



GRADO EN INGENIERÍA EN TECNOLOGÍAS DE
TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

**Autonomous Vehicle Network
Cyberattack Modeling and
Machine Learning-based Defense Scheme**

Autor

Javier del Águila Martos

Directora

Dr. Janise McNair

Madrid

2024

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

**“Autonomous Vehicle Cyberattack Vulnerability Modeling and
Machine Learning-based Defense Scheme”**

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2023/24 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Javier del Águila Martos

Fecha:3/ Julio/ 2024.



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Dr. Janise McNair

Fecha: ..3../ ..7../ 2024





GRADO EN INGENIERÍA EN TECNOLOGÍAS DE
TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

**Autonomous Vehicle Network
Cyberattack Modeling and
Machine Learning-based Defense Scheme**

Autor

Javier del Águila Martos

Directora

Dr. Janise McNair

Madrid

2024

Acknowledgements

These past four years have been very important and special to me, and I would like to thank everyone who has walked with me during this path of my career.

I would like to thank my professors, for giving me an excellent education both in professional knowledge and life values, and especially to Dr. Janise McNair for her help and support during this project.

I would like to give special thanks to my family, for always supporting me and making all of this possible, and also to my friends, for making these past years unforgettable.

Modelaje de Ataques contra la Red de Vehículos Autónomos y Defensa mediante Aprendizaje Automático

Autor: del Águila Martos, Javier

Directora: Dr. McNair, Janise

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

Resumen del Proyecto

Este proyecto presenta la simulación de las comunicaciones de la red de vehículos autónomos (AVs) con los servidores de datos en un escenario de movilidad urbana, estudiando el comportamiento de la red en condiciones normales y bajo ataques de Denegación de Servicios (DoS). De las simulaciones se extraen datos para entrenar y probar una red neuronal con objetivo de detectar e identificar exitosamente ataques de DoS en la red de AVs, permitiendo así a los operadores de la red mejorar la seguridad de los datos de la red.

Palabras clave: Vehículo Autónomo, Mininet-WiFi, Denegación de Servicios, Red Neuronal, Comunicaciones con la red.

Introducción

Los vehículos autónomos (AVs) dependen en gran medida de sensores externos y comunicaciones con la red para el intercambio de datos, permitiendo la toma de decisiones informada. Sin embargo, esta dependencia con la información externa expone a los AVs a potenciales riesgos de hackeo y otras ciberamenazas.

Esfuerzos previos para la seguridad de los AVs se han centrado en soluciones dirigidas al hardware, consiguiendo sensores más difíciles de atacar[1]. El aprendizaje automático (Machine Learning) también se ha utilizado para defender los sensores de los AVs, además de para desarrollar modelos entrenados para detectar ataques de red que puedan ser efectuados contra los AVs [2], teniendo resultados positivos cuando son probados contra ataques para los que han sido entrenados, pero resultados ineficientes contra nuevos ataques.

El modelo de aprendizaje automático entrenado en este proyecto es distinto a los modelos previos, ya que en lugar de utilizar datos de red generales, es entrenado con datos propios de un entorno creado para simular la red de los AVs. Este proyecto realiza simulaciones de red virtual de la actividad inalámbrica de la red de AVs mientras estos navegan por zonas urbanas, utilizando la ciudad de Madrid como caso de estudio. El rendimiento de la red móvil de los AVs es analizado en

condiciones normales y bajo ataque para cuantificar el daño realizado a la red, obteniendo datos de la red utilizando las simulaciones y el programa Wireshark. Estos datos son filtrados, limpiados, configurados y etiquetados para entrenar y probar una red neuronal para detectar e identificar ataques de DoS en la red de AVs, permitiendo así a los operadores de la red mejorar la seguridad de los datos de la red.

Descripción del sistema

Para representar y proteger las redes vehiculares y la infraestructura de comunicaciones de datos, es necesario (1) obtener una representación de movimiento vehicular, (2) modelar la infraestructura de comunicaciones, (3) capturar los datos de la red vehicular, y (4) utilizar los datos capturados para entrenar y probar el modelo de aprendizaje automático.

La simulación del movimiento vehicular ha sido obtenida utilizando SUMO, Simulador de Movilidad Urbana [3]. Datos de Open Street Maps [4] fueron incluidos en las simulaciones de SUMO para mostrar patrones reales de movilidad urbana, utilizando como caso de estudio zonas urbanas de Madrid. Para modelar la infraestructura de comunicaciones, el emulador de red Mininet-Wifi es utilizado para modelar la red inalámbrica de los AVs. Mininet-WiFi emula el comportamiento de canales de comunicación inalámbricos y soporta escenarios dinámicos con nodos en movimiento y transiciones de red [5]. Para analizar las redes de los AVs mientras los vehículos se mueven por los escenarios urbanos de Madrid, los valores de movilidad de las simulaciones de SUMO fueron integrados dentro del emulador de Mininet-WiFi. Una red celular ha sido emulada en Mininet-Wifi para proporcionar conexión inalámbrica a los servidores internos de la red vehicular. La red celular fue diseñada e implementada siguiendo la estrategia de separación de celdas con el mismo canal y la reutilización de frecuencias. Los accesos celulares fueron separados en grupos, a su vez divididos en celdas hexagonales, donde cada una utiliza un canal diferente de radio frecuencia (RF). Se aseguró que no hubiera celdas adyacentes transmitiendo en el mismo canal de RF, teniendo así una baja interferencia por RF y por ello aumentando la calidad de servicio (QoS). La simulación de red configurada con los AVs y la red celular está representada en la figura 1.

Para capturar los datos de la red vehicular, se generan datos de tráfico de la red mediante “Multi Generator Network” (MGEN), que utiliza códigos para programar las transmisiones de datos [6]. Tráfico siguiendo el protocolo de datagramas de usuario (UDP) es enviado para simular la transmisión de métricas de los sistemas del vehículo a los servidores de datos. Un ataque de DoS ha sido implementado utilizando MGEN para desbordar los servidores de datos con paquetes UDP, eligiendo aleatoriamente vehículos en la red para ser atacantes y transmitir cin-

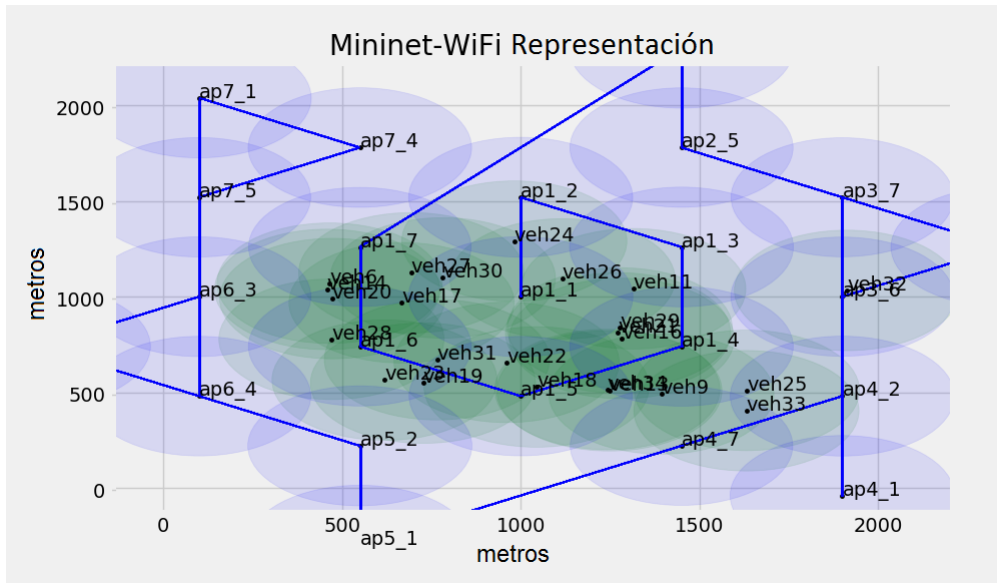


Figure 1: Representación de la simulación de red.

