Department of Computer Science

A Framework for Detection of Anomalies in Sensor Data for Prevention of Cyberattacks in Connected Autonomous Vehicles

Keane Fernandes

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science by advanced study in Computer Science in the Faculty of Engineering.

Department of Computer Science
September 27, 2021
Word count: 12345

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

SIGNED: .....

DATE: 27/09/21 .....

Executive Summary

Advances in sensor and communication technology have had a significant impact on our daily lives, and transportation is no exception. In today's major cities, where conventional transport systems have become overburdened and inefficient, there is a strong motivation for the development and implementation of intelligent transport systems, with increased levels of interoperability, connectivity and networking and automation. Whilst the increased connectivity can be a good thing, such networks of devices are susceptible to an increased risk of a cyber attack. Furthermore, increased automation would decrease the likelihood that an adversary is detected during execution of the attack.

The main aim of this project is to assess the suitability of a Pattern of Life inspired network monitoring system in the early detection of a denial of service (DoS) attack performed on in-vehicle networks. The hypothesis to be tested is that early indicators of an attack can be detected by analysing patterns in the data sent over the network with the help of supervised learning algorithms. A hardware demonstrator of automotive grade was provided by Thales Group to accomplish this. It provides an abstraction to the shared network and components one would expect to find in a real automobile. Pattern of life has been used in other industries, mainly military and maritime, but no such PoL implementation exists in an automotive cybersecurity. This project aims to contribute to this gap in the industry. There are two main aspects to this project, a software development component and a research component, with the aims being as follows:

1. The first, software development component, will aim to build a data collection and feature extraction tool to extract information and features from the hardware demonstrator
2. The second, research component, will aim to assess the effectiveness of a PoL inspired network monitoring system in an automotive cyber security context for the detection of a DoS attack and evaluate the performance of supervised machine learning algorithms using suitable statistical metrics.

The contributions of the project are:

1. Performed a literature review which summarises the key findings of Pattern of Life studies, on which literature is not very readily available.
2. Proposed a Pattern of Life processing framework in the context of automotive cyber security, with the aim of detection of DoS attacks.
3. Development of data collection and feature extraction tools that can be used by future users of the hardware demonstrator for their research pursuits.
4. The dataset generated during data collection of baseline operation of the board and for all DoS attacks can be used by the online community to apply their own ML techniques and is available online on GitHub.
5. The hypothesis that a DoS attack can be detected by using a carefully chosen feature vector of protocol throughputs was proven to be true by training a supervised binary classifier capable of predicting this.

Acknowledgements

First and foremost, I would like to thank **Dr. Kerstin Eder**, without whose vision and supervision this project would not have been possible. It was through her continued advice and feedback that I was able to bring this dissertation to its current standard. I greatly appreciate the time she took out to personally deliver the hardware demonstrator not only once, but twice.

I would also like to express my gratitude to **Mr. Peter Davis** from Thales Group for providing the hardware demonstrator, with which I was able to conduct the project.

Furthermore, I would like to thank **Mr Fergus Duncan** for talking me through the inner functioning of the automotive rig.

Last but not least, I would like to thank the University of Bristol for giving me the opportunity to work and connect with such brilliant individuals, with whose expertise and guidance, I was able to complete this thesis.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Advances in sensor and communication technology have had a significant impact on our daily lives, and transportation is no exception. In today's major cities, where conventional transport systems have become overburdened and inefficient, there is a strong motivation for the development and implementation of intelligent transport systems [45]. Connected and Autonomous Vehicles (CAVs) will play an important role in future transportation systems, unlocking invaluable social advantages such as reduced accidents, decreased energy consumption, lower congestion and improved air quality whilst enhancing transport accessibility [42]. To help enable the transition towards the mass adoption of CAVs, a V2X communications framework has been developed. V2X can be described as a communication framework between a vehicle and any other entity that may affect the vehicle, with 'X' denoting the entity. For example, intercommunication between CAVs and their manoeuvring intentions can be facilitated through the exchange of sensor data in V2V (Vehicle-to-Vehicle) and V2I (Vehicle-to-Infrastructure) communication frameworks. Figure 1 provides a visual description of how a V2X enabled ecosystem might operate.

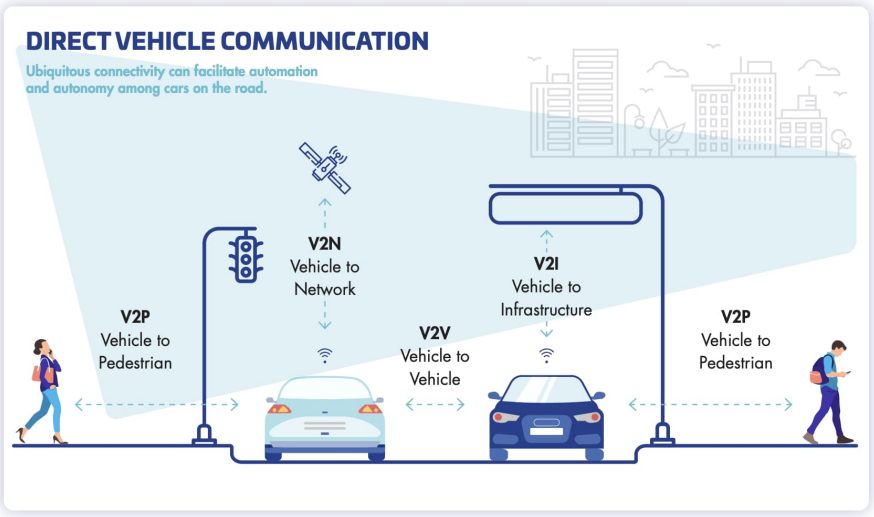


Figure 1: An infographic highlighting how the V2X communication framework is expected to function [10].

Increased levels of interoperability, connectivity and networking and automation would be fairly standard features of the future automobile in a V2X infrastructure. Whilst the increased connectivity can be a good thing, such networks of devices are susceptible to an increased risk of a cyber attack. Furthermore, increased automation would decrease the likelihood that an adversary (a name given to an entity performing a cyber attack) is detected during execution of the attack [37].

A major prerequisite to the successful implementation of a V2X communication platform would be to first ensure that the communication between the internal subsystems of an autonomous vehicle is

resilient to cyberattacks [21]. A previous masters project explored the current state of cybersecurity in the automotive industry and provides a thorough review of the vehicle attack surfaces and common attacks that can be used to disrupt normal functioning of internal vehicle communications. One of the key findings was that not enough cyber security based design considerations were implemented in typical automotive system development lifecycles [43]. The project aims to address this gap in the automotive industry by proposing an alternative solution to in-vehicle cyber security, one that is based on a Pattern of Life (PoL) approach. A PoL can be defined as the "repeatable behaviours of an entity which recur at normally distributed time intervals under a particular set of conditions" and its applications are mainly in the defence and military sectors. An implementation of this kind is unheard of in the automotive industry, and this forms the basis of the research in this project.

1.2 Aims and Objectives

The research question is to assess the suitability of a PoL inspired network monitoring system in the early detection of a denial of service (DoS) attack performed on in-vehicle networks. To help answer this question, a technique known as a knowledge discovery process (KDP) is employed, where a KDP is defined as the "nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data in a new application domain." [16] Building on the idea of a KDP, the hypothesis to be validated is that early indicators of an attack can be detected by analysing patterns in the data sent over the network with the help of supervised learning algorithms. A hardware demonstrator of automotive grade components including a CAN bus, vehicle speed set-point, cruise control and wheel movement through a motor has been provided to accomplish this.

The project is not purely research focused, rather it is a hybrid project comprising of a software development component and a research component. To further elaborate, the software development part will be carried out to enable automated extraction of information from the hardware demonstrator. This information will then be used to identify patterns in data using a supervised learning algorithm when the hardware demonstrator is attacked, which can then be used as the basis for validation of the above mentioned hypothesis and evaluation of the research question. The standalone steps in the KDP are not novel or revolutionary, however, application in a previously unseen domain (automotive cyber security domain in this case) can help boost the probability of discovery of patterns in the recorded data. Keeping in mind the two distinct phases of this project, the software development objectives are:

1. Build a data collection tool that automates the process of obtaining sensor data from the hardware demonstrator, and stores it in a suitable format for further processing.
2. Develop a feature extraction tool that works in tandem with the data collection tool to generate a set of features that can be used as the basis of a machine learning endeavour.
3. Generate a feature rich dataset for baseline operation of the demonstrator and for during DoS attack for provision on a remote git repository. This is so that members of the community that do not have access to the hardware can implement their machine learning techniques on the dataset.

And the research aims are:

1. Assess the effectiveness of a PoL inspired network monitoring system in an automotive cyber security context for the detection of a DoS attack.

2. Evaluate the performance of supervised machine learning algorithms for identification of patterns in network monitoring data for predictions on when a DoS attack has happened in the network.

The documentation is laid out as follows – Firstly, the background and context in which the work of this project has been carried out is described, more specifically the state of automotive networking, cyber security trends in the automotive domain and the need for the integration of cybersecurity design principles in the development lifecycle. Attack surfaces, types and motivations are also presented. Section 3 describes a potential alternative approach to security in automotive networks known as Activity-Based Intelligence (ABI) and a subset of ABI known as Pattern of Life (PoL). This is done because PoL framework elements will inspire the research methodology of this project. Keeping this in mind, existing PoL related literature is reviewed and key takeaways are summarised and used as a point of reference when drawing up the research methodology. The methodologies for both software development and research components of this project are then detailed in Section 4. The design and implementation of the two methodologies are then documented in Sections 5 and 6. The results of the DoS attacks performed on the network and the performance of the PoL-based network monitoring system is presented in Section 7. The discussion section critically evaluates the performance of the PoL processing framework, and where improvements could have been made. Furthermore, the extent to which this project meets the above mentioned objectives are discussed. The final section concludes the project by proposing further work that can be done in the future and summarising the key contributions of the project.

2 Background and Context

This section aims to give the reader an overview of the current state of the art technologies in automotive cyber security, including current in-vehicle network architectures, potential cyber threats and their associated consequences. Furthermore, the future direction of the automotive sector from a cyber security perspective will be touched upon. The information presented is by no means exhaustive, and the cited literature can be explored further if the reader wishes to gain further clarification on a specific topic. The information provided in this section is the bare minimum the reader shall needed to understand where the research aims of this project stem from.

2.1 Automotive Networks

Automobiles have evolved from purely mechanical devices into a complex network of electronic control units (ECUs) that are responsible for the implementation of all features of the modern automobile, ranging from the infotainment unit to automated parking systems. In the simplest of terms, a vehicle can be thought of as a system comprising of various subsystems that work together over a common shared network to provide functionality to the user. The subsystems can refer to the engine, the transmission or perhaps the suspension module. Typically, each subsystem will have its own ECU, and this will communicate with other ECUs in realtime to ensure smooth functioning of the vehicle. The most common network over which ECUs communicate is known as the CAN standard, a shared bus, where communication is facilitated by the broadcast of packets with an unique identifier indicating which ECU the message was intended for [41]. The number of ECUs in the vehicle have grown over time (currently around 150 [6]), which in turn have led to the integration of other communication standards providing features that the CAN standard may not have been appropriate for [33]. For example, applications such as camera assisted parking systems benefit greatly from the FlexRay protocol because of it's high bandwidth of 10Mbps as compared to the 1 Mbps of the CAN standard. Further examples can be found in Table 1 where a summary of the most common communication protocols and their typical use cases are presented and Figure 2 visually depicts how the ECUs of various subsystems might communicate with one another using using the relevant communication protocol stated in Table 1

2.2 Automotive Cybersecurity

2.2.1 The Present

A common association that the average human being would make when the term 'cyber security' is mentioned, would be to instantly think about it as the prevention of the 'hacking' of a computer, a web server or a smartphone. Strictly speaking, cyber security encompasses a much broader spectrum and includes areas such as application security, cloud security and infrastructure protection to name a few. The National Cyber Security Centre UK defines cybersecurity as the "means by which individuals and organisations reduce the risk of being affected by cybercrime [2]." Furthermore, they mention that a cyber security program's main priority should be to protect personal devices such as smartphones, tablets and computers from unauthorised access to prevent the theft of valuable personal information.

Protocol	Properties
CAN	Multi-master asynchronous serial network protocol for high reliability control applications
	Speed: Max. 1 Mbps
	Applications: Body Systems, Engine Management, Transmission
LIN	Low-speed, single-master, multiple-slave serial networking protocol. The LIN master node typically connects the LIN network with higher-level networks.
	Speed: Max. 20Kbps
	Applications: Door Locks, Climate Control, Seat Belts, Sunroof, Lighting, Window Lift, Mirror Control
FlexRay	Next-generation, deterministic and fault-tolerant network protocol to enable high-bandwidth, safety-critical applications
	Speed: Max. 10 Mbps per channel (dual channel)
	Applications: Drive-by-Wire, Brake-by-Wire, Advanced Safety and Collision Avoidance Systems, Steer-by-Wire, Stability Control, Camera-Based Monitoring Systems
MOST	High-speed multimedia network technology, daisy-chain topology or ring topology and synchronous data communication to transport audio, video, voice and data signals via plastic optical fibre (POF) or electrical conductor.
	Speed: Max. 10 Mbps or faster
	Applications: Car navigation system, audio system etc.

Table 1: Top-level summary of vehicle network protocols, typical use-cases and speeds.

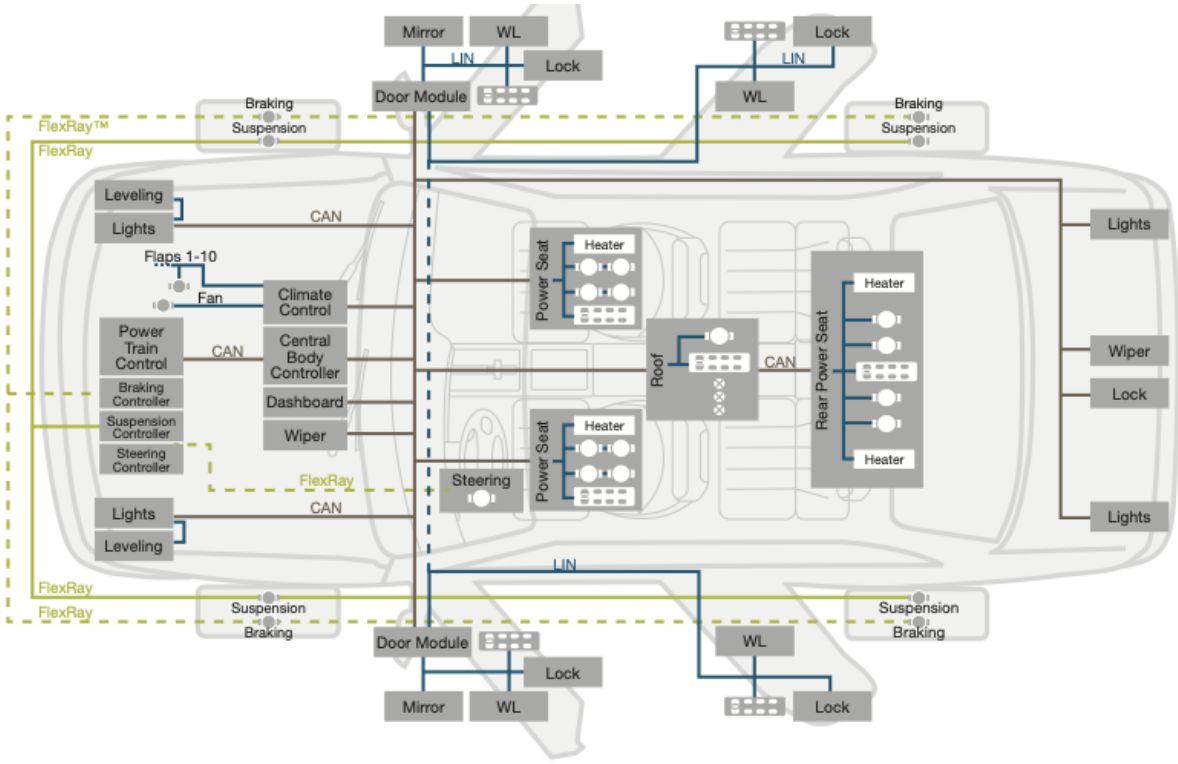


Figure 2: An overview of in-vehicle networking between automotive subsystems.

