showing good performance, the kernel methods used can be computationally intensive. The choice of the appropriate kernel is also a challenge.

Information-theoretic methods

These methods attempt to detect novelties through significant changes in the information content of the data. Typical metrics include entropy, relative entropy, information gain, etc.

4

STATE OF THE ART

Intrusion detection is, as defined in **?**, "the process of monitoring the events occurring in a computer system or network, and analysing them for signs of intrusions". It can be deployed directly on the network node (host-based), where it monitors the activity for hosts containing sensitive information, or it can be deployed on the network (network-based), where it monitors traffic using sensors. Recent literature classifies the system's detection approach as being either signature-, anomaly-, or specification-based (**?**).

4.1 DETECTION APPROACHES

A signature-based IDS raises an alert when it finds a pattern in the network that corresponds to a signature in the system database. Its advantage comes from its simplicity, as it only requires a database of known attacks to compare network traffic to. However, this reliance on the attack signature database means that it will not recognise new attacks that are not stored there, meaning that regular updates are required.

The specification-based approach attempts to detect deviations from a protocol specification (well-defined rules that describe component behaviour). It is, again, a simple solution that requires only a set of rules that trigger an alert when not followed. Nonetheless, manually setting specifications is an error-prone and time-consuming task and must be adapted for different domains.

In the case of an anomaly-based IDS, an alert is raised when the network state deviates from what it considers to be normal behaviour. Here, the challenge relies upon defining what is considered normal behaviour, as a poorly defined model can generate alerts on what would otherwise be considered normal packets. A high rate of false positives could cause the driver to ignore the alerts completely. Techniques for defining what constitutes normal behaviour include machine learning, packet frequency and statistical-based analysis, as well as a hybrid approach.

In the traditional desktop environment, a hybrid approach is a combination of several IDS techniques, such as network- and host-based IDS. Here, the host-based IDS aggregates information into a standard format, which is then transmitted to the network-based IDS, where it is aggregated and processed. In the case of anomaly detection in CAN, a hybrid-based approach consists of the combination of more than one detection method, considering various packet features, e.g., ID field, payload, timing interval and frequency **?**.

4.2 IDS EVASION TECHNIQUES

Intrusion detection systems are commonly used in many networks, and attackers have adapted in order to avoid being detected. Evasion techniques mostly make the use of fragmentation, flooding, obfuscation and encryption.

Fragmentation is the process of dividing a packet into smaller packets, which are then reassembled when received. To analyse these fragmented packets, an IDS is required to re-assemble them. Attackers then take advantage of fragmentation overlap, overwrite and timeouts to escape detection. Flooding occurs when an attacker overwhelms the detector with a massive stream of packets, which causes it to fail. Obfuscation works by making a message difficult to understand, typically by changing the code while maintaining its functionality. This reduction in the code readability makes both static analysis and reverse engineering much more difficult. An IDS using a signature database will also see a decrease in its ability to detect attacks. Although encryption offers many security benefits, attackers can also leverage it to their advantage. Attacks on encrypted protocols, for example, cannot be detected by an IDS that relies on a signature database, as it cannot interpret the packets contents (**?**).

4.3 INTRUSION DETECTION IN THE CONTROLLER AREA NETWORK

Although IDSs are commonplace in traditional IP networks, they are still being developed for use in vehicles, more specifically in the CAN bus. Therefore, there still is no single best approach, but multiple different attempts to build a reliable IDS.

4.3.1 *Based on packet frequency*

? proposed a frequency-bases analysis of network traffic flow in order to detect anomalies. Through the use of a OCSVM, they found that they were able to identify when packets were artificially inserted or removed by the mean-time interval between packets of the same ID. Another frequency-based detection method was proposed by **?**, in which they recorded no false positives during the packet injection tests.

An IDS that is based on the offset ratio and time interval between request and response was proposed by **?**. The system periodically sends remote frame requests and analyses the response time, and attacks were successfully identified because of the abnormalities in the instant reply ratio, lost reply ratio, correlation coefficient between offsets and time intervals, and average response time.

A clock-based IDS was developed by **?**, where it identified ECUs based on the intervals of periodic messages. This baseline then allowed for the detection of abnormal behaviour in the network, with a reported false positive rate of 0.055%. Unlike most IDSs, it also facilitated the identification of the misbehaving ECU.

4.3.2 *Based on network entropy*

An entropy-based anomaly detection mechanism was proposed in **?**. Here, the developed IDS analysed variation in the network entropy (*i.e.*, how much coincidence a given dataset contains) to detect anomalies. Through increased

message frequency, message flooding, and simulation of unrelated events, they concluded that "deviations from the normal behavior of in-vehicle networks can successfully be identified by an information-theoretic detection approach". Another entropy-based system was proposed by **?**, in which they claimed that this represents a "viable approach for identifying CAN bus anomalies caused by the activity of attackers that [are] injecting messages over the CAN bus". However, limitations include only being able to detect attacks where packets are inserted at a high rate.

4.3.3 *Based on network data probability*

? used probability-based feature vectors extracted from the network packets to train a Deep Neural Network. The training was performed offline, while the detection was done against incoming packets in the network. They achieved a 99% detection racio against a TPMS spoofing attack. Another deep learning model was presented in **?**, in which the authors developed an LSTM that attempts to predict the next data word originating from each sender on the bus, flagging anomalies. However, it worked only for a single CAN ID and did not support online learning.

In **?**, the authors proposed the utilisation of a HMM to model what is considered normal vehicle function. The IDS works by signalling low probability changes of state (like a sudden jump from 25 kph to 160 kph) as an anomaly. The system was able to detect the introduction of anomalous RPM data, and they state that advantages of this approach include the fact that it can be applied in a plug-and-play fashion to both new and older vehicles.

4.3.4 *Hybrid*

? presented a hybrid anomaly detection system that combines both specification-based detection and machine learning, drawing from the advantages of both. In a first stage, static checks are made, and then the machine learning part applied learning checks for temporal analysis of the CAN packets.

A Hierarchical Temporal Model (HTM) model was developed by **?**. This is a memory-based model that is capable of online learning, meaning that it can continuously learn with new CAN data. Results from packet insertion and modification attacks show that it outperforms existing CAN IDSs that used RNN and HMM models.

4.3.5 *Based on protocol specifications*

The authors of **?** investigated the use of a specification-based IDS with security rules derived from the e CAN v2.0 and CANopen draft standard 3.01 communication protocol and object directory sections. These enforce the correct functioning of the protocol according to specification, signalling deviation from the rules as anomalous traffic. They tested six types of attacks (spoof, replay, read, modify, flood, and drop), concluding that most attacks can be readily detected and that "both protective and detective measures will be required in future in-vehicle networks".

4.3.6 *Based on attack signatures*

4.4 CHALLENGES FOR AUTOMOTIVE

There are, however, several challenges that must be overcome to successfully implement an IDS in a CAN bus. First, most ECUs have very limited resources, meaning low memory capacity, computational power, and data transmission rate. Another consideration is the time-sensitive nature of the packets being transmitted (**?**). Proper vehicle function must be maintained, which means that CAN packets must be managed in real-time. Otherwise, the safety of the driver might be at risk, as there would be an additional delay in information processing. Typical IDS solutions were developed for conventional Internet Protocol (IP) networks. As this is environment is very different from the in-vehicle one, adopting existing solutions is not possible. The nature of the vehicle environment is also a constantly moving one, and at various speeds. This means that an IDS must have a certain degree of autonomy, as establishing a connection to an outside party may not always be possible. Lastly, it must be an efficient solution, both in terms of cost, for it to be implemented in mass-production, and its physical aspects, i.e., not requiring massive amounts of rewiring, as it may be infeasible for manufacturers (**?**).

In a traditional IT setting, a system administrator is usually in charge of maintaining the system functioning properly and up to date, and is also the one notified in case of an alert. In the automotive domain, however, the IDS cannot be maintained by a qualified administrator, and a typical user cannot be expected to be an IT expert. This means that tasks like maintenance and alert handling can be delegated to him in a limited way. This might motivate the design of the system to be as autonomous as possible, but the high safety requirements of automotive systems make this a challenging task. The authors of (**?**) state that "in general, automotive systems shall never take safety relevant decisions autonomously but may only support the driver in his reaction (final decisions regarding control or safety of the vehicle are only to be made by the driver)". Because of this, special care must be taken when deciding upon the way of communicating an alert to the driver. This communication must be done in a modern way, taking advantage of the car's multimedia system, and must not cause panic. If a driver reaction is required, it is advantageous to provide suggestions, and the need to disrupt the driver must be kept to a minimum. They suggested a gradual approach to this communication, depending on the severity of the incident, which can be seen in Figure **??**. From left to right, as the severity of the incident increases, the amount of time it takes to grab the attention of the driver reduces. A visual-only message would be read when the driver decides to look at the display, but a haptic intervention would cause immediate alert. However, the amount of information that can be communicated is reduced, as a visual or acoustic message would reveal more about the incident that a haptic intervention.

4.5 INCIDENT RESPONSE

Unlike in other settings where an IDS might be used, like in desktop IT, one can no expect that the typical vehicle user is an IT expert. Therefore, maintenance and alert handling can be delegated to the driver in only a very limited way.

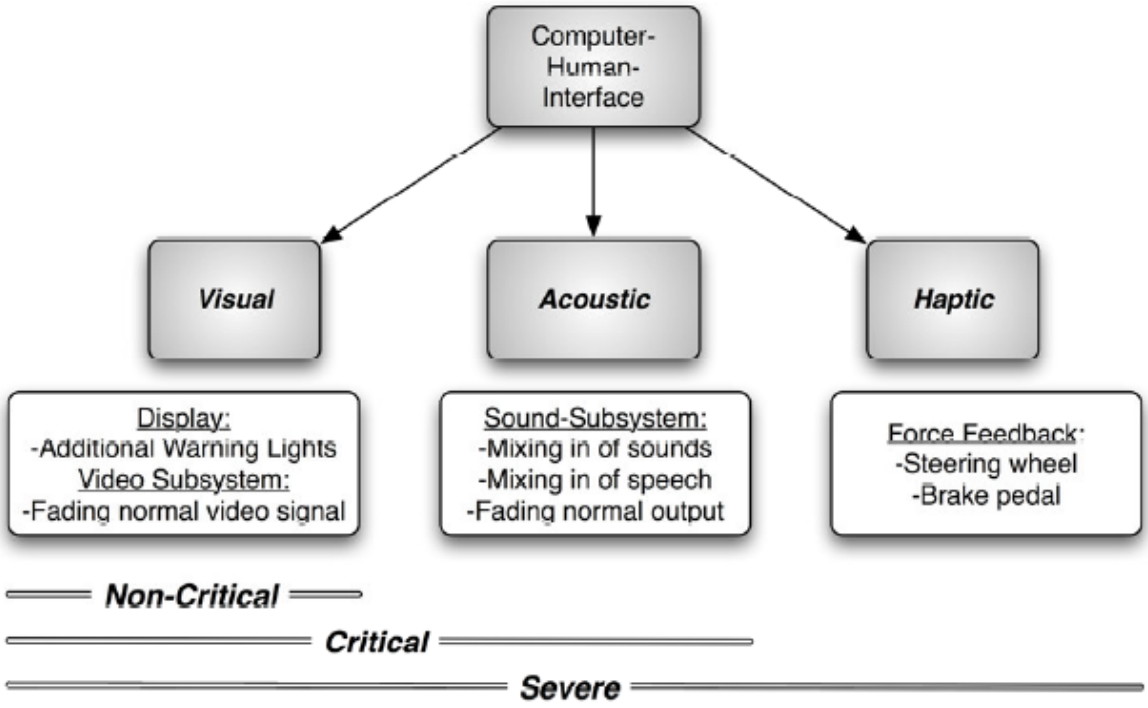


Figure 17: Stages for alert communication (**?**)

The lack of a knowledgeable user might motivate building a system that is as self-contained as possible. This would include being able to respond do intrusions. However, such an intrusion response mechanism is challenging to build given the high safety requirements of the automotive domain. Therefore, it is important to consider how the system should respond in case it detects an attack. **?** identified four aspects that should be taken into consideration when designing a way to communicate an alert to the driver: the use of a modern way of communicating, transmission of a sense of peace of mind, avoiding repetition, and support of the driver's decision making. They then propose a three-stage model, which can be seen of Figure **??** for alert communication to the driver depending on the severity of the intrusion. The three stages are visual, containing elements like the dashboard or central screen, acoustic, through the vehicle's sound system, and haptic, through force feedback in the break pedal or steering wheel.

Increasing severity is implicated by going from visual to acoustic and then to haptic. A visual-only message is used for non-critical alarms as the driver may not immediately look at the visual indication. Combining visual with acoustic signals indicates a critical alarm, as the driver would be immediately alerted. Haptic signals, like steering wheel vibration, is reserved for only severe alarms as it might frighten the driver. However, communication ability increases by going from haptic to acoustic and the to visual. Haptic signals are only able to indicate that something is wrong, without further information. Audible signals are able to provide more context, with visual messages informing the driver about what is the problem.

The severity of the alarm, however, is not the only factor to determine what stage should be used. Environmental context gathered from the vehicle's multiple sensors may aid in this decision. If the light sensors, for example,

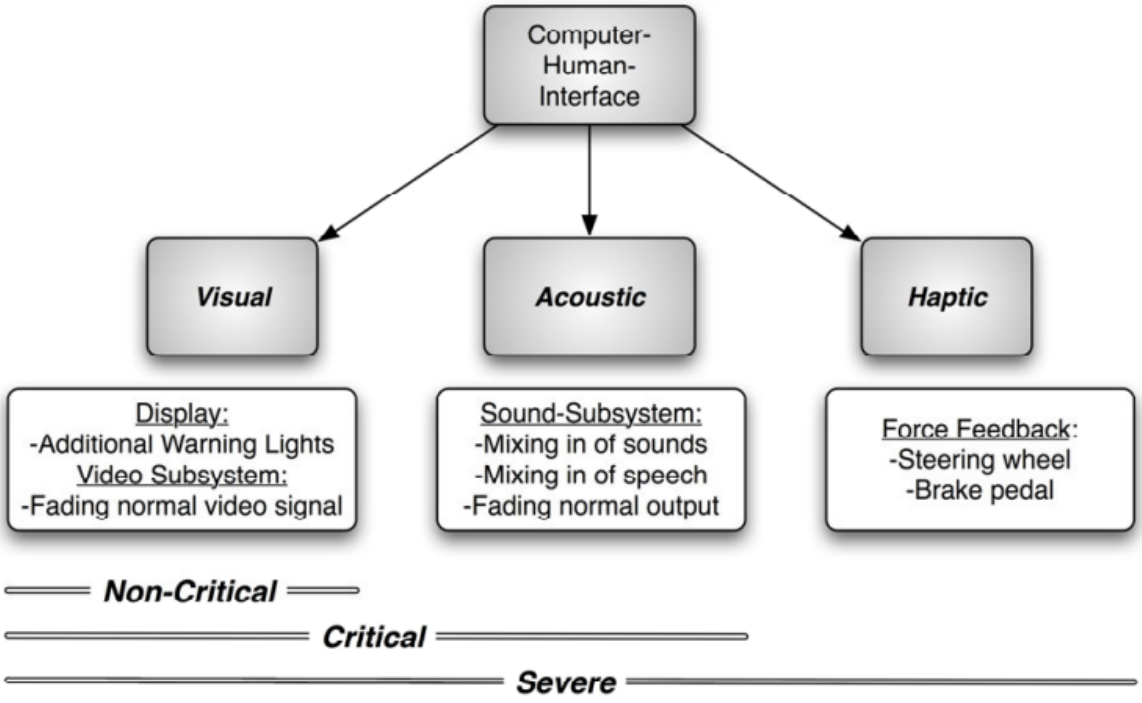


Figure 18: Three stages model for IDS alert communication (**?**)

detect a high level of brightness, the system may opt for audiovisual indication in a non-critical situation, as a visual-only warning would be difficult for the driver to notice.

