TÉCNICO LISBOA

An edge-based smart network monitoring system for the Internet of Vehicles

Pedro Manuel Augusto Ferreira

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Paulo Rogério Barreiros D'Almeida Pereira
Prof. Naércio David Pedro Magaia

Examination Committee

Chairperson: Prof.a Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Prof. Paulo Rogério Barreiros D'Almeida Pereira
Member of the Committee: Prof. Miguel Nuno Dias Alves Pupo Correia

January 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to thank my thesis supervisors, Professor Paulo Rogério Pereira and Professor Naercio Magaia, for their advice, guidance, encouragement, availability and shared knowledge that has made this thesis possible. I would also like to express my thankfulness to them for challenging me during the realisation of this project, they made me grow as a student, as a professional and as a person.

I would also like to thank all my family and friends for helping me in this journey. Thank you for your support and for always being there for me.

To each and every one of you. Thank you.

Abstract

The Internet of Vehicles (IoV) is the future of transportation. It will be present everywhere and will have a huge impact on our lives. Notwithstanding, there are plenty of aspects to take into consideration while studying these networks, such as: data dissemination, cybersecurity threats and vulnerabilities. For the network to work efficiently, data has to be able to spread through the network efficiently, therefore, to tackle the data dissemination problem, a cluster-based routing algorithm was developed, R-privo. Which, uses a machine learning clustering algorithm as well as a routing algorithm based on social relationships between nodes. The R-privo, as shown by its performance in simulations, obtains high delivery rates with low overhead. Besides, due to the amount of data needed for the IoV, it also requires a good cooperation between nodes. In this sense, a misbehaving node might have a huge impact on network performance. In light of the above, a deep learning based monitoring system that is capable of detecting anomalies in the network and identify known misbehaviours was implemented. The monitoring algorithm is capable of identifying the sybil attack and the id spoofing attack with a high success rate, 68% and 95% for the sybil respectively. Besides, the algorithm was also built capable of detecting other misbehaviours without labelling them.

Keywords

Internet of Vehicles, Machine Learning, Clustering, Deep Learning, Network Monitoring

Resumo

A Internet de veículos será o futuro do transporte, vai estar presente em tudo à nossa volta e terá um impacto tremendo nas nossas vidas. No entanto, existem vários aspetos que têm que ser tidos em conta no estudo destas redes, tais como: disseminação de dados e ciber ameaças e vulnerabilidades. Para que esta tecnologia funcione da melhor maneira possível, é necessário que haja um bom mecanismo para disseminação de dados. Com este objetivo, um algoritmo de encaminhamento baseado em clusters foi desenvolvido, chamado R-privo. Este algoritmo recorre a aprendizagem automática para fazer o clustering com base em relações entre nós. O R-privo em comparação com algoritmos tradicionais, obtém uma elevada taxa de entrega com um overhead baixo. Para além disto, as IoV requerem uma grande cooperação entre nós. Neste sentido, um nó com um comportamento incorreto poderá ter um impacto em toda a rede. Com base nisto, uma abordagem em deep learning foi desenvolvida para detetar e identificar qualquer anomalia. Neste projeto foram desenvolvidos dois algoritmos baseados em relações entre nós. O sistema de monitorização é capaz de identificar o ataque sybil, o id spoofing com uma taxa de sucesso de 68% e 95% respetivamente. Para além disso, o algoritmo foi desenvolvido para também ser capaz de detetar desvio ao normal comportamento na rede.

Palavras Chave

Internet de Veículos, Aprendizagem Automática, Cluster, Aprendizagem Profunda, Monitorização de redes

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Networks
CH	Cluster Head
DL	Deep Learning
DL4J	Deep Learning for Java
DTN	Delay Tolerant Network
HMM	Hidden Markov Model
IM	Interface Manager
IoT	Internet of Things
IoV	Internet of Vehicles
IDS	Intrusion Detection System
MTTE	Mean Time to Encounter
MEC	Mobile Edge Computing
ONE	Opportunistic Network Simulator
PM	Protocol Mapping
ReLU	Rectified Linear Unit
RH	Request Handler
RSU	Road Side Unit
R-privo	RSU based privo
SAM	Security Analysis Module
SPM	Security Profile Manager
SSM	Security Simulation Module

SVM	Support Vector Machine
UI	User Interface
V2X	Vehicle to Everything
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
VANET	Vehicular Ad-hoc NETwork
VEC	Vehicular Edge Computing

1

Introduction

Contents

This chapter provides an overview of the concept of the Internet of Vehicles (IoV) and the motivation behind monitoring the system using an edge-based approach. Furthermore, the scope of this thesis is explicitly stated and the structure of the document is outlined.

1.1 Motivation

With the rapid development of social economy and the process of industrialisation, the number of vehicles has been increasing dramatically in cities. However, due to limited capacity, roads become saturated, which causes a considerable number of issues, such as traffic congestion, traffic accidents, energy consumption, and environmental pollution. In the process of exploring solutions to the above-mentioned issues, the automotive industry introduces new energy and energy-saving vehicles to achieve the coordinated development of the industry and environment. In addition, it is essential to fully use the information and communication technologies for achieving the coordinated development of human, vehicle, and environment, which can alleviate traffic congestion, enhance transportation efficiency, and existing road capacity. [2]. The amount of data needed to achieve this leads to numerous cybersecurity threats and vulnerabilities. Which can be from merely data breaches where the attacker will gather data from other nodes to threats that can impact the environment itself. To detect these attackers an Intrusion Detection System (IDS), a system to detect any anomaly in the network, can be deployed to the network. Due to the amount of data and possible vulnerabilities most systems are exposed, nowadays, many IDSs use machine learning approaches.

For the Internet of vehicles to be used, the security of all data, nodes and users has to be ensured. Therefore, a system must manage all the data about vehicles, traffic, routes, signs, car crashes and people who interact with the network.

Besides, this system has to be flexible enough to work everywhere and adapt to every city, village or any other place it might be deployed. The algorithm has to learn to accomplish all goals it was designed to achieve. Therefore, a Deep Learning (DL) approach will be used. In addition, the above-mentioned algorithm will require a lot of computer power, notwithstanding, in such network, the response time is also of utmost importance. Hence, to bring part of the computer and storage power closer to the end-users the algorithm will be deployed in the edge layer of the IoV.

Nevertheless, there are some aspects to take into consideration. What if the algorithm stops working? What if the algorithm does not share with the vehicles the correct data? What if some nodes act in a way that may harm the others? This may occur if the network is attacked. Therefore, such algorithm should also be able to detect and act to eliminate these threats.

1.2 Objectives

This thesis aims to develop an edge-based smart network system for the IoV. Thus, in the first instance, a study of its environment, architectures, main vulnerabilities and critical challenges was needed. Which, considering the importance of data dissemination, lead to the research of a viable way to deliver data among the network. In this scope, two algorithms are needed, one for clustering and the other for routing. Which lead to the following objectives:

- Design and implementation of a clustering algorithm stable in dynamic networks such as the IoV.
- Design and implementation of a routing algorithm complying with the paradigm of the IoV and being capable of propagating data throughout the network.

At last, considering the numerous vulnerabilities of the IoVs, a monitoring system capable of handling them has to be developed, Which lead to the main goal of the project.

- Develop a monitoring system on the edge layer to detect any anomaly and classify behaviours in the network, hence, helping mitigate the impact of misbehaviour nodes.

1.3 Contributions

This work proposes a clustering algorithm based on the social relations between nodes, this algorithm is later used in a cluster-based routing algorithm based on the privo [8] algorithm. In addition, this work proposes a monitoring system for the IoV, to detect anomalies in the network and to identify possible malicious behaviours.

An article based on this thesis is being prepared to be submitted to an international conference.

1.4 Outline

In this section, a brief description of the chapters is provided.

Chapter 2: State of the Art - Firstly, a deeper study of IoV, its applications and architecture is presented. Secondly, a description of Edge computing focusing on vehicular edge computing is described. Then, a summary of some security challenges is presented. In addition, a survey on the main machine learning algorithms as well as some clustering approaches is also presented. Following, there is a brief description of the simulator that will be used to evaluate the system. This chapter finished with an introduction of some concepts that are going to be used during this thesis.

Chapter 3: Design and Implementation - A detailed description of everything implemented in this project, beginning with the deployment of the architecture on the simulator is presented. Followed by a

clustering algorithm based on the social relations between the nodes that will run based on the similarity between nodes and their ego in the network graph. Then, a cluster-based routing algorithm which consists of an adaptation of the privo routing protocol [8] will be explained. At last, the monitoring system as well as its two algorithms are clearly explained. The proposed system can detect anomalies in the network and identify attacks.

Chapter 4: Results and Discussion - This chapter contains a description of all tests and simulations done to prove the success of the solutions presented in the previous chapter, as well as the respective results and comparisons with known algorithms/baselines. In addition, in this chapter is a discussion on the results obtained.

Chapter 5: Conclusion and Future Work - This chapter provides a summary of what was accomplished, significant conclusions and suggestions for future work in the area.

2

State of the Art

Contents

This chapter presents an overview of the IoV regarding what it is, its main applications and its architecture. Then, a description of Edge computing focusing on vehicular edge computing is presented. A summary of the IoV security challenges and how to tackle them is also provided. Then, a survey on the main machine learning algorithms and their characteristics as well as some clustering approaches is also presented. Following, there is a brief description of the simulator that will be used to evaluate the system. This chapter finished with an introduction of some concepts that are going to be used during this thesis.

2.1 Internet of Vehicles

Vehicular Ad-hoc NETwork (VANET) [9] has been a very active area of research both in academia and industry. However, as the number of vehicles connected to the Internet increased, new requirements of VANETs were emerging (such as intervehicular and intravehicular communications). Therefore, VANET has brought the concepts of Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) communication. One of the main problems of VANETs is the limited capacity for processing all the information that is collected by each network node. However, VANETs are well suited for short term applications or small scale services such as collision prevention or road hazard control notifications services. In contrast, IoV focuses on the integration of objects such as humans, vehicles, things, networks, and environment to create an intelligent network based on computing and communication capabilities that supports services for large cities or even a whole country. In this context, the concept of VANETs is evolving into the IoV [10].

2.1.1 Applications

IoV is becoming the next transformation in the world of transportation. It aims to improve the user's experience and safety by coordinating human, vehicle and the environment. Its main goal is safety, comfort and prompt delivery of its occupants with minimum impact on the environment. With this goal in mind, there are several applications for this technology, such as: managing network traffic the cooperation of all vehicles leading to a reduction of traffic jams; alert the users about any hazard; in case of an accident, call for specific help and send information about the victims. The only way these goals can be achieved is through communication between vehicles, pedestrians, Road Side Unit (RSU)s, and public networks of Vehicle to Everything (V2X). This huge amount of data leads to some challenges for security, privacy for users and their data.

All in all, the predominant challenges for IoV are the human, vehicle and environment integration; and human and vehicle coordination. [2]

2.1.2 Architecture

Most of these challenges have also appeared in other fields of study such as Internet of Things (IoT). The similarities between IoT and IoV are evident. The same architecture schema is being used to model the networks, splitting them into tree layers, namely, Vehicles (Things), Edge and Cloud as shown in Figure 2.1 [11].

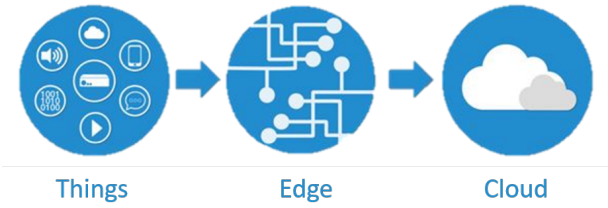

Figure 2.1: IoT based architecture (based on [1])

The Things layer is mainly responsible for data collection and actuation to control the physical world. Most devices in the Things layer are resource constrained in terms of computational power, storage, and energy. Thus, they must use various low-power, low-bandwidth communication protocols, and many of them cannot connect to the Internet directly. The Edge layer is introduced to help end devices. First, computation intensive tasks can be offloaded to edge devices. Second, the edge layer can mask communication heterogeneity among end devices and connect them to the Internet. Third, edge devices help manage IoT end devices. The resource-rich Cloud layer is utilised not only to store, process, and analysed the collected data but also to provide the additional support needed by many IoT applications. [4]

It is widely accepted that there are significant differences between IoT and IoV. Vehicles are not as low-power and constrained as common things in IoT. However, some IoV's capabilities also require knowledge of the rest of the network, e.g, traffic prediction, leading to an edge layer also in IoV.

In IoV, the idea of edge computing is also being used [12]. In fact, on the Internet of Vehicles there is a new paradigm, Mobile Edge Computing (MEC) [13]. As the authors describe this architecture as a set of four different components, the vehicles, RSU, MEC servers and the cloud. RSUs are fixed and vehicles can only connect to the ones located in the corresponding segment. Each RSU is equipped with a MEC server with limited computation resource [14]. The RSUs are connected with each other through wireless backhauls. Since for many applications the size of computation task-input is much larger than the outcome, the task-input file cannot be transmitted between RSUs. In other words, each MEC server only executes the computation task received from the RSU with which it connects. However, because the output data is small, the computation output can be transmitted between RSUs through wireless backhauls. [13].

There are two types of offloading: through direct V2I mode transmission and the predictive-mode

transmission. On the first one, the vehicle will upload the data to the nearest RSU and then, when the computation is done, the RSU will send the results to the RSU that covers the expected position of the vehicle. The other option is sending the data through vehicles, and the source will send the data from one vehicle to the other until it reaches the desired RSU, i.e., the RSU that covers the position where the vehicle estimates that it will be once the output is done [13].

A distinct perspective of the architecture is presented in [2]. The authors mention IoV is not only a service network for vehicle-to-vehicle communication or vehicle terminals, but also a complex system that has the feature of human-vehicle-environment tightly coordinative interaction and highly dynamic evolution. To achieve this target, they present an architecture, shown in Figure 2.2, divided into four layers: Vehicle network environment sensing and control layer; Network access and transport layer; Coordination computing control layer; and Application layer.

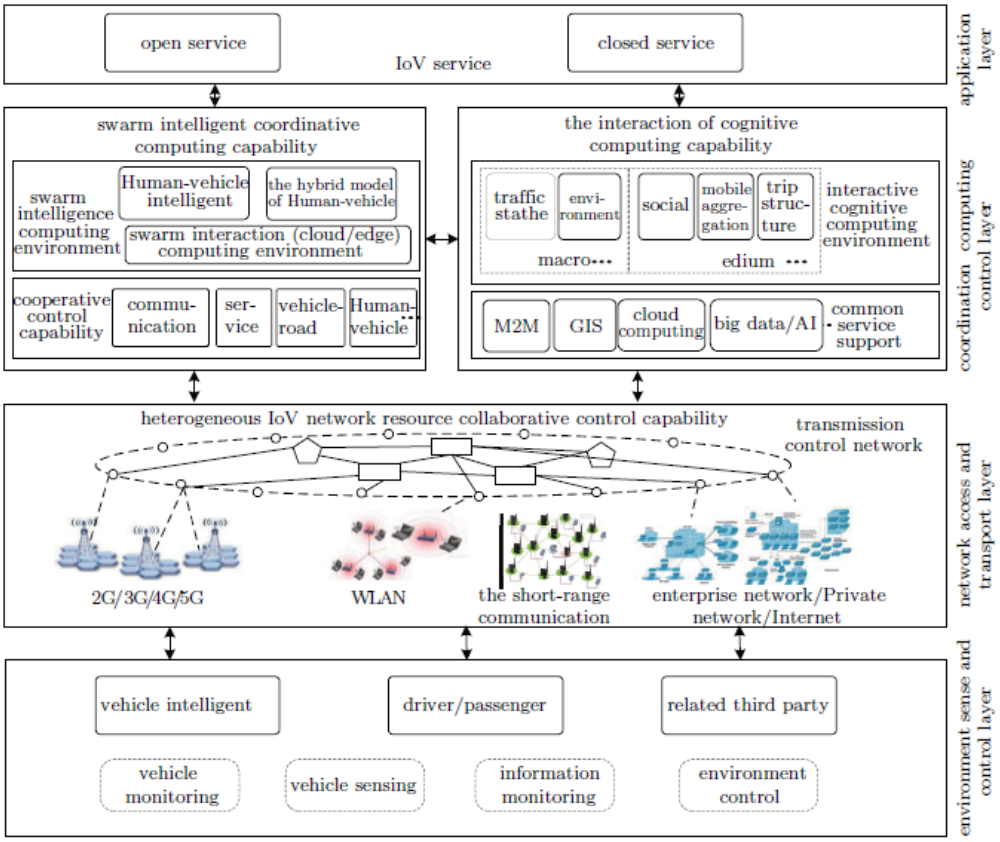

Figure 2.2: IoV architecture [2]

The first layer, vehicle network environment sensing and control layer, has the goal of retrieving information regarding humans, vehicles and the environment. In addition, it is also responsible for receiving and executing control instructions and implementing the desired sensing capabilities. The following layer, network access and transport layer, is responsible for the interconnection and information exchange

on the network. Regarding the coordination computing control layer, it provides IoV applications with the network-wide capability of coordinative computing and control for human-vehicle-environment [2]. At last, the application layer of IoV provides various types of services to achieve the requirements of human-vehicle-environment coordination services [2].