A 2019 report from Cybersecurity Ventures mentions that the cyber security market was worth a mere $3.5 billion back in 2004, whereas in 2017, the market was valued at over $120 billion, implying a 35-fold increase in spending [3]. Furthermore, in 2022, the market value is expected to cross $170 billion. The 10-15% year-on-year increase seems to be consistent with other sources such as Forbes [17] and TechRepublic [30], with cloud security being the top contributor to the rise in spending (approximately 41% year-on-year increase).

Historically speaking, cyber security in the automotive industry has not been a very common topic of discussion. This changed in the summer of 2015, when two white-hat security researchers going by the names of Chris Valasek and Charlie Miller took the automotive industry by surprise, and wirelessly broke into the internal network of a Jeep Cherokee through manipulation of the infotainment system [1]. The duo were able to perform tasks such as blast cold air through the air conditioning vent, turn on the windshield vipers and control the music system of the car. At first glance, these exploits may not seem to pose a threat to society, but it did alert car manufacturers around the world of the lack of design considerations given when developing the security features of the car. Prior to the attack, several car manufacturers claimed that it was 'virtually impossible' to enter the vehicle's internal networks without a physical connection via the OBD-II (On Board Diagnostics) port. An OBD port is a physical interface through which a user can gain access to a vehicle's subsystems for the purposes of retrieving information and diagnostics from the various ECUs. Therefore, the incident did come as quite a surprise to Jeep with one of the repercussions being a software recall of almost 1.4 million vehicles. This was not the first time Valasek and Miller performed such an attack. Back in 2013, the duo were responsible for a similar demonstration of the security exploits in Toyota and Ford vehicles, where they were able to remotely control the steering wheel and at one point, through manipulation of the transmission, were able to bring the vehicle to a complete standstill. In another incident, a team of researchers at KU University Leuven were able to break into the a Tesla Model X in under two minutes using a Raspberry Pi, a replacement Tesla ECU and the key fob, all amounting to a mere $200. They managed to successfully implement the attack by taking advantage of a security weakness in the Bluetooth communication channel between the key fob and the vehicle [4].

The above incidents have definitely pushed car manufacturers to consider adopting a 'security by design' approach, rather than waiting for something calamitous to occur, and then scrambling to patch up the security exploits via a software recall, as was the case with Jeep or an OTA, in the case of Tesla. The adversaries did not have any malicious intent when carrying out the attacks, but, the fact that they were able to do it so easily was definitely a matter of concern and prompted swift action. The modern vehicle has immense technological capabilities ranging from WiFi hotspots to sophisticated HMI functionalities such as automated sleep detection and heart monitoring systems via the steering wheel. All of these features and connectivity support our digital lifestyles, but they become unfeasible if the internal software of the vehicle is not strongly equipped to deal with unauthorised intrusions. Automobile manufacturers' goal for the future is a vehicle that functions more like a smartphone on wheels than a traditional automobile. Governments around the world hope to build on the automakers' vision by increasing spending towards making smarter cities via V2X communication platforms. Vehicles will undoubtedly become more personalised from here on out, and form a strong coupling with our personal online accounts in a similar fashion to how smartphones

did when they were introduced into our technological ecosystems. This can be a good thing, because as consumers we have an additional avenue of access to all of our digital services and subscriptions on the go. But, the increased connectivity and the pool of valuable information within a vehicle's network can turn into a strong incentive for an adversary to implement a cyber attack on a vehicle. According to TÜV Süd, a German organisation that specialises in inspection and certification of technical systems, estimated that the total number of ECUs that were in operation in 2018 were almost 100 million. Moreover, they predict that the number of fully autonomous vehicles on the road to reach 33 million in 2040. All of this information points to one thing, which is that now would be a good time for automotive OEMs to divert funding and research into making vehicles more cyber secure in the future.

2.2.2 Attack Types

Vehicle attacks fall into four major categories: interception, injection, modification and interruption. A broad summary of vehicle attack types are presented:

1. **Spoofing attacks** – a kind of interception attack, where an a malicious actor tries to disguise themselves on a network, to perform malicious activity [31].

2. **Denial of Service (DoS)** – any type of attack on a networking structure to disable a server from servicing its clients. Attacks range from sending millions of requests to a server in an attempt to slow it down, flooding a server with large packets of invalid data, to sending requests with an invalid or spoofed IP address [47]. Injection and interruption would be the category that DoS attacks fall under.

3. **Advanced Persistent Threats (APT)** – refers to the use of a persistent, multi-stage attack to gain access to the targeted system and breach it in order to obtain sensitive information and do severe harm [29]. This is a multi-stage attack and could comprise of all of the above mentioned categories, for example a spoof attack to enter the system, and then an injection in the form of a DoS attack to disable a system.

2.2.3 Attack Surfaces

An attack surface is effectively the complete area of a system that is exposed to the outside world and it encompasses all possible attack vectors (or vulnerabilities) through which an adversary could obtain access to the system [40]. This is the first step an adversary would consider in order to gain access to the vehicle's internal security systems, where a range of different attacks can then be performed. This is not a comprehensive list of the attack surfaces, but it does shed light to the issue of the increasing number of attack surfaces as vehicles get more advanced in the future. A short summary of common attack surfaces include:

1. **Infotainment Systems** – a typical starting point for an adversary would be through an infotainment system, with the primary reason being that it indirectly provides access to the internal vehicle CAN bus [14]. Another reason is that there is a wealth of information available from the manufacturer on the operating systems and firmware that infotainment systems use, for the purposes of maintenance and repair work carried out by independent garages. Assuming

that an adversary manages to get a hold of this, it should be fairly straight forward to use the telematics system as a backdoor into the internal CAN bus of the vehicle.

2. **OBD-II Dongle** – this is an interface that has been mandated to be present in cars manufactured after 1996, and is used by mechanics and technicians to access all CAN buses in the vehicle for the purposes of maintenance, updates and diagnostics. The interface can be a physical connection or a wireless one facilitated by bluetooth, Wi-Fi or 4G/5G. Nonetheless, it provides an additional point of entry to an adversary, as demonstrated by the security in Zubie's connected car OBD-II connector [46].

3. **Bluetooth** – a fairly standard feature in the present day automobile, based on a globally accepted standard, with the same protocol stack. However, the front-end implementations differ from vehicle to vehicle and this is where an adversary might identify a vulnerability, as demonstrated by Miller and Valasek. They managed to gain access *directly* by remotely identifying the Bluetooth MAC address and brute forcing their way in and *indirectly* by navigating their way through an already paired phone [1].

4. **In-car Wi-Fi** – a feature that is starting to become more common in the car [ref], can also potentially be one of the easiest way to gain access into the internal subsystems. Once again, Miller-Valasek demonstrated this vulnerability by reverse engineering the password in 30 seconds, to gain back door access into the vehicle network [1].

2.2.4 Attack Consequences

Now that the attack surfaces and attack types have been touched upon, it might be useful to see potential motivations as to why an adversary might want to perform a cyber attack. Table 2 presents a value chain analysis to show why an adversary might carry out a cyber attack in an automotive context.

2.3 Chapter Conclusions

The main goal of this chapter is to put the motivation and the research aims of this project into the wider context. Firstly, given the increased number of attack surfaces and information available in the form of increased sensors, one of the research aims of this project is to evaluate the suitability of a PoL style monitoring framework to take advantage of the enormous chunks of information being processed every second in a vehicle, and potentially detect when a vehicle has undergone or is currently undergoing a cyber attack. Secondly, it is important to realise that there are several types of attacks that an adversary might choose to perform, of which different patterns will be induced in the recorded information. This research aims to investigate the patterns induced by DoS attacks.

Attacker	Motivation	Outcomes
Organised criminal gangs	Financial gain	Vehicle theft, goods theft (in case of commercial fleets), social media accounts, denial of service of vehicles (which could be re-enabled after the owner pays a ransom), premises security and burglary vehicle data (reveals which businesses and homes are unoccupied)
Business competitor	Intellectual property theft	Access valuable algorithms in vehicle ECUs developed to increase fuel efficiency or stability control
Terrorists and Hacktivists	Terrorism	Denial of service to create mass disruption on the roads, manipulation of cyber-physical systems to cause harm to the occupants, abuse of infotainment systems for the distribution of propaganda material, tracking of VIP locations
Unhappy ex-employees	Competitor reputation	Causing damage to the reputation of an OEM by some kind of denial of service attack
White-hat hackers and Academics	Research and Exploratory Analysis	Remotely taking over vehicle controls with a motive to expose vulnerabilities, sabotage or degrading of vehicle and connected system performance.
Automotive enthusiasts and Script kiddies	Miscellaneous	Enabling features via software to avoid paying for expensive vehicle options, enhancing the performance by modifying the fuel-air map of the engine

Table 2: A value chain analysis that provides summary of potential automotive attackers, their motivations and consequences [26]

3 Literature Review

This section builds on the automotive cybersecurity conundrum and introduces a new approach towards combating cyberattacks, an approach based on Activity Based Intelligence (ABI). Existing literature shall be reviewed, knowledge synthesised and gaps in the knowledge will be identified. The work of this project shall aim to address these knowledge gaps.

3.1 Activity Based Intelligence

3.1.1 Introduction

The current approach to intelligence is still relatively linear, whilst the systems on which intelligence is being generated are highly non-linear. The present intelligence paradigm can be effectively summarised into four major steps: requirements definition, collection against those requirements, analyse and repeat. This technique have served us well up until now, however, it cannot be used to create large-scale, aggregated population-level systems, kinematic simulations of object motion, or statistical demographic-based models of behaviour, because they do not capture the nuances of an entity's actions, beliefs, and motivations [20].

Activity-based Intelligence (ABI) offers a truly novel approach of performing intelligence analysis, one that has the potential to completely disrupt our current tired and worn intelligence processing framework. ABI is an automated analysis process that swiftly combines data from numerous sources to detect significant patterns, determine and identify changes in them, and characterise those patterns in order to drive collection and provide decision advantage [13]. ABI practitioners have advanced the

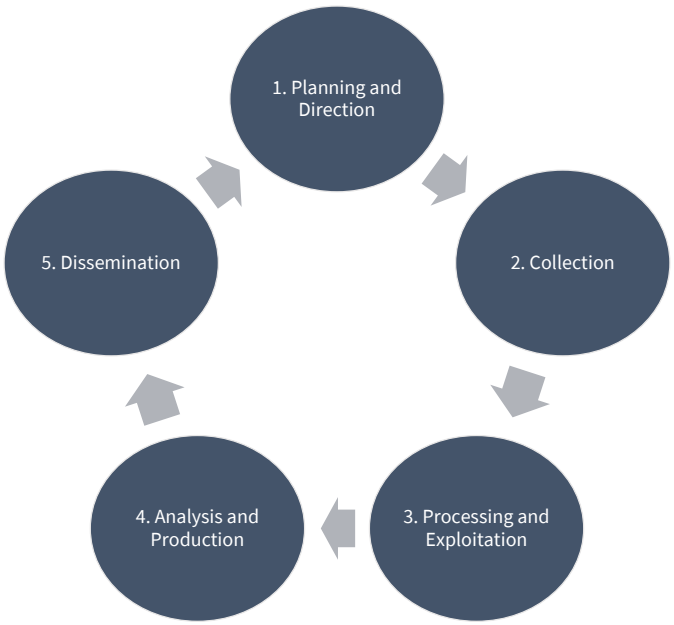


Figure 3: The steps involved in the ABI cycle [13].

concept of large-scale data filtering of events, entities, and transactions to develop understanding through spatial and temporal correlation across multiple data sets, as opposed to the traditional intelligence cycle, which decomposes multidisciplinary collection requirements from a description of the target signature or behaviour.

ABI consists of four basic pillars [13] that differ considerably from traditional intelligence and are described as follows:

1. **Geo-reference to discover** – everything and everybody has to be somewhere at a given time, more formally speaking, it means spatial and temporal data can be used to merge multiple sources of information that can potentially lead to invaluable insight into new, undiscovered patterns.

2. **Sequence neutrality** – realising that sometimes the answer has the likelihood of arriving before before the question has been asked.

3. **Data neutrality** – essentially all data, irrespective of source or form, is considered during intelligence generation.

4. **Integration before exploitation** – combining and correlating data and realising patterns is to be done as early as possible, rather than try to analyse further down the processing pipeline. Seemingly unimportant information from a single sensor may be important when combined and integrated across other sensor readings.

ABI has started to gain traction in recent years with the rise in information available from the increased number of sensors present in our cyberspace and the increased availability of highly performant computing devices.

3.2 Pattern of Life

3.2.1 Introduction

Pattern(s) of life, like many concepts in and around ABI, suffers from a lack of written literature and is normally defined based on the context on which it is applied [13].

3.2.2 Definitions

PoL analysis can be used to analyse the behaviour of both human and non-human entities in a variety of applications. Currently, humans generate PoL intelligence manually, which is a time-consuming task that frequently results in humans being overloaded with data. Based on [9] [18] [44] [13], a group of PoL-specific definitions shall be made to avoid any sort of ambiguity in the rest of the documentation.

1. **Entity** – a tangible or intangible thing (living or non- living, individual or group) which exists in an area of interest and possesses dynamic parameters.
2. **Behaviour** – a sequence of an entity's dynamic parameters which can be observed or inferred from observations.
3. **Pattern of Life** – the repeatable behaviours of an entity which recur at normally-distributed time intervals, under a particular set of conditions.
4. **Conditions** – in the context of a PoL, refer to the factors which influence an entity's behaviour.
5. **PoL history** – the set of PoLs which results from combining the PoL from current and past observations for one or more entities, the PoL for similar entities and the PoL which results from other sources.
6. **PoL processing** – the creation, maintenance and application of PoL history which is used to recognise, analyse and predict behaviours and anomalies occurring either currently or in the past.
7. **PoL intelligence** – a set of information which is the output of Pattern of Life processing.

A typical workflow how one might go about implementing the PoL framework can be seen in Figure 4.

3.2.3 PoL Related Literature

ABI and PoL applications mainly reside in the surveillance, military and defence sectors, with some of them being documented and published as research papers. However, in recent times researchers are starting to notice strong value in the application of PoL to their respective domains, the application domains of PoL sees to have become a lot more varied. The aim of this section is to review these studies to help fill any knowledge gaps, before proceeding to the design and implementation of an automated PoL processing framework in an automotive cybersecurity context.

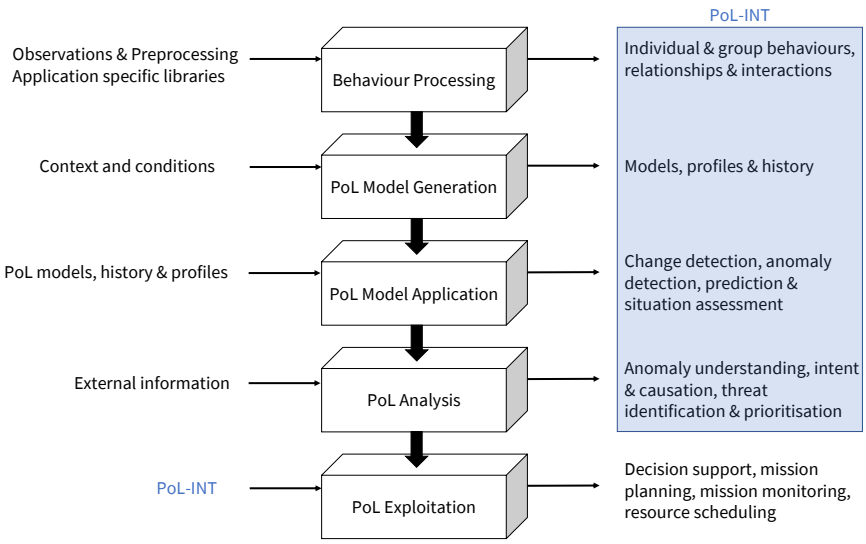


Figure 4: Overview of the PoL processing framework proposed by Craddock et al. [18].

Study 1 – In an article on 'weaponised media' using pattern-of-life analyses[25], the author looks at how highly automated data collection and algorithmic infrastructures using drones and satellites were used to help identify criminogenic patterns and schemas that arise from human populations. This is quite fascinating because it introduces a new 'vertical perspective' of surveillance that no longer relies solely on optical media alone. The endgame here is that these masses of information can be turned into actionable intelligence [25], and could perhaps be used for the policing of populations. The ethics of these endeavours are a topic of discussion for some other time, it is just the application of PoL analyses that is the focus here.

Key takeaways – This study highlights one of the most early and most common implementations of PoL processing frameworks, which is in a military context and what some of the key motivations were. Before, undertaking work in a field with a lack of literature, reviewing early implementations can provide unmatched perspective when venturing out to adapt the framework for application in the automotive cybersecurity domain.