cuenta veces más paquetes de lo habitual. Los datos de la red han sido capturados y recopilados utilizando Wireshark [7]. Estos datos han sido filtrados, limpiados, configurados y etiquetados mediante Python para ser utilizados en el modelo de red neuronal de aprendizaje automático, con el 80% de los datos destinados para su entrenamiento y el 20% restante reservados para probar el funcionamiento del modelo de aprendizaje automático.

Resultados

La red neuronal ha sido entrenada y probada para obtener un modelo con excelente funcionamiento tanto en métricas de predicción como en tiempo de ejecución. El modelo entrenado es una red neuronal de propagación hacia delante (feedforward) con dos capas ocultas y una única neurona de salida para predecir binariamente si los datos pertenecen a un ataque o no. Esta simple y eficiente estructura permite al modelo tener una rápida ejecución, utilizando solo 1.63 microsegundos para predecir cada dato. También ha sido configurado con adiciones, como capas de abandono (dropout layers) que ayudan a prevenir el sobreajuste y la dependencia de neuronas específicas, con el objetivo de mejorar su rendimiento [8], [9]. Los resultados de rendimiento obtenidos de la ejecución del modelo con los datos reservados para pruebas se encuentran en la tabla 1, obteniendo métricas de rendimiento con valores de éxito superiores al 99.63%. Además, la tabla 2 muestra la matriz de confusión resultante de los datos predichos, indicador adicional de su

eficiencia y excelente capacidad para detectar ataques de DoS.

| Métricas | Valores |
|----------------------------------|---------|
| Exactitud | 0.9963 |
| Precisión | 0.9969 |
| Exhaustividad | 0.9992 |
| Puntuación F1 | 0.9981 |
| Área bajo la curva ROC (ROC AUC) | 0.9993 |

Table 1: Métricas de rendimiento del modelo frente a datos no vistos.

| | Usuario Legítimo Predicho | Atacante Predicho |
|-----------------------|---------------------------|-------------------|
| Usuario Legítimo Real | 742 | 57 |
| Atacante Real | 14 | 18302 |

Table 2: Matriz de confusión obtenida con el modelo.

Para mejorar el proyecto, trabajos futuros podrían dirigirse hacia el desarrollo de acciones defensivas contra los atacantes una vez detectados por la red neuronal, y el desarrollo de más ataques en el simulador de red creado para entrenar y probar la red neuronal, permitiendo así la detección de más tipos de ataques y aumentando la defensa proporcionada a la red de AVs.

Referencias

- [1] Anastasios Giannaros et al. “Autonomous Vehicles: Sophisticated Attacks, Safety Issues, Challenges, Open Topics, Blockchain, and Future Directions”. In: *Journal of Cybersecurity and Privacy* 3.3 (2023), pp. 493–543. ISSN: 2624-800X. DOI: 10.3390/jcp3030025. URL: <https://www.mdpi.com/2624-800X/3/3/25>.
- [2] Qiyi He et al. “Machine Learning-Based Detection for Cyber Security Attacks on Connected and Autonomous Vehicles”. en. In: *Mathematics* 8.8 (2020). Number: 8 Publisher: Multidisciplinary Digital Publishing Institute, p. 1311. ISSN: 2227-7390. DOI: 10.3390/math8081311. URL: <https://www.mdpi.com/2227-7390/8/8/1311>.
- [3] Pablo Alvarez Lopez et al. “Microscopic Traffic Simulation using SUMO”. In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: <https://elib.dlr.de/124092/>.
- [4] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. <https://www.openstreetmap.org>. 2017.

- [5] R. R. Fontes et al. “Mininet-WiFi: Emulating Software-Defined Wireless Networks”. In: *2015 11th International Conference on Network and Service Management (CNSM)*. 2015, pp. 384–389. DOI: 10.1109/CNSM.2015.7367387.
- [6] *Multi-Generator Network*. Available: <https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN/>.
- [7] Wireshark Foundation. *Wireshark: Network Protocol Analyzer*. <https://www.wireshark.org/>. 2024.
- [8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [9] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

Autonomous Vehicle Network Cyberattack Modeling and Machine Learning-based Defense Scheme

Author: del Águila Martos, Javier

Supervisor: Dr. McNair, Janise

Collaborating Entity: ICAI – Universidad Pontificia Comillas

Abstract

This project presents the simulation of autonomous vehicle (AV) network communications with data servers in an urban mobility scenario, studying the behavior of the network in normal conditions and under Denial of Service (DoS) attacks. Data is gathered from simulations to train and test a neural network to successfully detect and identify DoS attacks in the AV network, therefore enabling network operators to improve network data security.

Key words: Autonomous Vehicle, Mininet-WiFi, Denial of Service, Neural Network, Network Communications.

Introduction

Autonomous vehicles (AVs) heavily rely on external sensors and network communications for data exchange, enabling informed decision-making. However, this dependence on outside information exposes AVs to potential hacking and other cyber threats.

Previous efforts in AV security have focused on hardware solutions to make sensors more resilient [1]. Machine learning has also been utilized to defend AV sensors, in addition to having some machine learning models trained to detect network attacks which could be performed against AVs [2], resulting in good performance when being used against attacks they had been trained for, but poor results against unseen attacks.

The machine learning model trained in this project is distinct to the previous ones, as instead of being trained with general network databases, it is trained with data obtained from an environment created to simulate AV networks. This project conducts virtual network simulations of AVs' wireless network activity as they navigate through urban street layouts, using the city of Madrid as a case study. The performance of AV mobile networks is analyzed in normal conditions and under attack to quantify its damage to the network, gathering network data from the simulations using Wireshark. This data is filtered, cleaned, formatted and labeled

to train and test a neural network to detect and identify Denial-of-Service (DoS) cyber-attacks in the AV network, therefore enabling network operators to improve network data security.

System Description

To represent and protect vehicular networks and the data communications infrastructure, it is necessary to (1) obtain a representation of vehicular movement, (2) model the communications infrastructure, (3) capture the vehicular network data, and (4) use the captured data to train and test the machine learning models.