2.1.3 Routing

As mentioned above, the goal of IoV is to change the world of transportation by coordinating human, vehicle and environment. To achieve this goal an enormous amount of data has to be constantly traded between the different types of nodes, therefore, a routing algorithm has to be deployed to the network.

Most routing algorithm work based on the property of shortest path to destination, however, in IoV most nodes are mobile, which means that the algorithm has to be able to change data between mobile nodes and has to be delay tolerant, in other words, where there is no guarantee that a fully connected path between the source and destination exists at any time.

The most trivial algorithm for this situation is the epidemic protocol [15], where nodes will transmit the message to every single node as long as it does not already possess a copy.

The main problem of the epidemic algorithm is the resource usage that it requires, the algorithm does not attempt to eliminate replicas that are unlikely to reach the destination. The prophet algorithm [16] is a variant of the epidemic [15] that by pruning the epidemic distribution tree, attempts to exploit the higher probability of certain nodes facing the destination, and therefore, the message will only be sent to a node with a higher probability of reaching the destination. This probability will be updated over time.

There is also the Bubble Rap algorithm [17] that groups the nodes into clusters our hubs and forwards the messages according to this social-based metrics which based on real data was proved to be more efficient that the prophet in environments where this social relations between nodes can be seen.

Some algorithms create more efficient ways of measuring the relations between nodes and also deploy some extra features to the network, such as the privo algorithm [8], an algorithm that besides enabling the communication between nodes, also preserves the privacy among the elements of the network.

2.1.4 The PRIVO protocol

This section contains a brief explanation of the Privo algorithm [8] focusing more on some metrics that will be used in this project.

The Privo algorithm is an efficient PRIvacy-preserVing Opportunistic routing protocol for Vehicular Delay-Tolerant Networks. This protocol models a Delay Tolerant Network (DTN) as a time-varying neighbouring graph where the edges correspond to the neighbouring relationship among pairs of nodes,

the weight of an edge indicates the strength of a relation at a given time.

The authors of [8] present two social metrics, Ego betweenness centrality and similarity. The centrality of a node is a quantitative measure of the structural importance of this node in relation to others within the network.

Betweenness centrality is defined as the number of geodesics (shortest paths) passing through a given node, it measures how well a node can facilitate communication among others. An ego network is defined as a network that consists of a central node along with its direct neighbours and all links among these neighbours. The shortest paths, due to the structure of the ego network, are either of length 1 or 2. Every single pair of non-adjacent direct neighbours must have a shortest path of length 2 which passes through the central node (ego). Shortest paths of length 1 do not contribute to the betweenness centrality computation. Considering A as the adjacency matrix of the graph of the network, then $A^2_{i,j}$ contains the number of shortest paths on length 2 connecting vertices i and j. Therefore, the number of shortest paths between i and j is given by $A^2[1 - A]_{i,j}$ (where 1 is a matrix of all 1's).

As far as the similarity is concerned, it is defined as the number of common neighbours among a pair of nodes. Therefore, the more common neighbours they have, the more similarity they have.

While constructing the graph, the authors of [8] also introduce average separation period, whose aim is to capture the evolution of social interactions in similar time periods (time slots). The separation period $x_{i,j}(t)$ is calculated based on the number of times two nodes were away from each other $n_{i,j}$ and the elapsed time τ. In this way, if $x_{i,j}(t) = 0$ the nodes i and j are in range at time t, otherwise, $x_{i,j}(t) = 1$. The average separation period is given by

$$\delta_{i,j} = \frac{\int_\tau x_{i,j}(t)dt}{n_{i,j}} \tag{2.1}$$

The normalised average separation period $\hat{\delta}_{i,j}$ is given by

$$\hat{\delta}_{i,j} = 1 - \frac{\delta_{i,j}}{|\tau|} \tag{2.2}$$

The authors also use daily timeslots and the average separation period in the same timeslot over consecutive days is updated using an exponential weighted moving average as seen in equation 2.3. During the first day, the value of the normalised average separation period will be assumed.

$$\Delta^t_\tau = (1 - \alpha) * \Delta^{t-1}_\tau + \alpha * \hat{\delta}^t_\tau \tag{2.3}$$

where α is the smoothing factor, with $0 < \alpha < 1$, and it is depreciated over consecutive timeslots as follows

$$\hat{\Delta}^t_\tau = (1 - \alpha) * \Delta^{t-1}_\tau \tag{2.4}$$

Besides the metrics above, the privo also used a metric called Mean Time to Encounter (MTTE) to determine the best message forwarder. This metric takes into consideration the average separation period at each time slot as well as the expected time necessary for the two nodes to re-encounter. Given that in Privo each node keeps an estimation of its average separation period at each timeslot, the algorithm also predicts the most probable timeslot for future contact.

2.2 Edge Computing

Edge computing encompasses various paradigms, (e.g., multiaccess edge computing, mobile edge computing, fog computing) that aims to decentralise the cloud layer and bring part of the computer and storage power closer to the end-users. Thus, edge computing will improve user experience reducing the network latency and the overall response time of the system. It will also curtail the bandwidth utilisation between the edge and the cloud. Moreover, this new paradigm reduces the need for direct communication among end-user devices since edge components can serve as network messages' relays and (pre)process them if needed [12].

2.2.1 Vehicular edge computing

Vehicular Edge Computing (VEC) can be seen as the application of edge computing in vehicular networks. Some authors consider vehicles as devices of the edge itself ([18]). Here, vehicles will be only end-users of the infrastructure. In addition, and according to [3], VEC is divided into two planes, users and infrastructure, as shown in Figure 2.3. The devices in the infrastructure are organised into the traditional edge and cloud layers. The authors also foresee three possible deployment models: Fog-Based; RSU-based; and Hybrid deployment. Regarding the fog-based deployment, it concerns the general purpose of fog/edge computing, using devices that have large storage and strong computation capabilities. They can also manage communications and computations with multiple vehicles simultaneously. The second model, RSU-based deployment, concerns the use of roadside devices to support the vehicular network (that can be attached to the road signs, traffic lights, etc.). However, these devices, typically have less storage and computational power than general purpose edge devices. At last, the hybrid deployment that combines the two deployments and takes advantage of the capabilities of each model [3].

2.3 Security

The amount of autonomy and data that a vehicle might have or acquire leads to a major concern about the security and privacy of such systems. This network can be seen as a distributed system and has

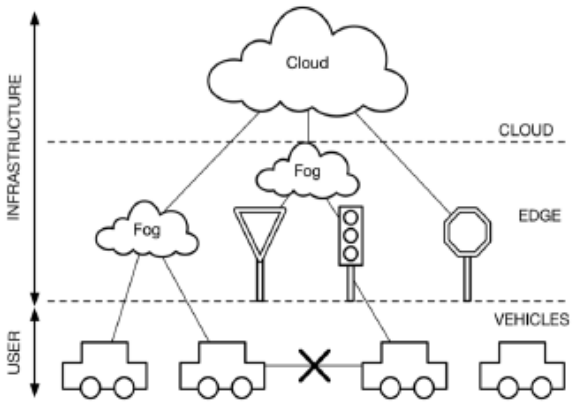

Figure 2.3: Vehicular edge computing [3]

some risks associated with it. There are risks linked with network elements/devices; unauthorised access to confidential or reserved information; loss or theft of information or equipment; personification; denial of service, among others.

Concerning the network itself, the risk of unauthorised access to confidential or reserved information and personification are also present, but, in addition, there are interception, modification and data replay [19]. To mitigate these risks, there are the following security requirements that have to be taken into consideration: Authentication; Intellectual Property Protection; Confidentiality; Integrity; Access Control; Message Freshness; Privacy; Availability.

Authentication is a key factor concerning the various entities that interact with each other. The interactions within the car, software and hardware create an openness that may allow the disruption of the normal behaviour of the car or the triggering of sophisticated attacks. Also, this factor must be taken into consideration in intravehicular communications, where, instead of software and hardware there are numerous nodes and entities. In light of the above, entities must authenticate each other to make sure they are the ones they claim to be.

Regarding intellectual property protection, all vehicles and its components should be protected against cloning and reverse engineering.

Confidentiality is also a major aspect that cannot be neglected. The information stored or exchanged between different components or elements of the network must be always protected, preventing the leakage of sensitive information that can endanger the users.

In any network, integrity is of utmost relevance. It refers to the detection of any undesirable manipulation of some data. It should also be possible to detect message injection.

As far as access control is concerned, vehicles should grant selective access to the data they acquire. Only authorised elements can have access to certain information (e.g. a driver does not need to know the destination of the cars next to it).

The network should also be able to ensure the freshness of the messages. However, vehicles shall also be protected against massages that may be delayed or replayed (by intra-vehicle and inter-vehicle components).

The privacy of users is essential. Sensitive information must be protected because it may compromise the privacy and the safety of the users (driver and passengers).

At last, the availability of any element of the network must be ensured at any point in time and space. [20]

To address the challenges discussed above, there is a novel security service used in IoT called EdgeSec [4]. First, this security approach is based on the edge layer, that, as mentioned before, has much more resources than the things layer. Second, it is much closer to the IoT things (or IoV vehicles) than the cloud. Third, the Edge layer has more information than end devices about the whole system; therefore, it is possible to deploy optimised algorithms at it. Fourth, edge devices can have a relatively stable relationship with the IoV vehicles, which is very beneficial to establish trust between the IoV vehicles and edge devices. Fifth, the Edge layer can be used to safeguard the privacy of individuals by enabling more user control over their personal information, or by applying secure aggregation algorithms. Sixth, the Edge layer has a high-speed connection with the Cloud layer and it is easy for them to get extra support from Cloud layer services. Finally, the Edge layer is flexible enough to support various services needed by IoV vehicles.

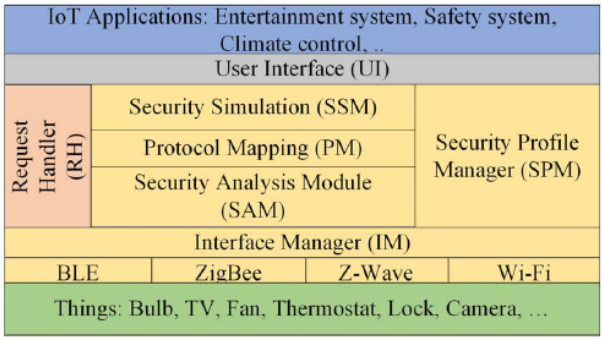

Figure 2.4: Design of EdgeSec [4]

As seen in Figure 2.4, the design of the EdgeSec consists of seven major components as follows: Security Profile Manager (SPM), Security Analysis Module (SAM) Security Analysis Module (SAM), Protocol Mapping (PM), Interface Manager (IM), Security Simulation Module (SSM), Request Handler (RH), and User Interface (UI). SPM registers IoT things to EdgeSec. It creates a security profile and collects the security requirements of each end device. Based on device security profile and requirements, SAM decides where to deploy the security functions, i.e., whether at the Things layer, the Edge layer, or the Cloud layer. Then, the PM module chooses appropriate protocols to satisfy the security requirements based on the decisions from SAM. IM is designed to mask communication heterogeneity

in IoT things. Requests which may cause physical damages are tested in SSM before they are handled by IoT things. Coordinating other EdgeSec components, the RH module helps handle requests to access IoT resources securely. Finally, interfaces are provided for administrators and users to interact with EdgeSec components through UI [4].

Nonetheless, the Edgesec was developed for IoT, by, transposing to IoV, all seven components would remain there with the same purposes.

SPM can be seen as an IDS whose goal is to detect and counter any intrusion. An intrusion is an action with the goal of compromising the integrity, confidentiality, or availability of a resource. Therefore, SPM aims to detect intrusions looking for attack signs, i.e., odd or suspicions actions or changes on the stored information. Moreover, this is not trivial due to detection errors. Specifically, the IDS can wrongly detect an intrusion, which is a false positive; or, on the other hand, fail to detect an intrusion, called false negative. This error rate is difficult to manage, if the system has too many false positives, it may be discredited, if it has too many false negatives, it is useless.

There are two methods to detect intrusions, misuse detection and anomaly detection.

The first one, also known as knowledge-based detection, scans the system activity searching for known attack or intrusion patterns. The advantage of this method is its efficiency, usually, this method as a low false positive rate. On the other hand, it can only detect problems whose signatures are already known by the system, leading to a high false negative rate.

Anomaly detection searches for deviations from the normal behaviour of the system. This method uses statistical analysis and Artificial Intelligence (AI) to learn the normal behaviour of the system. However, this information takes time to acquire. The main advantage of this method is the fact that it is theoretically able to predict new attacks. The main problem of this method is that the IDS can be deceived with unusual, but not harmful, behaviours. This would increase the false positive rate and, therefore would make the system less trustworthy.

method	advantages	disadvantages
misuse detection	high efficiency - low false positive rate	only detects known attacks - high false negative rate
anomaly detection	it may detect new attacks	it may be trained by the attacker to lower its accuracy

Table 2.1: IDS overview

Therefore, if possible, a hybrid IDS should be chosen to take advantage of the benefits of each paradigm, i.e., a highly efficient system that might detect unknown attacks.

Another distinction on IDSs is based on where they look for intrusive behaviour: network-based or host-based. A network based IDS identifies intrusions by monitoring traffic through network devices. A

host-based IDS monitors processes and file activities related to the software environment associated with a specific host [7]. As far as edge monitoring is concerned, a network based IDS should be used to oversee the IoV, since it has the goal of monitoring the network of a given area.

Moreover, the IoV is exposed to several attacks [20] such as Denial of Service which consists on explicit attempts to block legitimate users' system access by reducing system availability [21]; black hole attack which is when a malicious node uses the routing protocol to advertise itself as having the shortest path to the node whose packets it wants to intercept [22]; id spoofing, when a node successfully identifies itself as being another node; and the sybil attack [23], a node that takes multiple identities and use it to gain influence and power in the network [24].

2.4 Machine Learning

Machine learning is the computational process of automatically inferring and generalising a learning model from sample data [5]. Learning models use statistical functions or rules to represent the correlations between the input and the output of a given system. There are two types of problems in machine learning, regression, when the goal is a value; and classification, when the goal is a label. In this case, the goal is to label the inputs, therefore, only classification methods will be studied. Depending on the training data and the desired output, these algorithms are categorised in supervised and unsupervised learning. Regarding supervised training, pairs of input and output are given to a train function and a model is trained such that the output of the system can be predicted with the minimum cost. Artificial Neural Networks (ANN), Support Vector Machine (SVM) and decision trees are examples of this kind of learning. In unsupervised learning, the only input is given in sample data. These algorithms are designed to form natural clusters of input patterns and summarise key features of the data. Examples of these algorithms are k-means clustering and k-nearest neighbour.

2.4.1 Supervised Learning

Artificial Neural Networks

ANNs [25] are inspired by the human brain and composed of interconnected artificial neurons. These neurons are divided into layers and the output of a layer is the input of the following layer, additionally, there are weights related to each connection of neurons, as shown in Figure 2.5.

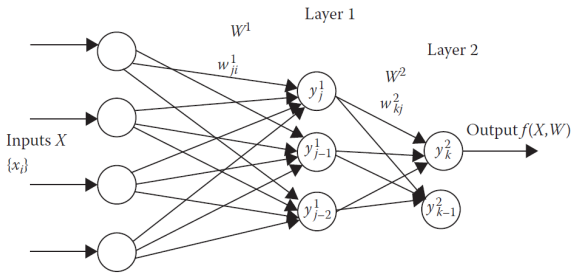

Figure 2.5: Example of an ANN [5]

In addition, each neuron has an activation function, that defines the output given an input or a set of inputs, the most used is a Rectified Linear Unit (ReLU) (Figure 2.6). On the light of the above, ANN can model nonlinear functions to represent the desired system [5, 7].

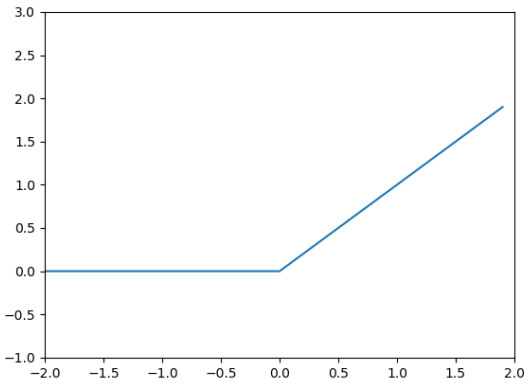

Figure 2.6: Rectifier Linear Unit

There are several possible architectures for Neural networks, changing the number of layers and neurons per layer it is possible to build a network with any size as desired. With the increase on the number of non-linear layers, it will be created an image of the raw data (input data) that will be transformed into a representation at a higher, slightly more abstract level, and this is called Deep Learning [26].

Most neural networks are composed of layers such as the ones described above, nevertheless, there are other types of layers, such as convolution layers that are used to 3D inputs. These are called Convolution neural networks [27].

One of these architectures is an Auto Encoder, a neural network that is composed of four parts, encoder, bottleneck, decoder and reconstruction loss. An example of the architecture of an auto encoder can be seen in Figure 2.7. The encoder is responsible for reducing the input dimensions and compress the data into an encoded representation. The bottleneck refers to the layer that contains the compressed data, it has the lowest possible dimensions of the input data. Regarding the decoder, it reconstructs the

data from the encoded representation as close to the original as possible. At last, the reconstruction loss measures how well the algorithm is performing and how close is the output data in comparison to the input. This algorithm uses the input data instead of labels [28].