Study 2 – A group of scientists at Airbus reviewed the current state of the art air and maritime surveillance systems [35] [19], and explain how modularised lambda architecture-based ABI and PoL processing frameworks can be integrated directly into existing surveillance systems. A lambda architecture is effectively three layer system – a batch layer that continuously stores and receives raw sensor data (a distributed filesystem), a serving layer that holds learned ML models and a speed layer that takes in data in real-time and interacts with the serving layer to help with classification and identification on the go. Figure 5 helps to better visualise the interactions between the various layers of this system. The main sources of raw data for the batch layer were from GMTI radar and satellite based AIS.

Their motivation was simple – given the volumes of data available to existing surveillance systems, it seems like a wasted chance that nothing is being done to exploit its availability. Moreover, they go on to describe how a combination of supervised ML, unsupervised ML and big data techniques can be used to create patterns of life that would facilitate the creation of advanced analytics and

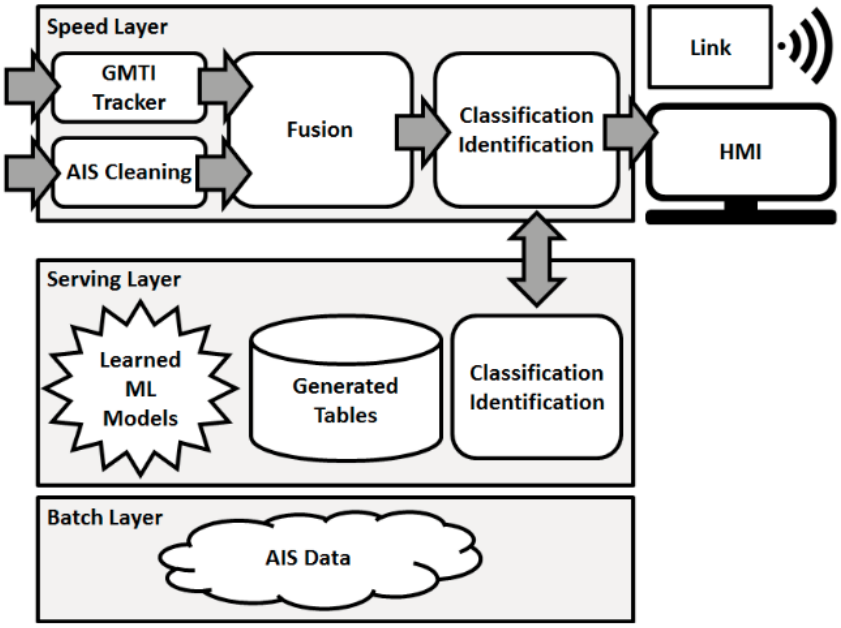


Figure 5: A lambda architecture based modern surveillance system that provides advanced analytics.

invaluable decision support for the user.

Key takeaways – An important design consideration is to ensure that the system architecture of the framework being developed mimics or at the very least is similar in nature to the architecture of the system it is to be integrated into. The advantage of this is that the developed framework would be highly adaptable and can fit seamlessly into other domains since system architects are more likely to utilise state of the art system architectures in their respective domains.

Study 3 – In a break from military and defence use-cases, one study used PoL techniques to gather a dataset of 37 million geolocated tweets to characterise the movement patterns of 180,000 individuals [24]. By deducing a PoL for each individual (entity), the authors successfully managed to characterise a change in word usage as a function of movement. This identified pattern led them to discover that expressed happiness increases logarithmically with distance from an individual's average location. The contributions of the study essentially encapsulated the fundamentals on which the PoL processing framework is based on – the combination of spatiotemporal data along with other variables of interest being captured from an entity over a period of team can lead to the identification of behaviours and patterns that aid in the discovery of unique characteristics of entity.

Key takeaways – The study highlights the need of PoL researchers to firstly identify sources of big data and know what combination of parameters can be combined to potentially uncover correlations that could, for example, enhance decision advantage. Sometimes, it is simply not about taking the first 'big data' source, more importantly it is to develop the intuition of knowing what arbitrary set of parameters can be combined to produce interesting behaviour patterns.

Study 4 – The authors of [9] used PoL in network analysis to improve the effectiveness of Intrusion Detection Systems (IDSs). Their key motivation was that next generation IDSs should be capable of adapting their detection mechanisms not only to be based on measurable network parameters, rather, they should incorporate contextual information, situational awareness and cognitive information within the context of the application of the network. Essentially, the key point here is to construct a PoL for the higher level behaviour of the network, and to use this extracted intelligence to improve detection rates of the IDSs and reduce the number of false alarms. They did this by using fuzzy cognitive maps (FCMs), a data fusion technique, and correlated the users of the network to the time of the day, and the usage of the network resources, to extract a contextualised behaviour that would improve decision advantage of the IDSs.

Key takeaways – The 'network' context in which this study was carried out aligns closely with the methodology expected in the work that shall be carried out in this project. For instance, network throughput, data fusion from different packets, timestamps and the composition of packets in a specified timeframe are some of the potential parameters that would need to be continuously monitored and checked for anomalies.

Study 5 – From an automation perspective, a group of scientists developed an automated PoL processing called POLIS (Pattern of Life Integrated System) [28]. In addition to being fully automated, their application was scalable (cloud capable), able to combine contextual information from raw sensor data and various other forms of soft data such as past reports and existing models to provide PoL analyses and alerts. Furthermore, they applied this framework to a maritime use case using a layered approach comprising of:

1. Baselines for normal operation was recorded.
2. AIS sensor data was collected.
3. Hard and soft data association and fusion.
4. PoL anomaly detection implementation.
5. Threat modelling, alerting, reasoning and visualisation.

Key takeaways – This paper depicts that PoL should be developed with scalability, deployability and automation as core design criteria. The importance of this is further reiterated in another study that reviews potential solutions towards scalable PoL implementations [22]. Currently, no such PoL implementation exists in an automotive cybersecurity context that provides functionality such as the one described above. Therefore, this project aims to fill this gap in the industry keeping in mind the above mentioned design criteria.

3.3 Putting It All Together

Having described the current state of the art in automotive networking and reviewing the current state of cybersecurity in the automotive industry, the purpose of this literature review was to introduce the reader to a slightly less known approach to security – activity based intelligence and PoL (a

subset of ABI). Moreover, it provides a wealth of architecture and design principles that would aid in achieving the main objective of this project, which is to implement a PoL processing framework in an automotive cybersecurity context.

In essence, these are the definite conclusions that shall be used in the design and implementation of the project:

1. Software architecture of the PoL processing framework should mimic the system it aims to be integrated into, to make the integration process seamless.
2. Automation is key, little to no time to be spent on manually analysing the data being collected (i.e. minimal human intervention)
3. There is no specific ML technique that is considered the gold standard, a combination of supervised and unsupervised ML should be used keeping in mind the application and the desired outcomes.
4. Scalability and deployability (cloud) of the developed PoL framework is imperative.

4 Methodology

4.1 Hardware Setup

An automotive test rig provided by Thales was used for the practical part of the project. The rig contains a user input panel consisting of set, resume, cancel cruise control and brake buttons. To the extreme right of the input panel is the throttle knob that rotates clockwise to increase throttle demand. There is a motor that rotates counter-clockwise depending on the inputs given to the user input panel. An input controller is connected to the input control panel and a motor controller connected to the motor, which are both in turn then connected using a five port TP-Link TL-SG105E Ethernet switch. The network protocol used by the two controllers is the IP protocol and the input controller communicates with the motor controller using customised UDP packets written in the application layer. The switch has five available ports, where ports two and four are connected to the input controller and motor controller respectively, port five is a mirroring port that can be used to monitor the packets being sent across the network leaving two ports available for the user.

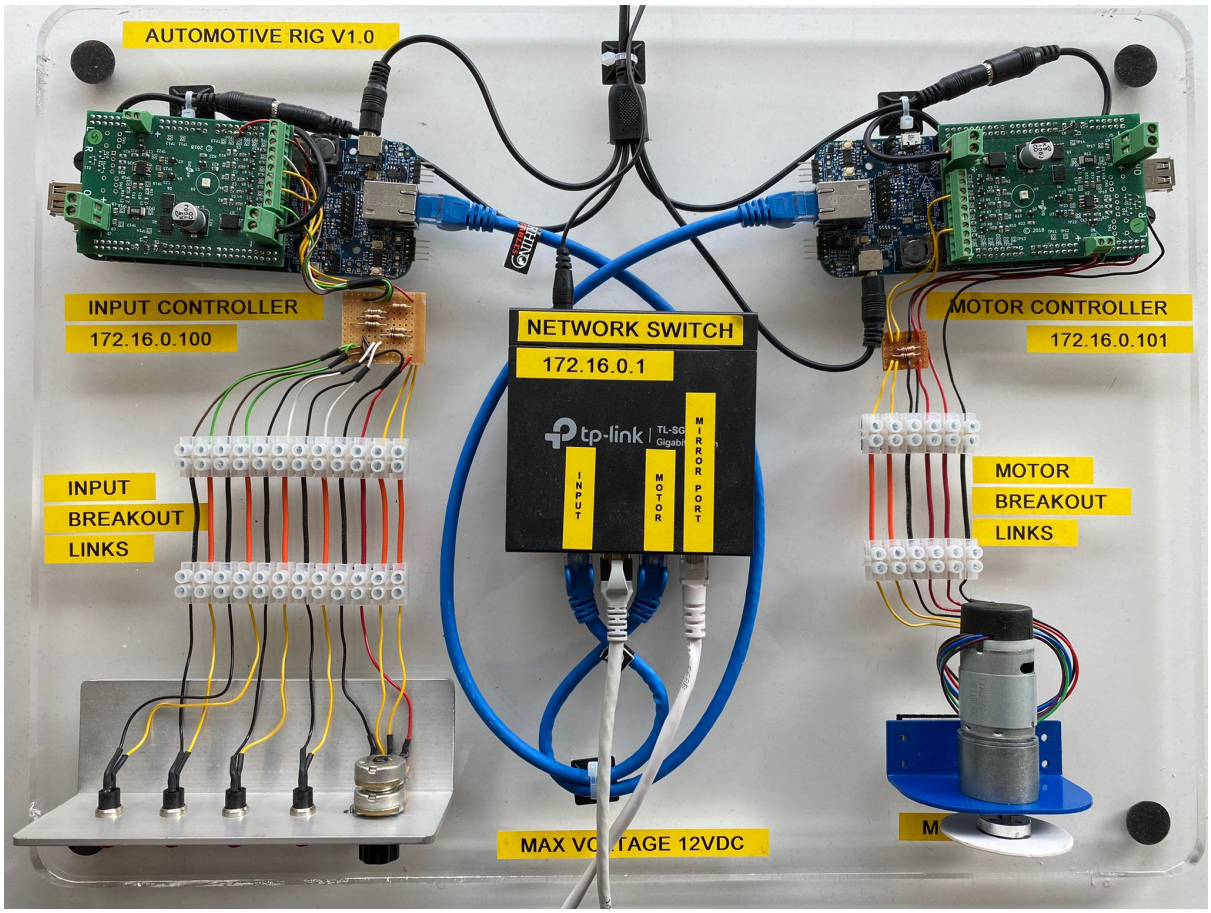
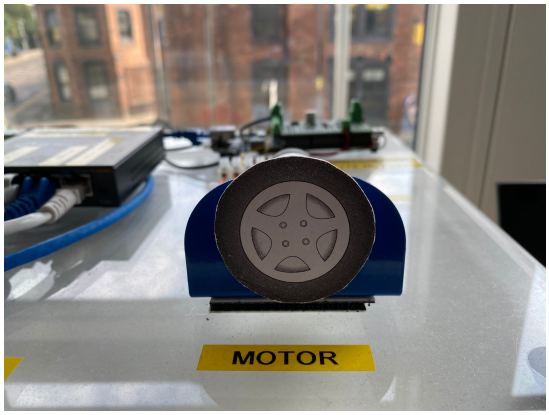


Figure 6: The automotive rig v 1.0 provided by Thales.

The hardware demonstrator provides an abstraction to the shared network and components one would expect to find in a real automobile, making it a suitable choice of hardware to address the research aims set out for the project. An important point to note is that wireless interfacing with

(a) User input panel



(b) Motor

Figure 7: An abstraction of the input and outputs of a real vehicle provided by the hardware demonstrator.

the demonstrator is not possible due to the absence of Wi-Fi and bluetooth modules, however, this should not pose a problem since the attack being investigated in this research is a DoS attack via packet injection. This can be done by connecting a computer to one of the two free available ports on the Ethernet switch, and running the packet injection script.

4.2 Project Division

Before proceeding further, there is an important distinction that needs to be made between the two major components to this project. The first is a software development component that builds the tools required to obtain data from the hardware demonstrator in a suitable format for further processing. The second is a research component, where an experimental methodology shall be detailed to answer the research question of whether a PoL inspired approach is a suitable one for the early detection of DoS attacks.

The overall workflow of the project is split into three separate layers as highlighted in Figure 8, where the data collection and feature extraction layers are implemented in the software development component and the DoS attack detection layer is implemented in the research component of the project. The architecture of the pipeline has been constructed to in a way that makes it possible to implement the key steps of a traditional KDP, which are data selection, data integration, data transformation, data mining and pattern evaluation [15].

4.3 Software Development Methodology

One of the key takeaways of the literature review was to ensure that the software development process should try to mimic the system it aims to be integrated into. Therefore, to make things contextually clear to researchers in the automotive domain, a workflow based on a commonly used engineering lifecycle in the automotive industry called the V-Model is used, with a summary of the key steps outlined in Figure 9. Taking into account the time available to deliver the project objectives, the entire left side of the V-model (steps highlighted in blue) and the unit testing part is what shall be implemented and documented going forward.

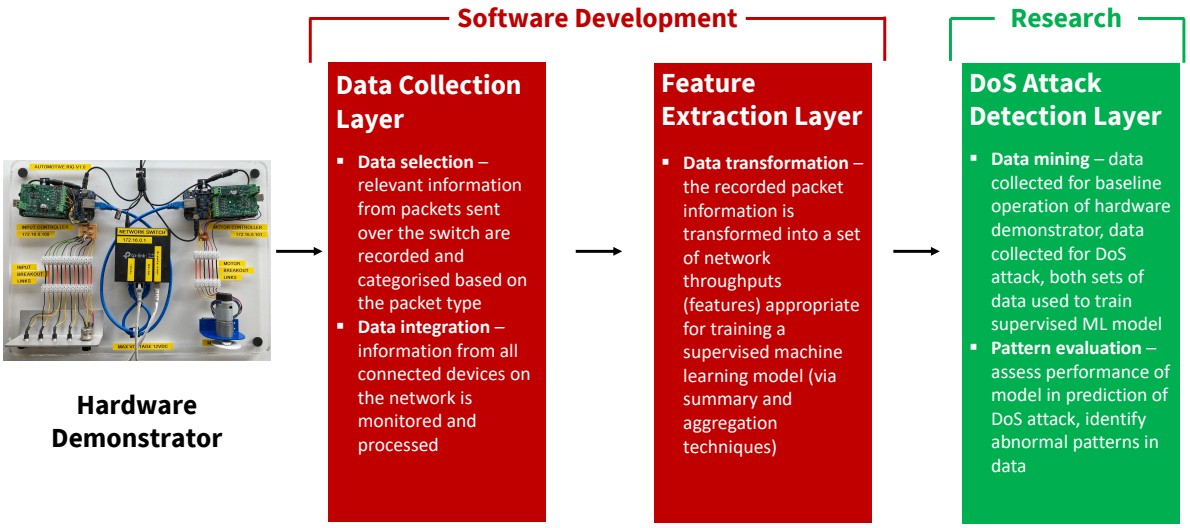


Figure 8: A three layered pipeline to perform the KDP on the provided hardware demonstrator.