Simulation of the vehicular movement was achieved using SUMO, Simulator of Urban Mobility [3]. Open Street Maps data [4] was merged into the SUMO simulations to show real street-level mobility patterns from Madrid urban road scenarios. To model the communications infrastructure, the Mininet-WiFi network emulator is used to model the AVs' wireless network. Mininet WiFi emulates wireless network channel behavior and supports dynamic scenarios featuring nodes in motion and network transitions [5]. To analyze vehicular network as the vehicles move in Madrid urban scenarios, the mobility values from the SUMO simulation were integrated into the Mininet-WiFi emulator. A cellular network was emulated in Mininet-WiFi to provide wireless connections to the vehicular network backbone servers.

The cellular network was designed and implemented following cellular co-channel separation and frequency reuse strategies. Cellular access was separated into clusters, with each cluster further divided into a set of hexagonal cells, each operating on a different radio frequency (RF) channel. It was ensured that there were no adjacent cells transmitting at the same RF channel, thus enabling low RF co-channel interference and thereby enhancing user quality of service (QoS). The network simulation configured with the AVs and the cellular network is represented in figure 2.

To capture vehicular network data, data traffic was generated with the "Multi Generator Network" (MGEN), which utilizes scripts to program data transmissions [6]. User Data Protocol (UDP) traffic is sent to simulate vehicle system metrics being transmitted from the vehicle to the data server. A DoS attack was implemented by using MGEN to overflow the data server with UDP packets, having vehicles of the network randomly selected to be attackers and transmit fifty times more packets than usual. Wireshark [7] was used to sniff and collect data from the network. This data was then filtered, cleaned, configured and labeled via python to be utilized in the neural network-based machine learning algorithm, with 80% of the data used to train and 20% to test the machine learning model.

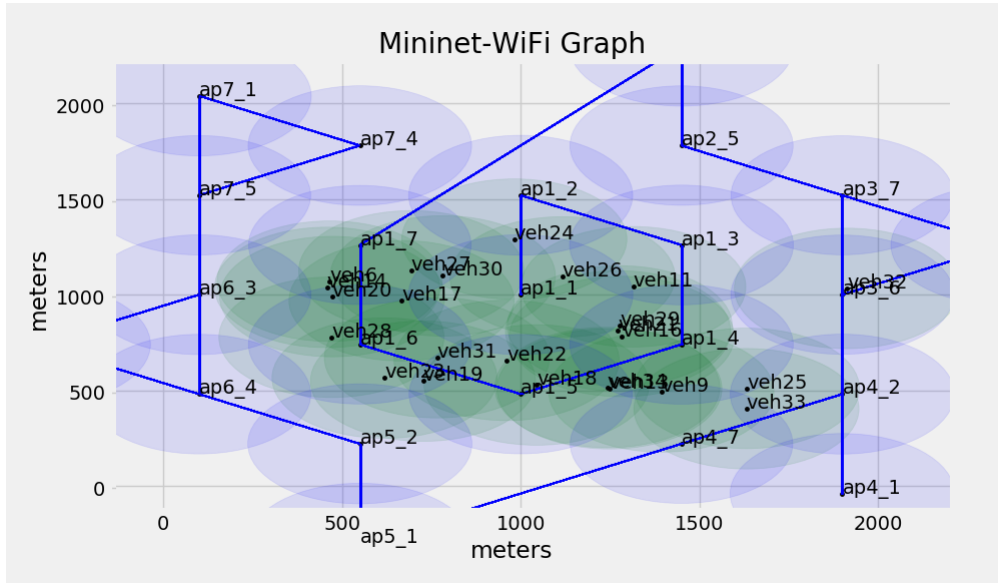


Figure 2: Network simulation representation.

Results

The neural network was trained and tested to achieve a model with excellent performance in both prediction metrics and execution speed. The model trained is a feedforward neural network with two hidden layers and a single output neuron to binary predict if the data belongs to an attacker or not. This simple yet efficient structure allows the model to have quick predictions, as it spends only 1.63 microseconds predicting each new data packet. It has also been configured with additions, such as dropout layers to help prevent overfitting and reliance on specific neurons, to further improve its performance [8], [9].

The performance results obtained from the testing data are shown in table 3, showing performance metrics values over 99.63% of success rate. Additionally, table 4 provides the confusion matrix that resulted from the data predicted, further proving its efficiency and excellent capability to detect DoS attacks.

| Metric | Value |
|-----------|--------|
| Accuracy | 0.9963 |
| Precision | 0.9969 |
| Recall | 0.9992 |
| F1 Score | 0.9981 |
| ROC AUC | 0.9993 |

Table 3: Performance metrics of the model against unseen data.

| | Predicted Legitimate User | Predicted Attacker |
|------------------------|---------------------------|--------------------|
| Actual Legitimate User | 742 | 57 |
| Actual Attacker | 14 | 18302 |

Table 4: Confusion matrix obtained with the model.

Future works would include the development of defensive actions against the attacker once detected by the neural network, and the development of new attacks in the network simulator to train the neural network and allow it to detect more attacks, thus improving the defense provided to the AV network.

Bibliography

- [1] Anastasios Giannaros et al. “Autonomous Vehicles: Sophisticated Attacks, Safety Issues, Challenges, Open Topics, Blockchain, and Future Directions”. In: *Journal of Cybersecurity and Privacy* 3.3 (2023), pp. 493–543. ISSN: 2624-800X. DOI: 10.3390/jcp3030025. URL: <https://www.mdpi.com/2624-800X/3/3/25>.
- [2] Qiyi He et al. “Machine Learning-Based Detection for Cyber Security Attacks on Connected and Autonomous Vehicles”. en. In: *Mathematics* 8.8 (2020). Number: 8 Publisher: Multidisciplinary Digital Publishing Institute, p. 1311. ISSN: 2227-7390. DOI: 10.3390/math8081311. URL: <https://www.mdpi.com/2227-7390/8/8/1311>.
- [3] Pablo Alvarez Lopez et al. “Microscopic Traffic Simulation using SUMO”. In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: <https://elib.dlr.de/124092/>.
- [4] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. <https://www.openstreetmap.org>. 2017.
- [5] R. R. Fontes et al. “Mininet-WiFi: Emulating Software-Defined Wireless Networks”. In: *2015 11th International Conference on Network and Service Management (CNSM)*. 2015, pp. 384–389. DOI: 10.1109/CNSM.2015.7367387.
- [6] *Multi-Generator Network*. Available: <https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN/>.
- [7] Wireshark Foundation. *Wireshark: Network Protocol Analyzer*. <https://www.wireshark.org/>. 2024.
- [8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.