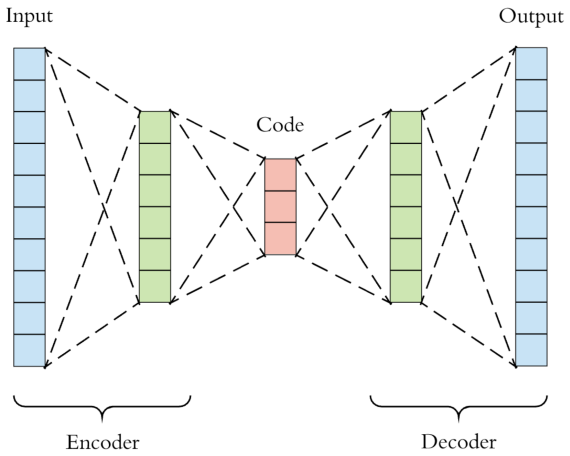

Figure 2.7: Architecture of an auto encoder [6]

Support Vector Machines

SVM is a classifier that aims to find a separating hyperplane in the feature space between two classes with the maximum distance to the nearest point on each side, called support vectors [29]. When two classes are not separable, slack variables are added and a cost parameter is assigned to the overlapping data points. This method can also represent nonlinear functions by using kernel functions that map the original feature space to a higher dimensional feature space where the training set is separable [5, 7]. An example of a hyperplane of separation can be seen in Figure 2.8.

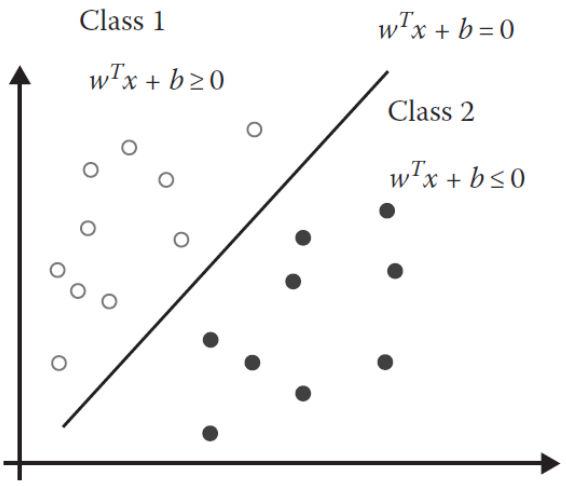

Figure 2.8: Example of SVM classification [5]

Bayesian Network

The Bayesian network is a model that represents a probabilistic relationship between variables [30]. The network is constructed with nodes as the discrete or continuous random variables and directed edges as the relationships between them, establishing a directed acyclic graph. The child nodes are dependent on their parents.

Naïve Bayes is a simpler Bayesian Network model that assumes that all variables are independent. Notwithstanding the strong independence assumption of the variables involved, the method gives good results even if the assumption is violated [5, 7].

Decision trees

A decision tree [31] is a tree-like structure that has leaves, which represent classifications and branches, which in turn represent the conjunctions of features that lead to those classifications, as represented in Figure 2.9. These decision trees are usually built according to the ID3 algorithm [31] that builds the model based on the concept of information entropy.

The advantages of decision trees are: intuitive knowledge expression, high classification accuracy, and simple implementation. The main disadvantage is that for data including categorical variables with a different number of levels, information gain values are biased in favour of features with more levels. Moreover, larger trees often have high classification accuracy but not very good generalisation capabilities. By pruning larger trees, smaller trees are obtained that often have better generalisation capabilities [5, 7].

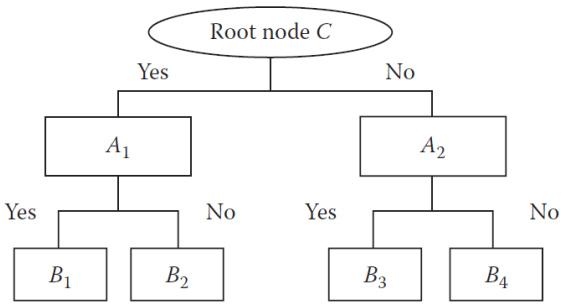

Figure 2.9: Sample structure of a decision tree [5]

To reduce the biased factor of decision trees, it was created a model called random forest. This model consists of various decision trees and the output of it is decided by the votes of all individual trees. Each decision tree is built by classifying the samples of the input data [5].

Hidden Markov Models

The previously discussed algorithms assume that all samples are independent and identically distributed. In this model, it is assumed that data is sequential, and, therefore, there may be a correlation between sequential states. In Hidden Markov Model (HMM) [32], each node represents a random variable with a hidden state and an observed value. HMM is also based on the Markov property that the current state is conditioned only by the previous hidden state but is independent of all other states (past or future). In addition, each observation only depends on the respective hidden state. [5]

The general architecture of this model is presented in Figure 2.10.

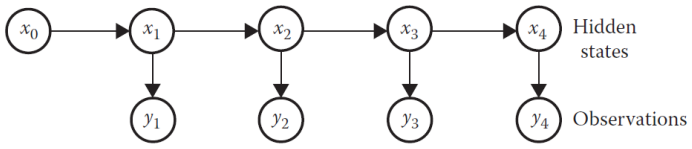

Figure 2.10: Architecture of HMM [5]

Genetic Algorithm

A genetic algorithm [33] is an algorithm based on the theory of natural evolution. The algorithm selects the fittest individuals of a population to create the next generation.

In this algorithm, an individual is composed of a set of variables known as genes, these genes are gathered and form a chromosome. Based on each chromosome, it is assigned a fitness score that will be used to choose which individuals will be used to reproduction.

The reproduction of new individuals is done by mixing genes of each parent and generate new chromosomes. This new individual may suffer some mutations, i.e., random changes on their chromosomes, to generate a bigger population based on the same parents.

2.4.2 Unsupervised Learning

k-Means Clustering

Clustering is the assignment of objects into groups (clusters) so that objects from the same cluster have more similarities than objects from different clusters. This method divides the given data points into *k* clusters and at each iteration assigns a new centroid based on the elements contained on each cluster [34].

Two key issues are important for the successful implementation of the k-means method: the cluster number k for partitioning and the distance metric. Euclidean distance is the most employed metric in k-means clustering. Unless the cluster number *k* is known before clustering, no evaluation methods

can guarantee the selected *k* is optimal. However, researchers have tried to use stability, accuracy, and other metrics to evaluate clustering performance [5].

k-Nearest Neighbour

In this method, each data is assigned to the label that has the highest confidence among the *k* datapoints nearest to the query point. Specifically, if *k* is 5, the nearest 5 points are queried, and each reply is counted as a vote. In the end, the answer with more votes will be the classification of that data point [5].

2.4.3 Computing Complexity

As mentioned before, an IDS can be used to monitor a given network. Most IDS use machine learning algorithms to achieve their goals and increase their capabilities.

The authors of [7] compare several machine learning algorithms concerning its typical time complexity during training and its streaming capabilities and their results are shown in Table 2.2. It is important to highlight that the complexity presented is based on an extensive literature and internet search done by the authors of [7]. Naturally, some complexities are debatable and may vary with the experience and skill of the implementer.

Algorithm	Typical Time complexity	Comments
ANN	$O(emnk)$	e: number of epochs k: number of neurons
Bayes Network	$>> O(mn)$	
k-means	$O(kmni)$	k: number of clusters i: number of iterations
Decision tree	$O(mn^2)$	
Naïve Bayes	$O(mn)$	
k-nearest neighbour	$O(nlogk)$	k: number of neighbours
Random Forest	$O(Mmnlogn)$	M: number of trees
HMM	$O(nc^2)$	c: number of states
SVM	$O(n^2)$	

Table 2.2: Complexity of Machine Learning algorithms during training [7]

Regarding the time complexity, linear algorithms, i.e. $O(n)$ and $O(nlogn)$, are usable for online approaches. $O(n^2)$ are still acceptable for most practices and above that are considered much slower and can only be used for offline approaches. In the results presented in table 2.2, *n* to refers to instances and *m* to attributes and it was assumed that *n* is much greater than *m*.

The computational complexity of the algorithm to be used is a key factor in an IDS. In these systems, models are trained more than once. There are examples when they are trained daily [35], whenever the analyst requires [36] or each time a new intrusion is identified and its pattern becomes known [37].

Traditionally, each model is trained from starch. However, if a model needs to be trained often because of only a few changes in data, it may be advantageous to, instead of training from scratch, start with a trained model and train it from there, or, turn to self-adaptive models.

2.5 Clustering

In order to ease the communication between the edge and the elements of the network, vehicles should group themselves into clusters. Each cluster has a cluster head who oversees the communication between these two layers, vehicles and edge.

The most common clustering approach is grouping based on the geographical locations of each vehicle, thus, each vehicle forms a cluster with the nearby nodes. The drawback of this approach is the fact that, in a vehicular network, the position of the nodes is always changing, therefore, the clusters would be too unstable.

Another clustering criterion is based on the social network of each node [38], i.e. clustering the nodes with the ones that it spends more time, for example, where the user lives or where the user works. This approach makes it possible to predict the nodes which a given node will be able to connect in the future. A distinct implementation of this approach is used in [39], where each node keeps a hashmap of the nodes which it has connected. This hashmap stores the social strength of each pair of nodes that will reduce over time and be reinforced when the nodes communicate with each other.

2.6 Simulator

In order to design and implement an edge-based smart network monitoring system for the Internet of Vehicle, it is necessary to model this network and all the case studies necessary to develop such system. Considering that vehicles and roads are not yet equipped with devices capable of testing these networks, a simulator must be used.

Several simulators can be used to study vehicular networks. In [40], the authors present a review of the main simulators for Opportunistic Networks, as well as the advantages and disadvantages of some simulators.

The first simulator was OMNet++[1], a general simulator with several frameworks available. According to the authors, it also had a good performance and good documentation. However, this simulator does not offer the possibility to pass a simulation model to a real implementation, which may be a key factor for some cases. The Veins simulator[2] is a vehicular network simulator based on OMNet++ and the

[1] https://omnetpp.org/
[2] http://veins.car2x.org/

SUMO simulator[3], a road traffic simulator, to generate data of vehicles and their environment. It offers a comprehensive suite of models for Inter-vehicular communication simulation.

The second simulator was ns-3[4], that is also a general simulator used for IP based networks. The main advantage of this simulator is the fact that it can interact with real-world networks and it is easy to move any simulation for the real world. On the other hand, this simulator does not have a visual interface and, according to [40], it has a bad user interface and a complex structure.

At last, the Opportunistic Network Simulator (ONE) [41], that is designed specifically for opportunistic networks, it has a graphical interface and, in addition, also has a solid user community. In this simulator, the radio interface is a simplified version where the programmer only defines the range and if the two nodes are within the range they communicate. On the other hand, this makes the simulation much faster.

At last, considering that our case the simulation will not be passed to a real implementation; the radio module is not of utmost importance; the project requires only an abstraction of a network of vehicles and this network in detail would slow down the simulation. The chosen simulator was the ONE simulator [41]. Nevertheless, some modifications should be implemented to bring the simulation closer to a real one.

The most significant changes were the addition of the edge layer where some of the most costly tasks are supposed to run; the implementations of RSU; the implementation of a new routing algorithm based on privo [39]; and the addition of the deep learning algorithm.

Moreover, the output file of the simulations was also changed to provide a more suitable analysis.

[3] https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/
[4] https://www.nsnam.org/

3

Design and Implementation

Contents

This chapter provides a detailed description and discussion of everything implemented in this project. It will start with the architecture, covering the types of nodes, their communications, and their position. Then, clustering and routing algorithms will be explained and analysed. At last, the monitoring of the network will be discussed, focusing on the algorithm used and what it can detect.

3.1 Network Architecture

This section describes the network architecture. It will cover the types of nodes in the network, the communication paradigm used; and the position of the stationary nodes, RSU.

The proposed IoV network has three layers, vehicles, edge and cloud. It is also assumed that all vehicles connect with each other using an 802.11p Wireless Access in Vehicular Environments (WAVE) interface, that, using the same values as [42], has a range of 100 m and a data rate of $10Mbits/s$. In addition, each vehicle is also equipped with a more powerful interface that can only be used to connect with RSU, this interface has a bigger range, 250 m but has the same data rate.

There are different types of nodes with different parameters, the RSUs, cars and buses. However, the only difference between them is the buffer sizes, an RSU has 1024MB while a car has a buffer size between 64MB and 256MB and all buses have 256MB.

As far as the offloading of data from the edge to the vehicles is concerned, in the previous chapter there were presented two distinct approaches, the one used is the one where a node uploads data to the nearest RSU to be processed, and, once the computation is finished, the output will be sent through vehicles to the direction of the desired node. Nevertheless, the RSUs are all connected through the edge and, thus, some data can be shared between them.

RSU Placement

The addition of RSU was one of the changes to the simulator than had to be done. These devices are supposed to be stationary and their position should be near a road. Therefore, the implementation of the RSU movement/position was based on two movement models that the simulator already had, *StationaryMovement* and *MapBasedMovement*. The first one had the property of generating nodes that could not move and the second one generates nodes that have to be on a road.

When choosing the number of RSUs and their position there are two options available.

In the first one, the user only chooses the number of RSUs and a position. From this position, 7 circles will be created and the RSUs will be distributed among them. The circles have the same centre but have a different radius that will increase with the number of the circle, as shown in table 3.1.

The circles will be divided according to the number of nodes assigned to each and compute a position to each RSU. With these coordinates, there will be checked if there is a valid position in their proximity

1 node	centre (0*radius)
5% of the RSU	circle (1*radius)
10% of the RSU	circle (2*radius)
15% of the RSU	circle (3*radius)
20% of the RSU	circle (4*radius)
25% of the RSU	circle (5*radius)
25% of the RSU	circle (6*radius)
rest	circle 7*radius

Table 3.1: Distribution of RSU

and, when found, the RSU will assume the valid location. If there is not any legit position, the node will be placed in a 7th circle. An example of this distribution is in Figure 3.1, with 50 RSUs centred in RSU_0 (whose range is marked in black).

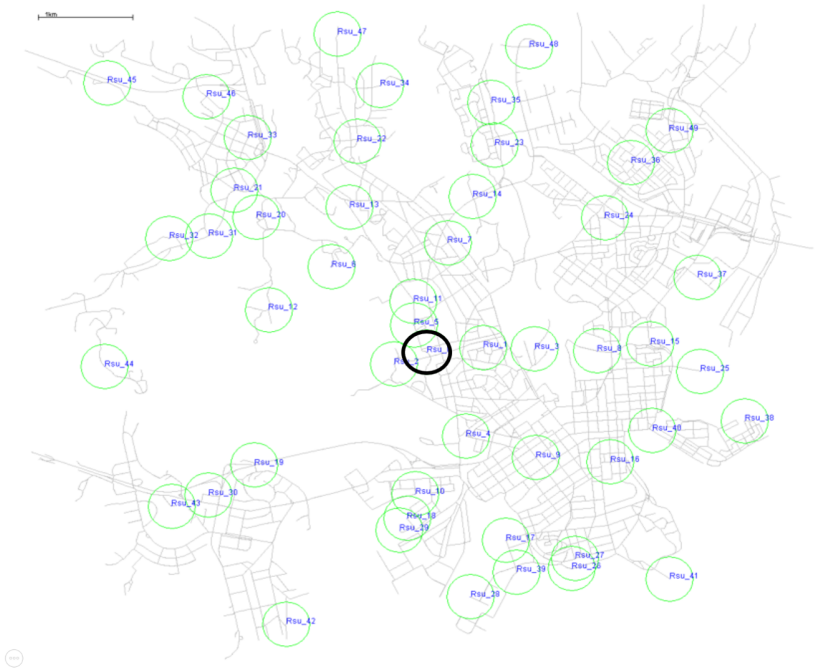

Figure 3.1: Distribution RSU - first approach

When testing this scenario, it was noticed that there were some important points and routes that were not covered by this model. Therefore, a different approach was needed to take this into consideration. Consequently, the map was studied and the RSU were placed one by one into strategic points, such as main interceptions, main accesses to some parts of the city and city centre. The simulator has data of some public transports routes that were also taken into consideration. An example of this approach is shown in Figure 3.2

This node distribution used only a portion of the number of nodes to cover all the city and presents much better results. However, to implement this approach some information about the city is required,

Figure 3.2: Distribution RSU - second approach

therefore, for a simple simulation in a new scenario, the first distribution approach would be advisable.

With more data the second approach could be transformed into an optimisation problem, reducing the number of nodes and improving results of the network.

3.2 Clustering

This section presents a discussion about nodes' clustering in the network. It will also cover different approaches to the problem as well as a description of the algorithm and the metrics used.

There are countless options to cluster nodes in a network, the most trivial one is clustering based on their position. However, this leads to a problem, the stability of the network. The aim of this algorithm is to create clusters of nodes in IoV. Therefore, most nodes are moving in completely different directions. For example, at a given time, a picture of all cars in Lisbon was taken, and, according to that, they were assigned to clusters based on their position. How different would these clusters be if the photo was taken 1 minute later?

In light of the above, another metric was used, social relationships. In this way, every time a node connects with another node their social strength will be enhanced. On the other hand, this metric will decrease if the two nodes do not connect during a given period. All nodes will update their social

relations every hour and they will be clustered according to the similarity of their relations, as explained below.