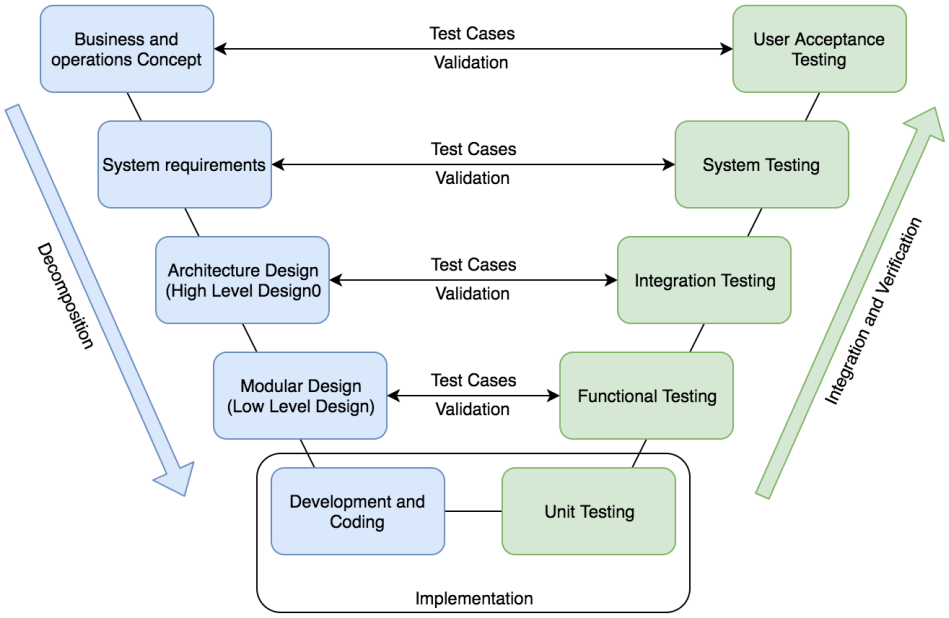


Figure 9: The V-Model development lifecycle.

4.3.1 Business and Operations Concept

A business and operations concept is not extremely relevant in this case simply because the software being developed here is not enterprise-grade, and will be used to aid research. For the sake of comprehensibility however, the main need for an application of this kind would be the collection and processing of information from the hardware demonstrator in a format suitable for further processing.

4.3.2 Requirements Definition

The next step of the V-model states the need for a formal definition of the system requirements. The system in our case is the application that is capable of parsing packets being sent across the networks and processing it into a form suitable for further exploitation. A formal definition of the application requirements is presented in Table 3 and acts as the blueprint for architectural design, modular design and development and coding found in Section 5 (Software Implementation).

Req. ID	Requirement	Implementation
R001	The application shall be able to parse the output of the mirror port on the ethernet switch for the retrieval of vehicle speed, brake, cruise control and throttle information.	Section 5.2
R002	The application shall be able to record information on the packets being sent across the network that would aid in the detection of a DoS attack.	Section 5.2
R003	The application shall be able to assign a driving mode to the current functioning state of the system.	Section 5.3
R004	The application shall be able to extract features from the recorded packet information that can be used as the basis for DoS attack detection.	Section 5.3

Table 3: System level requirements of the software development part and the section in the documentation where the implementation of the requirement can be found.

4.4 Research Methodology

4.4.1 PoL Analogy

The second component of the project is the research component, where the performance of a PoL based approach towards DoS attack detection is evaluated. Looking at the problem space in terms of a PoL perspective, the first step would be to identify the *entity* for which a pattern of life is being created. In this case, the *entity* being observed is the hardware demonstrator because it produces a set of observable *behaviours* at equally distributed time intervals. The assumption here is that the entity exhibits two distinct behaviours – a normal behaviour, which is basically the baseline operation of the entity, and an attack behaviour, which is when the entity has is undergoing or has undergone a DoS attack. The patterns in the observed behaviours of the entity may be different under several *conditions* and this would need to be kept in mind when devising an experiment plan to identify the "attack" behaviour exhibited by the *entity*. A summary of these analogies are presented in Table 4.

PoL Element	Project Analogy
Entity	Hardware demonstrator
Behaviour	Normal or Attack
Conditions	Cruising or Not Cruising
PoL History	Baseline data and DoS Attack data collection, feature extraction and storage
PoL Processing	Training of binary classifier using network throughputs as features to distinguish between normal / attack behaviour based on PoL history
PoL Intelligence	Prediction from binary classifier if the board is exhibiting normal or attack behaviour

Table 4: Drawing analogies between the work carried out and the PoL framework.

4.4.2 Hypothesis

The hypothesis is that the early onset of a DoS attack can be predicted by monitoring the throughputs of the various protocols on the network, with the throughputs of certain protocols making the early detection of the DoS attack possible. In mathematical terms, a set (X_t) of n network protocol throughput values for a timestamp t, where $X = [X_1, X_2, X_3 \cdots X_n]$, is assigned a label (y) where $y \in \{0, 1\}$. The 0 label indicates that the demonstrator is exhibiting "normal" behaviour and the 1 label means that the demonstrator is exhibiting "attack" behaviour.

4.4.3 Workflow

The research method to be followed is seen in Figure 10 and detailed in Sections 6 and 7. The steps to be followed are presented:

1. **Step 1 –** Collection of data that characterises baseline operation of the hardware demonstrator.
2. **Step 2 –** Perform a DoS attack on the hardware demonstrator in 3 different scenarios and observe the behaviour of protocol throughputs as the attack is performed.
3. **Step 3 –** Identify the list of protocols that exhibited a pattern in throughput behaviour during the DoS attack.
4. **Steps 4 and 5 –** Extract the set of throughputs of the identified protocols from Step 3 from the DoS Attack Data from Step 2 and label them 1 (Attack). Extract the set of throughputs of the identified protocols from Step 3 from the baseline data from Step 1 and label it 0 (Normal).
5. **Step 6 –** Train a binary classifier using this labeled data.
6. **Step 7 –** Evaluate the performance of the binary classifier on a validation test set never before seen by the classifier.

4.4.4 Supervised ML Models

This is a binary classification task, where a feature vector X maps to either 0 or 1. The are several supervised ML algorithms that will be evaluated and a brief description of the algorithms are presented below:

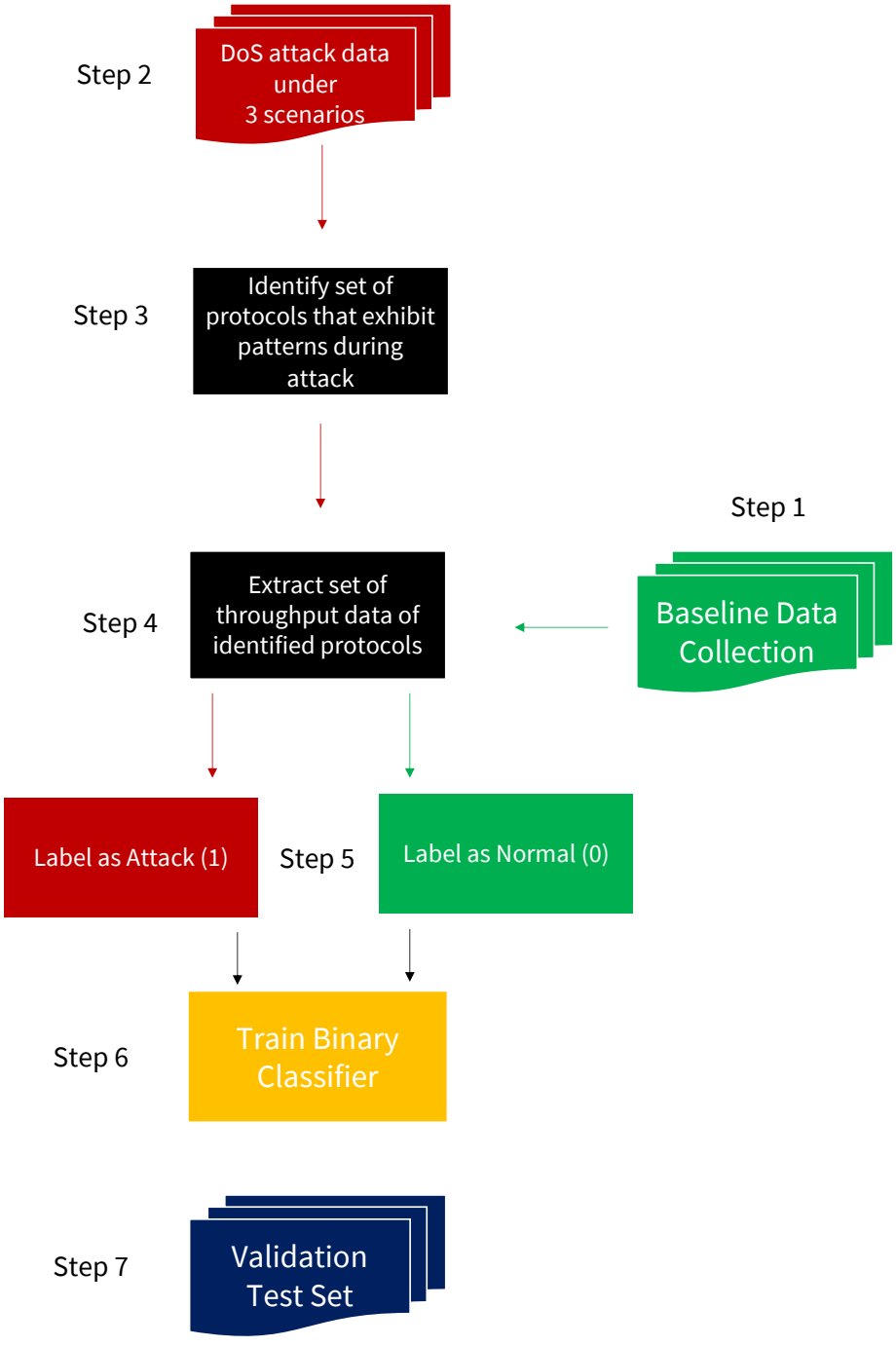


Figure 10: The steps involved in the research methodology to evaluate the Pattern of Life processing framework.

Logistic Regression – a model the relationship between a set of independent features where the dependent variable is categorical. It is an extension of the linear regression model, where the outcome of the logistic classification model outputs a probability between 0 and 1 [48].

Naïve Bayes Classification – one of the most simple Bayesian network models. It relies on the assumption that it is possible to predict the probability of an event based on prior knowledge regarding it. In the Naive Bayes classification model, there is an assumption of strong independence between the variables. While this may seem rather impractical, and not applicable to real-life data, where the variables are categorical, this model has shown to have good predictive accuracy [23].

Support Vector Machines – a model that was developed by Vapnik et el, and is based on the Vapnik-Chervonenkis theory. It takes training data and forms a "hyperplane" that acts as a gap between observations belonging to two separate categories. When new data is entered, it is plotted on this hyperplane based on its known features and then is assigned one of the two classes, depending on which side of the gap it falls on [38].

Decision Trees – a method through which classification is achieved via the means of multiple "decisions". These decisions are based on simple rules or features that are known or inferred about sets of data, which allows for the modelling and prediction of unknown data [39].

Random Forest – an ensemble classification model that relies on the use of many decision trees. For this reason, often, Random Forest classification often surpasses the Decision Tree model described above [12].

Gradient Boosted Trees – an ensemble learning technique, where weak learners (decision trees) are combined to form a strong learner, they have been gaining popularity in the ML community lately, hence have been chosen for evaluation [32]

4.4.5 Evaluation Metrics

The convention for the confusion matrix to be followed is that the majority class (Normal) is assigned as negative, and the minority class (Attack) is assigned positive.

		Prediction	
		0 (Normal)	1 (Attack)
Actual	0 (Normal)	TN	FP
	1 (Attack)	FN	TP

Keeping in mind the above convention, the metrics for evaluation of the above described supervised ML models are:

Execution Time – a typical use-case of an attack detection framework of this nature will be in an embedded system where computational resources are limited. The execution time here refers to the training time plus the time for testing on the validation set.

Precision (PPV)– the ratio $\frac{TP}{TP+FP}$, i.e. the rate at which the model predicts an attack correctly, as a proportion of all the instances it predicts that a DoS attack is taking place.

Recall (TPR)– the ratio $\frac{TP}{TP+FN}$, i.e. the rate at which the model predicts an attack has taken place, as a proportion of all the instances that the demonstrator is actually under a DoS attack.

F1-score – the harmonic mean of precision and recall given by $2 \times \frac{Precision \times Recall}{Precision + Recall}$, and provides a more generalised statistical index of the model's performance when the training set is greatly imbalanced, as is the case here.

Matthew's Correlation Coefficient (MCC) – a discrete implementation of the Pearson's Correlation Coefficient (PCC) used in binary classification problems and is used to quantify the correlation between the labels and predictions. According to [11], it can be viewed as stronger indicator of model performance for imbalanced datasets as compared to the F1-score, hence the inclusion.

5 Software Implementation

This section describes the steps involved in the implementation of data collection and feature extraction layers of the software development part of the project. The software tools, software design choices and a thorough description of each layer is presented.

5.1 Software Setup

On the software front, there were several decisions to be made because, apart from defence related implementations of automated PoL frameworks (the software is not typically open source for obvious reasons), there was not a lot in terms of guidance on what would be the most optimal pipeline. A summary of the software used is presented:

During the initial phases of the work, it was decided that a combination of Python and MATLAB were to be used for the project. Python was chosen due to the availability of its documentation and ML libraries. MATLAB was chosen because of it's comprehensive automotive specific toolboxes. However, about a month into the project, there was a lack of cross-compatibility between the MathWorks toolchain and Wireshark which made packet capture quite a task. The codebase was then migrated over entirely to Python.

Packet Capture – For packet capture, an open source network protocol analyser called Wireshark is used. To aid in the dissection of the customised UDP packets sent by both controllers, Thales provided custom Lua dissectors that were able to extract the payloads from the packets being sent across the network (vehicle speed, cruise control, brake status and throttle demand) and this was visible in the Wireshark GUI.

Scripting – Python was chosen as the main scripting language for a multitude of reasons. The primary reason for this decision is the cross-functional support with WireShark and tshark (tshark is the command line version of Wireshark). Secondly, the access to coherent and extensive documentation and ML libraries made it a solid choice for this application.

Version Control – Git was used for the entirety of the project with GitHub as the remote repository. This project aims to be open-source after the submission deadline, with the ambition that this framework gains more attention for application in other domains. Furthermore, it aims to provide future students with the opportunity to pick up exactly from the current state of the project, and contribute to this growing field. There is also the opportunity for members of the ML community to gain access to the dataset so that they can perform their own machine learning techniques.

Distribution – The implementation of the of the whole project was carried out in a virtual environment. A *requirements.txt* was exported with all relevant dependencies for the purposes of reproducibility for a future user of the hardware demonstrator.

5.2 Data Collection Layer

5.2.1 Overview

The data collection layer is responsible for capturing packet information from the mirror port on the switch and storing it an a format suitable for further processing and satisfies requirements R001 and R002. The relevant attributes in this context would be packet timestamps, time from previous packet (delta time), packet length (in bytes), source and destination IP addresses and source and destination UDP ports (if relevant). Furthermore, depending on the source of the packet (input controller or motor controller), the payload would be parsed accordingly, and be made an attribute of the packet.

5.2.2 Detailed Design

Packet Classification – Before starting development of the data collection layer, the system was allowed to run for 30 minutes to get a feel for the packets during normal operation. This meant letting the demonstrator run at the lowest and highest possible vehicle speeds (5 m/s - 30 m/s) with and without cruise control, at a constant speed, with low and high acceleration and deceleration and in a completely stationary state. The capture file was was then examined carefully to filter out the various kinds of packets being sent across the network. The main packets of interest from the demonstrator are the four UDP packets corresponding to vehicle speed, brake, throttle and cruise. There were several other protocols detected – SSDP packets (Simple Service Discovery Protocol), DHCP (Dynamic Host Configuration Protocol) and MDNS (Multicast DNS) packets, which were due to the fact that the computer connected to the mirror port was set to have an automatic IP configuration as opposed to the recommendation of a static one. This was done intentionally to account for the scenario that a future user of the application may not have a static IP address set when connecting to the demonstrator, and this may raise a false alarm of malicious packets in the network. There were also ARP (Address Resolution Packets) identified when either the input controller or the motor controller were not connected to the switch. Lastly, there was a broadcast protocol packet (0x8899) that was seen every second. This is in fact an RRCP (Realtek Remote Control Protocol) that is used by switches to detect network loops. According to Netgear, a network loop occurs when a network has more than one active path carrying information from the same source to the same destination [7]. The information loop amplifies itself using the additional path instead of stopping when it reaches the destination resulting in a slowdown of network traffic. Table 5 provides a summary of findings of the various protocols in use, and a classification column to denote the type of packet it was stored as during data collection.