- [9] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.nurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 2 | Background Information | 5 |
| 2.1 | Driving Automation Technologies | 5 |
| 2.2 | Vehicular Network Communications | 6 |
| 2.3 | Cyber-Attacks on Vehicular Communications | 7 |
| 2.3.1 | Man-in-the-Middle | 8 |
| 2.3.2 | Jamming | 8 |
| 2.3.3 | Denial of Service | 9 |
| 2.4 | Simulation Tools to Analyze Vehicular Networks | 9 |
| 2.4.1 | Vehicular movement | 9 |
| 2.4.2 | Network Simulator | 10 |
| 2.4.3 | Generate Network Traffic | 10 |
| 2.4.4 | Capture Network Traffic | 10 |
| 2.5 | Machine Learning in Data Driven Applications. | 11 |
| 2.5.1 | Types of Machine Learning Algorithms | 11 |
| 2.5.2 | Neural Networks | 12 |
| 3 | State of the Art | 15 |
| 3.1 | Hardware Solutions | 15 |
| 3.1.1 | GPS Systems | 15 |
| 3.1.2 | Light Detection and Ranging Systems | 16 |
| 3.1.3 | Cameras | 17 |
| 3.2 | Data Integrity | 17 |
| 3.3 | Machine Learning | 18 |

| | | |
|----------|--|-----------|
| 4 | Project Definition | 19 |
| 4.1 | Project Motivation | 19 |
| 4.2 | Project Goals | 20 |
| 4.3 | Methodology | 21 |
| 4.3.1 | Simulation of Urban Mobility | 21 |
| 4.3.2 | Simulation of Wireless Network Environment | 21 |
| 4.3.3 | Modeling Vehicular Network Data Traffic | 22 |
| 4.3.4 | Machine Learning Model for Denial of Service Attack De- tection | 22 |
| 4.3.5 | Study of the performance results | 22 |
| 4.4 | Planning | 23 |
| 5 | Simulation of Urban Mobility | 25 |
| 5.1 | Network Traffic Creation | 25 |
| 5.1.1 | Manual Creation | 25 |
| 5.1.2 | Network with OpenStreetMaps | 28 |
| 5.2 | SUMO Position Data Transfer | 30 |
| 5.2.1 | SUMO and Mininet-Wifi Simultaneous Execution | 30 |
| 5.2.2 | SUMO Position Recording | 31 |
| 6 | Simulation of Wireless Network Environment | 33 |
| 6.1 | Mininet-Wifi Concepts | 33 |
| 6.1.1 | Access Point | 34 |
| 6.1.2 | Station | 34 |
| 6.1.3 | End Device | 34 |
| 6.2 | Autonomous Vehicle Movement | 34 |
| 6.2.1 | Random Paths | 35 |
| 6.2.2 | Path Programming via Events | 35 |
| 6.2.3 | Replaying Mobility Configuration | 36 |
| 6.3 | Cellular Network | 37 |
| 6.4 | Mininet-Wifi Programming | 40 |
| 6.4.1 | Vehicular Configurations | 40 |
| 6.4.2 | Network Configurations | 41 |

| | | |
|----------|--|-----------|
| 6.4.3 | Attenuation in Wireless Communications | 43 |
| 6.4.4 | Data Server Configuration | 44 |
| 6.4.5 | Network Simulation Representation | 45 |
| 6.4.6 | Connectivity Performance | 47 |
| 7 | Modeling Vehicular Network Data Traffic | 49 |
| 7.1 | Generation of Data Traffic | 51 |
| 7.1.1 | Reception Scripts | 51 |
| 7.1.2 | Transmission Scripts | 52 |
| 7.1.3 | Denial of Service Attack Design | 55 |
| 7.2 | Data Capture from the Network | 55 |
| 7.3 | Data Cleaning | 56 |
| 7.4 | Data Format | 58 |
| 7.4.1 | Numerical Formatting | 58 |
| 7.4.2 | Data Normalization | 58 |
| 7.5 | Resulting Database for Model Training and Testing | 59 |
| 8 | Machine Learning Model for Denial of Service Attack Detection | 61 |
| 8.1 | Model Development for Vehicular Network Data Traffic | 61 |
| 8.1.1 | Number of Layers and Neurons | 62 |
| 8.1.2 | Model Training | 63 |
| 8.1.3 | Loss Function | 64 |
| 8.1.4 | Gradient Optimizer | 64 |
| 8.1.5 | Learning Rate | 64 |
| 8.1.6 | Epochs and Batches | 65 |
| 8.1.7 | Overfitting Prevention | 65 |
| 8.1.8 | Backpropagation Issues and Solutions | 66 |
| 8.2 | Metrics to Analyze Performance | 66 |
| 8.2.1 | Accuracy | 67 |
| 8.2.2 | Precision | 67 |
| 8.2.3 | Recall | 67 |
| 8.2.4 | F1 Score | 68 |
| 8.2.5 | ROC AUC | 68 |

| | | |
|-----------|---|-----------|
| 8.2.6 | Confusion Matrix | 68 |
| 8.2.7 | Execution Time | 69 |
| 9 | Performance Analysis and Numerical Results | 71 |
| 9.1 | Autonomous Vehicle Network Performance | 72 |
| 9.1.1 | Ping Performance Results | 72 |
| 9.1.2 | Iperf Performance Results | 73 |
| 9.2 | Model Performance | 73 |
| 10 | Conclusion and Future Works | 77 |
| 10.1 | Conclusion | 77 |
| 10.2 | Future Works | 78 |
| A | Project Alignment with the Sustainable Development Goals | 79 |
| | Bibliography | 81 |

List of Figures

| | | |
|-----|--|----|
| 1.1 | Scheme of AV main means of data collection. | 2 |
| 2.1 | Vehicular networks simple representation of its main communications. | 7 |
| 2.2 | Possible infection risks in AV wireless communications. | 8 |
| 2.3 | Example scheme of a neural network with two hidden layers. | 13 |
| 3.1 | General system architecture of an AV, picture from [16]. | 16 |
| 3.2 | Example of hardware redundancy in AVs, picture from [18]. | 17 |
| 4.1 | Gantt Diagram with the project's planning. | 23 |
| 5.1 | Creation process of a SUMO simulation in Madrid, Argüelles, using OpenStreetMaps data. | 29 |
| 5.2 | SUMO traffic simulation in Madrid, area of Argüelles. | 30 |
| 6.1 | Radio frequency interference representation between cellular nodes. | 38 |
| 6.2 | Hexagonal cellular architecture with a cluster size of seven cells, picture from [26]. | 39 |
| 6.3 | Network topology structure and channel allocation used in the sim- ulations. | 40 |
| 6.4 | Cellular network connection in the simulation. | 43 |
| 6.5 | Network simulation start. | 45 |
| 6.6 | Network simulation with more participants in the network. | 46 |
| 6.7 | Network simulation representation to compare the position of the AVs with the previous image. | 46 |
| 6.8 | RSSI measurement of two AVs in the network. | 47 |

| | | |
|-----|---|----|
| 7.1 | Network capture of the server being attacked. | 56 |
| 7.2 | Normal UDP packet capture showing the different layers of information available in Wireshark. | 57 |
| 9.1 | ROC curve plot. | 74 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | Autonomous drive technology levels. Table from [2]. | 6 |
| 7.1 | Distribution of labeled data in terms of packets from attackers and legitimate users. | 59 |
| 8.1 | Confusion matrix scheme. | 68 |
| 9.1 | Ping metrics comparison between normal and DoS attack scenario. | 72 |
| 9.2 | Iperf metrics comparison between normal and DoS attack scenario. | 73 |
| 9.3 | Performance metrics of the model against unseen data. | 74 |
| 9.4 | Confusion matrix obtained with the model. | 75 |