Part II

CORE

5

CONTRIBUTION

Many intra-vehicle IDS approaches have been published, as shown in **??**, but they have mostly relied on a few labelled datasets, such as **?**. Here, the proposed system aims to be an IDS based on unsupervised machine learning for anomaly detection that makes decisions based on the statistical and information-theoretic characteristics of network traffic gathered in real-time. In other words, the system is able to learn what constitutes normal traffic, and signal when it detects behaviour that it considers to be novel. This would allow manufacturers to deploy the IDS in a plug-and-play fashion, without having to tune it for every different configuration of the vehicle.

6

DATASETS

6.1 CAR HACKING DATASET

These datasets were generated as part of the study published by **?**. Traffic was collected from a Raspberry Pi connected to a real vehicle via the OBD-II port, while another Raspberry Pi was used to inject traffic. The vehicle was parked with the engine turned on throughout the data gathering process. Five datasets were produced, with an overview of the attack datasets being available on Table **??**.

- Normal: An attack-free dataset
- DoS attack: Packets with the ID set to 0 were injected every 0.3 ms, impairing network availability
- Fuzzy attack: Packets with random IDs and payloads were injected every 0.5 ms, aiming to cause vehicle malfunction
- Gear spoofing attack: Packets a specific ID and payload were injected every 1 ms, successfully changing the gear value on the instrument panel
- RPM spoofing attack: Packets a specific ID and payload were injected every 1 ms, successfully changing the RPM gauge on the instrument panel

Attack type	Normal messages	Injected messages
DoS Attack	3,078,250 (84%)	587,521 (16%)
Fuzzy Attack	3,347,013 (87%)	491,847 (13%)
Gear Spoofing	2,766,522 (82%)	597,252 (18%)
RPM Spoofing	2,290,185 (78%)	654,897 (22%)

Table 4: Car Hacking dataset overview

Purpose	Status	Flood	Spoof	Replay	Fuzzy
Training	Driving	77,373 (4.1%)	3,879 (0.2%)	23,775 (1.3%)	45,474 (2.4%)
	Stationary	76,807 (4.3%)	3,877 (0.2%)	23,818 (1.3%)	44,405 (2.5%)
Submission	Driving	96,559 (4.8%)	22,489 (1.1%)	37,869 (1.9%)	44,770 (2.2%)
	Stationary	95,120 (5.4%)	20,094 (1.1%)	25,012 (1.4%)	51,923 (3.0%)

6.2 IEEE CAR HACKING: ATTACK & DEFENSE CHALLENGE 2020

The datasets published by **?** were created to be used in the Car Hacking: Attack & Defense challenge. It had two rounds: preliminary and final. For the preliminary round, training and submission datasets were provided, both containing driving and stationary vehicle traffic. For the final session, a single dataset with multiple attacks was created.

An overview of the preliminary datasets can be seen on Table **??**. For both driving and stationary scenarios, the datasets contained flooding, spoofing, replay, and fuzzing attacks. Attack-free datasets were also provided.

6.3 AUTOMOTIVE CONTROLLER AREA NETWORK (CAN) BUS INTRUSION DATASET V2

The datasets made available by **?** consists of bus data from an Opel Astra, a Renault Clio, and a custom built prototype. For all three systems, attack-free datasets were provided along with datasets containing diagnostic, fuzzing, replay, spoofing and DoS attacks.

6.4 CRYSYS LAB

This dataset was obtained from **?** and consists of CAN traffic collected from 25 minutes of driving in the highway as well as in small streets. A variety of packet modification attacks were simulated using the tool publicly available in **?**, which allows for easy modification of CAN log files. These attacks consist of modifying the payload of packets with a given ID, akin to a man-in-the-middle attack. Five attack datasets were generated, along with the attack-free version, and the attacks target the payload of the packets with ID 120 starting at 50% and ending at 90% of the total file. The attacks consist of holding the payload at a constant value, modifying the payload to be a random value, adding an increasing amount to the original payload, decreasing the payload from 255 to 0 and repeating, and adding a delta of 1000 to the original payload. Unlike other datasets, were packet insertion is performed and the attack can mostly be detected via the interval between arriving packets, these attacks do not insert any packets. Attacks can only be detected by observing anomalies in the behaviour of the payloads.

Figure 19: Setup used to generate real CAN traffic

6.5 CES

The setup used to generate real CAN traffic can be seen on Figure **??**. It is composed of various real vehicle ECUs, along with a modern touchscreen, and a Raspberry Pi to capture CAN traffic and running the IDS here proposed. Three attacks were simulated (fuzzing, flooding, and spoofing) by inserting packets every 200ms. The following unlabelled datasets were generated:

- Baseline: 2,508,370 packets of attack-free traffic.
- Fuzzy attack: 184,833 packets containing two instances of a fuzzing attack.
- Denial-of-Service attack: 235,187 packets containing two instances of a Denial-of-Service attack.
- Left blinker spoofing attack: 375,569 packets containing two instances of a spoofing attack targeting the left blinker signal.

7

APPLICATION

7.1 IDS ARCHITECTURE

In **??**, various approaches to IDS deployment were presented, most notably its placement (host or network-based) and detection techniques (signature, specification, anomaly, or hybrid). Here, a host-based IDS for anomaly detection is put forward. Instead of the IDS residing in a separate ECU connected to the network, the IDS is deployed on all of the vehicle's ECUs. It then monitors the same messages as the ECU in which it is deployed. Since it would be present on all ECUs, an attack on any relevant message ID would, theoretically, trigger at least one alarm. Monitoring a small set of IDs also aids with detection because an anomaly in the behaviour of a certain ID becomes more noticeable in the average of the set when compared to a larger number of IDs being monitored.

Features are extracted from a rolling window with an overlap of 75% between consecutive windows. Given the same amount of packets, this allows for the extraction of a larger amount of features during the training phase, when compared to non-overlapping windows.

In this chapter, the IDS' training process is explained, along with how the system is structured. The feature selection process is then described. Finally, the prototype's deployment in an embedded environment is detailed.

A diagram of the overall system can be seen in Figure **??**.

7.2 TRAINING

Training the IDS can be done both in an online and offline fashion. Training online means that a baseline reading is gathered from live traffic, which must be attack-free to generate an accurate model because attacks are detected based upon significant deviations from the baseline traffic's characteristics. Offline training is done by providing the IDS with a Comma Separated Values (CSV) file containing CAN traffic that will serve as a baseline.

Features are extracted using a rolling window on the collected traffic, which are then scaled to fit in the [0, 1] interval. The minimum and maximum values of each feature are saved to perform proper feature scaling when the IDS is deployed.

The kernel method is employed using the Gaussian kernel with $\epsilon = 1$ and the model is trained with the ν parameter set to 0.01. The IDS is then saved to the filesystem so it can be recovered upon a system restart, and has the following components:

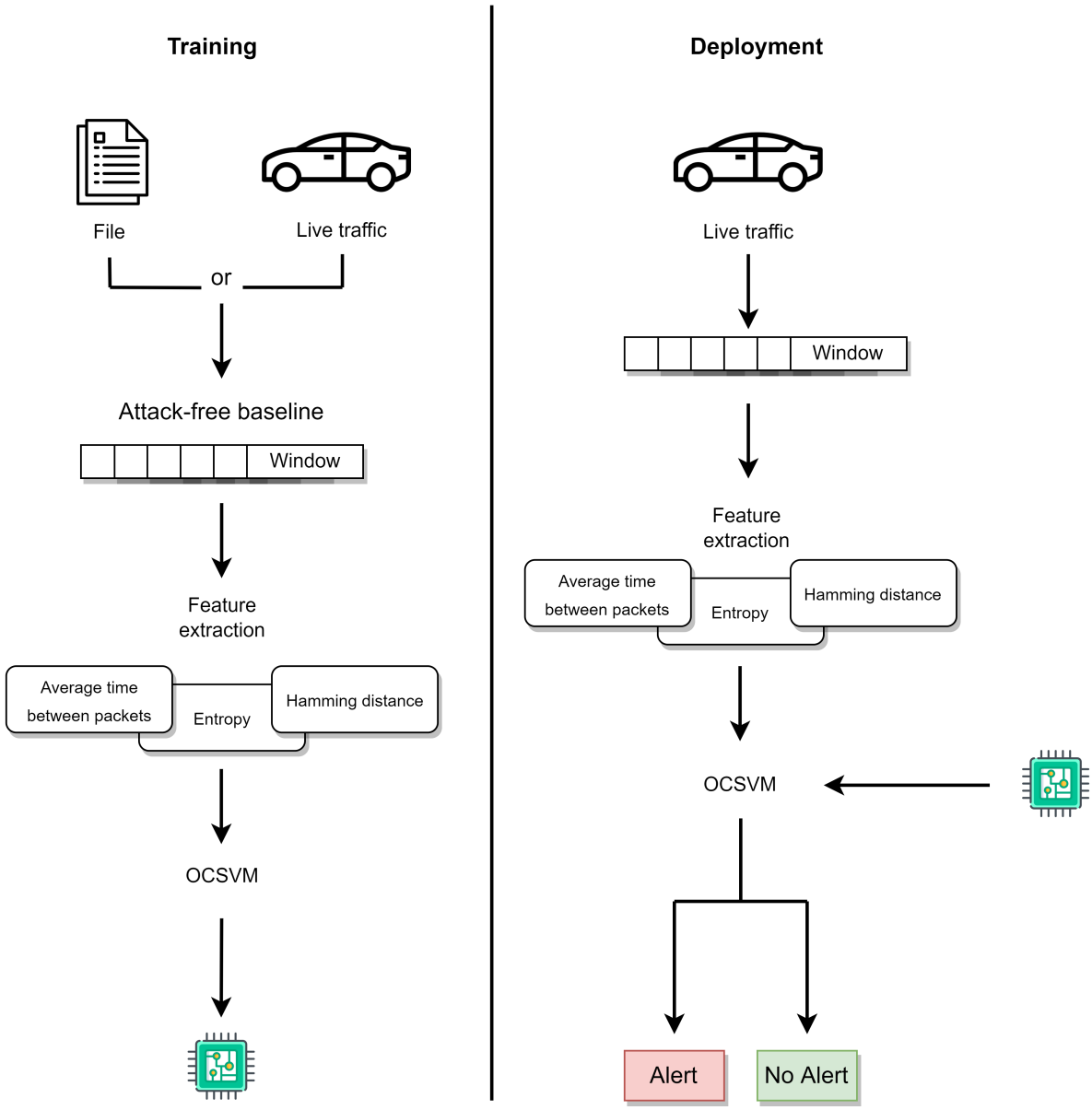


Figure 20: IDS architecture

- *model*: The OCSVM
- *scaler*: A vector of tuples containing the minimum and maximum values of each feature from the baseline
- *window*: The collection of packets from which features are extracted
- *size*: The amount of packets the window can hold
- *slide*: The amount of packets that leave and enter the window in a FIFO fashion
- *counter*: The number of packets that have left and entered the window between slides
- *monitor*: The collection of IDs to monitor

7.3 FEATURES

This section focuses on the features extracted from CAN bus traffic that are then used as input to the OCSVM model. Prior research on IDSs that focus on CAN has found success in the use of frequency analysis and information-theoretic algorithms, as shown in Section **??**. Frequency analysis was one of the first methods used to detect packet insertion attacks. Since many packets of the same ID are sent at a consistent interval (typically 10 or 20 milliseconds), a variation of these timings caused by a bad actor inserting packets into the network would be picked up as an anomaly. On the other hand, information-theory may aid in detecting anomalous behaviour caused by packet modification attacks, or packet low-frequency packet insertion.

Here, the proposed features are described, and the feature selection process is presented.

7.3.1 *Packet frequency*

Frequency analysis consists of measuring the average time between consecutive packets with the same ID. Since many packets sharing the same ID are sent at regular intervals, an attacker inserting new packets into the network would decrease the time between packets, which would be detected by the IDS. Likewise, packet suppression would also trigger an alert.

Calculating the arrival time between packets can done by

$$T(p_{t_0}, p_{t_1}) = t_1 - t_0$$

where t_0 and t_1 are consecutive arrival timestamps for packets of the same ID.

7.3.2 *Shannon entropy*

In the CAN domain, traffic is more restricted than in standard computer networks. This means that, generally, entropy in an automotive network is lower (**?**). If an attacker replaces the payload of a given ID, it likely would

change the network entropy, triggering an alert (an instance of this would be a constant payload on an ID that usually sends distinct values, or vice-versa).

The concept of information entropy was first introduced by **?**. Shannon entropy is often defined as being the amount of information or uncertainty of a random variable. For an alphabet X, and a random variable $x \in X$ distributed according to $p : x \to [0, 1]$, Shannon entropy is defined as

$$H(X) = -\sum_{i-1}^{n} P(x_i) \log_2 P(x_i)$$

An occurrence with probability 1 has no entropy, while a set of occurrences with a balanced probability distribution has an entropy value of 1.

7.3.3 *Hamming distance*

In information theory, the number of points where the matching symbols are different between two strings of equal length is known as the Hamming distance (**?**). For example, the Hamming distance between 0000 and 1111 is 4, while the Hamming distance between 0101 and 1101 is 1. The formula for evaluating the Hamming distance between two words of length k is

$$H_d(x, y) = \sum_{i=1}^{k} |x_i - y_i|$$

Here, the proposed approach consists on calculating the Hamming distance between consecutive packets in the network, similar to what was proposed by **?**. The authors formulated this as being

$$H_d(p_t, p_{t+1}) = \sum_{i=1}^{k} p_t^i \otimes p_{t+1}^i$$

where p_t is a generic payload at time t and p_t^i is the i_{th} element of that payload.

7.3.4 *Difference between bytes*

The calculation of the difference between bytes is done by

$$D(p_t, p_{t+1}) = \sum_{i=1}^{k} |p_t^i - p_t^{i+1}|$$

where p_t is a generic payload at time t and p_t^i is the i_{th} element of that payload.

The hypotheses is that this metric, similar to entropy and Hamming distance, contributes to the modelling of what constitutes normal packet payload behaviour.

7.3.5 *Selection*

For all datasets, five features were extracted from each rolling window of traffic: average frequency, average entropy, average bit-wise Hamming distance, average byte-wise Hamming distance, and average byte-wise difference. Since it would take a lot of computational power to extract all five features from each window while keeping up with incoming traffic, dimensionality reduction was performed. To do so, correlation between features and the datasets' labels was calculated. Some illustrative results are shown in Figure **??**, with all results obtained being present in Appendix **??**. All features were extracted from a 1000 packet rolling window, with a slide of 250 packets.

Figure **??** shows some illustrative examples of how these features are correlated with each other and the label. In the case of a modification attack, payload behavior will be altered, but no packets will be inserted. This is demonstrated in Figure **??**, where the average time between packets (*AvgTime*) shows no correlation with the dataset's label, but both entropy and Hamming distance-related features show a higher correlation. Figure **??** shows feature correlation values in the case where a variety of spoofing, replay, fuzzing, and flooding attacks are performed. All features except the difference between consecutive payloads of the same ID (*GapBytes*) show some correlation with the label, meaning that they are relevant predictors of many common attacks. Lastly, Figure **??** shows feature correlation values in the case of a replay attack. Here, only the average time between packets is relevant, since many previously observed packets are being inserted into the CAN bus. Payload-related features show no correlation with the label, which is to be expected because the attack consists of inserting packets whose payload is not abnormal.