In a network with the following nodes P, Q, S and T, each node has a map that links its social strength with each of its contacts and the value assigned to it. An example of how this map would look like is shown below where Q_p is the social strength between the nodes Q and P, in the nodes' Q map.

Node Q	Node P
$Q_p = 0.5$	$P_p = 1$
$Q_q = 1$	$P_q = 0.5$
$Q_s = 0.3$	$P_s = 0.7$
...	...

These nodes can be represented in a Euclidean space with N dimensions, where each dimension represents the relations with a given node, as seen in Figure 3.3. Hence, the similarity between nodes will be measured by the distance between them in this Euclidean space.

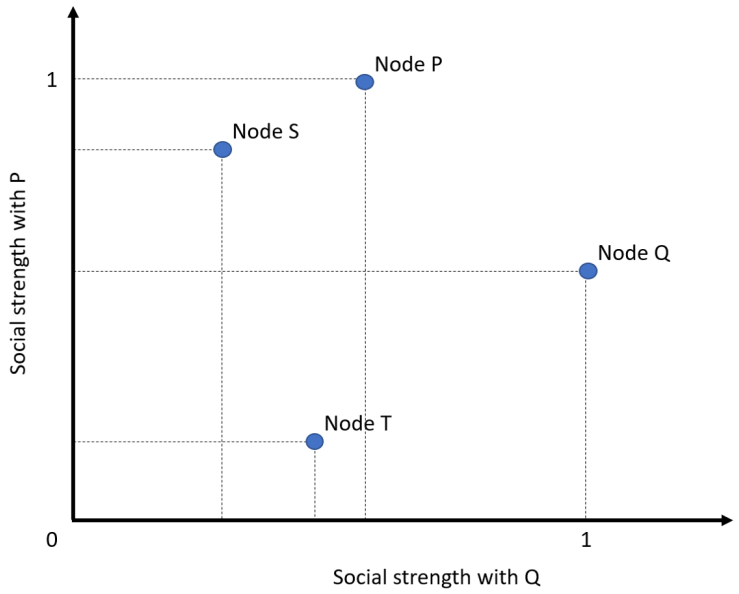

Figure 3.3: Simple example with only 2 dimensions P and Q

Considering that the Euclidean distance is given by:

$$\sqrt{\sum (q_i - p_i)^2} \tag{3.1}$$

The similarity between two nodes can be computed as

$$\sqrt{\sum (K_i)^2}, K_i = Q_i - P_i \tag{3.2}$$

Therefore, considering that the RSUs also store their social strengths, if the edge merges all these data it could be presented as a Euclidean space with N dimensions, where N is the number of RSUs and each node would be represented by its social strength, as shown in Figure 3.4, where node A has never connected with RSU 2 and node B has already connected with all RSUs.

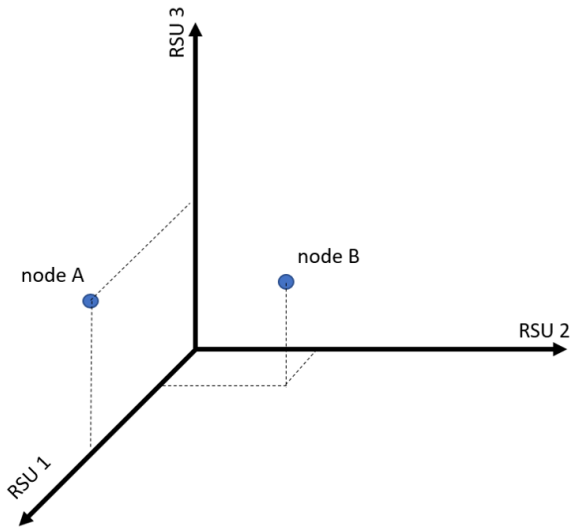

Figure 3.4: Simple Euclidean space with 3 RSU

Based on this representation, and keeping in consideration that the algorithm complexity, for a number of nodes much larger than the number of dimensions[1], is suitable for live computation (see chapter 2 for more information) the k-means algorithm (algorithm 3.1) was used to compute the clusters. The algorithm will run 3 iterations each time it is called. The number of iterations is a balance between resource consumption and the accuracy of the algorithm.

Just after the k-means finishes, the occupation of all clusters will be checked, and, in the case that one or more are empty, the centroid of this(ese) cluster(s) will be placed over a point of the largest cluster and the algorithm will run a few more iterations, then, there will be no empty clusters and a balance between their size.

Once all nodes are divided into clusters, the edge will choose the Cluster Head (CH), a node that represents the cluster and will gather the messages from the cluster to deliver them somewhere else in the network, reducing the network overload. The selection of CH is based on two metrics, the similarity between each node and the centroid that represents the cluster, and the ego betweenness centrality of

[1] i.e. number of vehicles/nodes much larger than the number of RSUs

Algorithm 3.1: K means Algorithm

```
begin
    numCluster ⟵ k (number of clusters)
    for 1 : numCluster do
        create centroid
    for each node do
        minDist = ∞
        for each centroid do
            dist ⟵ distance between node and centroid
            if dis < minDist then
                minDist ⟵ dist
                assign node to centroid
    for each centroid do
        centroid position ⟵ mean position of nodes assigned to centroid
```

each node in the network. The value that represents the similarity is usually between 1 and 0. However, the value that represents the ego[2] is not, thus, a logistic function was used to map the ego values to values that could be compared with the similarity, called "normalized ego", see Figure 3.5. This function was chosen since only a small percentage of the nodes have an ego larger than 7.5 and those are the ones that should be considered to CH. Then, the CH is the node whose metrics produces the lower value from equation 3.3.

$$normalized\ ego * 2 + similarity \tag{3.3}$$

Notice that the CH formula, equation 3.3, gives an extra weight to the ego in order to increase the importance of a smaller normalized ego over the similarity to the centroid.

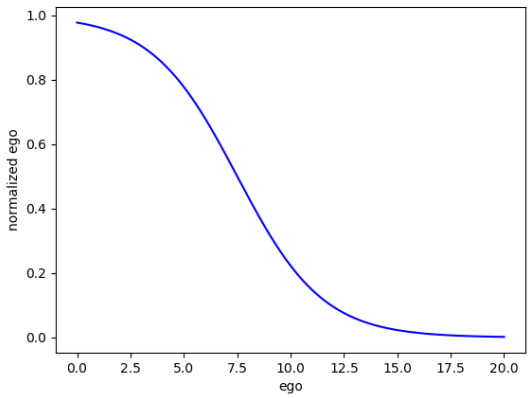

Figure 3.5: Function to map the ego

The main disadvantage of this approach is the fact that when a node does not connect with an RSU

[2]metric concerning the centrality of a node in a graph of all nodes and their connections

for a significant time, it will be closer and closer to the origin of the Euclidean space. Consequently, if this happens with a significant number of nodes, there will be a cluster containing the inactive nodes without any relation between them. Therefore, if the distance between the representation of the node and the origin of the Euclidean space is two orders of magnitude below the usual (clustering threshold - 0.0001), which represent nodes that do not connect with an RSU for a long time, these nodes will be ineligible for clustering and will be considered nodes without cluster.

In Figure 3.6 is shown an example of a network. There are 30 RSU marked with a blue label ”RSU_nn” with nn varying from 56 to 85. There are also 56 vehicles marked with a coloured label composed of a capital letter ”A” through ”H” (labelling its movement model) and a number from 0 to 55. The colours used in the label represent one of the 5 clusters the vehicle belongs to: red, yellow, black, pink and cyan. Each RSU has one circle around it representing the 250m transmission range. Each vehicle has two circles around it representing the 100m 802.11p V2V transmission range and the 250m V2I transmission range. The vehicles that were elected CHs are marked with red transmission circles, while other nodes are marked with green transmission circles.

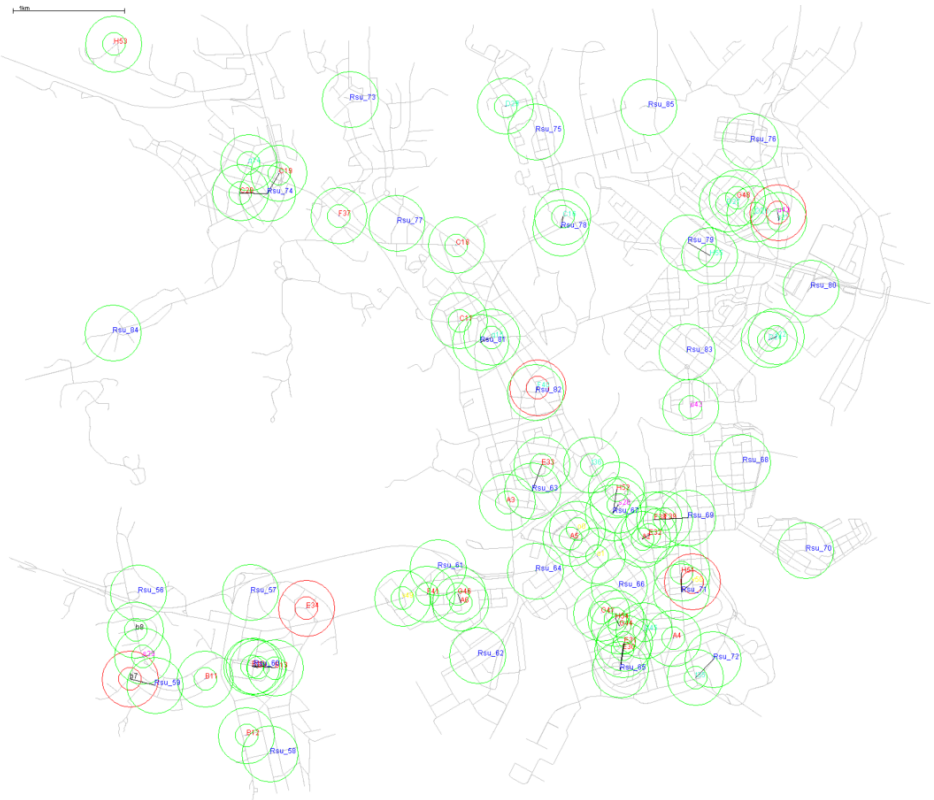

Figure 3.6: Nodes divided into clusters

3.2.1 Cluster Based Routing

It is essential to fully use the information and communication technologies for achieving the coordination of human, vehicle and environment. Therefore, a reliable routing algorithm is mandatory.

This algorithm must comply with the IoV paradigm presented above. Therefore, there are some aspects to take into consideration while designing this algorithm.

First of all, it is assumed that the messages are carried via nodes and not through the edge. This is important since even though the RSUs share this connection, it is just used to share data about the state of the network.

As far as the impact of the cluster in the routing is concerned, there are a few constraints to ensure that the connecting point between clusters are the RSUs and preferable the cluster heads.

In addition, it is important to refer that if two nodes are within range and one has a message for the other, even though they may belong to different clusters, the message will be delivered.

In light of the above, the RSU based privo (R-privo) was developed.

Metrics

The routing algorithm is built upon four mechanisms, inter-contact time, forwarding policy, routing metric and average separation period metric. Some of these metrics are explained in detail in chapter 2.

The inter-contact time metric is estimated for each pair of nodes using an exponential weighted moving average to update values based on previous data.

There are three forwarding policies available, direct single copy, direct multi copy and limited multi copy. In the first option, each message can only be sent from node to node without the sender keeping a copy. While on the second one, all nodes keep copies of all messages that pass through them. Finally, the last policy has a limited number of times an event/message can be copied.

As far as the routing metric is concerned, it depends upon three smaller metrics that are similarity to destination, betweenness centrality and average separation period.

At last, the average separation period is represented by the mean time to encounter.

The R-privo Protocol

The R-privo algorithm is all about complying with the rules of the IoV paradigm used. Therefore, each node will run algorithm 3.2 (Routing Rules) to select which of the nodes in range can receive the message. With this set of nodes, the algorithm 3.3 (Basic Routing Algorithm) will run to check if the other node is preferable comparing to the current node.

The connection between RSUs is only used to share which is the closest to the destination node and what is the RSU with the largest ego betweenness centrality. This information will be used to forward

the messages in the direction of the best RSU or to the most central one when the destination node has never made contact with any RSU. This connection will never be used to send messages from one to another.

In the algorithms presented, *MTTE*, *Sim* and *EgoBC*, refer to the metrics presented in chapter 2, representing MTTE, Similarity and Ego betweenness centrality respectively.

Algorithm 3.2: Routing Rules

```
begin
    if destination cluster = mycluster or destination node has no cluster then
        if the destination node has a cluster then
            run basic routing algorithm only with nodes from the same cluster and RSU
        else
            run basic routing algorithm with all nodes
    else
        if current node is a cluster head then
            send the message only to RSU
        else
            if current node is a normal node in a cluster then
                using the basic routing algorithm considering nodes in the same cluster, RSU
                or the cluster head
            else
                if current node is a RSU then
                    use the basic routing algorithm to send the message in the direction of the RSU
                    that has the highest social strength with the destination node
                    if it is already the best RSU send the message to a node from the same cluster as
                    the destination
                    if the destination has never connected with a RSU send to a node that has already
                    connected with the destination node or in the direction of the most central RSU
                else
                    if current node does not have a cluster then
                        use the basic routing algorithm to send the message considering all nodes
```

Algorithm 3.3: Basic Routing Algorithm

```
begin
    for all node's connections do
        otherNode ⟵ nodeconnected
        if currentNode MTTE < otherNode MTTEor
         currentNode Sim < otherNode Sim then
            send the message to the otherNode
        else if currentNode Sim == otherNode Sim then
            if
             currentNode EgoBC ∗ currentNode MTTE < otherNode EgoBC ∗ otherNode MTTE
             then
                send the message to the otherNode
```

In addition to routing rules, to avoid routing circles, any message will not be transferred to a node

that has already seen it before.

There is also one more aspect to take into consideration about the proposed routing algorithm. When the limited multi copy policy is used, there are times that instead of just transferring that message, the node will create a replica to the other node and keep the message. This change divides the copies of the messages among nodes with a higher probability of meeting the destination node.

As far as the implementation is concerned, the decision is based on the type and cluster of the node who carries the message. If the node is an RSU or it is from the same cluster of the destination node, it will always create a replica before transferring the message. In addition, if the similarity between the carrier of the message and the destination is larger than a threshold, the message will also be replicated and transferred. Otherwise, the node will just transfer the message.

3.3 Intrusion Detection System

In the near future, IoV will be present in our lives, they will change the way we see transportation and mobility. However, this responsibility brings a huge risk. What if something goes wrong? This leads to a need of an entity monitoring the network, overseeing all the nodes and raising flags if one of them is not responding as it should, or a node is intentionally misbehaving.

In this section, a Deep Learning (DL) approach that runs on the edge layer will be described. This algorithm runs based on data from each node, retrieving a probability of a misbehaviour by the given node, and if this output is high, it will give an indication of the most probable behaviour, as shown in the flowchart below (Figure 3.7).

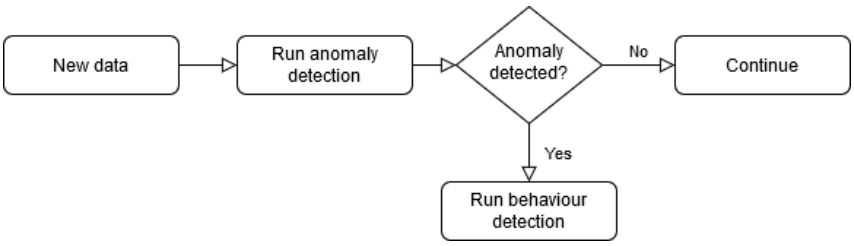

Figure 3.7: Monitoring flowchart

It is important to mention that the objective of this project is only to detect and, if possible, identify the malicious behaviour. The goal of this algorithm is to help the network manager by giving alerts, it will not affect any node or the network directly.

The data given to the algorithm are the type of node (normal node, cluster head or RSU); ego in the network; number of nodes in range; number of messages received; number of messages sent; buffer size; node location; and social strength with its three strongest connections. The location is given as a coordinate X, Y.

These data will be stored in a 2D array according to its data type and the time of acquisition. In the end, each sample contains data of a given node at three different times, as shown in Figure 3.8. The number of samples in each block of data was chosen to take into consideration the past of the node without a huge impact on the performance of the algorithm. Therefore, the input format will be 9×3.

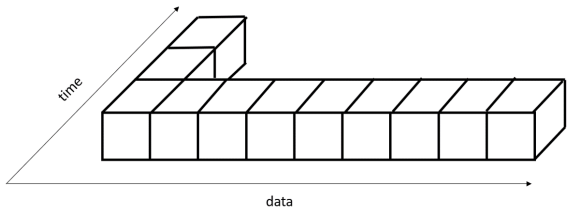

Figure 3.8: Input format

All data used is sent by the vehicles to the edge every time a node connects with an RSU. If a node does not share its data, the edge will not know about its existence, and after a while, the node will have no cluster. Consequently, its role in the network will be less important and the impact of its actions will be lower.

These data will enter in a convolution auto encoder that will use the convolution property to simulate a dependency of data during time. This algorithm aims to encode the input data into a smaller space than the original, and then, from the encoded data, decode it to restore the original input. The performance of the algorithm is measured by the mean squared error between the input and the output data. Therefore, the goal of the algorithm is all about finding a robust analysis of the data, thus, building a robust anomaly detector.