Listings

| | | |
|-----|---|----|
| 5.1 | Traffic light node creation using XML. | 26 |
| 5.2 | Type file example to determine edges with two lanes and a maximum speed of 50kmh. | 26 |
| 5.3 | Union of two traffic lights to create a bidirectional street in XML. . . | 27 |
| 5.4 | Vehicle configuration and creation of a route using XML | 27 |
| 7.1 | Scheme followed to configure a node to receive data in MGEN | 51 |
| 7.2 | Example of reception script in MGEN | 52 |
| 7.3 | Scheme and events used to design a transmission script in MGEN . . | 52 |
| 7.4 | Scheme configuration of Poisson transmission pattern | 53 |
| 7.5 | Example of a transmission script in MGEN | 54 |

Chapter 1

Introduction

Autonomous vehicles (AV) and autonomous drive (AD) technology have gradually increased their presence in today's society. Not only is it possible to see this technology in action in fully autonomous vehicles, but also in most new vehicles, as they now come with autonomous tools and safety features such as the Automatic Emergency Brake. As shown in figure 1.1, AVs work by using a variety of different technologies such as LIDAR(Light Detection and Ranging) sensors, GPS signal and cameras to gather information about their surroundings. All these systems need to be interconnected through the vehicle's internal network and with the vehicle's computer, which will analyze the data given and perform the appropriate actions. In addition to getting information via their near environment, AVs can transfer information through Vehicle to Vehicle (V2V), Vehicle to Infrastructure (V2I), Vehicle to Cloud (V2C) and Vehicle to Everything (V2X) communications.

These wireless communications enable AVs to transmit and receive real-time updates about road conditions, allowing the vehicular network to know the state of traffic in areas beyond their immediate vicinity, which lets AVs to proactively respond to traffic changes. For example, if one AV detects a traffic light turning red, it can inform other AVs nearby using V2V communications. This allows the vehicles to be proactive and gradually decelerate, instead of having a reactive response to the car ahead braking. Proactive behaviors are desired, as they increase traffic flow and safety. Additionally, AVs use these wireless communications to periodically transmit data such as their current route, position, battery status, tire

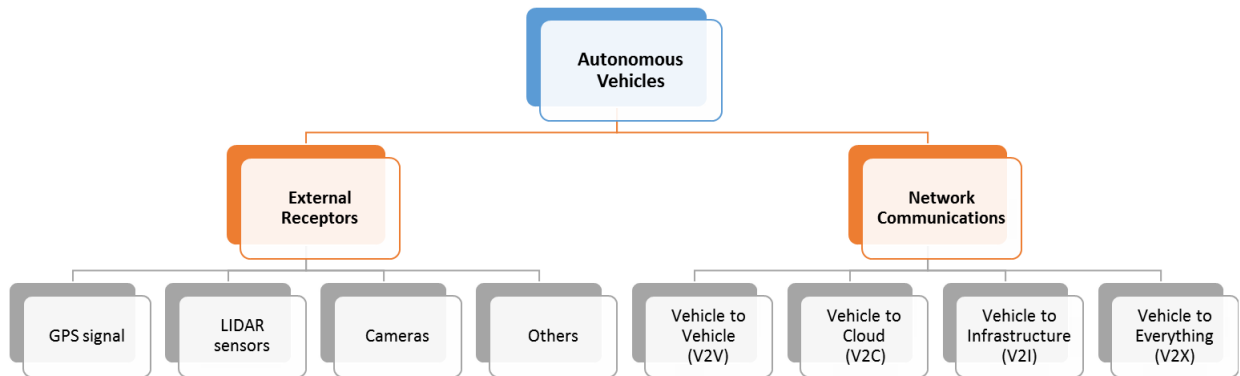


Figure 1.1: Scheme of AV main means of data collection.

pressure and other parameters to central servers, ensuring continuous monitoring and updates on the status of each vehicle. The dependency of external information makes AVs vulnerable to hacking attacks, which can lead to erroneous actions and possibly accidents. Vehicular security techniques are needed to protect against cyberattacks that impact data delivery and vehicle situational awareness.

This project's objective is to enhance autonomous vehicles' safety and resilience against cyberattacks, especially those targeting attack their vehicular network and wireless communications. This will be achieved by developing a machine learning model that allows the network to identify cyberattacks, enabling them to react and take appropriate measures.

This document and the work developed is structured as follows: the second chapter discusses background information relating AV technologies that are of special interest for the development of this project, in addition to providing a brief explanation about some security challenges in vehicular networks, the simulators and applications used to perform this project, and information about machine learning models and neural network models. The third chapter presents the state of the art in the matter of AV security, focusing on hardware solutions and machine learning applications to defend the vehicle. The fourth chapter looks at the project's motivation, goals to be achieved, methodology utilized and planning followed. The fifth chapter focuses on simulating the mobility of vehicles in an urban environment, with the goal of obtaining this data to model AV movement. The sixth

chapter presents the network simulator, the creation of a cellular network to provide connection to the AVs, the configurations utilized to model the vehicles and other elements in the network, simulation representations, and connectivity performance of the network. The seventh chapter includes all information pertaining to modeling vehicular network data traffic, including the processes of generating, capturing, cleaning and formatting the data to be utilized in the machine learning model; which is looked at in the eight chapter, explaining the creation and different configurations applied to the neural network, and the metrics utilized to measure its performance. The ninth chapter exposes the performance analysis of the network and provides numerical results and metrics obtained from testing the model. To finalize, chapter 10 includes the conclusion of the project, where the main achievements of this document are discussed, and a section about future works which could be developed to further improve this project and therefore AV security.

Chapter 2

Background Information

This chapter aims to provide an in-depth explanation of the technologies used in this project, improving the reader's understanding in key areas related to autonomous vehicles and their networks. It will focus on explaining autonomous technologies, vehicular communications, cybersecurity challenges, machine learning and artificial intelligence (AI), and the simulation and testing tools that have been utilized to fulfill this project.

2.1 Driving Automation Technologies

It is very common nowadays to find vehicles with tools that help the driver have a safe driving experience, such as the anti-lock brake system (ABS). These tools are driving automations, which help perform driving tasks for the user. There are different levels of driving automations depending on the number of tools used in the vehicle, and the extend of action that these tools have in the driving experience. According to the Society of Automotive Engineers (SAE) [1], there are six different levels of automation in vehicles, which are summarized in table. 2.1.

With the different automation levels clarified, it is important to mention that autonomous vehicles are those vehicles that include technologies corresponding to the levels three and above.

| Level of automation | Title | Description |
|---------------------|--------------------------------|---|
| Level 0 | No Driving Automation | The human driver performs all aspects of driving tasks, such as steering, acceleration, etc. |
| Level 1 | Driver Assistance | The vehicle features a single automated system for driver assistance, such as steering or acceleration/deceleration, with the anticipation that the human driver performs all remaining aspects of the driving tasks. |
| Level 2 | Partial Driving Automation | The vehicle can perform steering and acceleration/deceleration. However, the human driver is required to monitor the driving environment and can take control at any time. |
| Level 3 | Conditional Driving Automation | The vehicle can detect obstacles in the driving environment and can perform most driving tasks. Human override is still required. |
| Level 4 | High Driving Automation | The vehicle can perform all aspects of the dynamic driving task under specific scenarios. Geo-fencing is required. Human override is still an option. |
| Level 5 | Full Driving Automation | The vehicle performs all driving tasks under all conditions and scenarios without human intervention. |

Table 2.1: Autonomous drive technology levels. Table from [2].