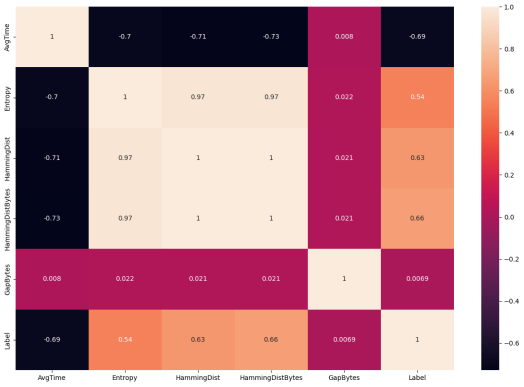
Thus, after observing several attack types, including with and without packet insertion, the average byte-wise Hamming distance and the average arrival time between packets of the same ID appear to be the most promising in predicting whether an attack is present or not.

7.4 EMBEDDED SYSTEM ENVIRONMENT

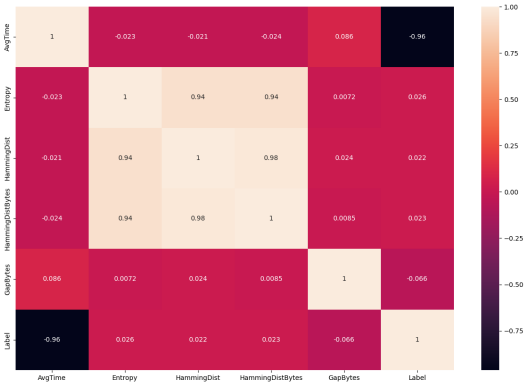
To simulate the environment where the proposed system would be deployed, the program was written in Rust 1.62.0 and compiled to the ARM v7 architecture to run on a Raspberry Pi 4 Model B connected to the ECU cluster shown in Figure **??**. Code documentation is available in Appendix **??**.

To verify if the system would be able to analyse real-time traffic, the number of packets per second it was able to process was recorded. This came at around 1600. However, it was unclear whether this was a bottleneck or if it was the real network throughput. To clear this, traffic was logged onto a CSV file and processed by the Raspberry Pi as if it was live traffic, but read from memory instead of a network socket. The average number of packets per second processed came at 39,195.426, while the average number of packets per second in the log file was 1,678.878. The proposed system should therefore be able to perform real-time analysis in an embedded environment. CPU usage was recorded as being 5%, signalling that its implementation in a real ECU would not raise serious concerns about resource usage.

(a) CrySyS Lab - Adding an incrementing amount to packet payload



(b) IEEE Car Hacking: Attack & Defense Challenge 2020 - Attacks while driving



(c) Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2 - Replay attack

Figure 21: Correlation between extracted features

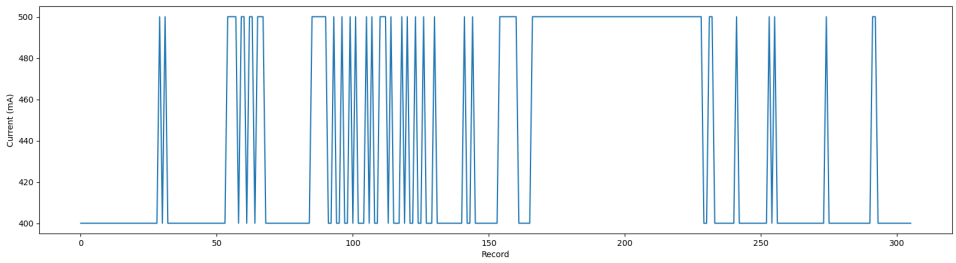


Figure 22: Power consumption from the Raspberry Pi while executing the proposed IDS

It is also important to assure that the algorithm does not consume too much power, as it would reduce the overall range of the vehicle. Figure **??** shows how much current the Raspberry Pi was drawing while executing the IDS. Applied voltage was constant at 4900mV. The baseline current was established to be at 400mA, and it rose up to 500mA when the IDS was executing. This comes at 0.5W, which is negligible.

8

RESULTS

8.1 CAR HACKING DATASET

Tests were made monitoring the IDs 350, 130, 131, 140, 2c2, 2b0, 316, 18f, 43f, and 440. The gear spoofing attack targets the ID 43f, and the RPM gauge spoofing attack targets the ID 316. Results are shown in Table **??**. Values in bold were obtained when using three features as input to the model: average time between packets, entropy, and Hamming distance. The remaining values were obtained using only the average time between packets and Hamming distance.

When using only two features, the F1-score for both spoofing attacks are above 90%. However, the occurrence of false positives remains high. Including entropy wields generally better results. Although the rate of false negatives is slightly worse, both precision and F1-score show bigger improvements. The fuzzing attack is harder to detect in both cases.

Graphics showing feature behaviour during tests can be seen of Figures **??**, **??**, **??**, and **??**.

8.2 IEEE CHALLENGE DATASET

The model was trained by gathering a baseline from the attack-free datasets in the stationary and driving modes, separately. Twelve IDs were monitored in each state (44E, 164, 260, 140, 541, 386, 490, 366, 367, 563, 4CB, 50E for the stationary state and 44E, 164, 130, 140, 3A0, 386, 490, 366, 367, 563, 4CB, 2B0 for the driving state). To test the IDS' performance, the stationary model was executed on the datasets provided for the preliminary and

Data	FN rate	Error rate	Precision	Recall	F1-score
Fuzzing	1.597%	20.300%	66.091%	98.403%	79.074%
	1.632%	**10.746%**	**79.134%**	**98.368%**	**87.709%**
Gear spoofing	0.869%	10.470%	86.554%	99.131%	92.416%
	1.062%	**2.910%**	**96.698%**	**98.938%**	**97.799%**
RPM spoofing	0.871%	10.256%	87.106%	99.129%	92.729%
	0.948%	**2.723%**	**96.891%**	**99.052%**	**98.034%**

Table 5: Performance metrics on Car Hacking dataset

Data	FN rate	Error rate	Precision	Recall	F1-score
Stationary Preliminary	0.601%	19.242%	71.264%	99.399%	83.013%
	0.701%	**16.635%**	**74.232%**	**99.299%**	**84.955%**
Stationary Final	0.403%	4.478%	93.283%	99.597%	96.337%
	0.403%	**8.004%**	**88.349%**	**99.597%**	**93.636%**
Driving	2.591%	1.218%	99.412%	97.409%	98.400%
	3.935%	**2.140%**	**98.427%**	**96.065%**	**97.184%**

Table 6: Performance metrics on IEEE Challenge dataset

final round of the competition, and the driving model was executed on the dataset provided for the preliminary round (the final round did not include a dataset for the driving state).

Test results are present in Table **??**. Values in bold were obtained from tests made using the average time between packets, Hamming distance, and entropy. Remaining values exclude entropy. Using only the average time between packets and Hamming distance. However, results for the stationary final and driving tests are above 90% in both the precision and F1-score metrics, the IDS did not perform so well in the stationary preliminary test. Adding entropy to the set of features reduces the amount of false positives, at the expense of a small increase in the amount of false negatives. However, the IDS performs slightly worse in the other two tests. Still, it may still be considered preferable to use the three features.

Graphics showing feature behaviour during tests can be seen of Figures **??**, **??**, and **??**.

8.3 AUTOMOTIVE CONTROLLER AREA NETWORK (CAN) BUS INTRUSION DATASET V2

Here, the results the tests performed on the datasets described in **??** are shown in Table **??**. For both the dataset created from traffic gathered from an Opel Astra and a Renault Clio, the IDS was tested on payload fuzzing, message deletion, and replay attacks. For the Opel Astra, the monitored IDs were 0C9, 1C8, 1E2, 232, 348, 34A, 451, 0F1, 1F3, 1E5, and 1A1. For the Renault Clio, the monitored IDs were 2C6, 186, 18A, 1F6, 211, 217, 214, 218, 4FA, and 090. All monitored IDs include the IDs targeted in the attacks, with the rest chosen at random.

The payload fuzzing attack goes unnoticed. This happens because, in this case, the attack consists of the modification of the payload of 10 consecutive messages on a given ID, which does not affect either the entropy or Hamming distance values enough to trigger an alert.

Graphics showing feature behaviour during tests can be seen of Figures **??**, **??**, and **??**.

8.4 CRYSYS LAB

Table **??** shows the results of the tests performed on the CrySyS Lab dataset and the generated modification attacks, as described in **??**. Tests were made monitoring 10 IDs (110, 120, 140, 180, 1A0, 280, 290, 295, 300, and 301) and only the attacked ID (120), also with and without entropy values. Values obtained from tests that included entropy as part of the feature set are displayed in bold.

Opel Astra					
Data	**FN rate**	**Error rate**	**Precision**	**Recall**	**F1-score**
Payload fuzzing	100.000%	0.408%	N/A	0.000%	N/A
	100.000%	**0.408%**	**N/A**	**0.000%**	**N/A**
Message deletion	18.750%	0.491%	100.000%	81.250%	89.655%
	18.750%	**0.491%**	**100.000%**	**81.250%**	**89.655%**
Replay	0.000%	0.000%	100.000%	100.000%	100.000%
	0.000%	**0.000%**	**100.000%**	**100.000%**	**100.000%**
Renault Clio					
Data	**FN rate**	**Error rate**	**Precision**	**Recall**	**F1-score**
Payload fuzzing	100.000%	2.800%	0.000%	0.000%	N/A
	100.000%	**2.800%**	**0.000%**	**0.000%**	**N/A**
Message deletion	27.273%	3.629%	100.000%	72.727%	84.746%
	27.273%	**3.629%**	**100.000%**	**72.727%**	**84.211%**
Replay	0.000%	0.000%	100.000%	100.000%	100.000%
	0.000%	**0.000%**	**100.000%**	**100.000%**	**100.000%**

Table 7: Performance metrics on the Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2

8.5 CES

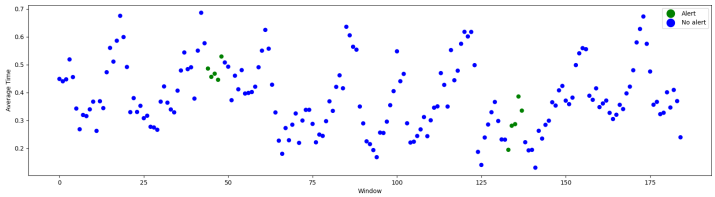
The IDS was trained with the attack-free dataset monitoring the IDs A0, A1, AB, CA, DE, F1, F3, F9, FA, 145, 231, 29D, and 510. Since this dataset is unlabelled, performance metrics, like precision, cannot be used. Instead, Figures **??** and **??** shows how feature values behave during the fuzzing and spoofing attacks, respectively.

In the case of the fuzzing attack, since packets were inserted at a rather slow rate (200ms), the average time between packets, shown in Figure **??**, does not behave abnormally. Network entropy also remained largely unchanged (Figure **??**). In Figure **??**, which shows Hamming distance values, two identifiable spikes corresponding to the two instances of the attack can be seen.

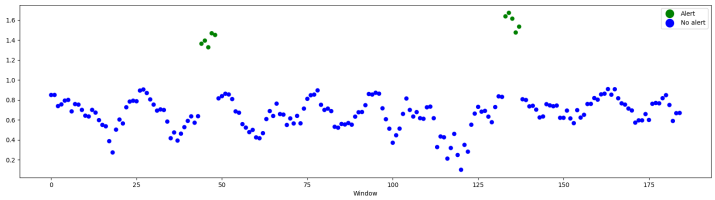
As for the spoofing attack, analysis of packet frequency (Figure **??**) does not show any anomalies given the slow rate at which the packets were inserted. Entropy, which can be seen in Figure **??**, was also not altered. Hamming distance, however, became anomalous during the two instances of the attack represented by the spikes in Figure **??**.

Monitoring 10 IDs					
Attack	**FN rate**	**Error rate**	**Precision**	**Recall**	**F1-score**
Constant value	100.000%	40.451%	0.000%	0.000%	N/A
	100.000%	**40.451%**	**0.000%**	**0.000%**	**N/A**
Random value	8.992%	3.687%	99.851%	91.008%	95.225%
	8.311%	**3.357%**	**100.000%**	**91.689%**	**95.665%**
Adding an increasing value	94.959%	38.415%	97.368%	5.051%	9.585%
	96.185%	**38.855%**	**100.000%**	**3.815%**	**7.349%**
Decreasing value	100.000%	40.396%	N/A	0.000%	N/A
	100.000%	**40.396%**	**N/A**	**0.000%**	**N/A**
Delta of 1000	99.728%	40.341%	66.667%	0.272%	0.543%
	99.864%	**40.341%**	**100.000%**	**0.136%**	**0.272%**
Monitoring 1 ID					
Attack	**FN rate**	**Error rate**	**Precision**	**Recall**	**F1-score**
Constant value	4.511%	1.911%	100.000%	95.489%	97.692
	5.263%	**2.229%**	**100.000%**	**94.737%**	**97.297**
Random value	1.504%	0.637%	100.000%	98.496%	99.242%
	0.752%	**0.318%**	**100.000%**	**99.248%**	**99.623%**
Adding an increasing value	38.346%	16.242%	100.000%	61.654%	76.279%
	24.812%	**10.510%**	**100.000%**	**75.188%**	**85.837%**
Decreasing value	100.000%	42.357%	N/A	0.000%	N/A
	100.000%	**42.357%**	**N/A**	**0.000%**	**N/A**
Delta of 1000	100.000%	42.357%	N/A	0.000%	N/A
	100.000%	**42.357%**	**N/A**	**0.000%**	**N/A**

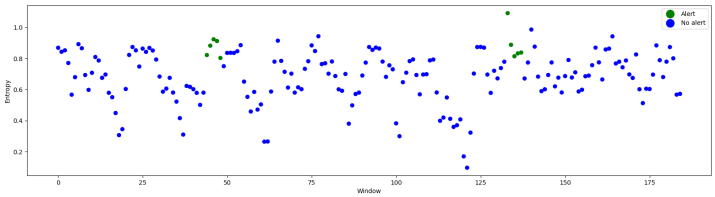
Table 8: Performance metrics on the CrySyS Lab datasets



(a) Average time between packets

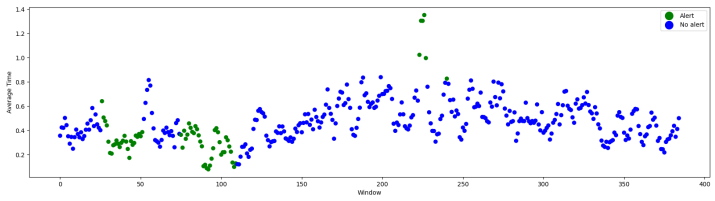


(b) Hamming distance

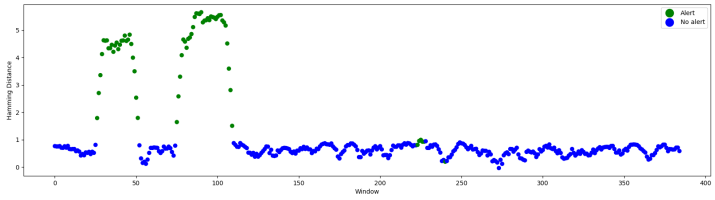


(c) Entropy

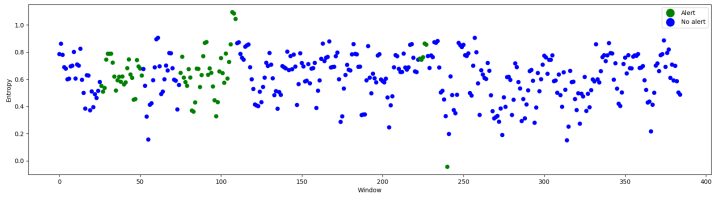
Figure 23: Features extracted from the fuzzing attack present in the CES dataset



(a) Average time between packets



(b) Hamming distance



(c) Entropy

Figure 24: Features extracted from the spoofing attack present in the CES dataset