The pitfall with the provided Lua dissectors was that they were only compatible with Wireshark and not with tshark. Despite being able to see the various throttle, speed, brake cruise payloads on Wireshark, manually exporting capture files each time is simply not an option in the context of an automated framework. To circumvent this roadblock, a solution comprising of two parts was implemented:

Classification	Protocol	Length (Bytes)	Src IP	Dst IP	Src Port	Dst Port
Speed	UDP	130	172.16.0.101	172.16.0.100	28091	28091
Throttle	UDP	122	172.16.0.100	172.16.0.101	28191	28191
Brake	UDP	122	172.16.0.100	172.16.0.101	28191	28191
Cruise	UDP	122	172.16.0.100	172.16.0.101	28191	28191
DHCP	DHCP	342	0.0.0.0	255.255.255.255	68	67
MDNS	MDNS	Various	169.254.226.217	224.0.0.251	5353	5353
	MDNS	Various	fe80:8ab:80dd:59bd:35c8	ff02:fb	5353	5353
SSDP	SSDP	217	Various	239.255.255.250	Various	1900
ARP	ARP	60	172.16.0.100	172.16.0.101	NA	NA
	ARP	60	172.16.0.101	172.16.0.100	NA	NA
Broadcast	0x8899	60	NA	NA	NA	NA

Table 5: Summary of protocols seen on the network.

Tshark Wrapper (Part 1) – A Python based tshark wrapper utility called Pyshark was used to access the packets contained in the capture files. The utility is open-source and maintained by the online community, and is publicly available on on GitHub [27]. Pyshark offers several methods that aid in the inspection of the layers in a packet and the attributes associated with each layer. This made extraction of the general packet attributes such as timestamps, IP addresses and UDP port numbers fairly straightforward. The caveat here was that the provided Lua dissectors would not work with Pyshark. A decision was made here to continue using Pyshark because of the ease of access to packet information and this led to the development of the second part of the data collector called the **payload dissector** which is described below.

Payload Dissector (Part 2) – The hexdumps of the controller packets were analysed in Wireshark, to obtain the byte positions which corresponded to the UDP payload. The bytes of the UDP payload were from byte 43 onwards till the end of the hexdump. This was 88 bytes for the throttle, cruise and brake packets and 96 bytes for the vehicle speed packets. After obtaining the byte field of the UDP payload, the type of packet (speed, throttle, brake or cruise) was determined based on the **sid** (Service ID) and **iid** (Interface ID? fields, with each of the four types of packets having a unique combination of the two fields. A visual representation of the four hexdumps is provided in Figure 11.

5.2.3 Performance

The criteria of performance was on equal footing with that of functionality, hence careful consideration was given to the choice of data structure into which the recorded information would be stored during data collection. There were three main contenders for this:

Approach 1: OOP – This involved creating a custom packet class containing instance variables (packet attributes), getter and setter methods and a counter as a class variable to record the number of packets being parsed. For every packet parsed, an instance of this custom packet class would be created with the appropriate attributes and appended to a native Python data structure such as a list or dictionary. The implementation can be viewed in the Github repository provided in the Appendix.

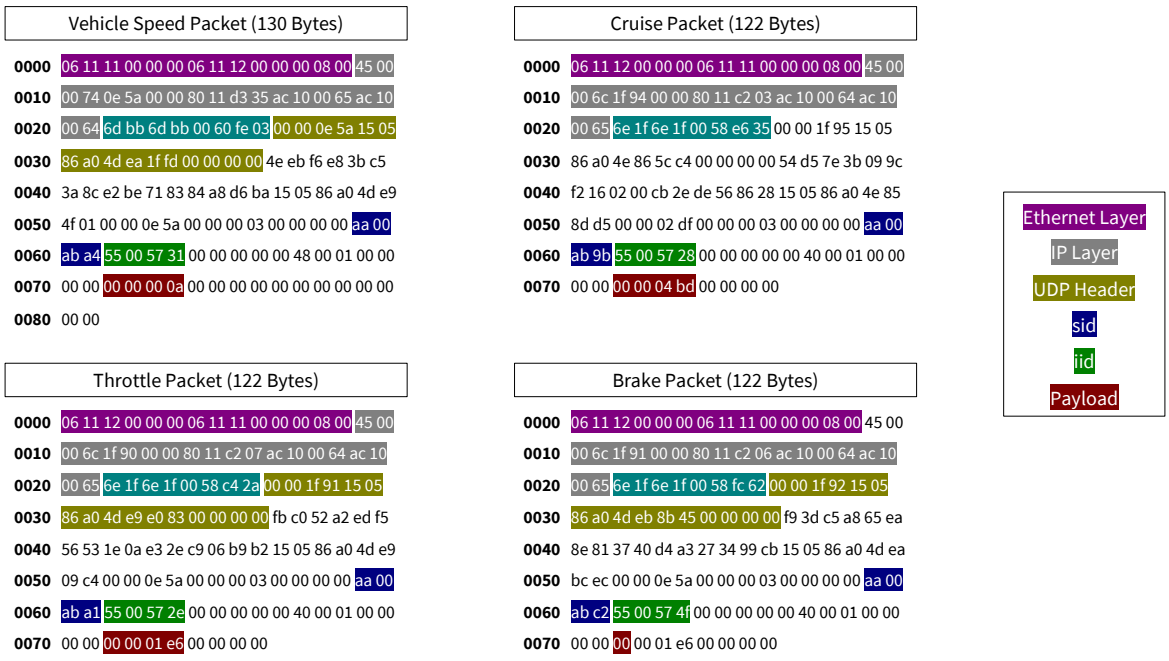


Figure 11: Hexdumps for the 4 types of packets. The areas of interest are the pid, sid and

Verdict – This approach was discarded early on because this solution felt over-engineered for the task and introduced unnecessary overhead into the application such as repeated packet object creation via the *__init__* method and the storage and retrieval of attributes via function calls. When scaled to hundreds of thousands of packets, these calls can become computationally expensive.

Approach 2: Nested Lists – Every packet is represented by a native Python list of strings, where each string would denote a packet attribute. Each 'packet list' was made into a Dataframe of one row and appended to a Pandas Dataframe containing all of the packets.

Verdict – This approach was initially the one chosen, however, it started to show its limitations when dealing with large capture files (∼100,000 packets). The parsing of the first 10,000 packets was fairly fast, taking approximately 50 seconds. Beyond the 10,000 packet mark, there was a considerable slowdown in performance of the data collector. The reason for this was when the `Dataframe.concat()` or `Dataframe.append()` method is called inside a **for** loop, the result is quadratic copying [36]. To elaborate further, `pandas.concat()` returns a new Dataframe each time it is called, which means memory needs to be allocated for the new Dataframe, and information from the old Dataframe would need to be copied into the newly allocated memory. For a capture file of n packets, using a **for** loop to iterate over the capture would result in $O(n^2)$ copies, justifying the enormous execution time. An append operation, in theory should be an $O(1)$ operation, but calling the `Dataframe.append()` method within the for loop resulted in similarly long execution times. As a result, the data collection layer was refactored to use the approach based on nested dictionaries instead which is described below.

Approach 3: Nested Dictionaries – Every packet is represented by a native Python dictionary, with the key being the name of the packet attribute, and the value being the value of the attribute

itself. The packet dictionary was then added to a master dictionary containing the other packet dictionary objects, with the key being the packet number (tracked by a counter within the data collection layer) and the value being the dictionary object of the packet being recorded. At the end of the capture file, the dictionary is exported to a Dataframe via the `pandas.from_dict()`, which is in turn exported as a CSV file.

Verdict – The only concern during implementation of this approach was if the order of insertion of the key-value pairs to the dictionary would be maintained during export to a Dataframe. This is a valid concern because a dictionary does not guarantee keeping track of order of insertion, and this would have resulted in a Dataframe with packet attributes not being in a fixed horizontal position. This concern was quickly resolved because since Python 3.6, dictionaries officially maintain order of insertion of key-value pairs, making this approach an extremely valid and optimised solution for data collection. To further justify that the nested dictionaries approach was the superior one, a benchmark was performed where data is collected from capture files of 10, 100, 1000, 10,000 and 100,000 packets, and the time for execution is recorded for the nested lists and nested dictionaries approaches and presented in Table 6.

No. of Packets	Approach 2 (List)	Approach 3 (Dict)
100	1.13234	0.81924
1000	3.43259	2.70842
10,000	49.84063	23.73492
100,000	377.23552	121.95863
200,000	649.81342	244.59186

Table 6: Benchmarks for packet parsing using a list approach versus a dict approach.

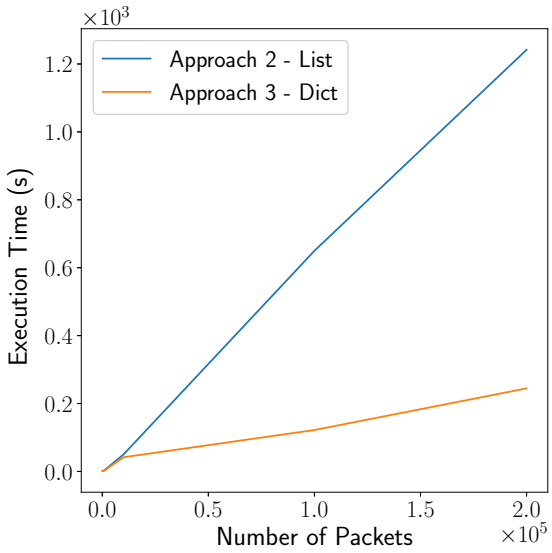


Figure 12: Execution times for increasing capture file size.

5.2.4 Output

The output of the data collection layer is a CSV file with the attributes shown in Table 7. CSV is a good choice of file format because data storage, retrieval and manipulation methods from the Pandas library are optimised to work with CSV files. Furthermore, CSV is one of the most common file format of choice when dealing with the various Python ML libraries. The extracted attributes and payloads were compared with the Wireshark output and graphically visualised to ensure that they were functioning as expected.

Attribute	Description
DateTime	The date and time of arrival of the packet, stored as a DateTime object.
Timestamp	The time (seconds) from when the capture started
PacketLength	The length of the packet in bytes.
TimeDelta	The time difference (seconds) between arrival of the current packet and the previous packet.
SourcePort	The UDP source port.
DestinationPort	The UDP destination port.
SourceIP	The source IP address.
DestinationIP	The destination IP address.
StatusType	The type of packet, according to the 'Classification' column in Table 5

Table 7: The recorded packet attributes of the data collection layer.

5.3 Feature Extraction Layer

5.3.1 Overview

Successful implementation of the data collection layer meant that packet information was now available in a format with which feature extraction could now be performed. At the point of completion of the execution of the data collection layer and just before the start of feature extraction, there is a list of packets conveying something different about the system at several timestamps. The purpose of the feature extraction layer is to combine these packets by aggregating small chunks of them to quantify the functioning state of the system at an average timestamp. Essentially, the goal is to perform data fusion, to produce useful information about the system that the individual packets, when used by themselves, would not have been able to. When implemented successfully, the feature extraction layer satisfies requirements R003 and R004.

5.3.2 Detailed Design

Time Window – The solution to extracting features from the output of the data collection layer was to split the data collected into chunks of packets, where each chunk was a predefined time

interval. Intuitively, a small time interval would aid in the detection of an anomaly a lot faster due to data being sampled at a higher frequency. However, the caveat with keeping the time interval small is that the computed features are filled with noise due to not enough packets being present to provide a smooth average. After trial and error, the value of 0.5s for the time window provided a sufficiently smooth signal for each of the protocols recorded on the network except for RRCP.

Network Throughputs – The throughput η, of a specified packet type with a size of x bytes, where N of these packets are recorded within a time interval δ, is defined as:

$$\eta = \frac{Nx}{\delta}$$

This was the formula used to calculate the throughputs of all of the identified packet types in Table 5.

Demonstrator Payloads – Within the same time interval δ, the speed, throttle, brake and cruise payloads were averaged.

The timestamps of the start and end packets of this chunk were averaged and added as the timestamp for this newly created entry. In addition to these four features, the network throughputs were computed for each of the identified packet types in Table 5 for each time interval.

Behaviour Extraction – A simple behaviour hierarchy is chosen with a the demonstrator, an entity, exhibiting the following three behaviours:

1. Stationary – when the vehicle speed $= 0$ and cruise control off
2. Moving – vehicle speed > 0 and cruise control off
3. Cruising – vehicle speed > 0 and cruise control on

Once these behaviours were assigned, training examples for each of these behaviours were fairly straight-forward to filter out.

5.3.3 Output

At the end of the feature extraction layer, there is now a aggregated time series of the features of the system, such as the speed, throttle, brake status and cruise control. Furthermore, the network state is also being monitored by computation of the throughputs of the various types of packets being sent across the network, which shall form the basis of the machine learning task in the research component of the project. The absolute average timestamp and the magnitude of the time window is also recorded.

5.4 Testing

5.4.1 Unit Testing

A test suite was built with the the *unittest* package which automates the written unit tests. The tests are run just before initialising the application, to ensure that all of the methods required for

core functioning of the feature collection software. The testing strategy was to split the unit tests into the following three classes:

1. **Packet Tests** – During initial inspection of the protocols on the network, small capture files were saved and these files are used as the basis for performing packet specific unit tests. An example of this would be to perform assertions on the number of layers (4), types of layers (Ethernet, IP, UDP, Data) and the packet sizes for a UDP packet (130 Bytes) from the input controller with a speed payload. Similar assertions are done for other such packets on the network and performed under a `PacketTests` class.

2. **Dissection Tests** – After ensuring that that all (known) packets on the network can be categorised, the next logical step would be to ensure that dissection of the custom UDP packets from the input controller and motor controller is functioning as expected. The `DissectorTests` class tests that the relevant payloads such as speed, throttle, brake and cruise values are being parsed correctly. Furthermore, extraction of the relevant bytes from the hexdumps for identification of the type of packet are assert tested.

3. **Utility Tests** – The `UtilityTests` class contains assertions for helper methods that were used across all of the layers. These were mainly assertions to check that file IO and methods returning DataFrames were behaving as expected. Typical assertions include checks that the attributes, dimensions and datatypes of the output dataframes of the methods contained within the collect and feature layers had the expected dimensions and attributes. The source code for the tests contained in the three described classes can be found in on the Github repository.

5.4.2 Integration Testing

The unit testing class ensured that the methods within the layers were functioning as they should. The next step was to combine the functionalities of the layers to ensure that the application can sniff packets being sent over the switch in realtime and extract the feature vector (throughputs) for the classifier to predict whether or not a DoS attack was to occur. There is no formal strategy here apart from importing the modules and calling the relevant methods to collect and extract a feature vector from the live packets being sent, and this is printed to the terminal or exported to a CSV for further inspection to check if the extracted features such as speed, brake, cruise and throttle are consistent with what is being recorded in Wireshark.

6 Experiment Execution

6.1 Training Data Collection

6.1.1 Baseline Data

Data was first collected to characterise baseline operation of the hardware demonstrator. There are two driving modes present in the system, which are cruise control on and cruise control off. Keeping this in mind, the following strategy for baseline data collection was executed:

1. With cruise control on, the demonstrator was allowed to run at intervals of 600 seconds, where in each interval the demonstrator was run at a constant speed. The speeds ranged from the minimum possible speed (7-8 units) up to the maximum (30 units). There is no unit for the speed because the calibration method for translating the rotational speed of the motor to what is being output as the "speed" from the motor controller is unclear.

2. With cruise control on, the demonstrator is accelerated by using the "set" button to increment the speed from the minimum speed to the maximum speed in a time interval of 120 seconds.

3. With cruise control on, the demonstrator is decelerated by using the "resume" button to decrement the speed from the maximum speed to the minimum speed in a time interval of 120 seconds.

4. With cruise control off, the demonstrator was allowed to run at intervals of 6 minutes, where in each interval the demonstrator was run at a constant speed. The speeds ranged from the minimum possible speed (5-6 units) up to the maximum (30 units).

5. With cruise control off, the demonstrator is accelerated by gradually increasing the throttle (turn the throttle knob clockwise) to increment the speed from the minimum speed to the maximum speed in a time interval of 60 seconds.

6. With cruise control off, the demonstrator is decelerated by gradually decreasing the throttle (turn the throttle knob counter-clockwise) to decrement the speed from the maximum speed to the minimum speed in a time interval of 120 seconds seconds.

7. With cruise control off, the demonstrator was allowed to stay at rest for 600 seconds at a speed of 0 units.

By the end of baseline data collection, approximately 2 million packets of data was collected, and after performing feature extraction resulted in 17,382 training examples to characterise baseline operation of the demonstrator, labeled as 0 (Normal), where every training example is a set of network protocol throughputs.