The auto encoder used is formed by convolution layers. These layers receive a 3D input and convolve it with a set of kernels, or filters, and applies an activation function to the filter outputs, in this case, a ReLU. Each kernel has a localised support in the first two spacial coordinates and a full range on the depth on the input (third coordinate). It will compute the value of each neuron of the next layer according to this paradigm. An example of this is shown in Figure 3.9, where each square represents a neuron, and shows that each neuron of the new layer ($layer\ l+1$) is computed based on all the neurons of the previous layer ($layer\ l$) that are covered by the (4×4) kernel with full depth, light blue parallelepiped.

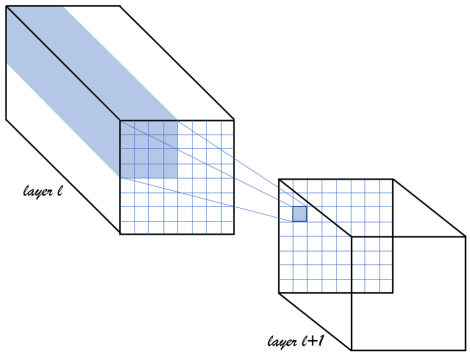

Figure 3.9: Example of the application of a kernel

All layers of the algorithm have the same parameters, a kernel of 3×2 and a stride of 1. The kernel can be seen as a filter, where its size is the filter size. The amount of the filter shift is given by the stride.

In Figure 3.10 is presented an example of an array of $9 \times 3 \times 1$ neurons, with a kernel and stride with the same parameters as used in the algorithm, after the first iteration. The blue represents the area covered by the kernel and the lighter blue the area that was already covered.

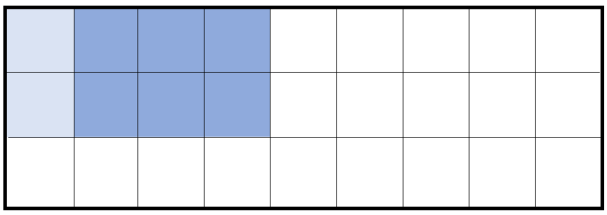

Figure 3.10: Example of the kernel(3x2) and stride(1) on a 9x3 array

At each iteration, the kernel will analyse the data that it is covering and compute a single value to represent it. In the end, the data is reshaped according to equation 3.4, that has to be applied to both dimensions.

$$output\_volume = \frac{(input\_volume - kernel\_size)}{stride} + 1 \tag{3.4}$$

The encoder consists of two convolution layers that will transform the input data into 5 neuron arrays of 7×2 and then 3 arrays of 5×1. Then, the decoder is two layers of the reverse function with the same parameters, as shown in Figure 3.11.

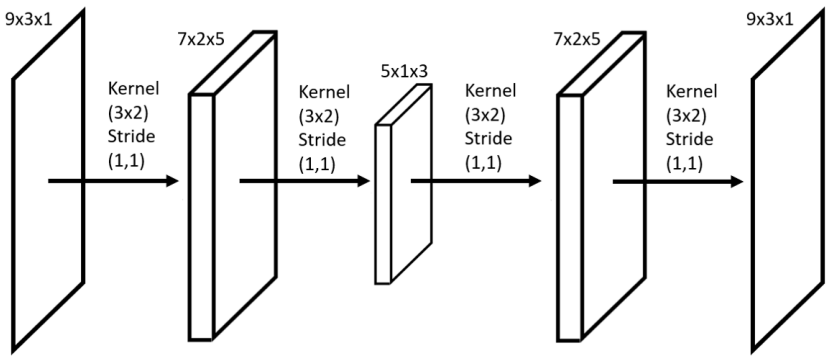

Figure 3.11: Architecture of the Convolution Auto Encoder used

To conclude, in the algorithm used (Figure 3.11), each neuron of the second layer will be computed based on the first layer neurons covered by a kernel (3×2) that will shift one position every time (stride) it is applied. In this case, the depth of the first layer is just one, thus, the kernel will only consider 6 neurons each time. Notwithstanding, for the other layers, as their depth is larger than one, the kernel will cover more neurons, for example, between layer 2 and 3, at each iteration, 30 ($3 \times 2 \times 5$) neurons will be considered.

To implement this neural network, a library called Deep Learning for Java (DL4J) [43] was used. It was trained in a different program, only the testing part runs at the same time as the simulations. As mentioned before, the algorithm will issue the mean squared error between the input and the output, therefore, values closer to 0 represent nodes whose behaviour is closer to the normal. When this value is larger than a threshold, the node will be labelled as misbehaving. This threshold will be adjusted during the simulation to be adapted to each case. Its initial value will be the largest value obtained while testing the algorithm. The mean score of this test as its relation with the maximum value will also be stored and will be used as a guideline to the update of the threshold.

During the simulation, every time the algorithm runs, the mid value will be adjusted as shown in equation 3.5.

$$mid\_value = mid\_value * 0.9 + new\_value * 0.1 \tag{3.5}$$

Then, the relation obtained during testing will be applied to this value, thus, the threshold will be updated.

Training data

All data used to train this network has been collected during several simulations with different parameters, such as number of nodes, movement models or message size, in order to build a model as flexible

and as accurate as possible.

The simulations were done by dividing the nodes into three groups, RSU, buses and vehicles. Considering that the movement of the buses and the position of the RSU is always the same, only the movement pattern of the vehicles was changed.

The simulator used has data concerning typical home, office and meeting locations in Helsinki, divided into 8 groups. Therefore, in the majority of the simulations, a movement model called working day movement was used. This model simulates daily routines. People wake up in the morning, go to work, go shopping or similar activities in the evening and finally go home to sleep. Parameters such as home and office relations, speed or number of nodes were changed over the simulations. In addition, there is a seed for a pseudo random number generator assigned to the movement model that was also changed.

Other movement models were also used to model some vehicles, such as shortest path movement that consists of selecting a point of interest (given by the user) as the next destination. Once the node arrives there, another destination is assigned (these nodes do not have a routine). The points of interest chosen were taken from the homes, offices and meeting spots data set. This movement model reduces the amount of time each node is in the same place and can be seen as an approximation of typical taxis movements.

During the simulation, the RSUs would write their data into a file every hour and the data concerning the other nodes were written during the contact with the RSUs.

3.3.1 Behaviours Implemented

As mentioned above, there are plenty of behaviours or malfunctions that may have a huge impact on the network. Therefore, the following behaviours were implemented.

Node not working

The node not working behaviour consists of a node dropping or simply not sending any of the messages it receives. In addition, the node may have a malfunction on its 802.11p communication interface, therefore, it can only communicate with RSUs. These actions were implemented on the three different types of nodes, RSUs, cluster heads and normal nodes.

Identity Spoofing

As far as the identity spoofing is concerned, it is expressed by a node that successfully identifies itself as another node, or has having a different role in the network. For example, a node claiming that it is an RSU or a cluster head when it is simply a normal node.

This behaviour consists essentially of lying when the node connects with any other node by changing its identification to the desired one or replacing the nodes' identification with the one of an RSU.

Sybil Attack

The sybil attack occurs when a node, i.e., the attacker, subverts the reputation system by creating a large number of pseudonymous identities and uses them to gain influence in the network. In this case, by creating these replicas, the attacker's reputation would rise in the network and in case of success would become a cluster head. The impact of this node would also be noticed outside of its cluster hence the attacker would occupy a central role in the network.

This attack lies on the creation of a group of nodes that will be always in range of the attacker. For the other nodes, these nodes will appear as normal vehicles.

One aspect to take into consideration when creating this behaviour is the fact that the replicas and the attacker has to share the same resources (computer power and buffer size), therefore, the replicas of the attacker usually do not have as many resources similar to a normal node.

3.3.2 Behaviour Recognition

Besides the detection of misbehaviour, the edge should also be able to label the above described actions. Therefore, when the previous algorithm raises a flag, the data will be forwarded to a new neural network.

This DL algorithm will have the same input as the previous one but will be a classification problem, thus, it will have only 5 different outputs: sybil attack; node not working; identity spoofing; normal behaviour; and new unknown behaviour. It will start with a convolution layer to simulate the dependency of data over time and will consist of 3 dense layers and a softmax function to discern the labels. The choice of the number of layers and the number of neurons was a balance between the network flexibility, as with more layers the network can synthesise a wider variety of nonlinear functions with fewer neurons, and the difficulty to train given that a deeper (more layers) and denser (more neurons) network requires more computation (Figure 3.12).

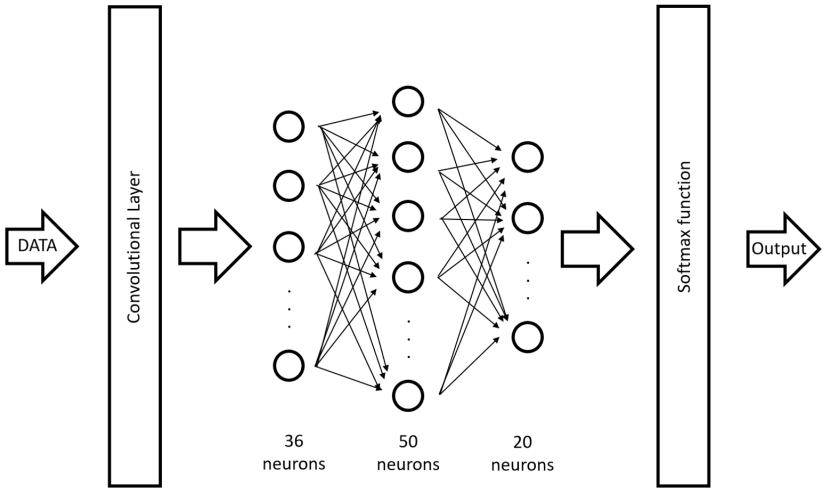

Figure 3.12: Behaviour recognition neural network

The dataset used to train this network was divided into two: the training and validation datasets. This model uses the validation set to implement the early stopping, a technique used to ensure that the network is not over-fitted to the trained data.

Another aspect to be taken into consideration, is the fact that the network cannot be trained to detect new unknown behaviours, as that would be a paradox. Therefore, the neural network is trained to label the 4 known behaviours, and, when the confidence on the result is low, the behaviour will be classified as an unknown one. The output of this algorithm is an array with four positions, one for each label, that represent how confident is the network in assigning the label to the data set. The sum of the four positions of the array is always one. Therefore, given a data set, if there is not confidence larger than 0.5 for any label, the data can be labelled as unknown behaviour.

4

Results and Discussion

Contents

In this chapter, the results of the algorithms and implementation mentioned in the previous chapter will be presented and discussed. It will start with a presentation of the testing scenarios and then, the order of chapter 3 will be followed.

4.1 Testing Scenarios

While testing the network architecture, clustering, routing and the network monitoring three different scenarios were used.

The most used, **Helsinki workday**, consisted of 56 nodes divided into 5 group of home, office and meeting spots following the workday movement model provided by the simulator. There were also 2 buses for each of the 8 routes available on the Helsinki scenario. In addition, the nodes buffer size is set to 64MB, 128MB or 256MB, and all buses have the largest buffer (256MB). The workday movement simulates the usual movement of a person, spending the night at home, then going to work for 8 hours and after that there are a few meeting points replicating stores, cinemas, restaurants and other places where people might go after work.

Another scenario using the Helsinki map was used, **Helsinki fast movement**. This scenario uses the same number of nodes with the same configurations. However, instead of the working day movement, the shortest path movement was chosen. This movement model assigns a set of points of interest to each group of nodes, and the nodes will go from one point of interest to the other in random order. With this movement model each node would only stay in the same spot for a few minutes, increasing the average movement of the network and reducing the time the network takes to converge.

At last, the **Barcelona** scenario is based on the map provided by the simulator. For this scenario there were a total of 30 nodes, 20 auto placed RSUs and 30 vehicles. In addition, all vehicles have a buffer set to 256MB while the RSUs' is 1024MB. Besides, the movement model used was the shorted path movement between points of interest. All the routing rules and communications interfaces were the same as for the Helsinki scenario.

4.2 Network Architecture

As presented in the previous chapter, two implementations of the RSUs were made. One where the user gives the number of nodes and the central position and the others are placed automatically; and the other where the positions were assigned one by one. To compare these approaches the following metrics were measured: the average number of nodes seen per RSU; the number of RSUs that never had contact with other nodes; and the frequency of contacts. The first metric indicates how close is the average of the RSUs to the centre of the network neighbouring graph, an RSU with a higher number

of nodes seen has more connections than the other. The second metric shows the number of RSUs that never made contact with a node, i.e., RSU that until the time when the date was acquired had not contributed to the network. At last, the frequency of contacts shows how closely the average of the RSUs are from crowded areas.

The following tests were done in the Helsinki workday scenario. All simulations were run 3 times and the results below refer to the average obtained, with a confidence interval at 0.95.

At first, a study regarding the number of RSUs needed for the auto-placing approach was done. The goal of this was to find where was the bottleneck point for this problem, thus, the minimum number of RSUs needed to obtain the best results. With this in mind, the simulation was run with five different configurations, 20, 30, 40, 50 and 60 RSUs.

The first test measured the coverage of the RSU network by counting the number of contacts with different nodes each RSU has done, as shown in Figure 4.1. This metric was chosen since it can measure how well the RSUs are positioned. The highest values represent a network where the average RSU have multiple contacts with different node, therefore, are placed next to main roads. In addition, if there are a lot of RSUs located at places without any movement, the average will also be lower. As seen in Figure 4.1, the coverage of the network was similar for all simulations.

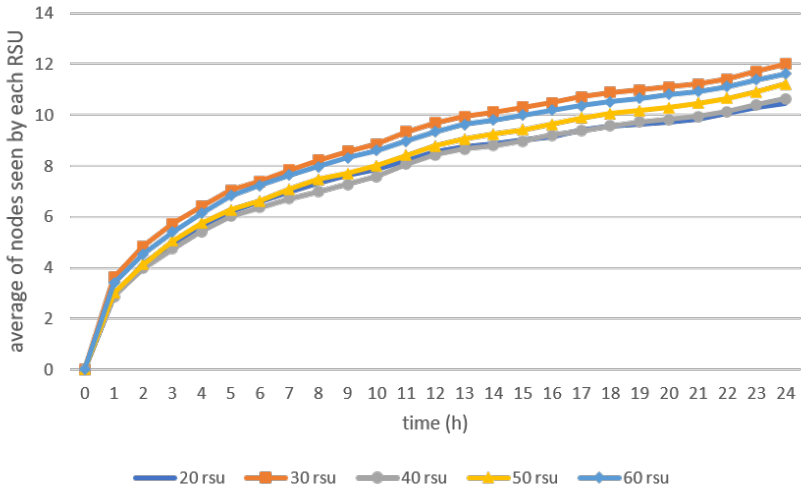

Figure 4.1: Average number of nodes seen by RSUs over time

As seen in the Figure above, the average number of nodes seen by each RSU, even though there are 56 nodes, is only 12. This is caused by the fact that the movement model used assigns a routine to each node, and therefore, there is a high probability of a node moving only in part of the map, leading to not making contact with many RSUs. Regarding the results obtained for each configuration, using only 30 nodes appears to have the best results. This fact is explained by the fact that having a reduced number of RSUs, the configuration also has fewer RSUs with zero contacts, allied by some chance that

some of the RSUs were placed closed to optimal places.

Regarding the second test, the percentage of RSUs that never had contact with other nodes over time was measured to identify how many RSUs were being used and were not contributing to the network. The corresponding results are shown in Figure 5.2 using a logarithmic scale.

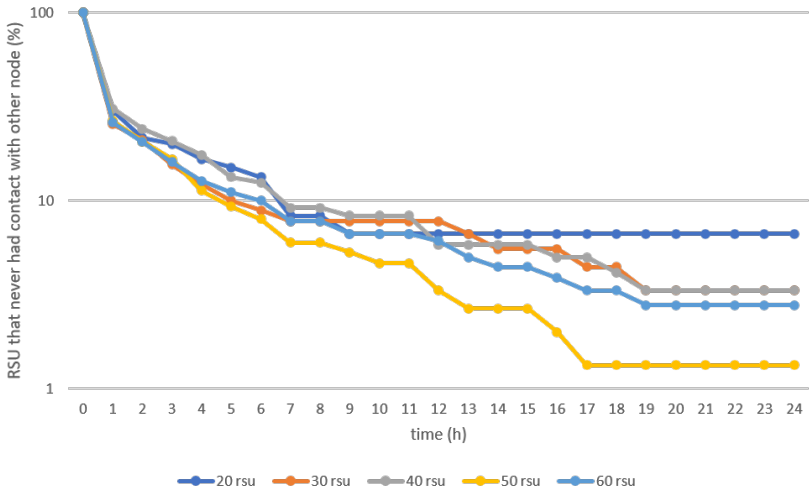

Figure 4.2: Percentage of RSUs that never had contact with other nodes over time

As far as the results presented in Figure 4.2 are concerned, after 7 hours, the percentage of RSUs that never had contact with other node is below 10% for all simulations. The Figure also shows a higher RSU percentage for the test with 20 nodes, however, 10% of 20 is just 2 nodes. In addition, the relation between the values of the simulations for 30, 40 and 60 nodes should be noticed, as the values are almost identical.