Part III

CONCLUSION

9

FUTURE WORK

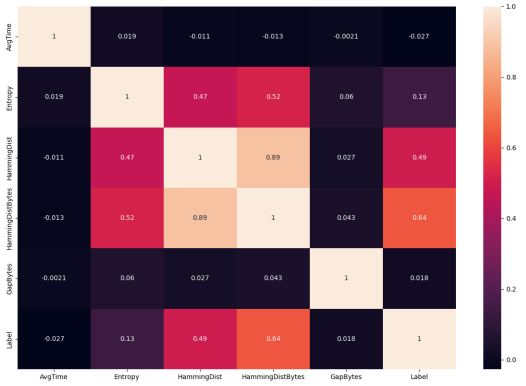
10

CONCLUSION

A

FEATURE SELECTION

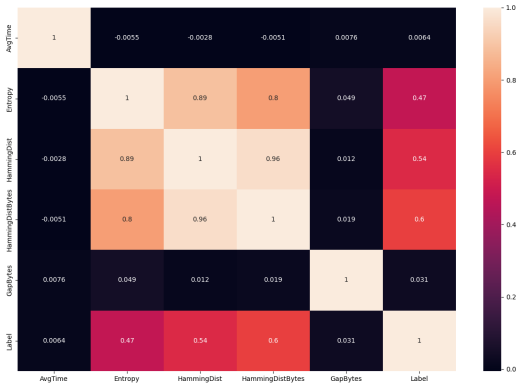
(a) Fuzzing attack



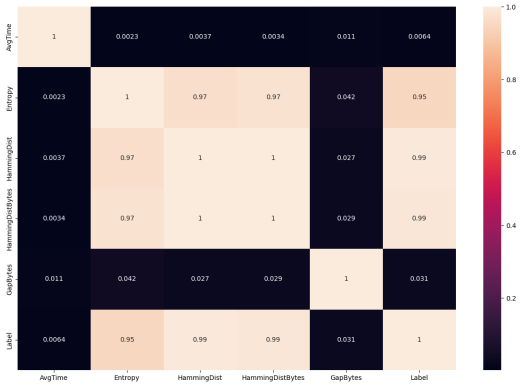
(b) Gear spoofing attack

(c) RPM spoofing attack

Figure 25: Correlation between extracted features obtained from the Car Hacking dataset

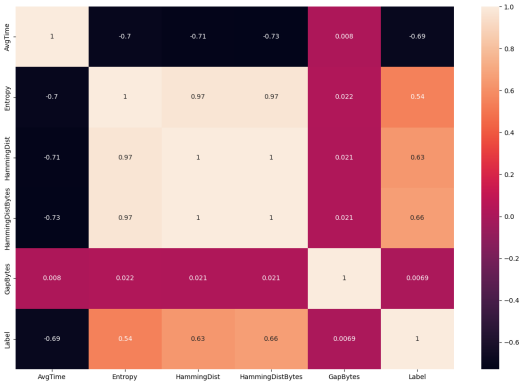


(a) Incremental payload modification attack

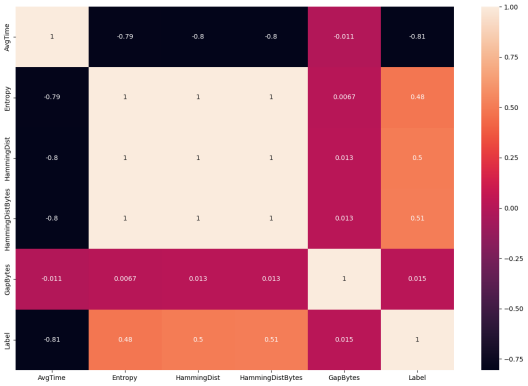


(b) Random payload modification attack

Figure 26: Correlation between extracted features obtained from the CrySyS Lab dataset



(a) Attacks while driving



(b) Attacks while stationary

Figure 27: Correlation between extracted features obtained from the IEEE Car Hacking: Attack & Defense Challenge 2020 dataset

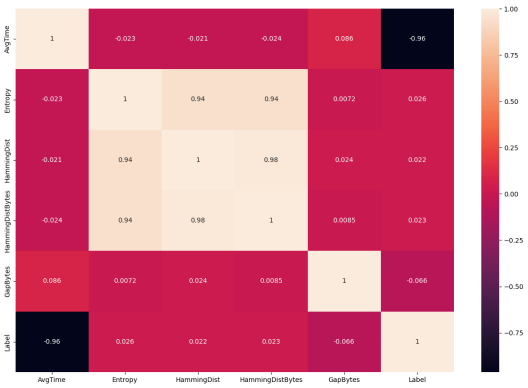
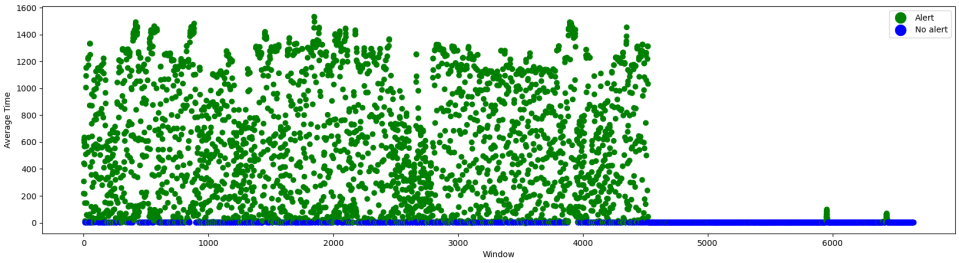


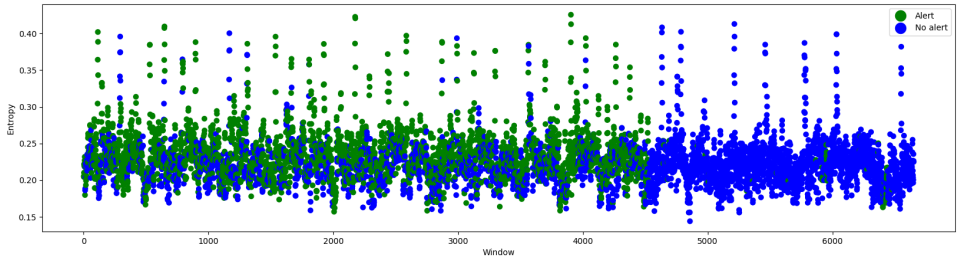
Figure 28: Correlation between extracted features obtained from the Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2