6.1.2 DoS Attack Data

The demonstrator was then attacked using a packet injection script (see Github Repository), where the input controller was flooded with a UDP packet every 0.5 seconds *with the key assumption being*

that the input controller is able to process malicious traffic injected at a rate slower than this. This is a valid assumption to make, since the process of finding the minimum rate of UDP flooding at which the input controller ceases to function is a time consuming task and deviates from the main aim of the study. Keeping in mind the assumption that the minimum packet flooding rate of 0.5s, a UDP flood was performed on the input controller up to the point that the input controller ceases to function, and this is identified by the increase in throughput of ARP packets on the network as the motor controller sends out ARP requests to locate the input controller. The UDP flood was performed and data recorded under the following three conditions:

1. With cruise control on and speed > 0.
2. With cruise control off and speed $= 0$.
3. With cruise control off and speed > 0.

The point at which the 'Attack' label was assigned to is when the throughput of the malicious packets was greater than 0. This resulted in 259 training examples to identify when a DoS attack has happened, and was labelled 1 (Attack).

6.2 Validation Set Data Collection

The network was subjected to a UDP flood at rates of 0.1s, 0.2s, 0.3s, 0.4s and 0.5s under a randomised mix of driving conditions. The data was collected and labeled using the same method as was done for the DoS Attack data collection above. This data was not used in the training of the model and used only for the purposes of testing. The validation set contained 93 test examples with the label 0 (Normal) and 368 test examples with the label 1 (Attack). The results for the tests are presented in Section 7.3.

7 Results

This section presents the results of the research methodology part of the project. The plots are all of the same format, with time on the x-axis, and throughput in bytes/second on the y-axis. The protocol for which the throughput is being measured is defined in the legend in the top right corner of each subplot. A definition of each subplot legend entry is presented:

1. **Network** - the total combined throughput of all protocols observed on the network.
2. **Speed** - the throughput of the UDP packets sent from the motor controller to the input controller, with the motor speed payload.
3. **Throttle** - the throughput of the UDP packet sent from the input controller to the motor controller, with the throttle payload.
4. **Brake** - the throughput of the UDP packet sent from the input controller to the motor controller, with the braking payload.
5. **Cruise** - the throughput of the UDP packet sent from the input controller to the motor controller, with the cruise control payload.
6. **RRCP** - the throughput of an RRCP packet broadcast from the ethernet switch, for the purposes of prevention of network loop detection.
7. **Malicious** - the throughput of malicious packets detected on the network that do not fit the description of any of the packets mentioned in this list.
8. **ARP** - the throughput of ARP packets sent from either the motor or input controller for the purposes of resolving a MAC address for a given IPv4 address.
9. **NBNS** - the throughput of a NetBIOS Naming Service packet sent from the device performing the DoS attack, similar to DNS and performs function of translating human readable names to IP addresses [5].
10. **LLMNR** - the throughput of a Link-Local Multicast Name Resolution packet sent from the device performing the DoS attack, used for name resolution as an alternative to DNS local link [34].
11. **Malformed** - the throughput malformed UDP packets sent from either the input or motor controller.

7.1 Baseline Operation

An example data collection interval is presented in Figure 13. Since this is during normal operation of the board with a malicious actor injecting packets, the throughput of the malicious packets, NBNS, LLMNR and ARP are recorded as being constantly 0. Additionally, the throughputs of the speed, brake, throttle and cruise packets remain fairly constant at their baseline values without any sudden drops or fluctuations.

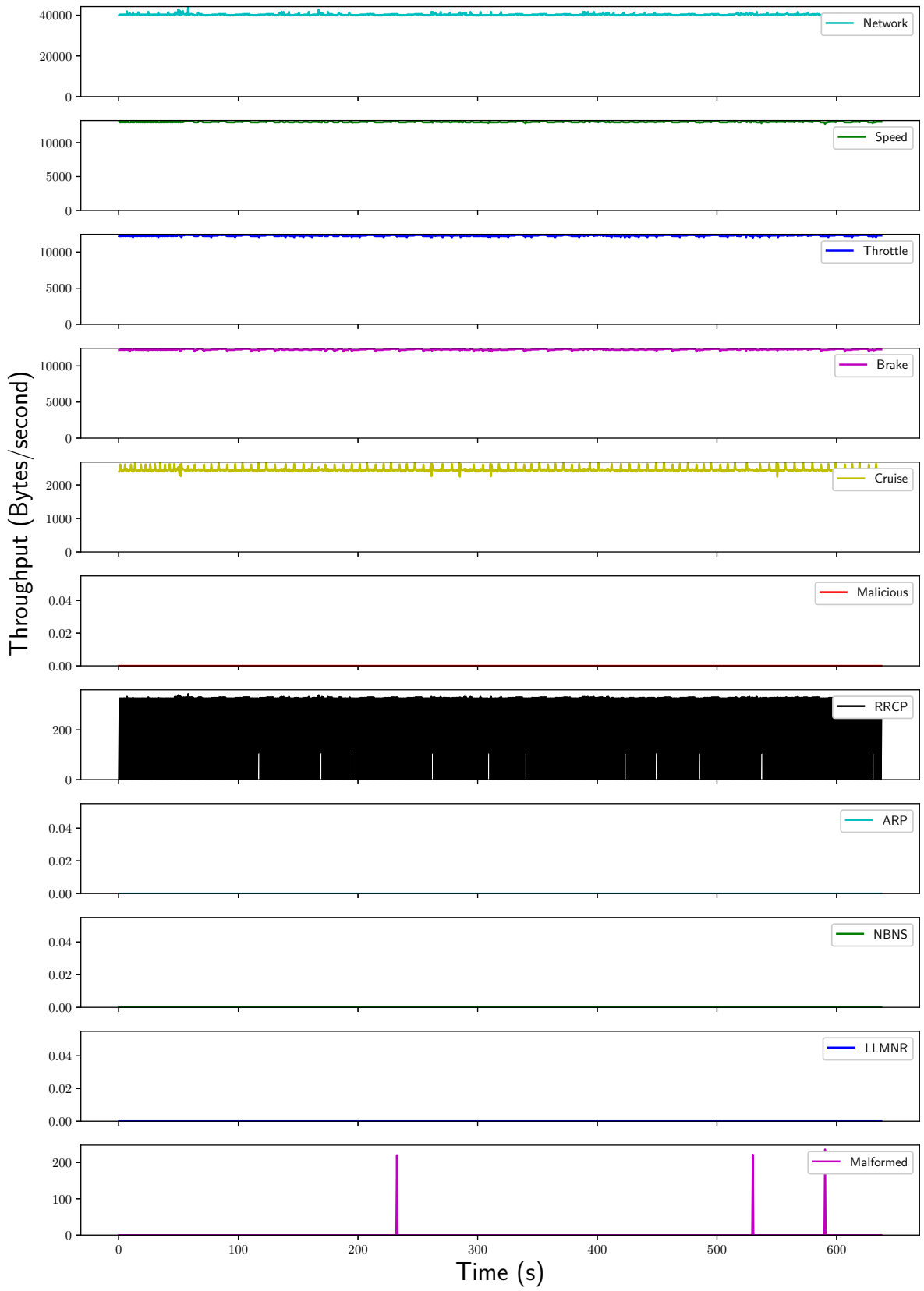


Figure 13: Network throughputs during baseline data collection.

During baseline data collection, there was a packet from the input controller and motor controller that kept getting flagged as malicious. Upon inspection, the packet had 5 layers, instead of the usual 4 layers found in a typical UDP packet from both controllers. According to the Wireshark documentation, this is a malformed packet, with its UDP payload corrupted. This can happen for a variety of reasons which include incorrect reassembly of packet, using the wrong dissector, or the packet is malformed [8]. Further investigation would be needed to find out the true cause of this.

The RRCP signal is quite noisy due to the fact that this packet is broadcast every second, whereas the rate of information capture was 0.5 seconds (for the purposes of DoS attack detection). This resulted in there periodically being a timestamp where there were no RRCP packets, which explains the erratic fluctuation in this signal.

7.2 DoS Attacks

The throughput results for DoS attacks performed at a rate of 0.5s under the three scenarios of [CC off, vehicle stationary], [CC off, vehicle moving] and [CC on, vehicle moving] are presented in Figures 14, 15 and 16. The packet injection attack is done on the input controller, which is responsible for sending the brake, throttle and cruise packets. The results for each of the attack scenarios are explained below.

7.2.1 CC off and Vehicle Stationary

Start of Packet Injection – The speed, throttle, brake and cruise packet throughputs remain fairly constant up until the point of the start of packet injection which occurs at roughly $t = 43$. There are spikes observed at around $t = 39$ in the NBNS and LLMNR throughputs right before the attack, but it is difficult to conclude if these spikes are related to the attack or not. They may have occurred because the device performing the attack was a Windows device, and with DNS being disabled on the switch, the NBNS and LLMNR may have been sent by the Windows machine for the purposes of name resolution, and this may be considered a reason for this spike. There is a spike in the throughput of ARP packets that can be attributed to the start of packet injection. The attack is being performed on the input controller by a malicious actor, and since the input controller does not have the MAC address of the malicious actor in its ARP table, it sends out an ARP request to resolve the MAC address of the malicious actor, which explains the spike in ARP throughput at the start of packet injection.

During Packet Injection – The malicious traffic is continuously injected, and at $t = 69$ there is a significant drop in network throughput with the major contributor to that drop being that the throttle, brake and cruise throughputs instantly drop to 0. This is because the input controller's computational resources are completely drained and it is unable to send any further packets. The complete drainage in the computational resources of the input controller is further evidenced by the increase in ARP throughput at $t = 78$, because the motor controller is sending out ARP requests to locate the input controller. An important metric to note is the time lag between the start of packet injection to the point of the input controller ceasing to function which is approximately 26 seconds.

After Packet Injection – The speed packets from the motor controller continue to maintain a steady throughput till $t = 140$, after which the speed packet throughput also rapidly drops down to 0, as it is unable to locate the input controller. An interesting observation is that the frequency of the RRCP signal decreases and the magnitude of the throughput nearly halves after the input controller ceases to function.

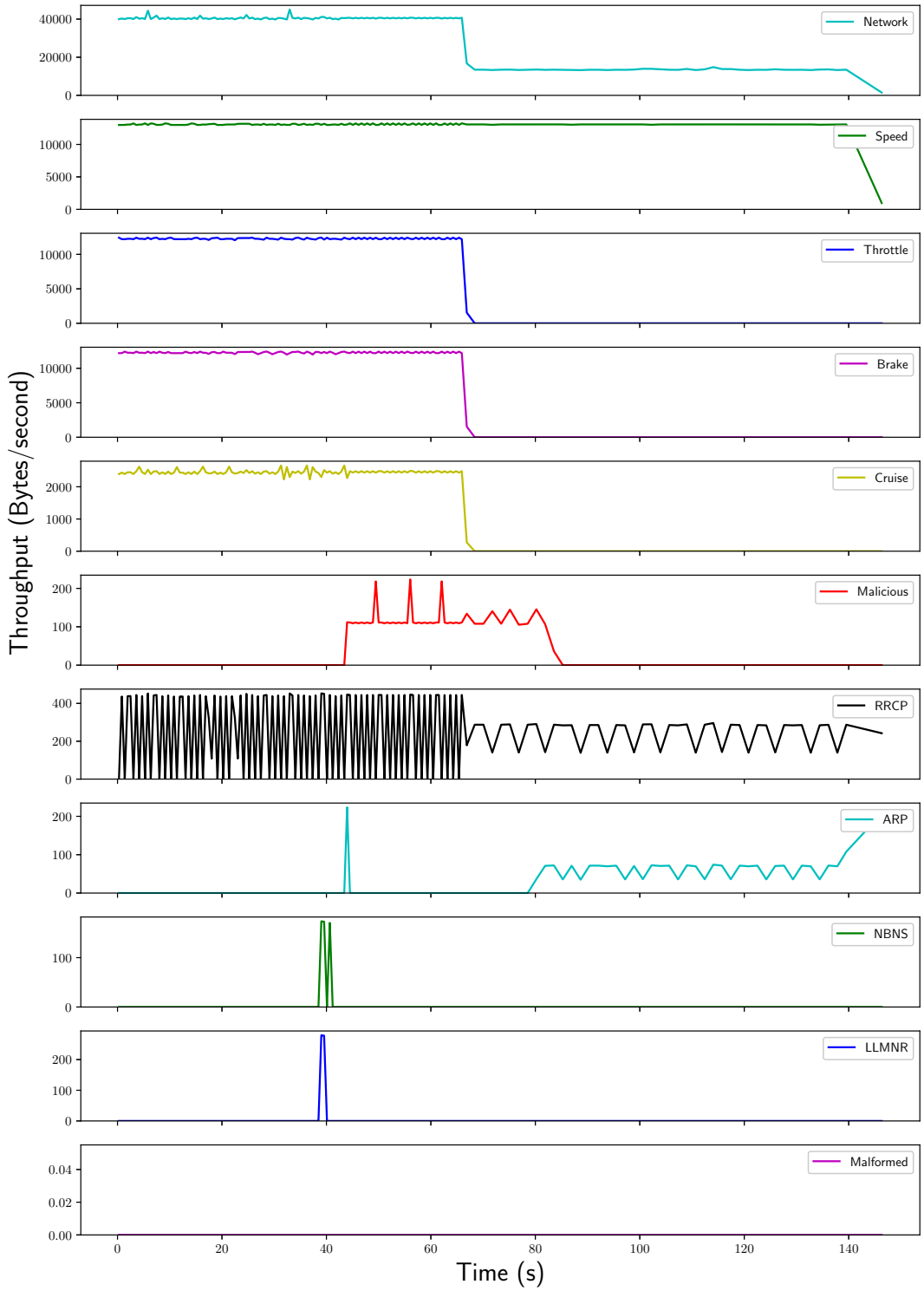


Figure 14: Network throughputs during DoS attack with conditions: CC off and vehicle stationary

7.2.2 CC off and Vehicle Moving

Start of Packet Injection – The speed, throttle, brake and cruise packet throughputs remain fairly constant up until the point of the start of packet injection which occurs at roughly $t = 20$. There are spikes observed at around $t = 22.5$ in the NBNS and LLMNR throughputs right after the attack, but once again it is difficult to conclude if these spikes are related to the attack because in the previous scenario these spikes occurred before the start of packet injection. There is a spike in the throughput of ARP packets at $t = 20$ that can be attributed to the start of packet injection, as the input controller tries to resolve the address of the malicious actor.

During Packet Injection – The malicious traffic is continuously injected, and at $t = 45$ there is a significant drop in network throughput because the input controller reaches its packet processing limits. At $t = 55$, approximately 10 seconds after the input controller ceases to function, the increase in ARP throughput is because the motor controller is sending out ARP requests to locate the input controller. The time lag between the start of packet injection to the point of the input controller ceasing to function is approximately 22.5 seconds.

After Packet Injection – The speed packets from the motor controller continue to maintain a steady throughput till $t = 115$ the speed packet throughput also rapidly drops down to 0, as it is unable to locate the input controller. A similar observation is seen in RRCP signal where the frequency decreases and the magnitude of the throughput also halves after the input controller ceases to function at around $t = 45$.

7.2.3 CC on and Vehicle Moving

Start of Packet Injection – The speed, throttle, brake and cruise packet throughputs remain fairly constant up until the point of the start of packet injection which occurs at roughly $t = 30$. There are no spikes observed for NBNS and LLMNR throughputs. There is a spike in the throughput of ARP packets that can be attributed to the start of packet injection at $t = 30$, as the input controller tries to resolve the address of the malicious actor.

During Packet Injection – The malicious traffic is continuously injected, and at $t = 53$ there is a significant drop in network throughput with because the input controller's computational resources are completely drained and it ceases to function. There is an increase in ARP throughput at $t = 67$, because the motor controller is sending out ARP requests to locate the input controller. The time lag between the start of packet injection to the point of the input controller ceasing to function is approximately 23 seconds.

After Packet Injection – The speed packets from the motor controller continue to maintain a steady throughput till $t = 127$, after which the speed packet throughput also rapidly drops down to 0, as it is unable to locate the input controller. An interesting observation is that the frequency of the RRCP signal decreases and the magnitude of the throughput nearly halves after the input controller ceases to function.