At last, the frequency of the contacts made by each RSU was also measured and Figure 4.3 shows this data for all different configurations tested (each one was tested 3 times). The higher the frequency, the better the RSU position found is. Therefore, if the average frequency is high, the average nodes are at an optimal position. In addition, it should be noted that the variation in the values obtained to the frequency of contacts is due to the movement model used.

As the results show (Figure 4.3), the architectures with the best frequencies were with 50 and 30 RSUs respectively.

To sum up, for these scenarios, the average of nodes seen by each RSU was the same. However, regarding the percentage of RSUs that were placed in irrelevant places (Figure 4.2), that with 30 nodes, the result was already the same as the one with 60 RSUs, only beaten by 50 nodes but by a small margin. In addition, the frequency of contacts of the scenarios with 30 and 50 nodes were the best results. Therefore, the number of nodes that had the best performance was for 50 nodes, nevertheless, using 30 nodes the results were similar and it is more economical.

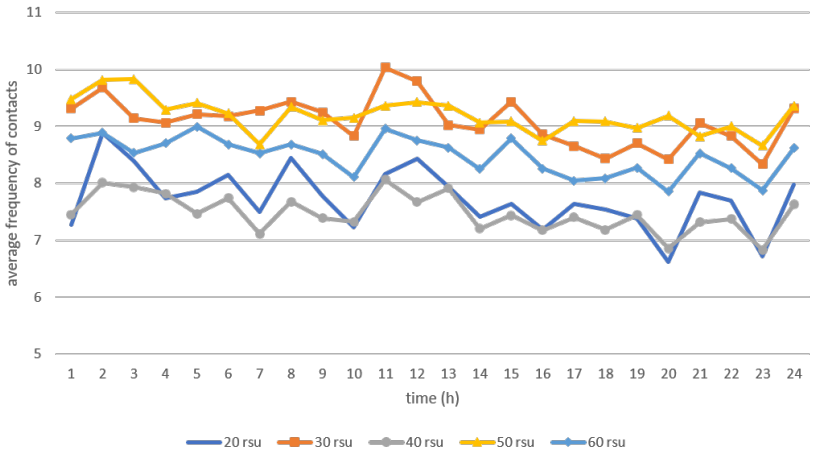

Figure 4.3: Frequency of RSU contact over time

At last, the two approaches were compared. In light of the above and based on the fact that the number of nodes has a huge impact on the price of the solution, the number of nodes chosen for the first implementation was 30 RSUs, the same number as used on the second approach.

Figure 5.4 presents a comparison of the cumulative number of nodes seen by each RSU along 1 day for the two RSU placement approaches under study: automatic placement and manual placement. As shown, the average number of nodes that made contact with each RSU is always increasing in both implementations. However, when the nodes where hand placed in strategical locations, the average number of nodes seen by each RSU is always higher, presenting a tendency of more 30% than the automatic approach.

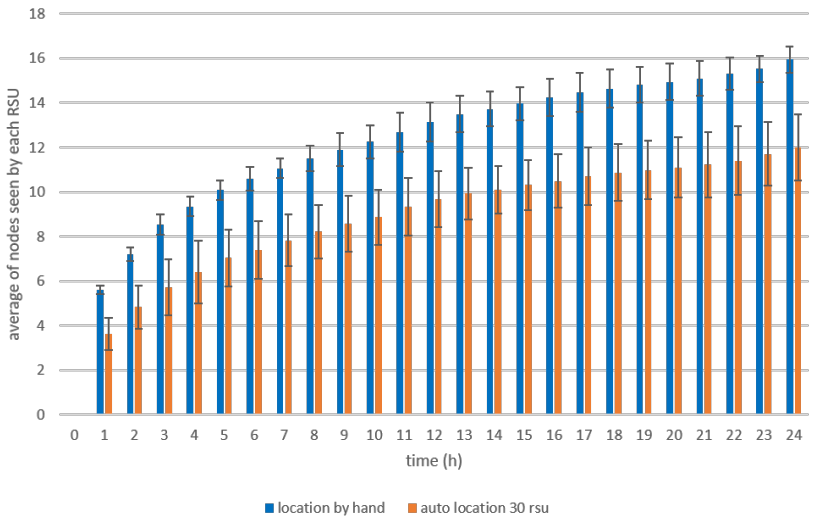

Figure 4.4: Average number of nodes seen by 30 RSUs over time

Regarding the number of RSUs that never had contact with a node, Figure 4.5 shows a comparison

between the values of the two approaches per hour. In the first analysis, a tendency to zero can be seen for both placement approaches. However, when placed automatically, the RSUs needed more than 24 hours to reach that value. On the other hand, when placed in strategical locations, zero is reached in only 4 hours.

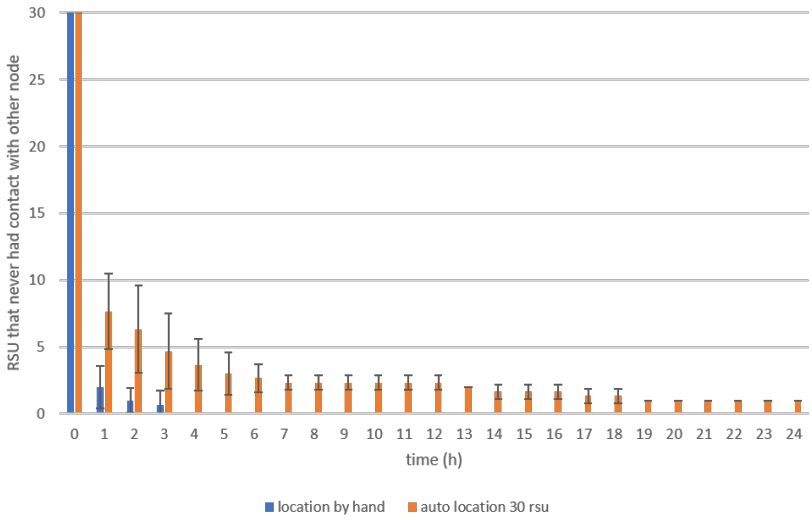

Figure 4.5: Number of RSUs that never had contact with other nodes over time

As far as the frequency of contacts is concerned, i.e. the average number of contacts per hour made by each RSUs, in Figure 4.6 is presented a comparison between the two approaches. This figure shows the main difference between the two techniques and the impact of choosing optimal locations. When the RSUs are hand located, the frequency of contacts is more than two times higher than when using the auto placing approach. Consequently, the edge can have a major influence on the network.

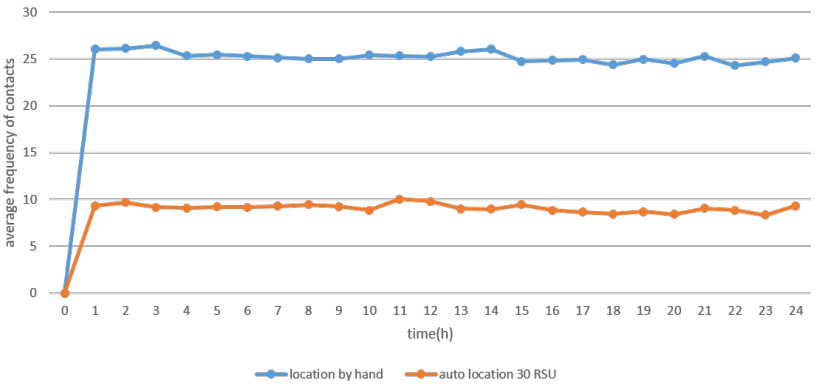

Figure 4.6: Frequency of RSU contacts over time

To conclude, RSUs have a huge impact on the performance of the network and they are desired to reach the highest number of nodes possible. In addition, the complexity of several IoV applications, such

as the clustering algorithm presented in this thesis, also, the cost of the deployment is conditioned by the number of RSUs.

4.3 Clustering

4.3.1 Clustering Evaluation

As discussed before, the network will divide the nodes into clusters. The main goals of dividing network nodes into clusters is covering as many nodes as possible and creating the clusters as stable as possible. In addition, the larger the number of clusters, the more specific information a cluster gives about its nodes. Nevertheless, the size of each cluster shall be taken into consideration, i.e. the number of clusters shall be much lower than the number of nodes.

Clusters were made using a k-means algorithm based on the social relations of the nodes. To evaluate the performance of the algorithm, the number of nodes eligible for clustering and the number of cluster changes per hour were measured during several simulations.

The following test was done in the Helsinki workday scenario. In addition, there are also 30 RSUs that were hand placed to improve the performance, as shown in the previous section. Each simulation was done 3 times with different seeds and the results presented refer to the average of the results obtained.

Regarding the number of nodes covered by the algorithm, it is important to mention that only nodes that already made contact with an RSU are known by the edge and therefore eligible for clustering. In addition, this result does not depend on the number of clusters. The algorithm is based on the relations with the RSUs, therefore it only depends on their position. Nevertheless, it is an important metric that should be taken into consideration when studying the clusters, therefore the results are presented in Figure 4.7.

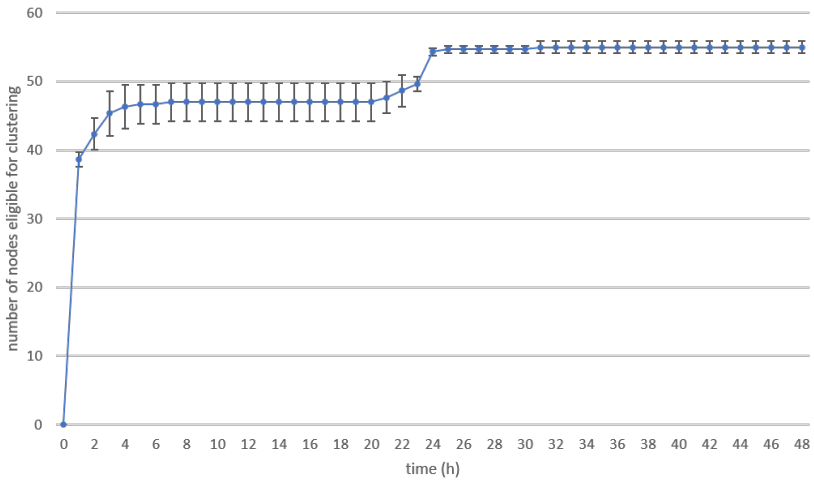

Figure 4.7: Number of nodes eligible for clustering over time

As expected, during the first two days, the nodes start to make contact with the RSUs which leads to an increase in the number of nodes covered. The most significant changes occur in the morning and in the evening that is when the nodes are supposed to move between home, office or meeting spots (following the workday movement model). During the work time, the values are much more stable. These values and changes should be taken into consideration when studying the stability of the algorithm.

Although it cannot be seen in Figure 4.7, there are a few times that the value decreases. This is explained by a node who did not make contact with an RSU and its relation has decayed below the clustering threshold mentioned in chapter 3. When this happens, the nodes affected are excluded from the algorithm and are not assigned to any cluster.

To further analyse the clustering algorithm, a cluster changes metric is defined where all cluster changes count as one, when a node does not have a cluster and joins one and when a cluster changes from one cluster to another, both count only as one cluster change.

In the simulation with only one cluster, all nodes eligible for clustering belong to the same cluster. Therefore, all changes will be nodes that were added to the clustering set. In addition, based on the total number of nodes in the network, the number of clusters was varied between one and eight.

Once the number of clusters and the number of nodes covered by the clustering algorithm are independent,and the addition of a new node to the algorithm will occur exactly at the same time. Therefore, with the objective of presenting clear data, the changes related to nodes that were added to the algorithm were subtracted from the number of cluster changes per hour.

In Figure 4.8, each colour represents a different simulation scenario, each vertical bar shows the sum of nodes who had a cluster change during each hour. For example, during the second hour, there were a total of 65 cluster changes, but when using two clusters, there was just an average of 7 changes. In addition, as it can be seen, there is not any cluster change counted in the first hour, this is because

all the changes in this period were nodes making the first contact with the RSUs.

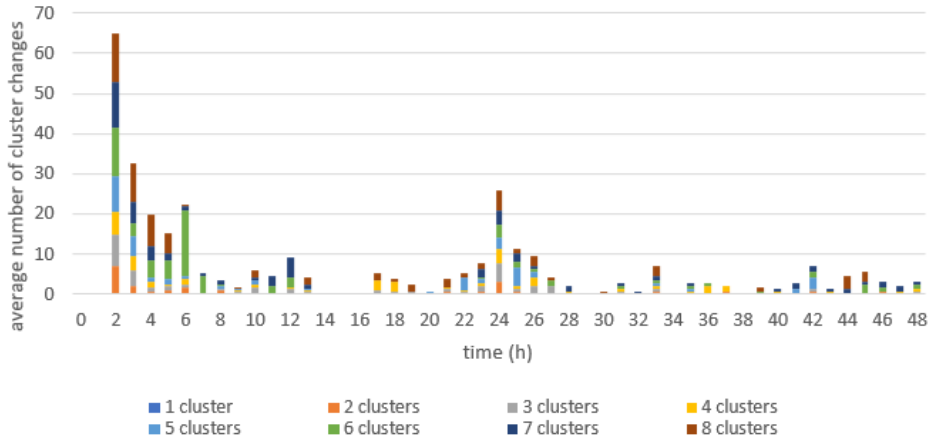

Figure 4.8: Sum of average cluster changes over time

To give a clearer image of when the network starts to converge, an average of all these simulations were done and it can be seen in Figure 4.9. In addition, in this figure, in contrast with the previous one, all cluster changes are represented, both nodes who connect for the first time with an RSU and nodes who change from one cluster to another. That is why, in the second figure, there are values on the first hour of the simulation, all these cluster changes are nodes who were not registered for clustering and made the first contact with one RSU during this time slot.

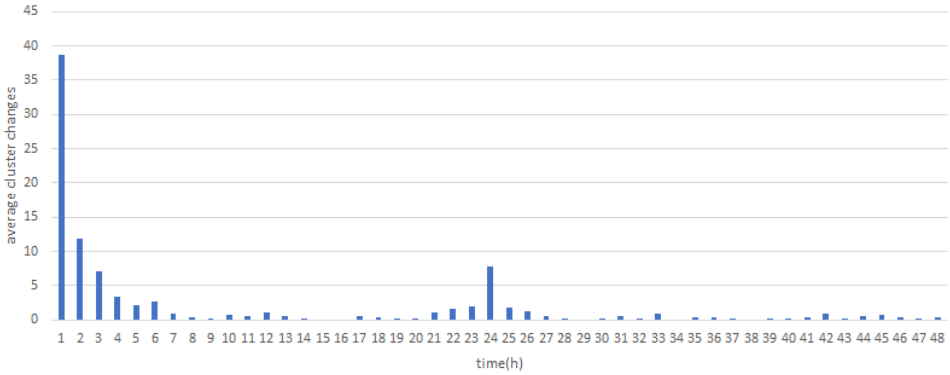

Figure 4.9: Average cluster changes over time

As seen in Figure 4.9, it takes an average of six/seven hours to reach a stable point. Furthermore, with the increase in the number of clusters a decrease of the stability can be seen (Figure 4.8). However, as it can be also be seen that even for the less stable configurations, after 48 hours the number of changes per hour tends to less than 1, which shows that the approach developed leads to a stable clustering solution as desired.

In addition to the above studied approach, the stability of clustering based on proximity was also

tested. The goal of this test was only to give a comparison with the most trivial approach to the problem, therefore, only a superficial analysis was done. Here, the contacts that a node had during two consecutive time slots were compared and the number of changes counted. During the first slot the number of changes was 11, then dropped to 8 and every time the node moved the values were similar. When the node was not moving these values were near 1. Nevertheless, these values are just for one node, which means that it represents only 1/50 of the cluster changes. Therefore, the difference between the stability of the social based approach and this one is evident.

All in all, when clustering a network of vehicles, there are some aspects to take into consideration, such as, what are the clusters representing, the size of the clusters and the network, and the stability that the algorithm used will provide. As presented, clustering based on social relations leads to stable results, and it is advisable to be used when the nodes are always moving and have some movement patterns, such as in the IoV case.

4.3.2 R-privo

As far as the routing algorithm is concerned, it should be noticed that the algorithm is an adaptation of another one built to DTNs, the Privo algorithm [8].

To evaluate the performance of the R-privo, the following metrics were considered: delivery probability, probability of a message reaches its destination; overhead ratio, excessive messages traded; average latency, the delay between the creation of a message and its delivery to the destination; average hop count, the number of nodes needed to deliver a message; and average buffer time, the time that a message stays in the buffer of a node.

The simulations done in this area were slightly different from before. As seen in the previous section, it takes almost 8 hours for the network to converge. In addition, using this movement model most nodes stay still for long periods of time. Therefore, instead of using the Helsinki workday scenario, Helsinki fast movement with 30 hand placed RSUs was chosen for this study. In this way, all nodes are eligible for clustering after one hour and the relation between housing, working and meeting locations are preserved. Nevertheless, taking into consideration the number of nodes in the network as well as the results obtained in the previous section, the clustering algorithm was chosen to run with 6 clusters.