B

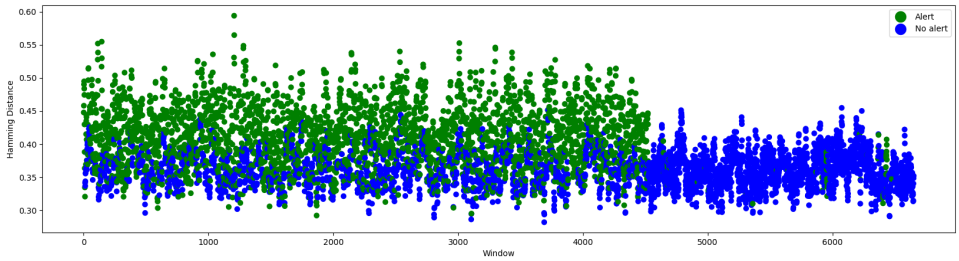
FEATURE EXTRACTION



(a) Average time between packets

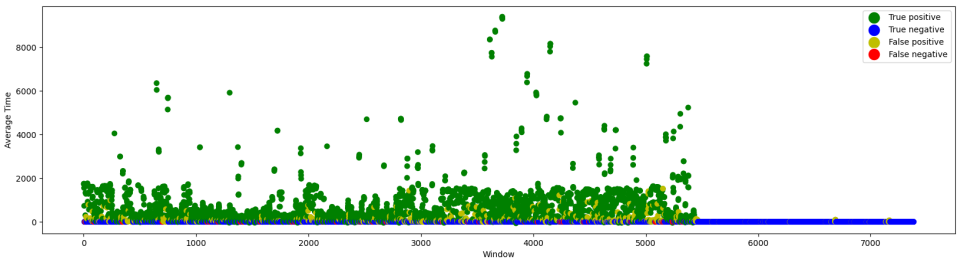


(b) Average entropy

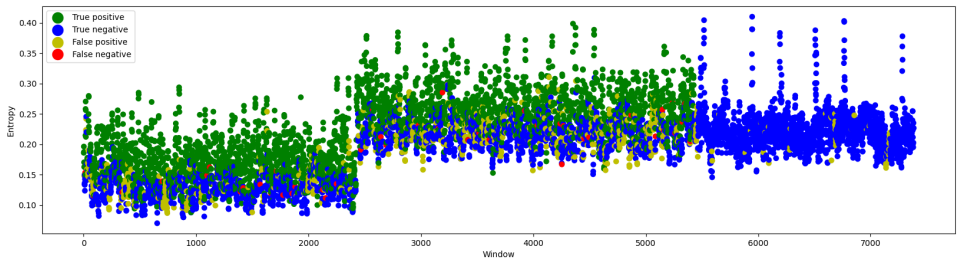


(c) Average Hamming distance

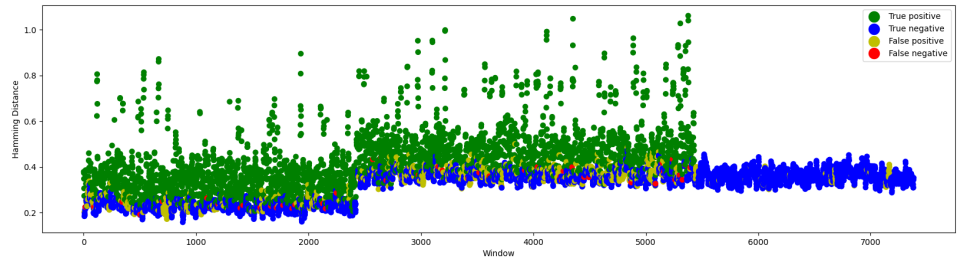
Figure 29: Car Hacking - DoS



(a) Average time between packets

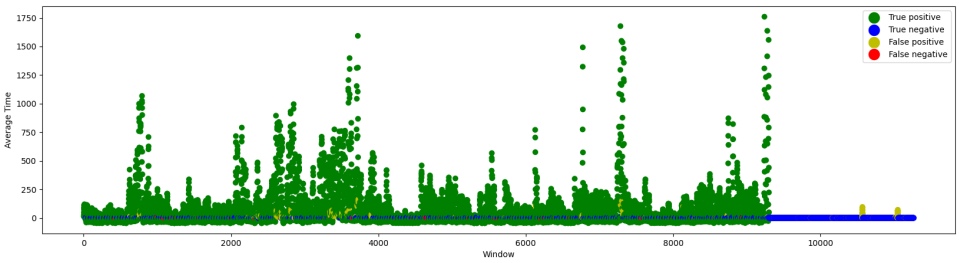


(b) Average entropy

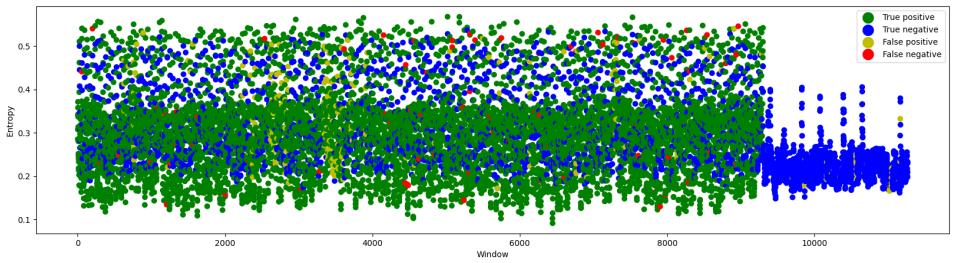


(c) Average Hamming distance

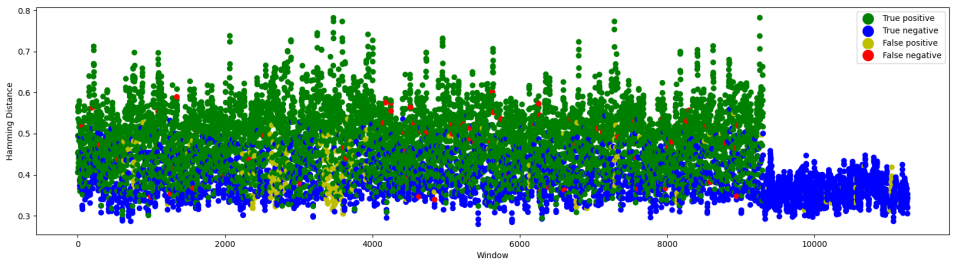
Figure 30: Car Hacking - Fuzzing



(a) Average time between packets

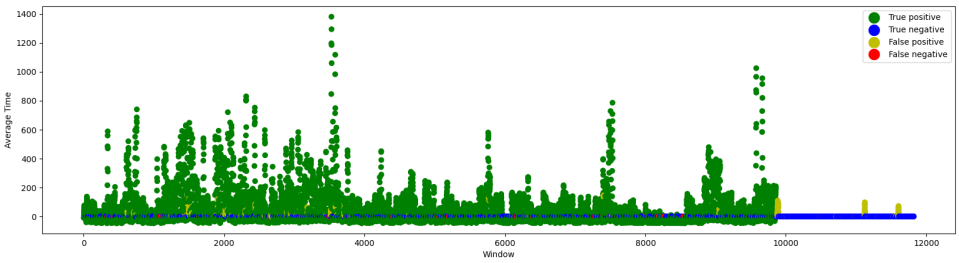


(b) Average entropy

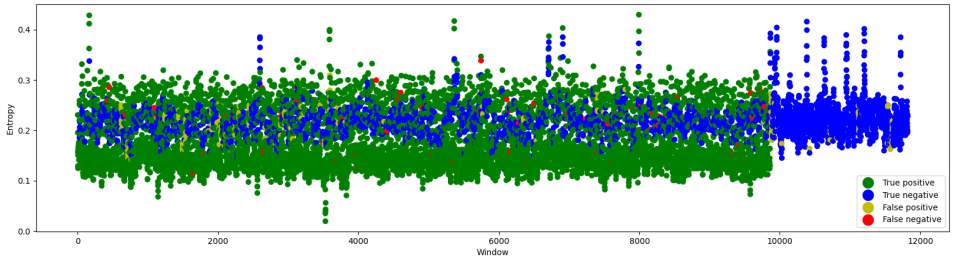


(c) Average Hamming distance

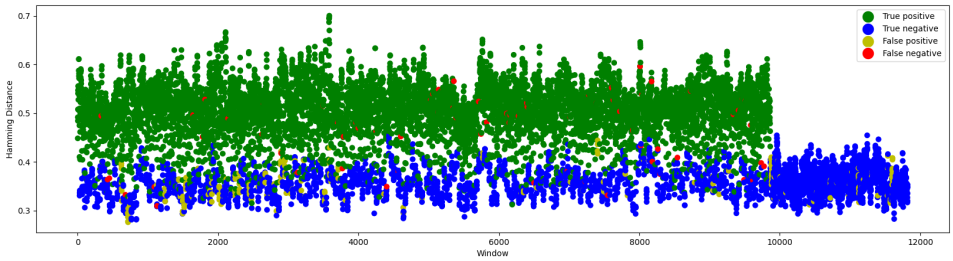
Figure 31: Car Hacking - Gear spoofing



(a) Average time between packets

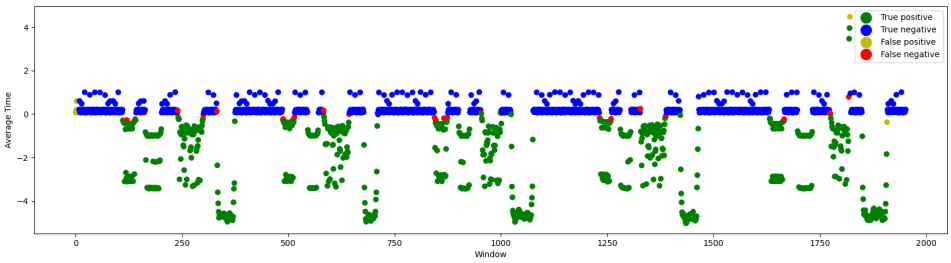


(b) Average entropy

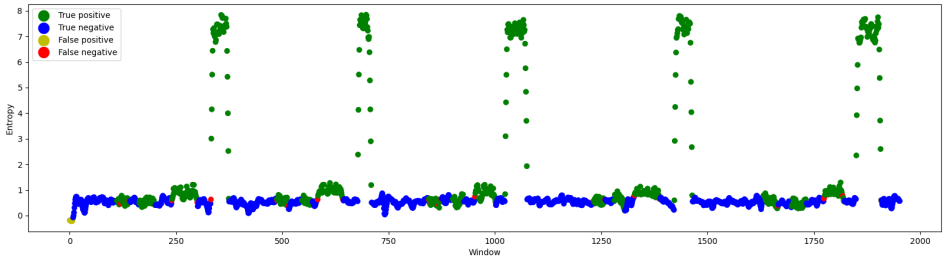


(c) Average Hamming distance

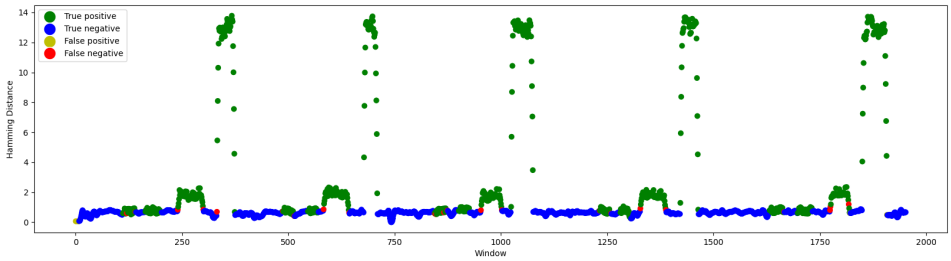
Figure 32: Car Hacking - RPM spoofing



(a) Average time between packets

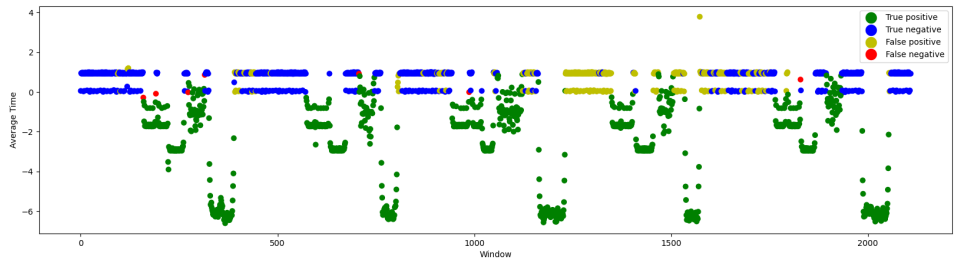


(b) Average entropy

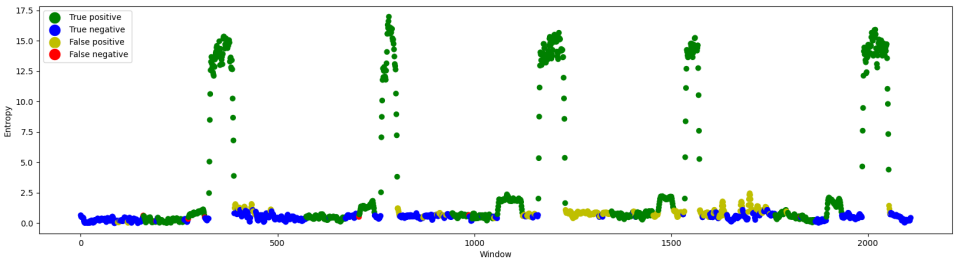


(c) Average Hamming distance

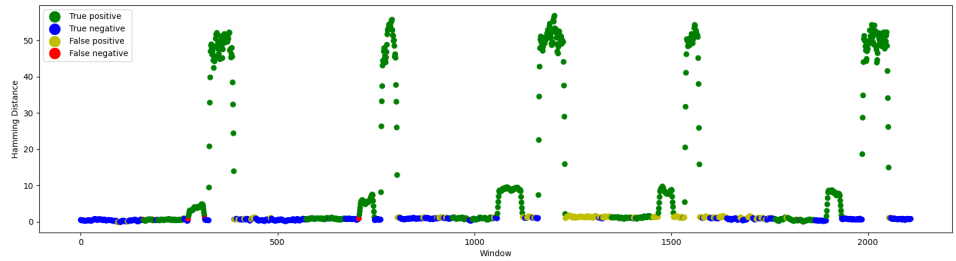
Figure 33: IEEE Car Hacking: Attack & Defense Challenge 2020 - Driving Submission



(a) Average time between packets

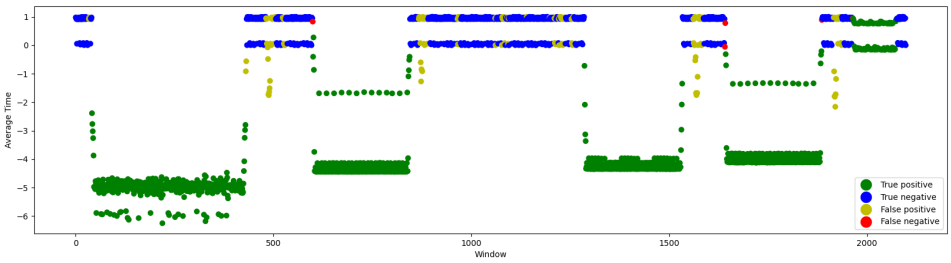


(b) Average entropy

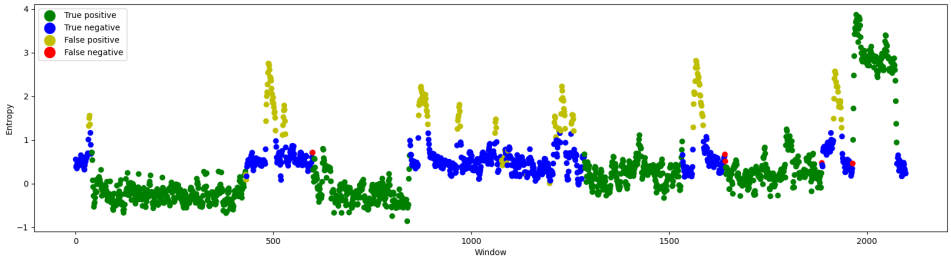


(c) Average Hamming distance

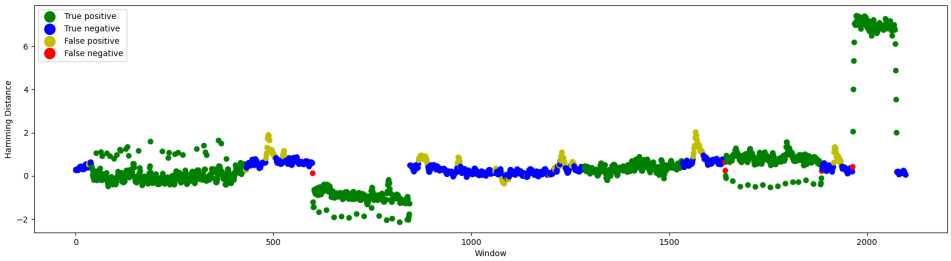
Figure 34: IEEE Car Hacking: Attack & Defense Challenge 2020 - Stationary Submission



(a) Average time between packets

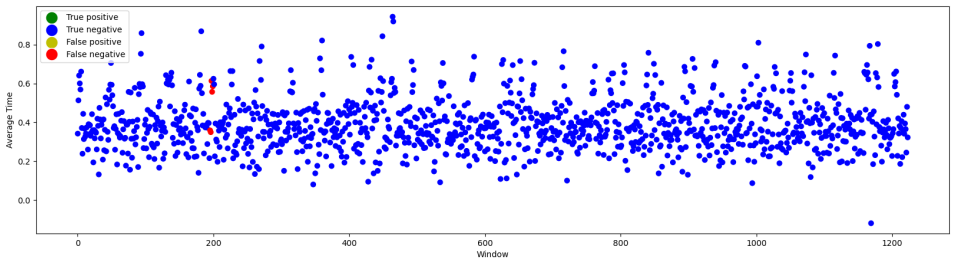


(b) Average entropy

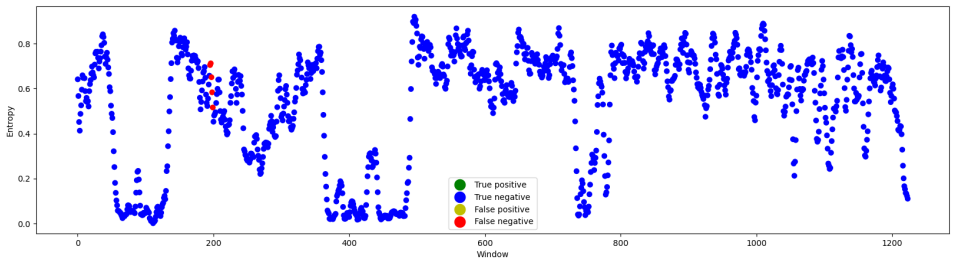


(c) Average Hamming distance

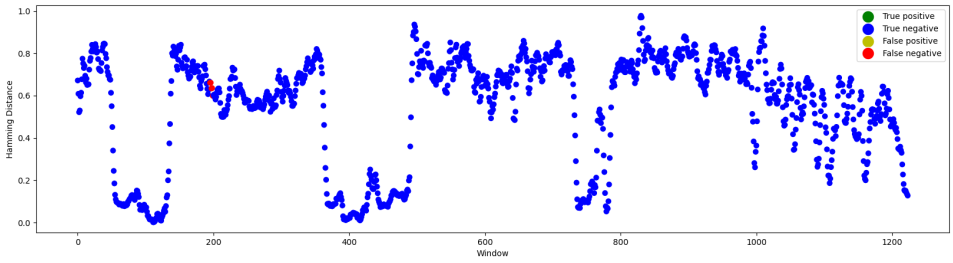
Figure 35: IEEE Car Hacking: Attack & Defense Challenge 2020 - Stationary Final



(a) Average time between packets

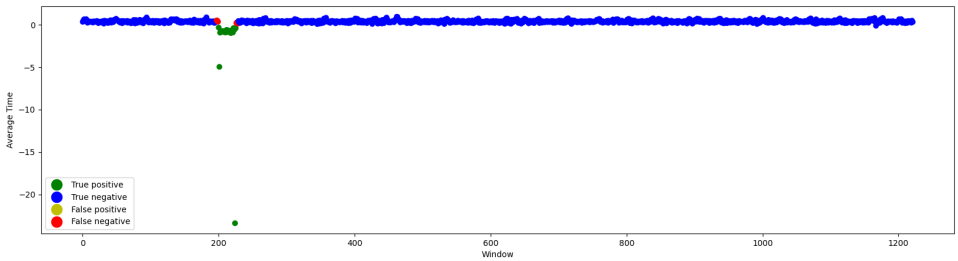


(b) Average entropy

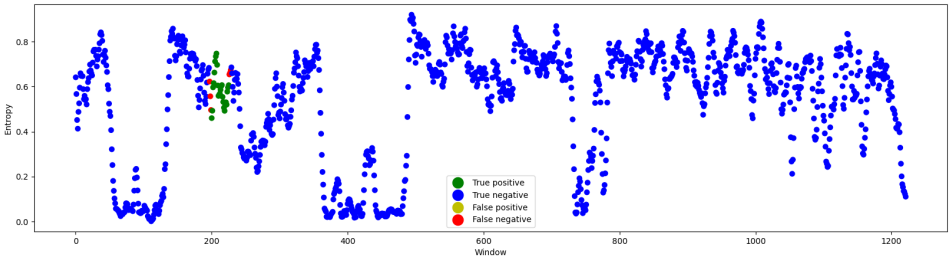


(c) Average Hamming distance

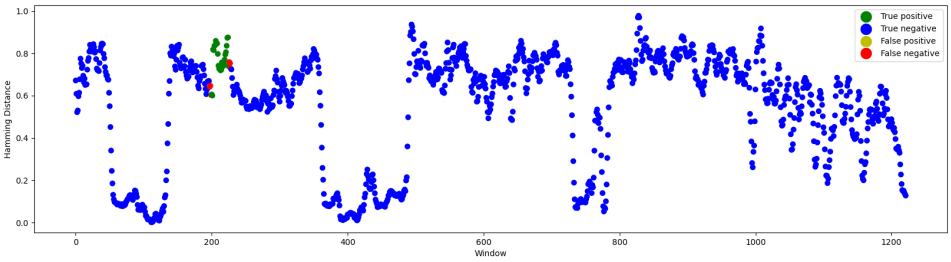
Figure 36: Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2 - Opel Astra - Fuzzing attack



(a) Average time between packets

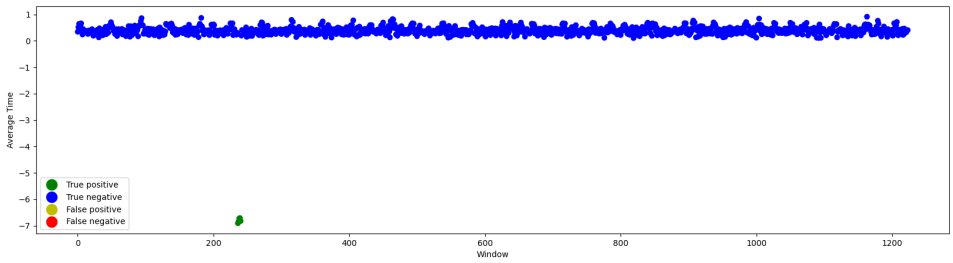


(b) Average entropy

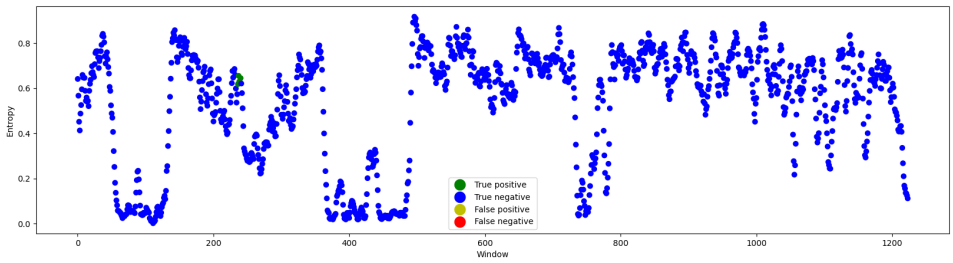


(c) Average Hamming distance

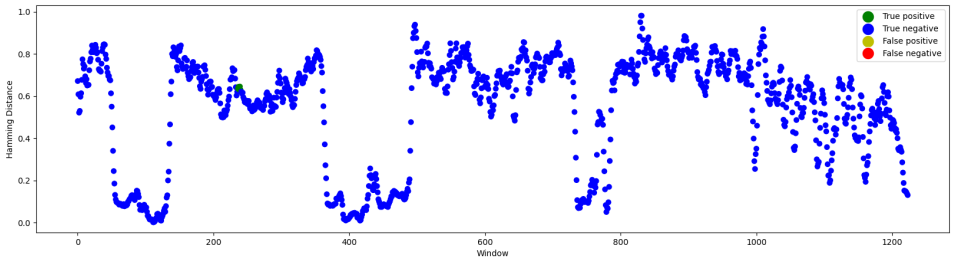
Figure 37: Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2 - Opel Astra - Message deletion attack