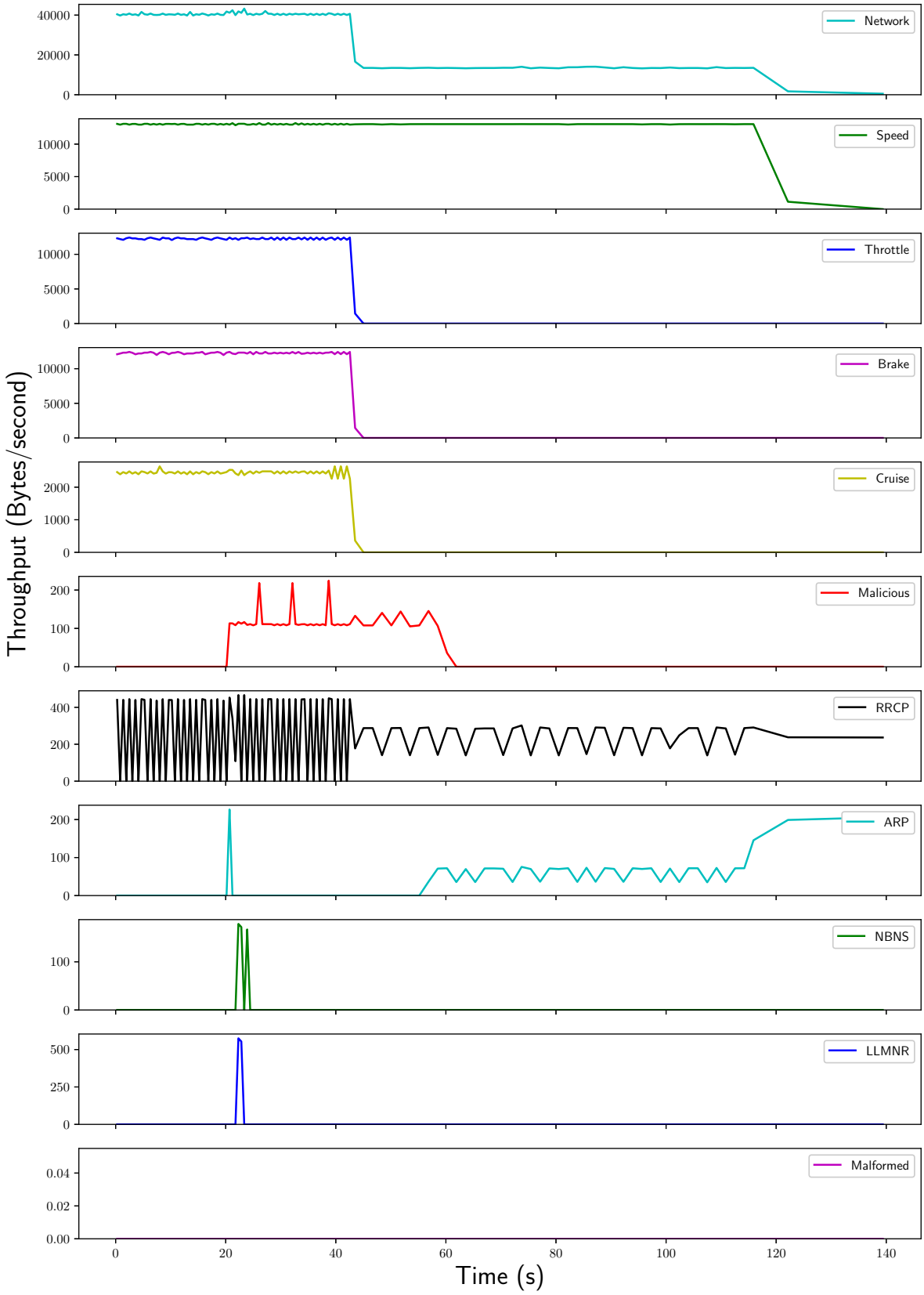
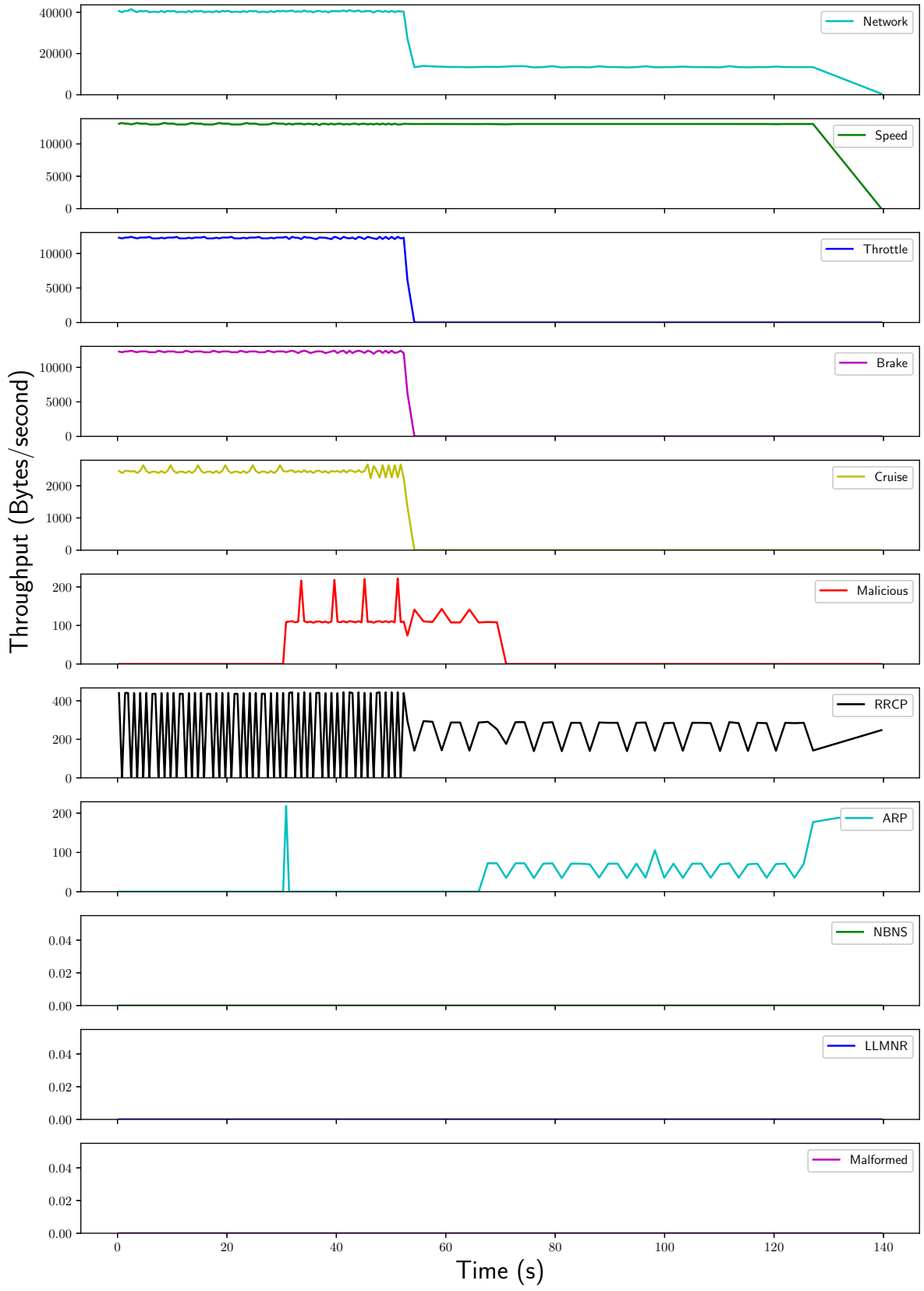


Figure 15: Network throughputs during DoS attack with conditions: CC off and vehicle moving

Figure 16: Network throughputs during DoS attack with conditions: CC on and vehicle moving



7.3 DoS Attack Detection Model

7.3.1 Choice of Protocols

Step 4 of the research methodology states that the protocols that exhibited a pattern when the DoS attacks were performed were to be included in the feature vector used in the training of the classification model. Based on the three attack scenarios observed in the previous section, these protocols are: Speed (UDP), Brake (UDP), Throttle (UDP), Cruise (UDP), RRCP and ARP. RRCP and ARP are chosen for the consistent patterns they exhibited during the three DoS attack scenarios.

7.3.2 General Performance

Table 8 presents an overview of the ability of the trained binary classifiers to predict if a DoS attack has occurred on the demonstrator. The classifiers are evaluated based on the metrics defined in the research methodology, which are execution time, precision, recall, F1-score and MCC. The training set consisted of 17,382 "Normal" training examples and 259 "Attack" training examples. The best performing model overall was the Naïve-Bayes classifier, which recorded the highest precision value, recall, F1-score and the lowest execution time (training time plus testing time). The testing set here is the validation dataset consisting of totally 368 attack examples and 93 normal examples, collected for packet injection rates ranging from 0.1 seconds to 0.5 seconds.

Classifier	Execution Time (s)	Precision	Recall	F1-score	MCC
Logistic Regression	0.082	1.000	0.954	0.946	0.898
Naïve Bayes	0.063	1.000	0.957	0.949	0.903
Linear SVM	0.078	1.000	0.954	0.946	0.898
Decision Tree	0.084	1.000	0.954	0.946	0.898
Random Forest	0.166	1.000	0.948	0.940	0.887
XGBoost	0.482	1.000	0.954	0.946	0.898

Table 8: Overall performance of the trained binary classifiers on the validation dataset.

7.3.3 Categorised by DoS Attack Rates

Table 9 presents the results for the same test and validation dataset, but the precision, recall, F1-scores and MCC are now calculated on the basis of the validation test set being split and categorised by packet injection rate.

DoS Attack Rate (s)	Classifier	Precision	Recall	F1-score	MCC
0.1	Logistic Regression	1.000	1.000	1.000	1.000
	Naïve Bayes	1.000	1.000	1.000	1.000
	Linear SVM	1.000	1.000	1.000	1.000
	Decision Tree	1.000	1.000	1.000	1.000
	Random Forest	1.000	0.964	0.972	0.945
	XGBoost	1.000	1.000	1.000	0.898
0.2	Logistic Regression	1.000	1.000	1.000	1.000
	Naïve Bayes	1.000	1.000	1.000	1.000
	Linear SVM	1.000	1.000	1.000	1.000
	Decision Tree	1.000	1.000	1.000	1.000
	Random Forest	1.000	1.000	1.000	1.000
	XGBoost	1.000	1.000	1.000	1.000
0.3	Logistic Regression	1.000	1.000	1.000	1.000
	Naïve Bayes	1.000	1.000	1.000	1.000
	Linear SVM	1.000	1.000	1.000	1.000
	Decision Tree	1.000	1.000	1.000	1.000
	Random Forest	1.000	1.000	1.000	1.000
	XGBoost	1.000	1.000	1.000	1.000
0.4	Logistic Regression	1.000	1.000	1.000	1.000
	Naïve Bayes	1.000	1.000	1.000	1.000
	Linear SVM	1.000	1.000	1.000	1.000
	Decision Tree	1.000	1.000	1.000	1.000
	Random Forest	1.000	1.000	1.000	1.000
	XGBoost	1.000	1.000	1.000	1.000
0.5	Logistic Regression	1.000	0.833	0.747	0.587
	Naïve Bayes	1.000	0.843	0.757	0.601
	Linear SVM	1.000	0.833	0.747	0.587
	Decision Tree	1.000	0.833	0.747	0.587
	Random Forest	1.000	0.833	0.747	0.587
	XGBoost	1.000	0.833	0.747	0.587

Table 9: Performance of the trained binary classifiers on test data categorised by packet injection rate.

8 Evaluation

This section discusses how the methodology and results contribute to the assessment of the suitability of a PoL inspired network monitoring approach in an automotive cyber security context. Moreover, the performance of the trained binary classifiers for the detection of DoS attacks are discussed. Furthermore, the objectives of the software development component of the project are evaluated. The CyRes methodology is discussed and is used to put the work of this project into the wider cyber security context.

8.1 DoS Attack Detection Model

Referring back to Table 8, the overall computed metrics indicate strong predictive capabilities of the trained binary classification models. All of the trained models showed a precision of 1, meaning that the classifier predicted "Normal" when the demonstrator was functioning as normal. This is a plausible result because of the abundance of baseline data that was used to train the classifier. The recall scores were in the ranges of 0.948 – 0.954, which is a strong indicator of the classifier's ability to predict that an attack is happening, as a proportion of all of the instances that an attack is actually happening. Furthermore, the F1-scores of the minority class (Attack) and the computed MCC between the attack labels and predictions are in statistically strong regions, which confirm that under the operating conditions of the hardware demonstrator, a binary classifier that accepts a set of network protocol throughputs, is an acceptable predictive tool for the detection of a DoS attack.

However, one could argue that the computed metrics in Table 8 are misleading and Table 9 explains why this is the case. By computing the metrics based on test data categorised by packet injection rates, it is observed that the test data between 0.1 seconds up to a packet injection rate of 0.4 seconds, the precision, recall and F1 scores are 1.0, meaning that the DoS attack is almost always detected. However, this is not the case with an injection rate of 0.5 seconds. The average recall was 0.84, which means that 16% of attack instances were missed. In the context of a security application, a false negative classification can prove to be quite costly, as it has the potential to cost one time recording window at the very least. Furthermore, the F1 and MCC scores for the minority class were 0.747 and 0.587. Once again, in the context of an attack detection application, these scores wi A potential reason as to why this might have happened is because the time window during which information was recorded was close to 0.5 seconds. Perhaps this might have not produced a consistent pattern in the recorded data that the algorithm could have picked up, potentially explaining the subpar performance at injection rates of 0.5 seconds.

The task of trying to distinguish between extremely specific patterns in attack data and an abundance of baseline training data created a significant imbalance in the training dataset, with the imbalance ratio being almost 1:67. There was a possibility to oversample the minority class (Attack) or undersample the majority class (Normal). However this was decided against for a multitude of reasons. Firstly, training times and recall values are at an acceptable level, suitable for DoS attack detection, and the introduced computational overhead from resampling the two classes does not

outweigh the potential gain in recall. Secondly, by oversampling, there is a possibility that "unrealistic" samples are created that might affect the predictive abilities of the classifier negatively. Moreover, by undersampling, one always runs the risk of losing potentially useful features about the system. This goes against the PoL methodology as well, where one of the core foundations of the methodology is to capture everything that occurs in the life of an entity, and use it as the basis of a prediction.

Going back to the hypothesis, it was stated that the throughputs of some of the identified protocols on the network made it possible for detection of a DoS attack. An observation noticed in all of the performed DoS attacks was the spike in ARP throughput. Another significant pattern detected during the DoS attack was the frequency and amplitude of the RRCP protocol throughput signal. A recommendation for a future implementation could be to potentially just monitor the ARP and RCCP throughputs, where the distinct DoS attack induced patterns could be identified and flagged as "Attack" behaviour.

8.2 PoL Framework Relevance

Keeping in mind the defence and surveillance roots of the PoL framework, a typical implementation in an autonomous vehicle would be to split the framework into two distinct components.

1. **Background Component** – this component would be responsible for automated collection of baseline network throughput data for the vehicle. This baseline data is used along with pre labelled attack data (potentially performed in an OEM Research Center and sent over 5G/OTA) to train a binary classifier, similar to the research done here. Network throughputs have the potential to exhibit dynamic behaviour, and may change periodically. For example due to a firmware update in the car, a higher throughput of protocol X now occurs because of a new software feature implemented in one of the internal subsystems. For this reason, a periodic redefinition of what a "normal" is, is needed to account for these dynamics.

2. **Monitoring Component** – this component would be responsible for monitoring the throughputs of the various protocols on the network, and a combination of the throughputs of certain protocols can be fed into classifier to identify what behaviour is being exhibited (Normal or Attack).

The frequency of automated collection of baseline data is user defined, but for a system like this one, approximately 120 minutes of baseline data was sufficient to detect early onset of a Dos attack on the demonstrator. Needless to say, careful consideration would need to be given to optimise the two major elements of this process - the amount of baseline data collected and the frequency of collection. Collecting data infrequently may result in an inaccurate representation of what "normal" is in a system. On the other hand, assuming an optimal collection frequency, collecting too little data would negatively affect the accuracy of detection of any malicious activity, as the system would struggle to differentiate between normal and anomalous. A fairly obvious solution to this would be to collect more data more frequently, but there are computational and storage costs that come along with this approach, and collecting more data more frequently than necessary would not be an optimal use of resources. On the other hand, collecting too little data not infrequently could potentially result in attacks being missed and not being detected quickly enough. The optimal choice of the

frequency of data collection and the amount of data being collected would definitely need to be kept in mind as a design criterion in future implementations.

The "attack" behaviour in the model is characterised by manually attacking the demonstrator using UDP flood attacks, and the recorded data then being labelled as "Attack". This is one of the limitations of the model, because this is a manual task that would be need to performed by an individual with domain specific knowledge. This violates the automation principle of the PoL processing framework. The are alternative approaches that to circumvent this and is to use unsupervised learning algorithms such as one class SVMs. This is an algorithm where instead of a hyperplane being used to differentiate between the two classes as is the case of SVMs, a hypersphere is used to encompass the provided training data, or more formally speaking, the representation of what "normal" is. The idea is to then minimise the size of this hypersphere, and data points lying outside the hypersphere are classified as anomalous.