2.2 Vehicular Network Communications

In vehicular networks, communication encompasses both internal and external interactions. Internally, vehicles send sensor and camera data to its onboard computers to be processed. Externally, vehicles perform communication with road infrastructure (V2I), other vehicles (V2V), networks (V2N), and various devices (V2X). Given the need for rapid decision-making at high speeds, external vehicular communications rely on Long Term Evolution (LTE) and 5G technologies to obtain the necessary bandwidth and low latency needed for these exchanges of information [3]. Figure 2.1 shows a simple representation of the main external communications and actors in vehicular networks. Additionally, AVs utilize Dedicated Short Range Communications (DSRC) to exchange information over short distances, such as in V2V and V2I communications [4]. This protocol uses IEEE 802.11p, which supports data exchanges at rates ranging from 3 to 27 Mbps at the distance of 300 meters over a bandwidth of 10MHz [5]. DSRC performs an important role in scenarios that require quick and reliable communication, such as collision avoidance, traffic signal coordination, and emergency vehicle prioritization. The usage of both long-range cellular communications (LTE and 5G) and short-range (DSRC) communications allows AVs to establish a resilient network, ensuring that vehicles can obtain and utilize critical information from both their

immediate surrounding and online data servers, improving safety and optimizing traffic flow.



Figure 2.1: Vehicular networks simple representation of its main communications.

To ensure rapid decision-making, vehicular data transmission is categorized by priority levels. High-priority traffic includes route queries and roadwork status checks, while lower-priority tasks would include transmission of data such as tire pressure, which are also important but less time-sensitive [6]. Not all data transferred is related to vehicle metrics and road traffic data, there is also a significant amount of user data exchanged from applications such as in-car entertainment. For instance, Spotify is a popular application used to reproduce music or podcasts, which would add to the vehicular communication load, requiring priority considerations to meet quality of service requirements without compromising vehicle efficiency.

2.3 Cyber-Attacks on Vehicular Communications

Vehicular networks use wireless communications to exchange important data, both in short and long distance communications. The usage of these communications makes the network inherently vulnerable, specially due to the open nature of wireless channels, the high mobility of vehicles which complicates communications, and the need to receive rapidly and reliably the data transmissions. A malicious user in the network could represent an important threat for the whole AV network. This section will focus on giving background information about different types of attacks that could be performed against a vehicular network [7]. It is vital to understand these threats to develop countermeasures against them, and therefore

protect the network. Fig. 2.2 shows a simple representation of users that could be infected and attack the network.

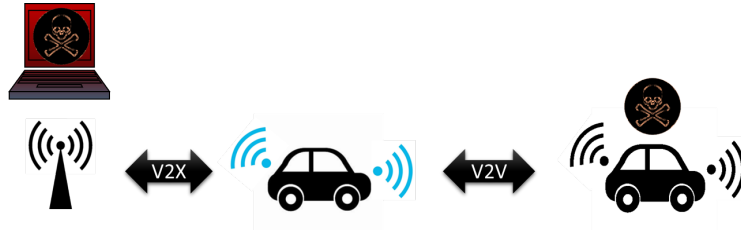


Figure 2.2: Possible infection risks in AV wireless communications.

2.3.1 Man-in-the-Middle

Man-in-the-Middle (MitM) attacks are a type of attack where an attacker intercepts communications between two parties without their knowledge. This interception allows the attacker to eavesdrop the communications, receive sensitive data, modify packets sent to inject false information, or even to impersonate one of the parties. In vehicular networks, this attack could be used to steal sensitive vehicular data. More importantly, it could be used to transmitting false information to the vehicles, misinforming them by modifying the packets sent from the data servers, which could lead to AVs taking wrong decisions based on the wrong information they received, potentially causing accidents or traffic congestion.

2.3.2 Jamming

Jamming attacks consist of the creation of high power interferences to make the network unable to properly receive communications, hence rendering it unusable. In vehicular networks, jamming attacks could make the vehicle unable to communicate in long or short distance. The elimination of this source of information would force the vehicle to make its driving decisions with much less data than normally, which could lead to mistakes in navigation systems and safety mechanisms.

2.3.3 Denial of Service

Denial of Service (DoS) are attacks whose goal is to render a service, machine or network resource unavailable to its users. To achieve this, the attackers overwhelm the target with a flood of requests to consume all its resources such as CPU, memory and bandwidth, making the target unable to respond to true requests. In the context of AV networks, DoS attacks can target data servers, making them unresponsive and therefore incapable of giving and receiving vehicle information and respond to their queries [8].

This project focuses on enhancing the resilience of data servers against DoS attacks. Given their role in managing numerous transmissions and communications with various AVs, servers are both critical components and vulnerable targets. Servers have an extensive knowledge of the vehicular network's status, making them a strategic point for developing defense mechanisms against cyberattacks

2.4 Simulation Tools to Analyze Vehicular Networks

This section will provide information about the different tools and simulators that have been used to model and analyze vehicular networks. These tools have played an important role for the project, being used for tasks such as simulating real-world traffic scenarios, emulating the AVs' network, and generating and capturing network traffic.

2.4.1 Vehicular movement

To simulate vehicular movement and traffic law obedience, the tool utilized has been Simulation of Urban Mobility (SUMO). SUMO is an open-source simulator of urban mobility, designed to simulate and analyze the behavior of entities such as vehicles, pedestrians and cyclist within traffic networks [9]. SUMO plays an important role in this project by simulating vehicles properly obeying traffic rules, just like AVs would do in a real life situation. To provide realistic scenarios, Open Street Maps data has been merged into the SUMO simulations, therefore

generating vehicle routes and its behavior in real locations [10].

2.4.2 Network Simulator

To simulate the network and transfer information, the tool utilized has been Mininet-WiFi. Mininet-WiFi is an extension of the Mininet network emulator, specifically designed to simulate wireless network environments. Mininet is used to create virtual networks to simulate the behavior of real-world networks, which for this project has been used to simulate the behavior of AVs communicating with data servers [11]. Mininet-WiFi requires a Linux-based operating system to run, with Ubuntu being the recommended option. Therefore, a virtual machine was installed with Ubuntu, which was beneficial to the project as the network simulations could be performed without other programs consuming the machine's resources.

2.4.3 Generate Network Traffic

MGEN is an open source software developed by the U.S. Naval Research Laboratory designed to generate and load traffic patterns using scripts. It can emulate the traffic of User Datagram Protocol, UDP/IP(Internet Protocol), and Transmission Control Protocol traffic, TCP/IP. MGEN has been used to emulate the traffic exchange made between AVs and data servers [12].

2.4.4 Capture Network Traffic

Wireshark is an open source network protocol analyzer that allows the user to capture network traffic and get detailed information of the packets transmitted over the captured network [13]. Wireshark has been used to monitor and capture the data exchanged between AVs and the data servers. It is able to filter and isolate specific traffic protocols or patterns, making it easier to focus on the relevant data sent among the network, such as UDP and TCP packets in AVs networks.