To evaluate this algorithm three tests were made for different forwarding policies. One using the direct single copy policy (DSCP) and two using limited multi copy (LMCP), the first one with a limit of six copies and at last with a limit of ten. The results are presented in the Table below.

policy	delivery rate	overhead	latency	hopcount	buffertime
DSCP	0.6288 $\pm$ 0.077	0 $\pm$ 0.0	9682.28 $\pm$ 996	1 $\pm$ 0.0	9682.28 $\pm$ 996
LMCP 6	0.8172 $\pm$ 0.077	5.53 $\pm$ 0.65	6224.7 $\pm$ 972	1.8 $\pm$ 0.018	4188.45 $\pm$ 522
LMCP 10	0.8486 $\pm$ 0.05	8.62 $\pm$ 0.59	6343.21 $\pm$ 1367	1.86 $\pm$ 0.03	3458.85 $\pm$ 720

Table 4.1: Routing results

The discussion below will focus on the delivery rate and the overhead ratio of each solution. The values seen in Table 4.1 are according to what was expected hence an increment of the number of copies leads to a boost on the delivery rate, however, this increment also has a negative impact on the overhead. Comparing the gain of having 10 copies over 6, the overhead has increased 55% while the delivery rate has only raised 3%.

In light of the above, the routing algorithm using limited multi copy policy of size 6 was chosen to represent the proposed algorithm. This evaluation was done by comparing the delivery rate and the overhead ratio with four different algorithms, epidemic, prophet, bubblerap, and privo.

The following tests were done under the same circumstances as the comparison of the forwarding policies. All tests were run 3 times with different seeds to generate pseudo random movements and all results presented are the average of the 3 simulations. In addition, all simulations have both vehicles and RSUs, however, while in the R-privo, the RSUs have a different behaviour than the other nodes, on the other algorithms, the only difference between them is the buffer size and the movement, the RSUs are static.

In the first instance, the delivery rate of each algorithm was compared. As shown in Figure 4.10, all algorithms have delivery rates between 78 and 93%. Prophet had the best performance, followed by bublerap, R-privo, privo and epidemic respectfully.

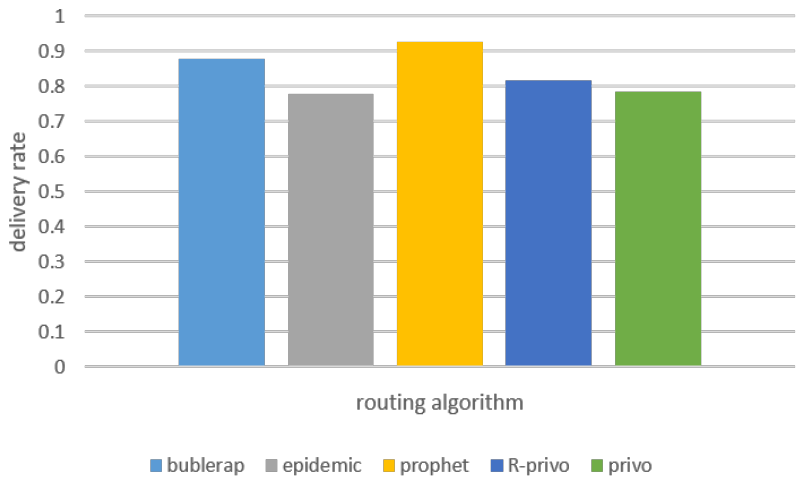

Figure 4.10: Delivery rate of the 5 routing algorithms

As far as the overhead is concerned, there is an enormous difference between the algorithms, with values ranging between 2753 and 5, with the R-privo and the privo algorithm presenting the best overheads by far. This disparity of values was expected hence the epidemic and the prophet do not assign any limit to the number of copies, the first one copies the message to every node it makes contact with and the other one copies to all nodes that have a better metric than him. As far as the bubblerap, has its restrictions on whom to copy to. However, with these configurations, the privo and the R-privo have a limit on the number of copies of 6. Figure 4.11 shows the overhead of each algorithm as a table printing the values. To encompass all values in the same figure, a logarithmic scale was used.

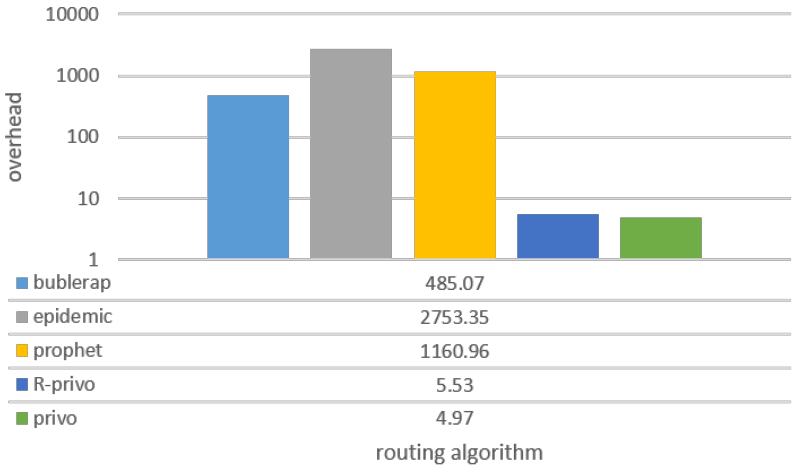

Figure 4.11: Overhead ratio of the 5 routing algorithms

At last, the same tests were done using the Helsinki workday scenario with 30 hand placed RSUs. This scenario represents a situation closer to reality with fewer movement, thus fewer contacts than the situation before.

As seen in Figure 4.12, the delivery rate of the algorithms has reduced. This change emphasises the best algorithms to use in a situation like the one described. In addition, the bubblerap, epidemic and prophet had a huge drop on their metrics.

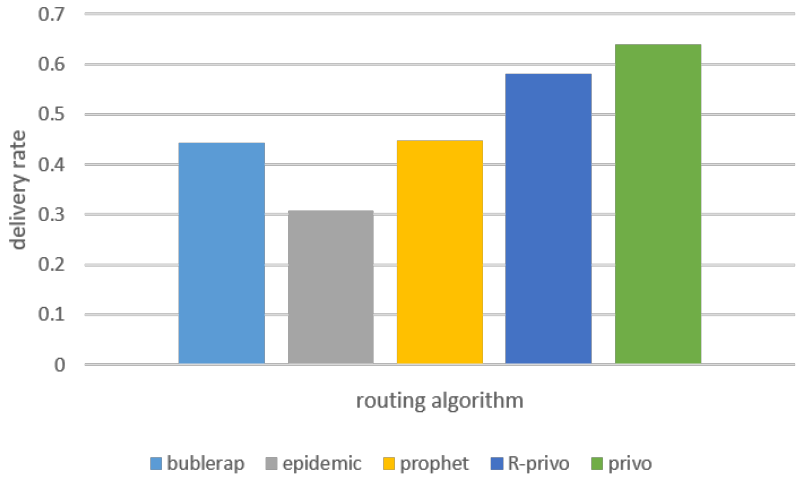

Figure 4.12: Delivery rate of the 5 routing algorithms (workday)

As far as the overhead is concerned, changing to this movement model the changes did not have the same impact as on the delivery rate, hence the comparison between the algorithms is similar.

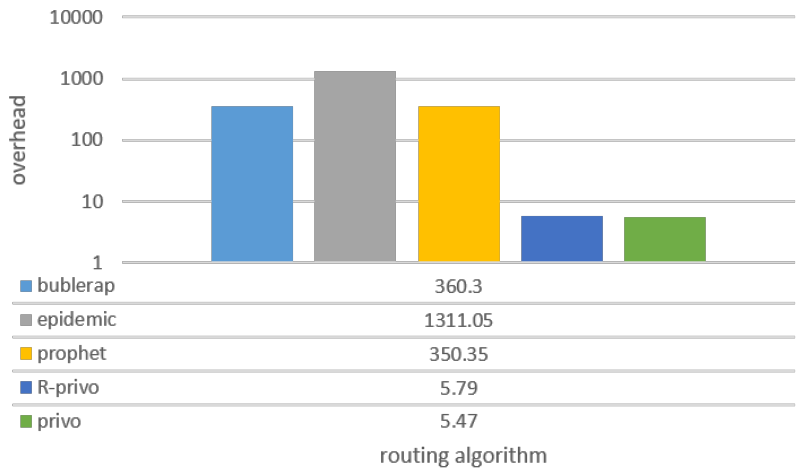

Figure 4.13: Overhead ratio of the 5 routing algorithms (workday)

As seen in the figure above, in the second situation, the R-privo presented worse results than the privo. This is due to the fact that one of the main differences between these algorithms relies on the restrictions to transfer a message, the R-privo will only share the messages within its cluster while the privo will share them with any node with a greater chance to meet the destination.

To conclude, the R-privo is an algorithm that was developed to take advantage of the architecture of the IoV, it can be seen as an evolution of the privo algorithm to this paradigm. The R-privo has a delivery rate in the same range of the other standard routing algorithms with an overhead much lower. In addition, this algorithm needs, on average, only 1.8 hops to deliver a message when the others need

3.0, 4.7, 3.8 and 1.9 for bubblerap, epidemic, prophet and privo respectively. On the other hand, both the latency and the average buffer time of this algorithm are much worst than the others, notwithstanding, this is to be applied to a network of vehicles where each node have more resources that a usual node in a network where these algorithms are used.

4.4 Intrusion Detection System

The monitoring of the network consists of two different algorithms, one that detects anomalies and the other that tries to classify that anomaly.

This study was done over several simulations when some nodes were programmed to misbehave and the edge would try to detect. Every time the edge made a decision about a node, this would be reported and stored in a confusion matrix. In a confusion matrix, each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class. This means that the number on the first row, third column represent the number of cases that belong to the first class but were labelled as third. From this matrix, the number of true/false positives/negatives can be taken.

In this work, a true positive is when an attacker situation is correctly labelled and a false positive when a normal is classified as a misbehaving one. A true negative is when a normal node is labelled correctly and a false negative is when an attacker situation is mislabelled.

Every time the first algorithm detects an anomaly, the data will also pass through the second one to classify the behaviour.

Regarding the classification of behaviours, in the following results, they are identified by an ID as presented in Table 4.2.

ID	Behaviour
0	Sybil attack
1	Id spoofing
2	Node not working
3	Normal behaviour
4	New behaviour

Table 4.2: Behaviour id's

Before using it in the real simulation, the algorithms were trained and tested. The training of the algorithms consisted of uploading a data set and the algorithm updated its weights to best perform on the given data. As mentioned in the previous section, this data was obtained in several simulations with different parameters. In addition, the anomaly detector takes approximately 6 minutes to train while the behaviour identifier takes 2 minutes.

Following this training, the algorithms were tested with a set of already known data (independent from the training set). The first algorithm, the anomaly detection system, was trained to extract the main features of a dataset, and based on those, replicate the input data. During the evaluation of this algorithm, it was capable of reproducing the input data with, in the worst case, only a mean squared error of 0.2. The second algorithm, the misuse detector, whose goal is to identify which misbehaviour is occurring, was tested with 150 data samples and the following confusion matrix was obtained. It is also important to mention that this set does not have examples of data labelled as a new behaviour, therefore, the outcome will be a matrix 4×4.

$$\begin{array}{c} \\ 0=0 \\ 1=1 \\ 2=2 \\ 3=3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 58 & 0 & 0 & 0 \\ 0 & 33 & 0 & 0 \\ 0 & 0 & 27 & 0 \\ 0 & 0 & 10 & 22 \end{pmatrix} \end{array}$$

As the confusion matrix above shows, during the test, the algorithm was able to identify the syblil attack and the id spoofing every time they appeared, 58 and 33 times respectively. As far as the other two behaviours are concerned, the algorithm also identified the node not working every time it appeared, however, sometimes (10 times out of 32), a normal behaviour was identified as a node not working. This result is acceptable, once there is a fine line between the data that represents a node that is not working and a completely isolated node.

During all the following tests, the attackers will behave maliciously from the beginning until the end of the simulation. Their data will be monitored every time they made contact with an RSU and the algorithm will not take into consideration any label given to that node before, i.e, in this study, instead of testing a node based on all its history, the tests will only take into consideration the data sample acquired at each moment. Therefore, they are presented below as different attack situations. Usually, each simulation has 5 misbehaving nodes and during the simulation time, each node is inspected by the algorithm 20 to 30 times, depending on its movement patterns.

After all the training, the algorithms were deployed to the simulations. In the first instance, the algorithm was used to monitor a network where all nodes had a normal behaviour. To this analysis, 1500 tests were done over 106 nodes. From these 1500 tests, 117 were labelled as a potential anomaly, however, from this 117, the second algorithm labelled 108 as normal behaviour. This indicates that from 1500 samples, only 9 were false positives, only 0.6%.

Considering that at the beginning of the simulations all the nodes signatures are the same, if the algorithm is applied, all nodes will have the same classification. Over the simulation, the signatures

will become more and more distinct. Therefore, in the first instance, the time that the system needs to warm up, i.e. there is a difference between the signature of a normal node and an attacker, was studied. With this objective, several simulations focusing the first hours of the simulation were done, these simulations concern the first two behaviours (sybil and id spoofing) with distinct parameters. In this study, the evolution of the true positives, false positives and false negatives were collected and once the values of the true positives is larger than the values of false negatives and false positives, it indicates that the system is warmed up and the algorithm is ready to run.

As seen in figures 4.14 and 4.15, and as expected, in the beginning of the simulation, the number of false positives and false negatives is much higher than desired. These values will decrease over the warm up period.

Regarding the sybil attack, during all these simulations there were not any false positive, therefore, only the evolution of the true positives and false negatives will be taken into consideration for the choice of the warm-up period. During the first hour, the number of nodes evaluated was lower since before any classification, a node had to connect with an RSU at least 3 times. Nevertheless, during the second hour, the number of tests has increased and a clearer image of the network state can be seen. During this hour the number of false negatives and true positives was approximately the same, However, from the third hour, the number of false negatives starts to be lower than the number of true positives, with the first one decreasing over time and the last one increasing, as seen in Figure 4.14. In addition, the number of false positives was always zero during these simulations.

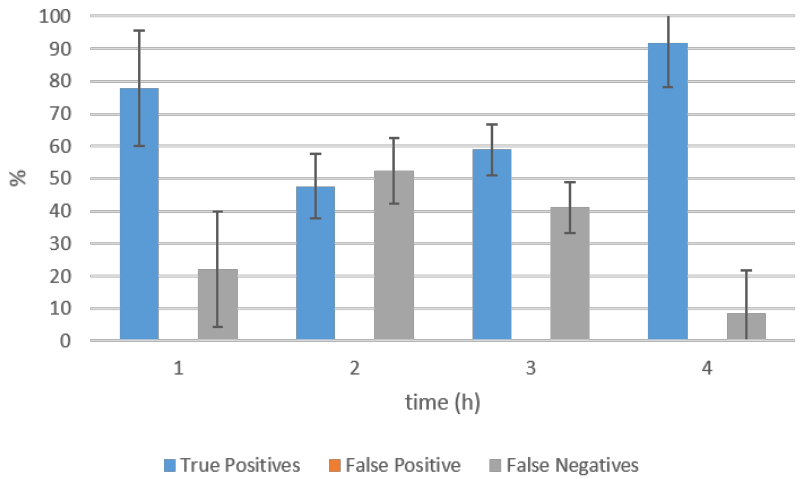

Figure 4.14: Evolution of classification metrics for the sybil attack

As far as the id spoofing attack is concerned, and as seen in Figure 4.15, during the first hour, the number of false negatives was much larger than desired, however, during the following hour, the number of false positives had a huge drop while the number of true positives had started to grow. During the

third and the fourth hour, all values look like tending to a value with lower deviations and a lower error margin.

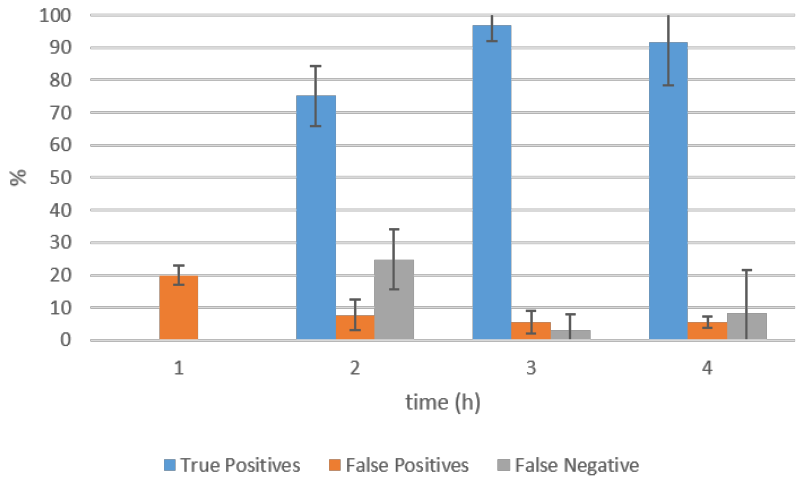

Figure 4.15: Evolution of classification metrics for the id spoofing attack

To conclude, after 3 hours the signatures of the nodes are distinct enough to start the classification, therefore, the warm-up period chosen was 10800 seconds (3 hours).

4.4.1 Sybil attack

With all tests and simulations for a normal situation done, it was time to start testing against misbehaviour. At first, the sybil attack was studied. For this, several simulations were performed, varying the movement model of the attacker and the number of replicas it creates. Among all these simulations, the monitoring system had run 597 attacks, from these, 407 were labelled as having an anomaly and were studied by the behaviour identifier who labelled all 407 as sybil attackers. This means that once an attacker is captured by the anomaly detector, the behaviour identifier will be able to label this attack correctly. In addition, this false negative rate (approx. 32%) derives from the similarity of an attacker with a node with an important role in the network, i.e. high Ego betweenness centrality. Moreover, during this study, a total of 8000 nodes were monitored and only 16 were mislabelled as sybil attackers which leads to a false positive rate of only 0.22%.