(a) Average time between packets

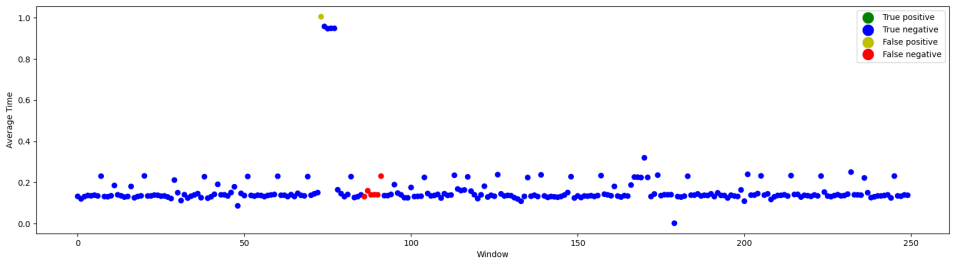


(b) Average entropy

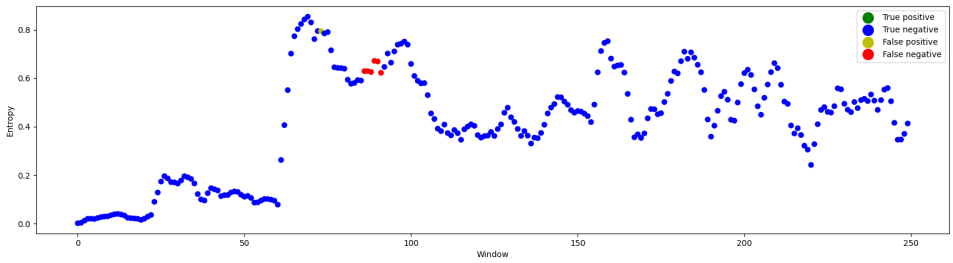


(c) Average Hamming distance

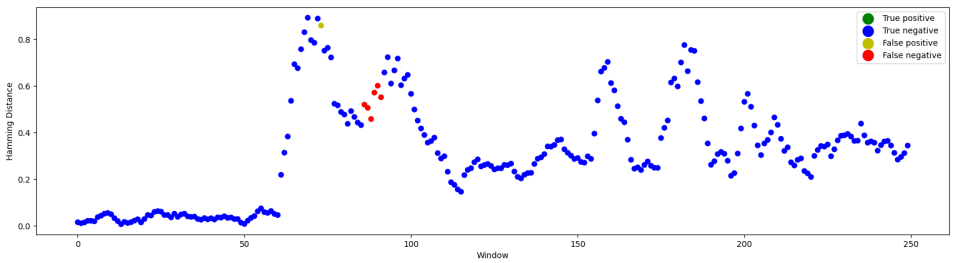
Figure 38: Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2 - Opel Astra - Replay attack



(a) Average time between packets

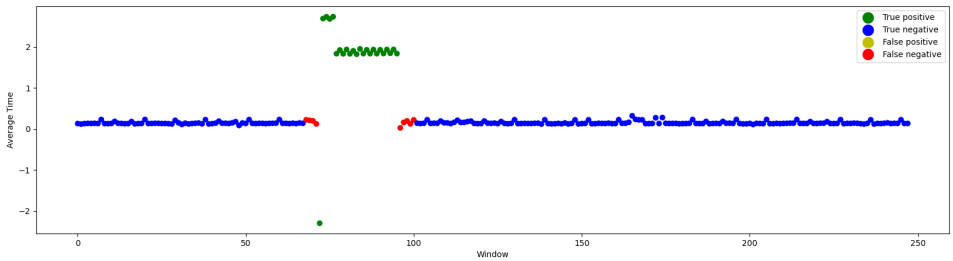


(b) Average entropy

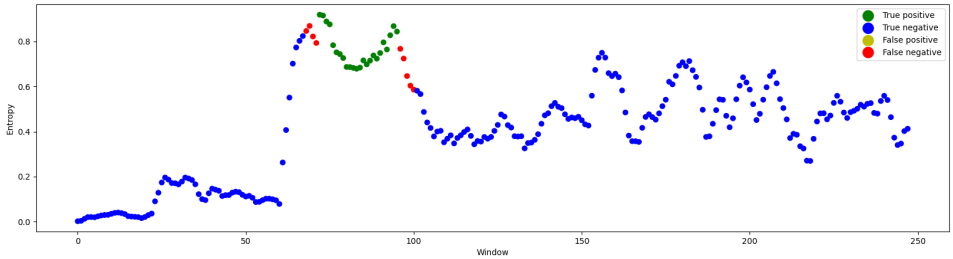


(c) Average Hamming distance

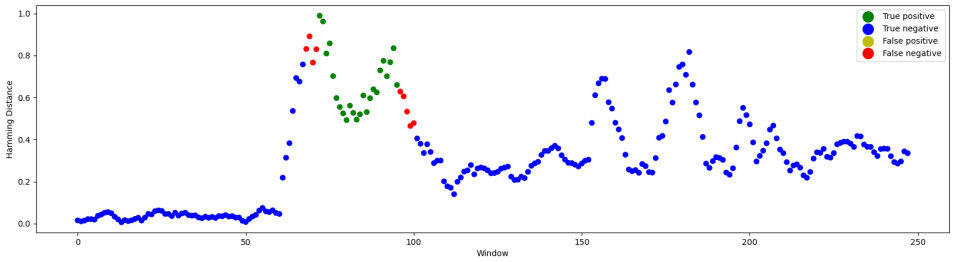
Figure 39: Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2 - Renault Clio - Fuzzing attack



(a) Average time between packets

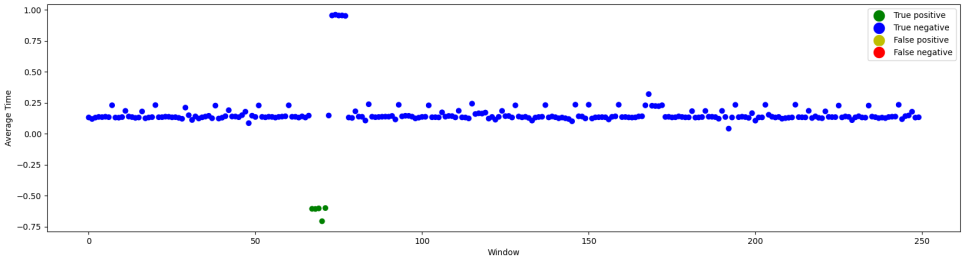


(b) Average entropy

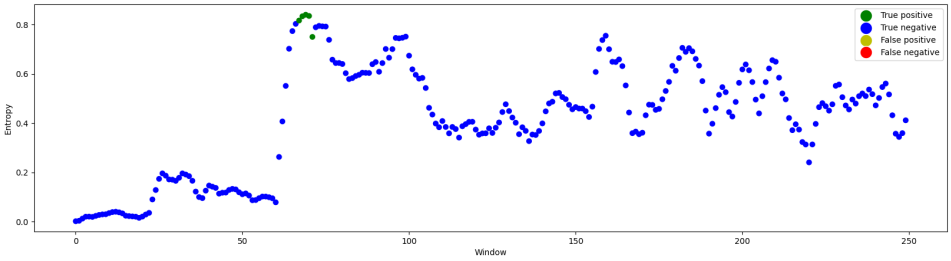


(c) Average Hamming distance

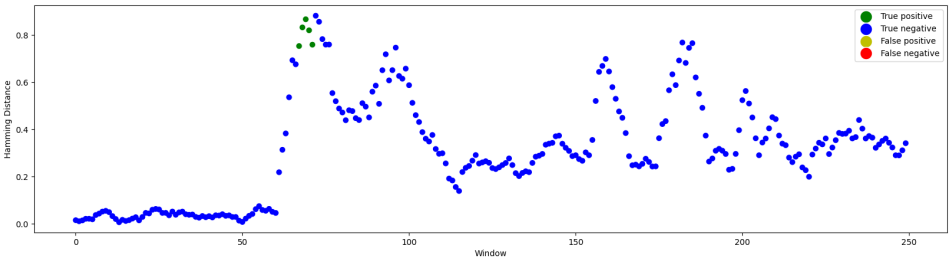
Figure 40: Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2 - Renault Clio - Message deletion attack



(a) Average time between packets

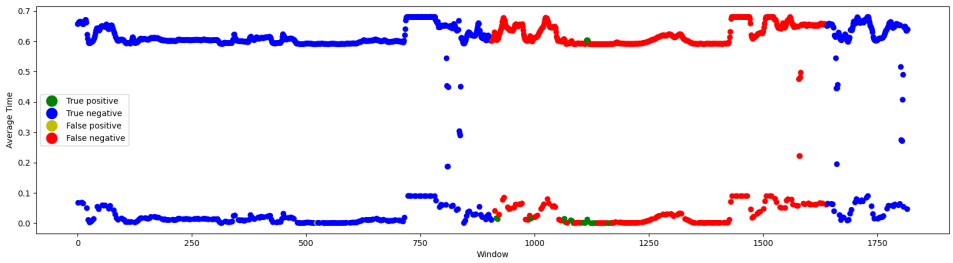


(b) Average entropy

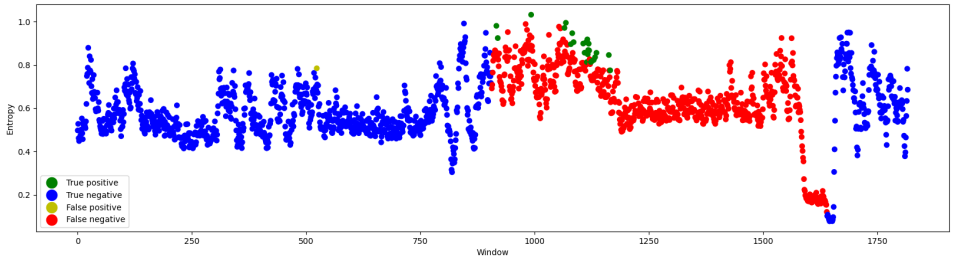


(c) Average Hamming distance

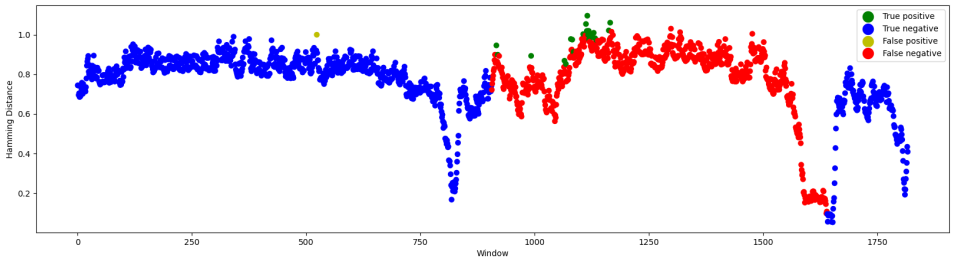
Figure 41: Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2 - Renault Clio - Replay attack



(a) Average time between packets

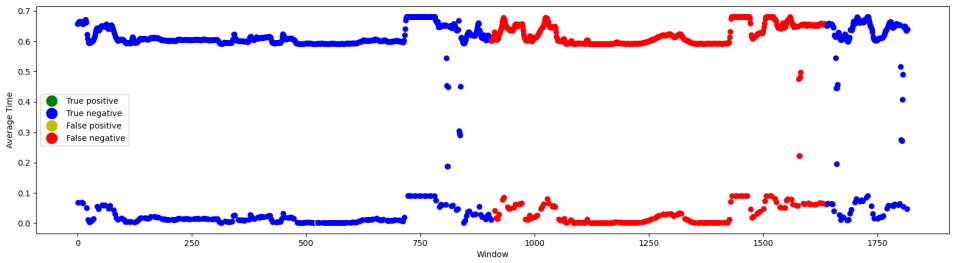


(b) Average entropy

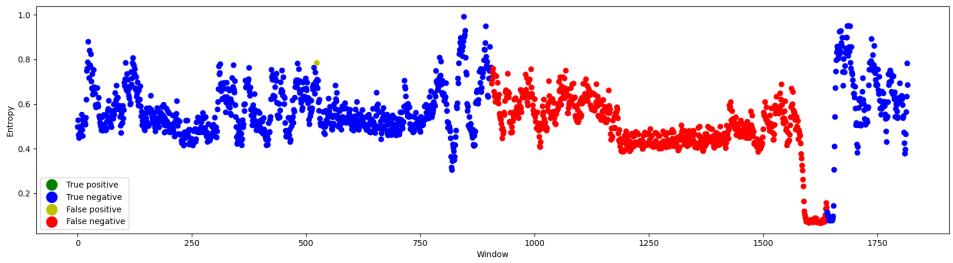


(c) Average Hamming distance

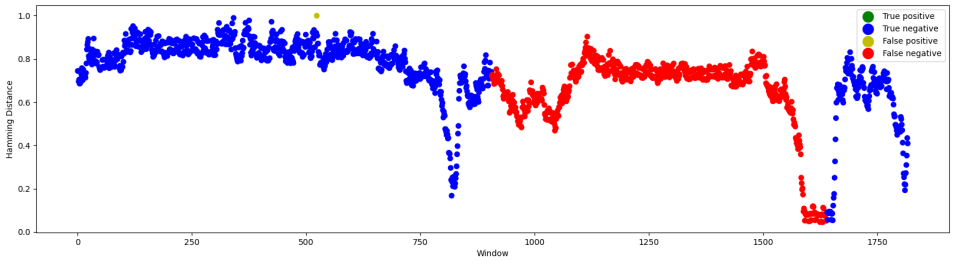
Figure 42: Modified CrySyS Lab - Adding an increasing value



(a) Average time between packets

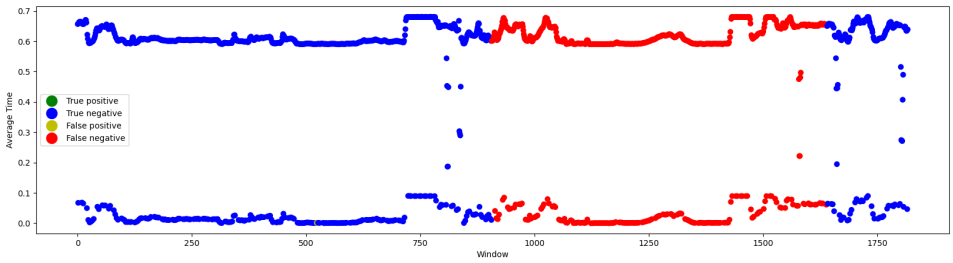


(b) Average entropy

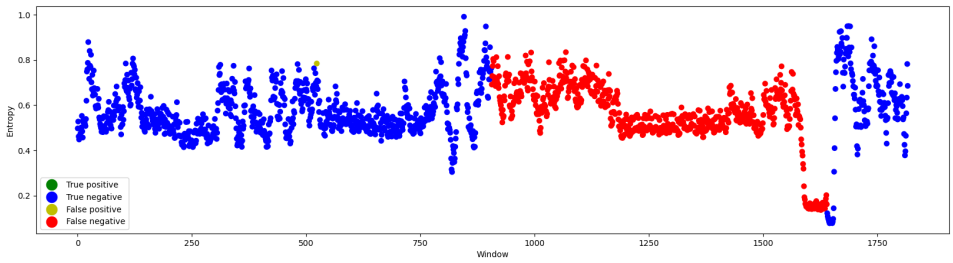


(c) Average Hamming distance

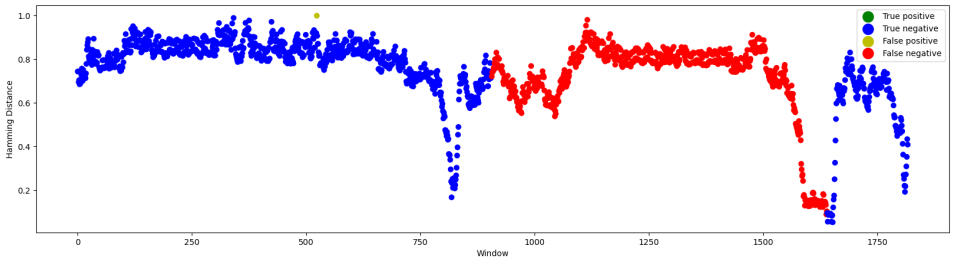
Figure 43: Modified CrySyS Lab - Constant value