The objective of this project is to use the data sent through the network to design a model that is capable of identifying which packet transmissions are malicious; therefore making the usage of ML a great tool to accomplish this task.

2.5 Machine Learning in Data Driven Applications.

Machine learning (ML) is a key element in data driven applications. ML focuses on understanding the data and its inner relations to produce algorithms and models that are able to make predictions based on the data they have been trained on. ML is a subset of Artificial Intelligence (AI), sharing both the aim to complete tasks that would normally require human intelligence such as learning, improving from experience and reasoning.

2.5.1 Types of Machine Learning Algorithms

Deciding what type of ML model to use is vital in every project. To make the decision, it is important to understand the different options, characteristics and outcomes that each algorithm provides. This section will discuss the two main types of ML models, the goal and outcome that each group aims to get, and the characteristics of the data needed to execute these algorithms.

Supervised learning

Supervised learning is utilized when the goal of the algorithm is to understand the relationship of the data and its output, with the objective of predicting the outcome for new and unseen data. To teach the model how to predict the desired output, the data given needs to include labels, the correct output. Given this labeled data, the model is trained to understand the relationship between the input data and their outcome, allowing it to predict the label of future data.

It is mainly used in regression problems, where the goal is to predict values, and in classification model, with the goal of categorizing data into different groups. Other models, such as neural networks, also utilize supervised learning to learn the structure and understand the data to be able to predict the desired outputs.

Unsupervised Learning

In contrast with the previous type of algorithm, unsupervised learning is utilized when data provided does not include labels. The goal with this data is to understand it and identify underlying patterns and key features within the data, being therefore used in clustering models, where the goal is to group data into clusters based on similarities, and in dimensionality reduction, models whose aim is to analyze and identify which features of the data are valuable to be kept and which are redundant. Dimensionality reduction is desired specially when the data is going to be used in another model, as having less complex data can simplify the model and therefore reduce the computational cost and time of running it. In addition, it can also help visualize complex data, allowing for a better understanding of it.

2.5.2 Neural Networks

This project aims to analyze network data with the goal of predicting and detecting whether the AV network is under attack, making therefore supervised learning models the most appropriate approach for this task. Neural networks have been the model chosen to fulfill the defense of the AV network against DoS attacks, as their capacity to understand the characteristics and complexity of the data makes them a great candidate to achieve complicated applications, such as to predict attacks in an AV network.

Neural networks work utilizing layers of interconnected nodes, also called neurons, which process information and recognize patterns in the data, being inspired by the human brain and its system of neurons to transmit information. Each neuron receives one or more inputs, processes them, and produces an output. The neural network that will be utilized consists of an input layer, two hidden layers, and an output layer. In all layers, each neuron is connected to all neurons in the next layer, allowing the model to understand and utilize the relationship between all parameters analyzed. To understand the structure of neural networks, a generic example is provided in figure 2.3, showing a neural network with two input nodes, two hidden layers having the first one four nodes and the second one three neurons, and an output layer with a single neuron.

The input layer receives the raw data, having one neuron per property or

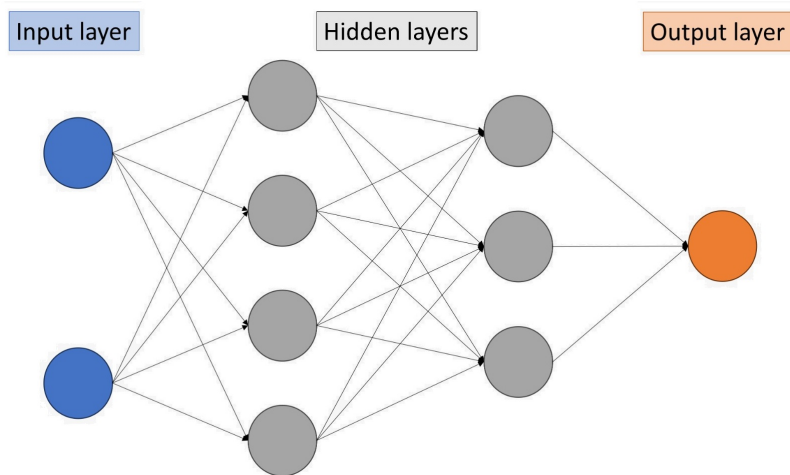


Figure 2.3: Example scheme of a neural network with two hidden layers.

column of the data to be analyzed. This information is passed to the first hidden layer, where each neuron processes the inputs given. Each neuron performs a weighted sum of the inputs given, giving more weight to those values the model considers important to detect anomalies and attacks. It is followed by the execution of an activation function to introduce non-linearity to the model.

As real-world data is complex and has non-linear relationships, adding non-linearity to the model is desirable. It allows the model to learn patterns and create decision boundaries in the data, therefore having a better performance. For the model designed, the activation functions utilized have been the Rectified Linear Unit (ReLU) and Sigmoid function [14].

Rectified Linear Unit Function

ReLU is a function from the torch library that introduces non-linearity by outputting zero to all negative values, or its actual value if it is positive, having the result of this function inputted to the next layer of neurons. The ReLU function is defined as written in equation 2.1.

$$\text{ReLU}(x) = \max(0, x) \quad (2.1)$$

It has also been utilized to reduce the effect of the vanishing gradient problem,

as this model includes back-propagation in training, which is explained in chapter 8.

Sigmoid Function

Sigmoid is another non-linear function, utilized in the output layer of the model. This model desires to obtain a binary result, detecting whether the packet received is from a hacker or not. To accomplish this, the sigmoid function is used to obtain a value between 0 and 1 that is interpreted as the probability of the prediction belonging to a class, in this case to being a hacker. The sigmoid function is defined as written in equation 2.2 [15].

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.2)$$

Once the first hidden layer has finalized processing the data, it passes its outputs to the second hidden layer, which continues the process until reaching the output layer.

The output layer in this neural network consists of only one neuron, as the current objective is to obtain a binary prediction; to determine if the data analyzed is from an attacker or not. This neuron utilizes all inputs and the sigmoid function, which as previously mentioned, obtains an output number between 0 and 1 which represents the probability of the data predicted to be an attacker.

Chapter 3

State of the Art

Security countermeasures against hacking attacks have become an increasingly important and relevant topic as the popularity of AVs has risen. Security for AVs has been explored through hardware solutions, data integrity protection methods like blockchain, and machine learning models.

3.1 Hardware Solutions

Via hardware, the focus has been to make the primary sensing devices more resilient. These devices play a key role in informing the AV about its nearby environment, and its information greatly influences short term decisions made by AVs. Figure 3.1 provides an overview of the sensors that are generally equipped in AVs, covering all angles to inform the vehicle about its surroundings. These sensors, while being great sources of information, can also become vulnerabilities for the vehicle if not properly defended. Work in their protection has been directed into the following areas.