To sum up, for this attack, the algorithm achieved the following results:

- true positive rate approx. 68%
- false positive rate approx. 0.22%
- true negative rate approx. 99.78%

- false negative rate approx. 32%

During the study above, the fact that when the attacker creates more replicas, it is easier to detect was confirmed. When testing with 3 replicas, the false negatives were significantly higher than the most used configuration, 5 replicas.

4.4.2 Identity Spoofing

As far as the id spoofing attack is concerned, when a node identified as an RSU makes contact with the edge, and this node is not registered as an RSU, the monitor system already knows that something not normal is occurring. Therefore, it will always run the behaviour identifier.

During the testing period, the algorithm run the data of 18000 tests, from which 389 were attack situations. Considering the attackers, 370 were labelled as attackers. In addition, during these tests, there were 100 false positives, i.e. nodes that were labelled as attackers and had normal behaviours. Therefore, the following results represent the performance of the algorithm against this behaviour.

- true positive rate approx. 95%
- false positive rate approx. 0.6%
- true negative rate approx. 99.4%
- false negative rate approx. 5%

4.4.3 Nodes Not Working

Regarding the detection of nodes not working, the difference between a node not working and a node whose role in the network is not important is small. I.e. a node that at a given time stopped working and a node who went to an isolated place and cannot connect to anyone result in a similar behaviour. This behaviour was not tested on RSUs hence if one of these nodes was not working it would be unable to connect with the edge and therefore, it was assumed that the edge would be able to detect this anomaly without the algorithm.

Regarding the results obtained, for the node not working behaviour, the true positive rate was only around 50%. As mentioned before, this behaviour has a signature similar to that of a normal one and even when the anomaly detector captures these nodes, it is not guaranteed that the behaviour identifier will not label it as a normal node with a normal behaviour. To have greater certainty about this behaviour, it would be advisable to wait for at least one more iteration on the algorithm and check if the classification stays the same.

On the other hand, the false positive rate on this behaviour, just like on the others, is close to zero, in a total of 5000 evaluations, only 33 were misclassified as a node not working.

To conclude, for this behaviour, the algorithm had the following performance:

- true positive rate approx. 52%
- false positive rate approx. 0.8%
- true negative rate approx. 99.2%
- false negative rate approx. 48%

4.4.4 Results validation in other scenarios

Helsinki Fast Movement Scenario

Following the study on a scenario similar to the one where the algorithm was trained, the algorithm was deployed to another scenario, the Helsinki fast movement scenario. This new test environment was built with the same map, however, the behaviour of the nodes is completely different. In this scenario, all vehicles are moving between points of interest without stopping for more than 30 seconds. Regarding the RSUs, their location stays the same. The scenario chosen is similar to the one used on the first study of the routing algorithm.

When the simulation was run in the new scenario, the results were slightly worse, as expected. However, the algorithm was still able to successfully identify some of the behaviours tested.

As far as the sybil attack is concerned, there were 3700 situations tested where 186 were attacks. From this set, 95 were correctly classified and 91 were labelled as normal behaviour by the anomaly detection system. From the nearly 3700 tests, there were only 3 false positives (normal nodes identified as attackers). Which leads to a true positive rate of 51% and a true negative rate near 99%. In addition, for this attack, all attackers identified as possible anomalies were still labelled correctly, which shows that, for this behaviour, the algorithm which had the biggest impact was the anomaly detection system.

Regarding the id spoofing attack, the impact of the scenario change was less, the true positive rate was not as higher as the 95% obtained before but was near 91%. Regarding the false positive rate, has raised to approximately 3%.

At last, when running the algorithm to identify the nodes that were not working, in general the results were similar to the first scenario, a true positive rate of 52%. However, as mentioned above, the signature of this behaviour is similar to the signature of a normal behaviour.

As seen above, when changing the scenario, the false negative rate of the sybil attack is much higher than the one where the data was collected. This is due to the fact that the anomaly detection system works by detecting deviations on the expected data. When changing a scenario, all the data will be different, therefore, the threshold that differentiates an anomaly and normal behaviour will be much higher, leading to a less meticulous analysis. This increase of the false negative rate in the two latest

behaviours is a lot less expressive, since when a node is behaving like a node not working or an id spoofing attacker, the algorithm will run the behaviour classification. Which will not take in consideration what is a normal behaviour of the network and how big the deviation to the normal is. Nevertheless, as shown by the results obtained, the performance of this algorithm is slightly lower compared with the original scenario.

Barcelona Scenario

In addition, to prove that with the data collected the algorithm can be deployed in any scenario, one more simulation was done using Barcelona scenario.

As seen in the previous simulation, the change on the usual node's behaviour had already had a significant impact on the anomaly detection system, nevertheless, the impact on the behaviour identifier was much lower. In this sense and considering that for the id spoofing attack only the behaviour identifier is used, only this attack was chosen to be tested.

In this simulation, there were 500 tests done, where 45 were attack situations, from this 45, 29 were correctly labelled, leading to a 64% true positive rate. During the simulation, there were also 34 false positives, which results in a 7.5% false positive rate.

In addition, the monitoring system was not trained with data from this scenario. The Barcelona map scenario is less detailed when compared with the Helsinki one, since the first one only has a portion of the city centre and the majority of the map is merely the main roads. Thus, there is a high number of roads in the city centre (higher than the city centre of the Helsinki scenario). However, on the rest of the majority of the map, the number of roads is much lower and the portion of areas where two nodes can make contact is less than on the Helsinki scenario. All the differences between these scenarios will have a huge impact on the performance of the monitoring systems. Nevertheless, the monitoring algorithm is still able to detect misbehaving nodes.

From the two latest simulations, the variation of the Helsinki scenario and the Barcelona scenario, the monitoring system has shown a huge dependency on the data used to train. Notwithstanding, it is shown that the algorithm can be deployed to any environment.

To conclude, the monitoring system has a high success rate identifying the attacks above and despite the results obtained for the nodes not working, the algorithm can be a huge help when managing a network. As expected, the anomaly detection system has a strong dependency on the training scenario, while the behaviour identifier has more flexibility. Nevertheless, the performance improves with the training of the algorithm, and, to obtain the best results, the algorithm should be trained with data of the network where it is deployed.

In a real world application, the algorithm should be collecting data, and from time to time, it should add the data collected to the training set. By doing so, the algorithm would learn more about the network

it is monitoring and it will be more suitable to detect any anomaly on it. Moreover, to increase the accuracy of classification, the algorithm could group the classifications by the respective node, thus, combining consecutive measurements to reduce uncertainty.

In addition, an increment on the deviation to the normal behaviour could be seen on all nodes when there is a node misbehaving, proving that even one node misbehaving has an impact on the network as a whole.

5

Conclusion and Future Work

Contents

5.1 Achievements and Contributions

The main objective of this thesis was to develop an edge-based monitoring system to the internet of vehicles. To achieve this, a clustering algorithm for these networks, a routing algorithm to ease the communication between nodes and a deep learning algorithm to detect and identify possible attacks and misbehaviour were also developed.

The project was developed with the ONE, a simulator designed for opportunistic networks and therefore the incorporation of the IoV architecture had to be done. Consequently, a new type of node had to be added to the simulator, the RSUs. In addition, two new communication interfaces were also added, one to enable the communication between vehicles (V2V) and another for the communication between vehicles and RSUs (V2I). Moreover, a study concerning the impact of the number and location of the RSUs was done, which lead to the conclusion that the RSUs should be strategically placed to increase their impact on the network.

Following the above mentioned implementations, the simulator could run all the nodes needed for the IoV with the respective communication interfaces. However, a routing algorithm that could comply with the IoV paradigm was needed. With that objective, the R-privo was developed, an algorithm based on the privo algorithm to take advantage of the edge layer and the location of the RSUs.

In addition, the R-privo also divides the network into clusters by their social relationships, grouping the nodes that most likely will share the same connections. While studying the clustering approaches on the IoV, two metrics were used, similarity and ego betweenness centrality. During this approach, higher importance was given to the similarity when choosing the cluster head. All in all, clustering based on social relationships between nodes was elected as an advisable approach to this problem, providing a stable solution that due to the choice of the RSUs as anchors, is also robust to an increment of the number of nodes.

In addition, the R-privo, even though being built with more constraints to comply with the IoV paradigm, was able to beat the traditional algorithms (epidemic, prophet and bubblerap) and reach the same performance levels of the privo algorithm, presenting itself as an ideal algorithm for these networks. Furthermore, the R-privo's routing rules are strongly related to the cluster of each node, therefore, an increment on the number of clusters will decrease the delivery rate of the algorithm. Besides, when deciding if a node has a higher chance of delivering a message, the algorithm will, in a first instance, compare the mean time to encounter and the similarity to the destination node and only later take into consideration the ego betweenness centrality of each node. When testing the R-privo, the limited multi copy policy was also chosen to provide a balance between the delivery rate and the overhead of the algorithm.

At last, with the environment set to properly simulate an IoV network, the edge based monitoring system was developed. This system is formed by two different deep learning algorithms, one to detect anomalies in the network and the other to identify those anomalies.

The first one, the anomaly detection system, has the purpose of detecting deviations from a normal behaviour in the network. It is very useful to identify nodes who may be misbehaving, however, this algorithm cannot distinguish the different misbehaviours.

The second algorithm, the behaviour identifier, is able to identify any known attack, i.e. the algorithm has to be taught how to detect each studied attack, therefore, the identifier cannot detect any new attack.

The monitoring system is built to take advantage of both algorithms. In addition, the choice of the data to be collected and used to evaluate each node was also studied as well as the architecture of each algorithm.

At last, in a scenario similar to the one used to train, the monitoring system can detect and identify 68% of the sybil attacks, 95% of the id spoofing attacks, and 52% of the nodes not working. In addition, the behaviour identifier was able to correctly classify all the Sybil attackers that were sent by the anomaly detector. Nevertheless, the poor results obtained for the nodes not working is due to the fact that this behaviour and a node who are in an isolated zone have similar signatures. Notwithstanding, the monitoring system can identify the above mentioned behaviours and can be a great tool for any network manager.

To conclude, in this thesis the ONE simulator was adapted to work with IoVs, an algorithm was developed to comply with the network paradigm and to take advantages of what the IoVs add to the system, and an edge-based monitoring system capable of identifying several behaviours was developed, fulfilling all the objectives presented in chapter 1.

5.2 System Limitations and Future Work

With more time and resources some improvements could be done.

As far as the location of the RSUs is concerned, an optimisation problem could be developed to find the optimal location of each RSU.

Regarding the clustering algorithm, the G means algorithm [44] could be tested and compared with the one used. The G means algorithm has the advantage that it will choose the most suitable number of clusters, providing greater flexibility to the algorithm.

At last, the monitoring system could be retrained from time to time with live data, adapting the system to the network it is monitoring. In addition, the classifications of each node could be saved to have the evolution of each node over time as it would also increase the certainty of a given classification by comparing it with the previous ones. The algorithm could also be able to act to reduce the impact of a misbehaving node in the network. Moreover, some behaviours could be added to the system, for example, the black hole or grey hole attack. Finally, the complexity of the deep learning algorithms could be greater, which would improve the performance of the monitoring systems but would require more

computation power.

Bibliography

[1] K. Curran, "Image compression using autoencoders in keras," Available at https://blog.paperspace.com/autoencoder-image-compression-keras/ (2020/09/03).

[2] F. Yang, J. Li, and T. Lei, "Architecture and key technologies for internet of vehicles: a survey," *Commun. Inf. Netw. 2, 1–17 (2017)*, 2017.

[3] J. A. Onieva, R. Rios, R. Roman, and J. Lopez, "Edge-assisted vehicular networks security," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8038–8045, 2019.

[4] K. Sha, R. Errabelly, W. Wei, T. A. Yang, and Z. Wang, "Edgesec: Design of an edge layer security service to enhance iot security," *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 81–88, 2017.

[5] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*. CRC Press, 2016. [Online]. Available: https://books.google.pt/books?id=1-FY-U30lUYC

[6] A. F. Gad, "Knuth: Computers and typesetting," Available at https://kevincurran.org/security/edge-computing-use-cases-must-be-driven-by-business-value/ (2020/03/31).

[7] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[8] N. Magaia, C. Borrego, P. Pereira, and M. Correia, "Privo: A privacy-preserving opportunistic routing protocol for delay tolerant networks," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, 2017, pp. 1–9.

[9] S. Yousefi, M. S. Mousavi, and M. Fathy, "Vehicular ad hoc networks (vanets): Challenges and perspectives," in *2006 6th International Conference on ITS Telecommunications*, 2006, pp. 761–766.

[10] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez, "Internet of vehicles: Architecture, protocols, and security," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3701–3709, 2018.

[11] W. Wei, A. T. Yang, W. Shi, and K. Sha, “Security in internet of things: Opportunities and challenges,” *2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI), Beijing*, pp. 512–518, 2016.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[13] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, “Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading,” *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, June 2017.

[14] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[15] A. Vahdat, D. Becker *et al.*, “Epidemic routing for partially connected ad hoc networks,” *Technical Report CS-200006, Duke University*, 2000.

[16] A. Lindgren, A. Doria, and O. Schelén, “Probabilistic routing in intermittently connected networks,” in *Service Assurance with Partial and Intermittent Resources*, P. Dini, P. Lorenz, and J. N. de Souza, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 239–254.

[17] P. Hui, J. Crowcroft, and E. Yoneki, “Bubble rap: Social-based forwarding in delay-tolerant networks,” *IEEE Transactions on Mobile Computing*, vol. 10, no. 11, pp. 1576–1589, 2011.

[18] Y. Xiao and Chao Zhu, “Vehicular fog computing: Vision and challenges,” in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 6–9.

[19] A. Zúquete, *Segurança em Redes Informáticas*, 5th ed. Lisboa, Portugal: FCA - Editora Informática, Lda, 2018.

[20] C. Bernardini, M. R. Asghar, and B. Crispo, “Security and privacy in vehicular communications: Challenges and opportunities,” *Vehicular Communications,Elsevier*, vol. 10, pp. 13–28, October 2017.

[21] G. Carl, G. Kesidis, R. R. Brooks, and Suresh Rai, “Denial-of-service attack-detection techniques,” *IEEE Internet Computing*, vol. 10, no. 1, pp. 82–89, 2006.

[22] H. Deng, W. Li, and D. Agrawal, “Routing security in wireless ad hoc networks,” *Communications Magazine, IEEE*, vol. 40, pp. 70 – 75, 11 2002.

[23] J. R. Douceur, “The sybil attack,” in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260.

[24] J. Newsome, E. Shi, D. Song, and A. Perrig, "The sybil attack in sensor networks: analysis defenses," in *Third International Symposium on Information Processing in Sensor Networks, 2004. IPSN 2004*, 2004, pp. 259–268.

[25] J. J. Hopfield, "Artificial neural networks," *IEEE Circuits and Devices Magazine*, vol. 4, no. 5, pp. 3–10, 1988.

[26] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[27] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[28] A. Ng *et al.*, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[29] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[30] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[31] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[32] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[33] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[34] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[35] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis." in *Ndss*, 2011, pp. 1–17.

[36] F. Jemili, M. Zaghdoud, and M. B. Ahmed, "A framework for an adaptive intrusion detection system using bayesian network," in *2007 IEEE Intelligence and Security Informatics*. IEEE, 2007, pp. 66–70.

[37] J. V. Hansen, P. B. Lowry, R. D. Meservy, and D. M. McDonald, "Genetic programming for prevention of cyberterrorism through dynamic and evolving intrusion detection," *Decision Support Systems*, vol. 43, no. 4, pp. 1362–1374, 2007.

[38] N. Magaia, P. D. Gomes, and P. R. Pereira, “Finalcomm: Leveraging dynamic communities to improve forwarding in dtns,” in *Proceedings of the 14th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, ser. PE-WASUN ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 55–62. [Online]. Available: https://doi.org/10.1145/3134829.3134836

[39] N. Magaia, C. Borrego, P. R. Pereira, and M. Correia, “eprivo: An enhanced privacy-preserving opportunistic routing protocol for vehicular delay-tolerant networks,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 154–11 168, 2018.

[40] J. Dede, A. Förster, E. Hernández-Orallo, J. Herrera-Tapia, K. Kuladinithi, V. Kuppusamy, P. Manzoni, A. bin Muslim, A. Udugama, and Z. Vatandas, “Simulating opportunistic networks: Survey and future directions,” *IEEE Communications Surveys Tutorials*, vol. 20, no. 2, pp. 1547–1573, 2018.

[41] A. Keränen, J. Ott, and T. Kärkkäinen, “The ONE Simulator for DTN Protocol Evaluation,” in *SIMUTools ’09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. New York, NY, USA: ICST, 2009.

[42] N. Magaia and Z. Sheng, “Refiov: A novel reputation framework for information-centric vehicular applications,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1810–1823, 2019.

[43] “Deep learning for java,” Available at https://deeplearning4j.org/ (2020/12/10).

[44] G. Hamerly and C. Elkan, “Learning the k in k-means,” in *Advances in neural information processing systems*, 2004, pp. 281–288.