(a) Average time between packets

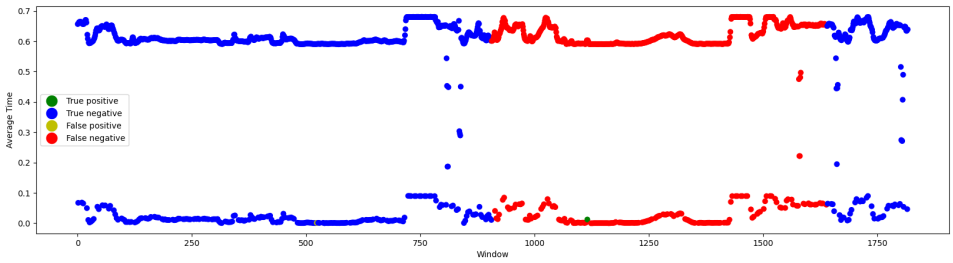


(b) Average entropy

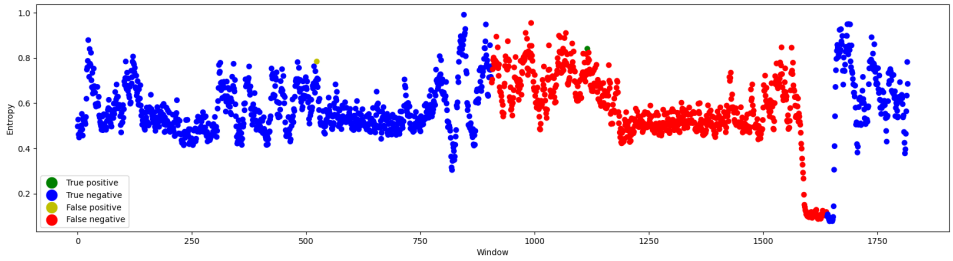


(c) Average Hamming distance

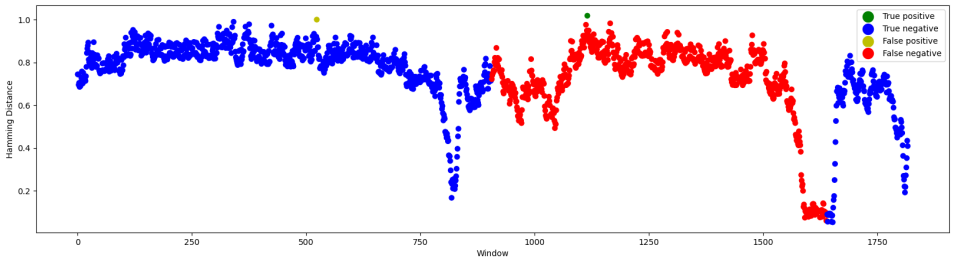
Figure 44: Modified CrySyS Lab - Decreasing value



(a) Average time between packets

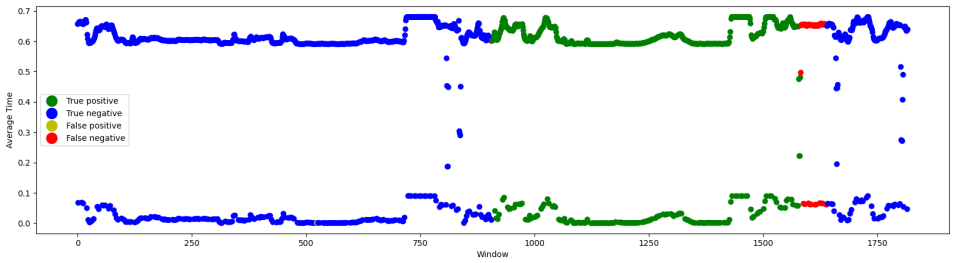


(b) Average entropy

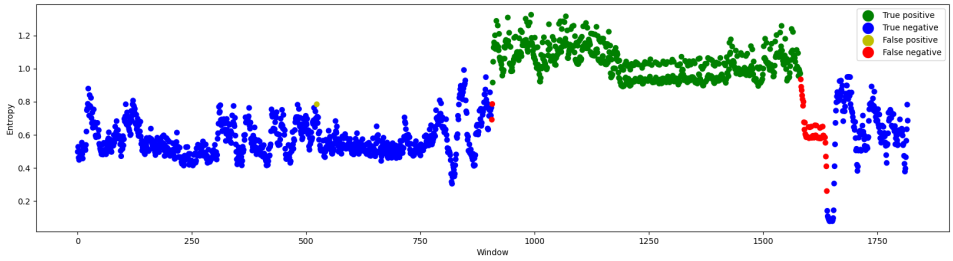


(c) Average Hamming distance

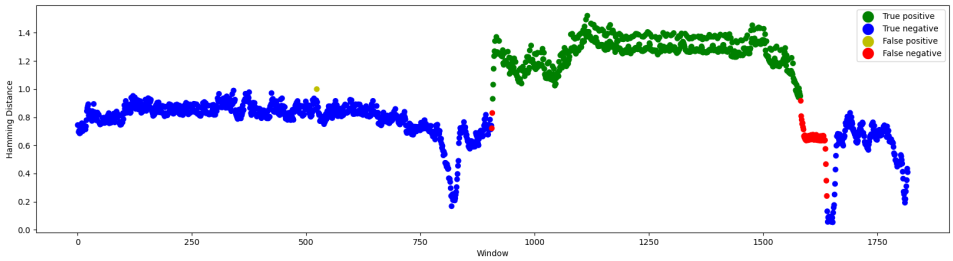
Figure 45: Modified CrySyS Lab - Delta of 1000



(a) Average time between packets



(b) Average entropy



(c) Average Hamming distance

Figure 46: Modified CrySyS Lab - Random value

C

CODE DOCUMENTATION

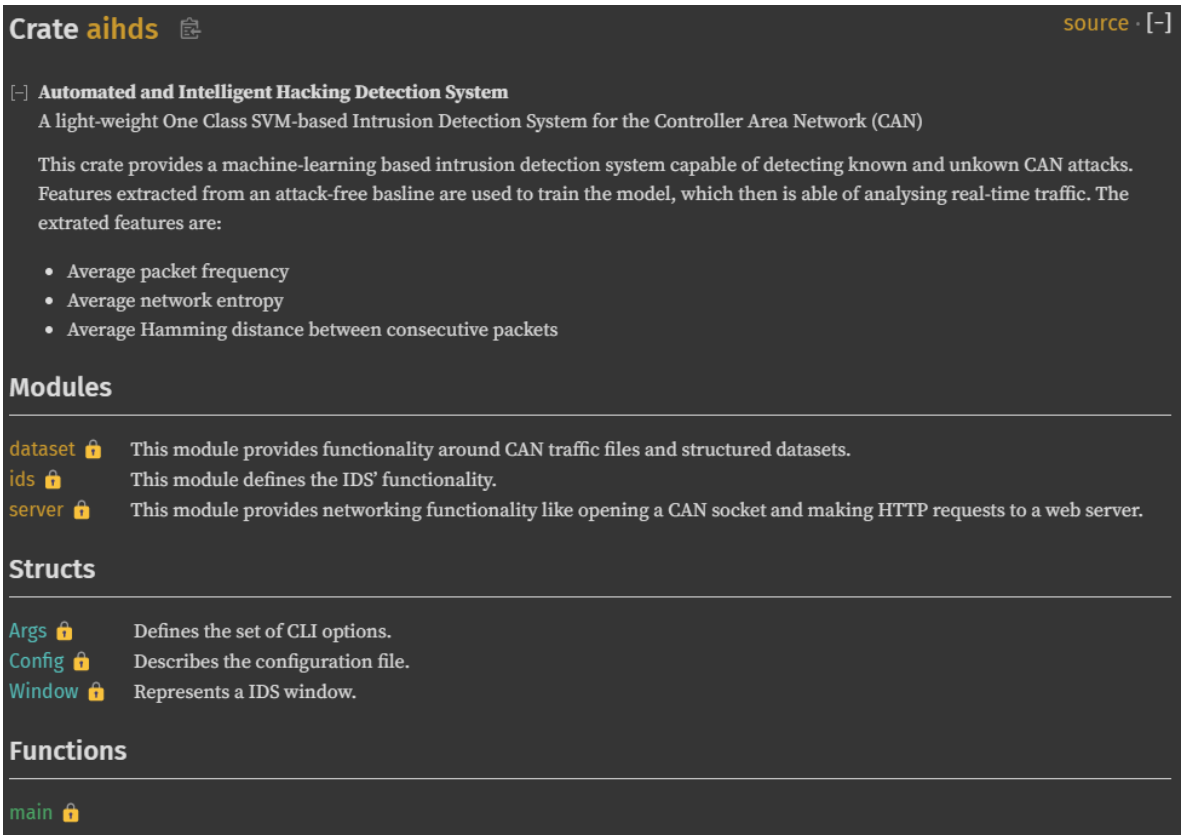
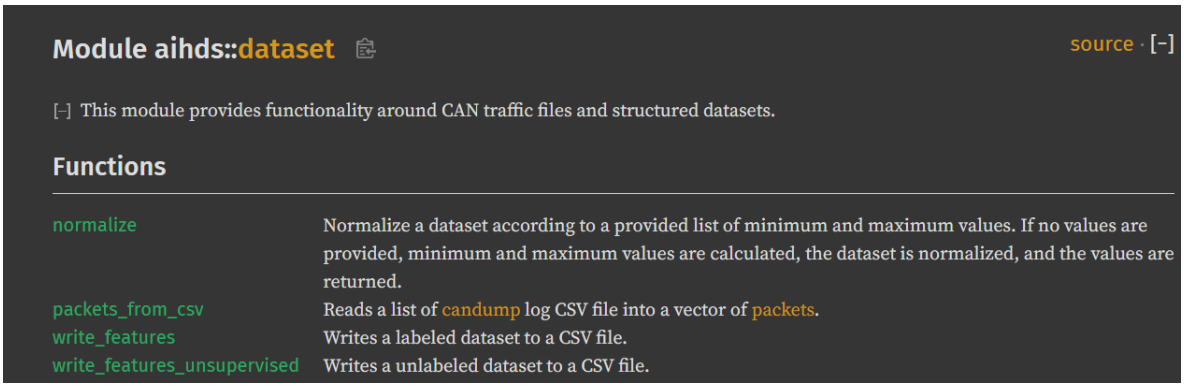


Figure 47: Crate AIHDS



Module aihds::dataset

source · [-]

[-] This module provides functionality around CAN traffic files and structured datasets.

Functions

normalize	Normalize a dataset according to a provided list of minimum and maximum values. If no values are provided, minimum and maximum values are calculated, the dataset is normalized, and the values are returned.
packets_from_csv	Reads a list of candump log CSV file into a vector of packets.
write_features	Writes a labeled dataset to a CSV file.
write_features_unsupervised	Writes a unlabeled dataset to a CSV file.

(a) Module

Function aihds::dataset::normalize

source · [-]

```
pub fn normalize(
    dataset: &mut Dataset<f64, (), Ix1>,
    params: &Option<Vec<(f64, f64)>>
) -> Option<Vec<(f64, f64)>>
```

[-] Normalize a dataset according to a provided list of minimum and maximum values. If no values are provided, minimum and maximum values are calculated, the dataset is normalized, and the values are returned.

Examples

```
use linfa::dataset::Dataset;
use ndarray::{Array1, Array2};

let ds0 = Dataset::new(Array2::from(vec![[1,2][3,4]]), Array1::from([0,1]));
let ds1 = ds.clone();

let minmax = normalize(&mut ds0, &None);
normalize(&mut ds1, &minmax);

assert_eq!(ds0, ds1);
```

(b) *normalize* function

Function aihds::dataset::packets_from_csv

source · [-]

```
pub fn packets_from_csv(paths: Vec<&Path>) -> Result<Vec<Packet>, Error>
```

[-] Reads a list of candump log CSV file into a vector of packets.

The log file must have the following format:

Timestamp,ID,DLC,Data,Label
1478195722.758421,0430,8,00,00,00,00,00,00,00,00,Normal
1478195722.766397,02b0,5,ff,7f,00,05,2f,Normal
1478195722.769240,0350,8,05,20,74,68,78,00,00,41,Normal
1478195722.775132,00df,8,8c,ab,f2,26,7a,29,1a,0c,Attack
1478195722.775957,06ea,8,25,10,9c,ed,5b,16,2c,18,Attack

Examples

```
use std::path::Path;

let packets = packets_from_csv(vec![Path::new("log.csv")])
```

(c) *packets_from_csv* function

Figure 48: *Dataset* module

Function aihds::dataset::write_features source · [−]

```
pub fn write_features(
    path: &Path,
    dataset: &Dataset<f64, u8, Ix1>
) -> Result<(), Error>
```

[−] Writes a labeled dataset to a CSV file.

Examples

```
use linfa::dataset::Dataset;
use ndarray::{Array1, Array2};
use std::path::Path;

let ds = Dataset::new(Array2::from(vec![[1,2][3,4]]), Array1::from([0,1]));
write_features(Path::new("features.csv"), ds);
```

(d) *write_features* function

Function aihds::dataset::write_features_unsupervised source · [−]

```
pub fn write_features_unsupervised(
    path: &Path,
    dataset: &Dataset<f64, (), Ix1>
) -> Result<(), Error>
```

[−] Writes a unlabeled dataset to a CSV file.

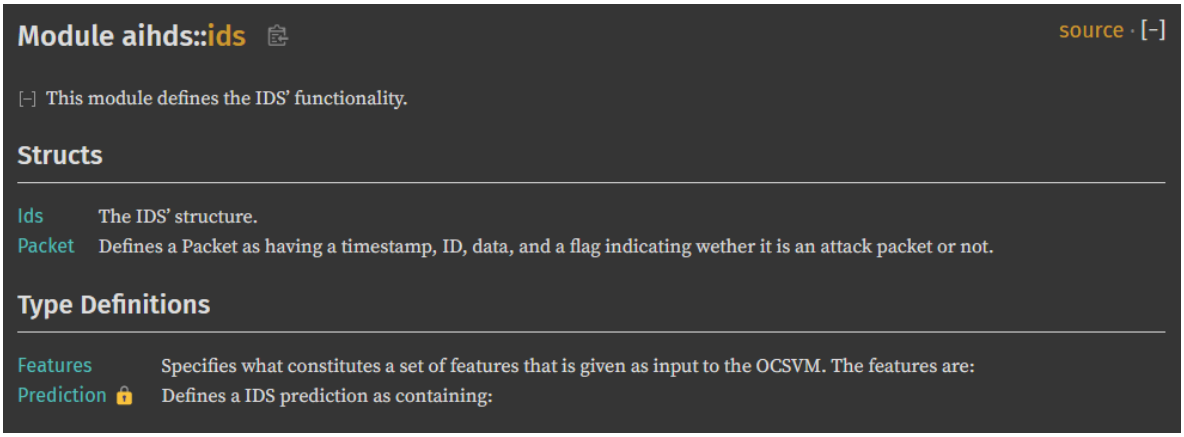
Examples

```
use linfa::dataset::Dataset;
use ndarray::{Array1, Array2};
use std::path::Path;

let ds = Dataset::new(Array2::from(vec![[1,2][3,4]]), Array1::from(vec![(), ()]));
write_features_unsupervised(Path::new("features.csv"), ds);
```

(e) *write_features_unsupervised* function

Figure 48: *Dataset* module (cont.)



Module aihds::ids

source · [-]

[-] This module defines the IDS' functionality.