In theory, an unsupervised learning approach would seem almost perfect for anomaly/novelty detection in the case of the hardware demonstrator, but this is only under the assumption that the system has truly seen *everything* that fits the definition of "normal". One can never be truly sure that this is the case, and an element of doubt would always persist in the minds of the system developers. Moreover, it is quite difficult to evaluate the performance of unsupervised learning algorithms, because the metrics used to evaluate their performance would be based on the network patterns induced by known attacks. Strictly speaking, for a truly holistic assessment of the ability of an unsupervised learning system, one would need to check if the system is capable of detecting *unknown* network patterns both in the past and in the future. The information in the past, is well, in the past and there is no access to the information in the future. Therefore, a purely unsupervised approach is simply not feasible and this issue presents itself as a significant limitation.

A more practical approach to the problem would be to combine the two learning methods in a sort of semi-supervised learning approach. The idea is to first train classifiers capable of recognising known attack-induced patterns in data. After this has been accomplished, the collected patterns can be fed into the unsupervised learning algorithm along with an abundance of normal baseline data, and any outliers outside of this, would signify a true anomaly that is worthy of review. The scope of such an implementation is definitely out of the question for this project in the given timeframe, but it certainly can form the basis of future work.

8.3 Data Collection Tool

Prior to development of the data collection layer, the only way to extract information from the demonstrator was to manually save a capture file using Wireshark, export it to a CSV file, and import it into Python for further processing. With the help of pyshark, and some custom written parsing methods, the data collection tool certainly adds value to the hardware demonstrator as it provides future users with the ability to capture (and view) the data from the board in realtime or as a background process, where the capture can be stored away and retrieved later for further processing.

However, there is one pitfall with this tool, which is that a major part of its functionality is highly dependent on a python package maintained by an independent developer, who may or may not maintain it in the future. For the purposes of research, this is an acceptable risk. Part of this risk is mitigated by carrying out the work in a virtual environment, with known package versions. However, in a professional development setting, one would ideally want to eliminate the dependence on a library from an unknown developer and build one in-house instead.

8.4 Generated Dataset

In hindsight, the development of the dissectors for the data collection tool to extract the custom UDP payloads from the demonstrator was imperative to generate a feature-rich dataset of significant value on which machine learning could be performed. Without this, it would have been been a fairly standard dataset that could have been acquired from a base installation of Wireshark, and it would not have been worthy of classification as a contribution to the community. The baseline data pre-feature extraction and post-feature extraction are publicly available on GitHub. By doing so, it provides the opportunity to members of the ML community who do not have access to the hardware to perform their own machine learning implementations on the dataset.

8.5 CyRes Methodology

CyRes is a cyber resilience methodology, that relies on the Detect-Understand-Act cycle, to reduce the probability of a random cyber attack having catastrophic consequences. It is an extension to the original engineering V-model, using newer, more innovative techniques (such as monitoring, real time simulation, and planning and mitigation techniques) to accommodate the modern systems in the transport industry. While the V-model is a very useful tool in the verification and validation of an old car, new levels of connectivity and automation calls for an update to these techniques. The basis of the methodology is that "to avoid a catastrophic outcome, a cyber threat must be identified, analysed with respect to its potential impact and understood sufficiently well so that effective countermeasures can be developed and deployed before the point of catastrophe." In the context of the Detect-Understand-Act cycle, the proposed framework hopes to contribute to the the Detect-Understand part of this, because there is a detection of a cyber attack that is happening and there is an understanding from the pattern identified in the ARP and RRCP throughputs that a DoS attack is indeed taking place.

9 Conclusion

9.1 Future Work

The data collection and feature extraction tools are publicly available on GitHub with build instructions and the documentation required for initial setup. This would certainly facilitate seamless adoption of the existing state of the repository for the purposes of future work. The layered approach during the development of these tools means that in the event that there is a more optimised approach, to say perform data collection from the hardware demonstrator, this could be implemented without affecting the rest of the codebase as long as the output of the data collection layer is a CSV with the same attributes and dimensions as the previous implementation.

There are multiple directions that that this project can taken into. One possible direction would be a PoL implementation using unsupervised algorithms such as Isolation Forest or One Class SVM for detection of anomalous behaviour. This would be an an implementation of immense value, since it will enhance the automation element of the PoL framework.

A limitation of the current project was that the patterns induced by DoS attacks were the only ones studied. It would be useful to look at other attack types such as ARP spoofing and characterise the data patterns that it induces when performed.

The trained binary classifiers of this project were done with the base implementations with the default parameters. A recommendation for further work would be to consider the hyperparameter tuning of each model, and reevaluating the performance of the binary classifier on the dataset.

9.2 Contributions

A literature review was performed which summarises the key findings of Pattern of Life studies, on which literature is not very readily available. The project then implemented a Pattern of life inspired network monitoring framework for the detection of DoS attacks, that aims to fill the lack of cyber security considerations in the automotive industry.

The dataset generated for definition of the baseline operation of the hardware demonstrator is publicly available online on GitHub for members of the community that do not have access to the hardware. They now have the opportunity to perform their own learning algorithms on the feature-rich dataset. There are two distinct datasets available – the first is time series of packets. And the second dataset is the feature extraction. Future users will have the option of performing their own data aggregation techniques if need be.

The DoS attack detection task was implemented as a binary classification task, where a set of features in the form of network throughputs were mapped to a 0 (Normal) or 1 (Attack). The hypothesis that a DoS attack can be detected by using a carefully chosen feature vector of protocol throughputs was proven to be true by training a supervised binary classifier capable of predicting this.

The algorithm with the lowest training and classification time was the Naïve Bayes classification algorithm with a combined execution time of 0.063 seconds, recall of 0.954, F1-score 0.949 and MCC of 0.903.

Distinct patterns were detected in ARP protocol packets and RRCP packets sent over the switch when a DoS attack was performed.

The development of the data collection and feature extraction tools can be used by future users of the hardware demonstrator for their research pursuits.

References

[1] Car hacking: The definitive source. URL: http://illmatics.com/carhacking.html.

[2] Cybersecurity information for individuals & families. URL: https://www.ncsc.gov.uk/section/information-for/individuals-families.

[3] Global cybersecurity spending predicted to exceed $1 trillion from 2017-2021. Section: Cybersecurity Market Report. URL: https://cybersecurityventures.com/cybersecurity-market-report/.

[4] KU leuven researchers demonstrate serious flaws in tesla model x keyless entry system – news. URL: https://nieuws.kuleuven.be/en/content/2020/ku-leuven-researchers-demonstrate-serious-flaws-in-tesla-model-x-keyless-entry-system.

[5] NetBIOS/NBNS - the wireshark wiki. URL: https://wiki.wireshark.org/NetBIOS/NBNS.

[6] A single automotive cyber security standard is coming at last. URL: https://www.thalesgroup.com/en/worldwide-digital-identity-and-security/iot/magazine/single-automotive-cyber-security-standard.

[7] start [OpenRRCP]. URL: http://openrrcp.org.ru/doku.php.

[8] Wireshark . Appendix a. wireshark messages. URL: https://www.wireshark.org/docs/wsug_html_chunked/AppMessages.html.

[9] Francisco J. Aparicio-Navarro, Jonathon A. Chambers, Konstantinos Kyriakopoulos, Yu Gong, and David Parish. Using the pattern-of-life in networks to improve the effectiveness of intrusion detection systems. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE. URL: http://ieeexplore.ieee.org/document/7997374/, doi:10.1109/ICC.2017.7997374.

[10] Fabio Arena and Giovanni Pau. An overview of vehicular communications. 11(2):27. URL: http://www.mdpi.com/1999-5903/11/2/27, doi:10.3390/fi11020027.

[11] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus A. F. Andersen, and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. 16(5):412–424. _eprint: https://academic.oup.com/bioinformatics/article-pdf/16/5/412/476945/160412.pdf. doi:10.1093/bioinformatics/16.5.412.

[12] Manuele Bicego and Francisco Escolano. On learning random forests for random forest-clustering. pages 3451–3458, 01 2021. doi:10.1109/ICPR48806.2021.9412014.

[13] P. Biltgen and S. Ryan. *Activity-Based Intelligence: Principles and Applications*. Artech House electronic warfare library. Artech House. URL: https://books.google.co.uk/books?id=KGyPCwAAQBAJ.

[14] Jennifer Bruton. *CANBusSecurity Thesis2014 Final JBruton Public*. PhD thesis, 09 2014. doi:10.13140/RG.2.2.22154.93127.

[15] Peter Butka, Peter Bednár, and Juliana Ivančáková. Chapter 1 - methodologies for knowledge discovery processes in context of AstroGeoInformatics. In Petr Škoda and Fathalrahman Adam, editors, *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, pages 1–20. Elsevier. URL: https://www.sciencedirect.com/science/article/pii/B9780128191545000102, doi:https://doi.org/10.1016/B978-0-12-819154-5.00010-2.

[16] Krzysztof J. Cios, Roman W. Swiniarski, Witold Pedrycz, and Lukasz A. Kurgan. The knowledge discovery process. In *Data Mining: A Knowledge Discovery Approach*, pages 9–24. Springer US. doi:10.1007/978-0-387-36795-8_2.

[17] Louis Columbus. Cybersecurity spending to reach $123b in 2020. Section: Enterprise Tech. URL: https://www.forbes.com/sites/louiscolumbus/2020/08/09/cybersecurity-spending-to-reach-123b-in-2020/.

[18] Rachel Craddock, Doug Watson, and William Saunders. Generic pattern of life and behaviour analysis. In *2016 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, pages 152–158. IEEE. URL: http://ieeexplore.ieee.org/document/7497803/, doi:10.1109/COGSIMA.2016.7497803.

[19] Kaeye Dastner, Bastian von Hasler zu Roseneckh-Kohler, Felix Opitz, Michael Rottmaier, and Elke Schmid. Machine learning techniques for enhancing maritime surveillance based on GMTI radar and AIS. In *2018 19th International Radar Symposium (IRS)*, pages 1–10. IEEE. URL: https://ieeexplore.ieee.org/document/8447961/, doi:10.23919/IRS.2018.8447961.

[20] Steven Douglas. The state and future of geoint 2017 build the community. 01 2018.

[21] L. Erhan, M. Ndubuaku, M. Di Mauro, W. Song, M. Chen, G. Fortino, O. Bagdasar, and A. Liotta. Smart anomaly detection in sensor systems: A multi-perspective review. 67:64–79. URL: http://arxiv.org/abs/2010.14946, arXiv:2010.14946, doi:10.1016/j.inffus.2020.10.001.

[22] J T Folsom-Kovarik, Sae Schatz, Randolph M Jones, Kathleen Bartlett, and Robert E Wray. Scalable models for patterns of life. page 3.

[23] Eibe Frank, Leonard Trigg, Geoffrey Holmes, and Ian H Witten. Technical note: Naive bayes for regression. page 21.

[24] Morgan R. Frank, Lewis Mitchell, Peter Sheridan Dodds, and Christopher M. Danforth. Happiness and the patterns of life: A study of geolocated tweets. 3(1):2625. URL: http://www.nature.com/articles/srep02625, doi:10.1038/srep02625.

[25] Nina Franz. Targeted killing and pattern-of-life analysis: weaponised media. 39(1):111–121. URL: http://journals.sagepub.com/doi/10.1177/0163443716673896, doi:10.1177/0163443716673896.

[26] S. Ghosh and Society of Automotive Engineers. *Automotive Cybersecurity: From Perceived Threat to Stark Reality*. Electronic publications. Society of Automotive Engineers. URL: https://books.google.co.uk/books?id=EAcoswEACAAJ.

[27] Dor Green. pyshark. original-date: 2013-12-28T14:38:22Z. URL: https://github.com/KimiNewt/pyshark.

[28] Geoff A Gross, Eric Little, Ben Park, James Llinas, and Rakesh Nagi. Application of multi-level fusion for pattern of life analysis. page 8.

[29] Krishna Kant. Advanced persistent threats in autonomous driving. *SIGMETRICS Perform. Eval. Rev.*, 47(4):25–28, April 2020. doi:10.1145/3397776.3397783.

[30] Lance Whitney in Security on May 17, 2021, and 9:59 Am Pst. Cybersecurity spending report for 2019. URL: https://www.techrepublic.com/article/cybersecurity-spending-to-hit-150-billion-this-year/.

[31] J. Rossouw van der Merwe, Xabier Zubizarreta, Ivana Lukčin, Alexander Rügamer, and Wolfgang Felber. Classification of spoofing attack types. In *2018 European Navigation Conference (ENC)*, pages 91–99, 2018. doi:10.1109/EURONAV.2018.8433227.

[32] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 12 2013. doi:10.3389/fnbot.2013.00021.

[33] T. Nolte, H. Hansson, and L. lo Bello. Automotive communications - past, current and future. In *2005 IEEE Conference on Emerging Technologies and Factory Automation*, volume 1, pages 985–992. IEEE. URL: http://ieeexplore.ieee.org/document/1612631/, doi:10.1109/ETFA.2005.1612631.

[34] openspecs office. [MS-LLMNRP]: Overview. URL: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-llmnrp/6806998e-034d-4c39-8398-5af8bfd52d7e.

[35] Felix Opitz and Kaeye Dästner. Data analytics and machine learning based on trajectories. page 12.

[36] Stack Overflow. Python - why does concatenation of dataframes get exponentially slower? URL: https://stackoverflow.com/questions/36489576/why-does-concatenation-of-dataframes-get-exponentially-slower.

[37] Simon Parkinson, Paul Ward, Kyle Wilson, and Jonathan Miller. Cyber threats facing autonomous and connected vehicles: Future challenges. 18(11):2898–2915. URL: http://ieeexplore.ieee.org/document/7872388/, doi:10.1109/TITS.2017.2665968.

[38] Derek A. Pisner and David M. Schnyer. Chapter 6 - support vector machine. In Andrea Mechelli and Sandra Vieira, editors, *Machine Learning*, pages 101–121. Academic Press. URL: https://www.sciencedirect.com/science/article/pii/B9780128157398000067, doi:https://doi.org/10.1016/B978-0-12-815739-8.00006-7.

[39] Lior Rokach and Oded Maimon. Decision trees. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 165–192. Springer US. doi:10.1007/0-387-25465-X_9.

[40] Martin Salfer and Claudia Eckert. Attack surface and vulnerability assessment of automotive electronic control units. pages 317–326, 01 2015. doi:10.5220/0005550003170326.

[41] Michele Scalas and Giorgio Giacinto. Automotive cybersecurity: Foundations for next-generation vehicles. In *2019 2nd International Conference on new Trends in Computing Sciences (ICTCS)*, pages 1–6. IEEE. URL: https://ieeexplore.ieee.org/document/8923077/, doi:10.1109/ICTCS.2019.8923077.

[42] Barry Sheehan, Finbarr Murphy, Martin Mullins, and Cian Ryan. Connected and autonomous vehicles: A cyber-risk classification framework. *Transportation Research Part A: Policy and Practice*, 124:523–536, 2019. URL: https://www.sciencedirect.com/science/article/pii/S096585641830555X, doi:https://doi.org/10.1016/j.tra.2018.06.033.

[43] Anastasia Shmyglya. Breaking the brakes: Spoofing and denial of service attacks for safety critical vehicle components.

[44] Barry G Silverman, Gnana Bharathy, and Nathan Weyer. What is a good pattern of life model? guidance for simulations. 95(8):693–706. URL: http://journals.sagepub.com/doi/10.1177/0037549718795040, doi:10.1177/0037549718795040.

[45] Agachai Sumalee and Hung Wai Ho. Smarter and more connected: Future intelligent transportation system. *IATSS Research*, 42(2):67–71, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0386111218300396, doi:https://doi.org/10.1016/j.iatssr.2018.05.005.

[46] Agnieszka Szmelter-Jarosz. The concepts of connected car and internet of cars and their impact on future people mobility. *Information System in Management*, 6:234–245, 09 2017. doi:10.22630/ISIM.2017.6.3.7.

[47] Nikhil Tripathi and Babu Mehtre. Dos and ddos attacks: Impact, analysis and countermeasures. pages 1–6, 12 2013.

[48] Raymond E. Wright. Logistic regression. In *Reading and understanding multivariate statistics*, pages 217–244. American Psychological Association.

A GitHub Repository

All of the source code and datasets can be found under this link: `https://github.com/keane-fernandes/python-pol-ml-cybersecurity`.