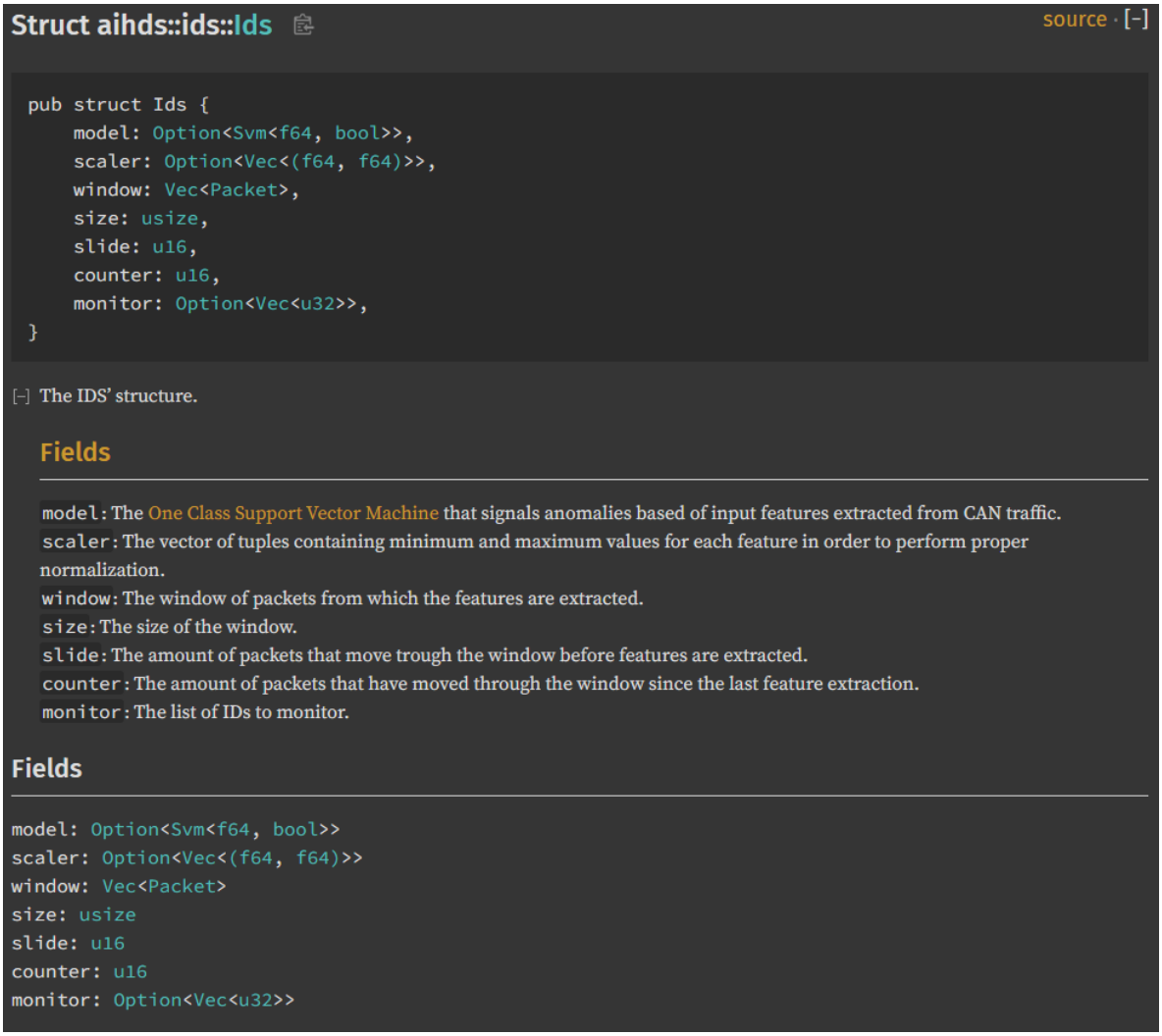
Structs

Ids	The IDS' structure.
Packet	Defines a Packet as having a timestamp, ID, data, and a flag indicating wether it is an attack packet or not.

Type Definitions

Features	Specifies what constitutes a set of features that is given as input to the OCSVM. The features are:
Prediction	Defines a IDS prediction as containing:

(a) Module



Struct aihds::ids::Ids

source · [-]

```
pub struct Ids {
    model: Option<Svm<f64, bool>>,
    scaler: Option<Vec<(f64, f64)>>,
    window: Vec<Packet>,
    size: usize,
    slide: u16,
    counter: u16,
    monitor: Option<Vec<u32>>,
}
```

[-] The IDS' structure.

Fields

`model`: The One Class Support Vector Machine that signals anomalies based of input features extracted from CAN traffic.
`scaler`: The vector of tuples containing minimum and maximum values for each feature in order to perform proper normalization.
`window`: The window of packets from which the features are extracted.
`size`: The size of the window.
`slide`: The amount of packets that move trough the window before features are extracted.
`counter`: The amount of packets that have moved through the window since the last feature extraction.
`monitor`: The list of IDs to monitor.

Fields

```
model: Option<Svm<f64, bool>>
scaler: Option<Vec<(f64, f64)>>
window: Vec<Packet>
size: usize
slide: u16
counter: u16
monitor: Option<Vec<u32>>
```

(b) Struct

Figure 49: *IDS* module

```
pub fn new(                                                    source
    model: Option<Svm<f64, bool>>,
    scaler: Option<Vec<(f64, f64)>>,
    window_size: usize,
    window_slide: u16,
    monitor: Option<Vec<u32>>
) -> Ids
```

Create a new instance of the IDS.

(c) IDS' *new* function

```
pub fn load(path: &Path) -> Ids                                source
```

Load an existing IDS from the file system.

Examples

```
use std::path::Path;

let ids = load(&Path::new("example.ids"))
```

(d) *load* function

```
pub fn get_monitor(&self) -> &Option<Vec<u32>>                 source
```

Returns the list of IDs that the IDS is monitoring.

Examples

```
if let Some(m) = ids.get_monitor() {
    println!("The IDS is monitoring the following IDs: {:?}.", m);
} else {
    println!("The IDS is monitoring all packets.");
}
```

(e) *get_monitor* function

Figure 49: *IDS* module (cont.)

```
pub fn train(                                                    source
    &mut self,
    socket: Option<&CANSocket>,
    files: Option<Vec<&Path>>,
    baseline_len: usize
)
```

Train a One Class Support Vector Machine for the IDS.

Training can be done in both an online and offline fashion. Online training is done by supplying a socket from which attack-free traffic is to be collected until the limit specified in `baseline_len`. Offline training is done by supplying a list of files containing attack-free CAN traffic obtained from the candump tool. All packets will be read from the files.
The trained model is saved in `models/ids`.

Examples

Online training

```
use crate::server;
use std::path::Path;

let socket = server::open_socket("can0", &monitor);
let mut ids = new(None, None, 1000, 250, monitor);
ids.train(Some(&socket), None, 1_000_000);
```

Offline training

```
let mut ids = new(None, None, 1000, 250, monitor);
ids.train(None, Some(Path::new("can_traffic.csv")), 1_000_000);
```

(f) *train* function

```
pub fn test(                                                     source
    &mut self,
    packets: Vec<Packet>
) -> (Vec<bool>, Vec<([f64; 3], bool, (i64, i64))>, f32)
```

Executes a performance test on the IDS.
Extracted features and model outputs are placed in `features/test.csv`.
Returns:

- the real labels
- the model's predictions
- number of packets per second that were processed

Examples

```
let result = ids.test(packets);
println!("Real values: {:?}", result.0);
println!("Predictions: {:?}", result.1);
println!("Packets per second: {}", result.2);
```

(g) *test* function

Figure 49: *IDS* module (cont.)

```
pub fn feature_set(                                                    source
    &mut self,
    packets: Vec<Packet>
) -> DatasetBase<ArrayBase<OwnedRepr<f64>, Dim<[usize; 2]>>, ArrayBase<OwnedRepr<bool>,
Dim<[usize; 1]>>>
```

Returns a dataset containing the extracted features from the list of packets.

Examples

```
use crate::dataset;

let ids = load(&Path::new("example.ids"));
let packets = dataset::read_from_csv(&Path::new("can_traffic.csv"));
let ds = ids.feature_set(packets);
```

(h) *feature_set* function

```
pub fn push(&mut self, packet: Packet) -> Option<([f64; 3], bool, (i64, i64))>   source
```

Pushes a packet into the IDS' window.
If the number of packets inserted corresponds to the configured window slide, a prediction is returned.

Examples

```
for packet in packets {
    if let Some(result) = ids.push(packet) {
        if result.1 {
            println!("An attack was detected inside window {} based on {}.",
                result.2,
                result.0
            );
        } else {
            println!("No attack was detected inside window {} based on {}.",
                result.2,
                result.0
            );
        }
    }
}
```

(i) *push* function

```
fn predict(&self) -> Option<([f64; 3], bool)>                          source
```

Performs a prediction based on the window's contents.

Panics

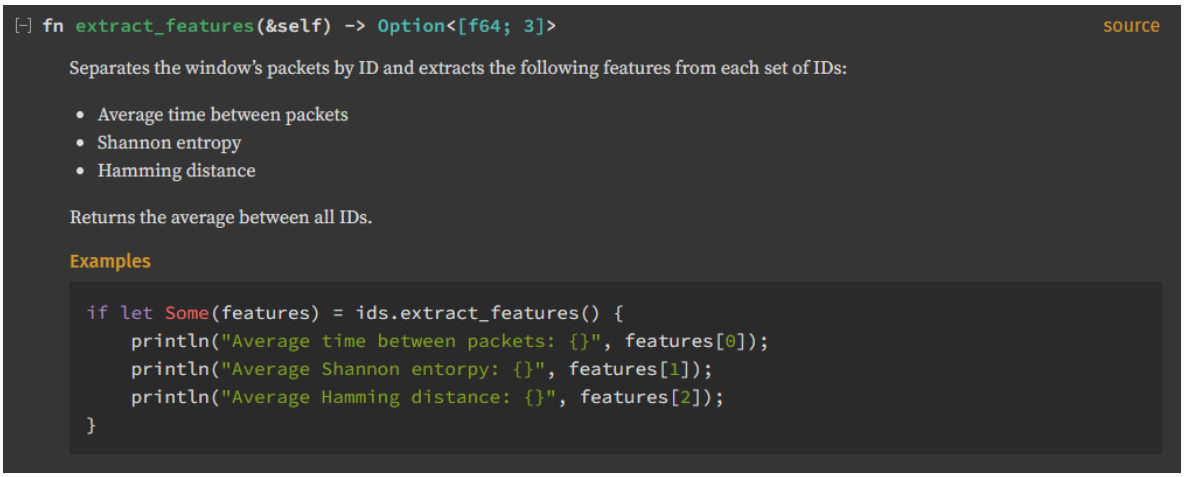
If there is no model or scaler present in the IDS.

Examples

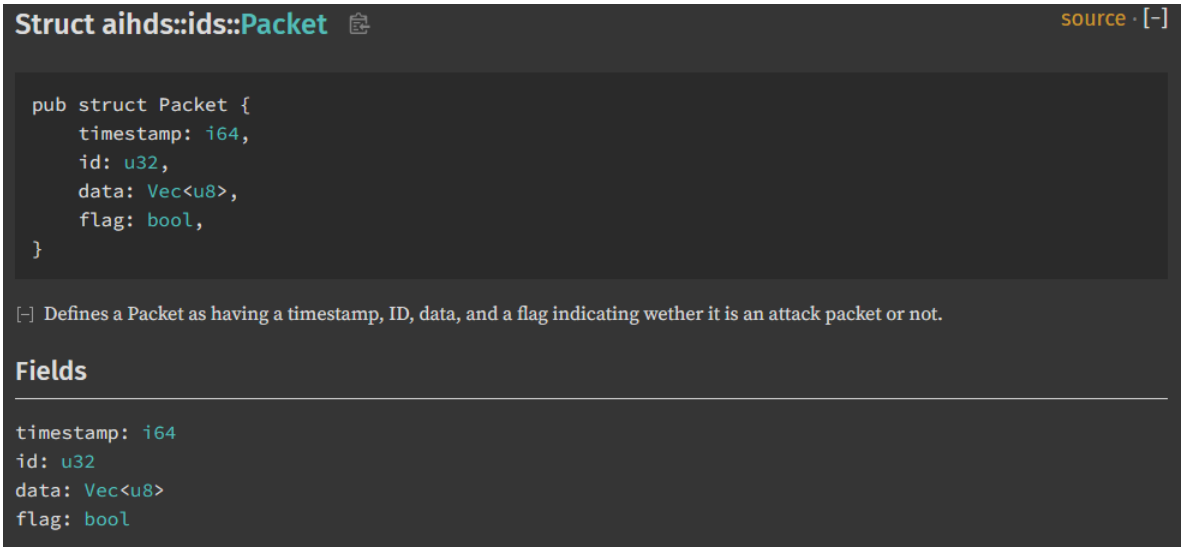
```
if let Some(pred) = ids.predict() {
    if pred.1 {
        println!("Alert based on {}", pred.0);
    } else {
        println!("No alert based on {}", pred.0);
    }
}
```

(j) *predict* function

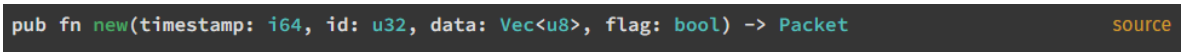
Figure 49: *IDS* module (cont.)



(k) *extract_features* function

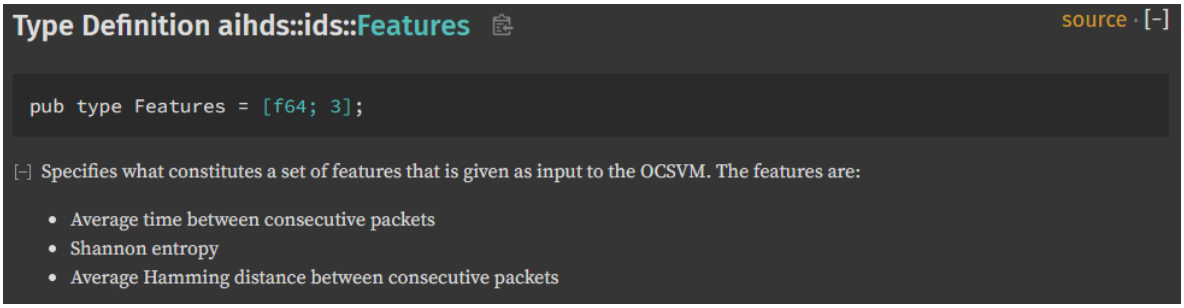


(l) *Packet* struct



(m) Packet's *new* function

Figure 49: *IDS* module (cont.)



(n) *Features* type

(o) *Prediction* type

Figure 49: *IDS* module (cont.)

Module aihds::server

source · [-]

[-] This module provides networking functionality like opening a CAN socket and making HTTP requests to a web server.

(a) Module

Function aihds::server::open_socket

source · [-]

```
pub fn open_socket(interface: &str, filter: &Option<Vec<u32>>) -> CANSocket
```

[-] Open a CAN socket with an optional filter.

Examples

```
let socket = open_socket("can0", &Some(vec![123,456]))
loop {
    match socket.read_frame() {
        Ok(frame) => // Process frame
        Err(why) => // Manage error
    }
}
```

(b) *open_socket* function

Function aihds::server::post

source · [-]

```
pub async fn post(
    client: &Client,
    url: &str,
    features: Vec<f32>,
    result: &bool,
    message: Option<String>
) -> Result<Response, Error>
```

[-] Make a POST request to a HTTP server.

This function makes an asyncronous HTTP POST request to a specified URL using the reqwest library. It allows the communication of an alarm and the extracted features, as well as an optional message.

Examples

```
let client = reqwest::Client::new();
post(
    &client,
    "localhost:1337",
    vec![0.0, 1.0, 0.0],
    &true,
    Some("Intrusion detected!")
)
```

(c) *post* function

Figure 50: *Server* module