3.1.1 GPS Systems

GPS systems and signals need to be secure, as AVs rely heavily on them to know their location and which direction to take. GPS receivers work by using the most powerful signal received, therefore hackers can build powerful signals and pretend to be the original sender or jam the system. Countermeasures against these attacks

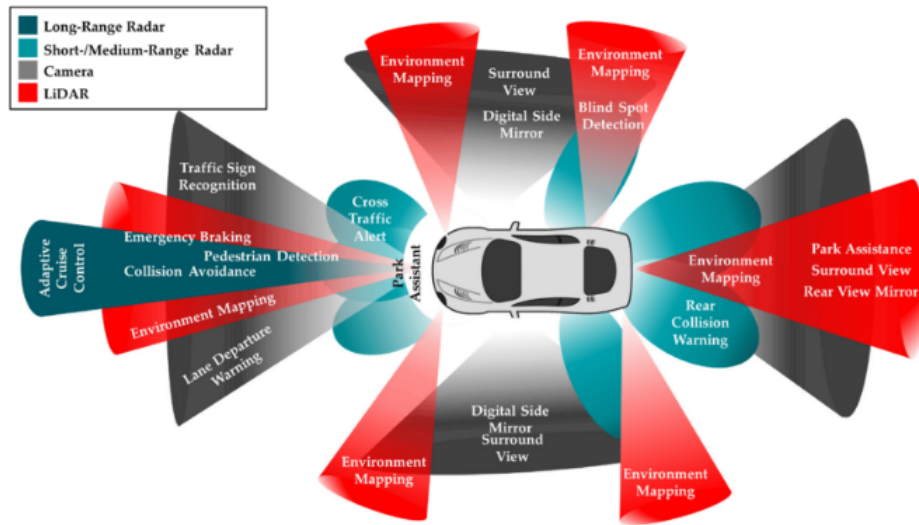


Figure 3.1: General system architecture of an AV, picture from [16].

include the usage of authentication methods and the denial of any signal received with a higher power than 163db done through antenna arrays, as the regular GPS signal never exceeds that amount, meaning that any signal with higher power levels is an attempt of hacking or jamming the device [8].

3.1.2 Light Detection and Ranging Systems

As these sensors are vital to determine obstacles in the road, Light Detection and Ranging systems (Lidar) have also been studied. They work through transmission of pulses to be reflected into surfaces. The AV waits, and if the pulse comes back, it will determine how far the obstacle is. Hackers could exploit this waiting period by sending a pulse to the sensor before the actual one arrives, making the car believe there are obstacles, misrepresenting the situation. Countermeasures against them would be reducing the time that the sensor waits, meaning objects will now be detected closer than before. Another solution analyzed has been using probabilistic approaches to determine if the new signal is real or not based on previous data [17].

3.1.3 Cameras

Cameras are also an easy target to attack, as any strong beam, light or laser can temporarily blind them, rendering them unable to analyze the environment. Protection measures explored are the addition of redundancies in the camera systems by adding extra cameras to get a similar view from different angles. Another measure would be the addition of filtering lenses to block and let through specific wavelengths of light [8],[17]. These measurements have already been applied in the market, as seen in figure 3.2, having system redundancy of cameras and other sensor devices.



Figure 3.2: Example of hardware redundancy in AVs, picture from [18].

3.2 Data Integrity

To protect the AVs integrity of data, blockchain implementation into AVs communications has also been proposed. Blockchain's communication is done via blocks which are unalterable and are linked between one another. Each block is unique and has its own identification, making the task of changing its information difficult for a hacker. Blockchain's ability to set rules in communication via smart contracts could also be useful to only let verified users communicate with AVs, and to only receive information from their own sensors which would incorporate the blockchain rules in its data transmissions. [19]

3.3 Machine Learning

While most approaches in AV security are directed into making a receptors more resilient and harder to hack, others aim towards defending the vehicle with machine learning. Machine learning has been used in a variety of applications, including the defense of AVs against cyberattacks. It has seen great results to solve some issues previously explained, such as jamming attacks on the GPS signal of the vehicle. Project [20] built a machine learning algorithm to detect jamming attacks on the vehicle, achieving successful results preventing their attacks. Machine learning projects typically use network datasets with packets labeled as legitimate or belonging to cyberattacks. Project [21] trained a Decision Tree and Naive Bayes model to detect different types of attacks which could be performed against AVs, including DoS. They utilized the KDD99 data set [22], and filtered it to only have those attacks which could be performed to AVs, creating a new benchmark called CAV-KDD. Using this data, both algorithms were trained, resulting in good performance when being used against attacks they had been trained for, but poor results against unseen attacks, which is crucial as attackers are constantly trying new types of cyberattacks.

In the context of AVs, machine learning applications are usually trained using generic network data, which is appropriate for general network applications, but not ideal for AV networks as its data has different characteristics, such as the mobility of its users, and use different protocols to perform AV communications. The machine learning model that has been developed in this project is different to the previous ones, as instead of being trained with general network databases, it is trained with data obtained from an environment specifically created to simulate AV networks. This is one of the goals of this project, to create an environment capable of reproducing the movement and data transfers that occur in AV networks. Using this created environment, the main objective of the project to create and obtain network data from the AV network in normal and under attack conditions, specifically under DoS attacks, to build a neural network to detect this type of attacks and defend the AV network.

Chapter 4

Project Definition

This project's objective is to enhance autonomous vehicles' safety and resilience against cyberattacks, especially those targeting attack their vehicular network and wireless communications. This will be achieved by developing a machine learning model that allows the network to identify cyberattacks, enabling them to react and take appropriate measures.

4.1 Project Motivation

Driving is an inherently dangerous activity. Any mistake could be fatal, not only for the driver but also for the rest of road users. Driving requires drivers to be fully present on the activity and to not make any errors, however, humans are not error free. In fact, 94% percent of traffic accidents are caused because of human error [23]. On the other hand, machines when programmed correctly do not make mistakes. AVs do not get tired on the road, or divide their attention with other tasks, they are only focused on the driving activity and information relating to it, ensuring the best decision is always made.

Autonomous vehicles could potentially reduce the number of accidents on the road to zero, and also improve circulation and traffic efficiency thanks to their wireless communications, which would allow them to know which routes are less congested and therefore take better routes. Additionally, AVs could also reduce pollution, as they are programmed to function as efficiently as possible, reducing

the amount of energy or fuel needed and therefore having lower emissions. AVs work perfectly when the data received is correct, however, this dependence of outside information exposes the vehicle to potential hacking threats, which could lead to mistakes and accidents. With data estimating around 20.3 million vehicles in active use with various levels of autonomous features, and 60 million units for 2030 [24], detection and development of measures against vehicular attacks is needed. Every contribution relating to enhance security, user safety, and potentially saving lives is important and worth pursuing; which is why this project's objective is to improve AVs security.

4.2 Project Goals

The main goal of this project is to increase traffic safety and road efficiency by utilizing AVs. Specially, this project aims to help in the following areas:

- **Enhance Road Safety:** Data gathering from sensors and network communications allows AVs to consider its nearby area and also the state of the road ahead to make the best decision possible. Additionally, they have quicker reaction times than human drivers and eliminate the risk of human error from its driving, hence eliminating a significant factor in traffic accidents [23].
- **Increase Traffic Flow:** AVs get information from its surroundings and from the network, being able to communicate with other vehicles (V2V), traffic infrastructure (V2I) and with data servers (V2X, V2C). These communications allow AVs to optimize their routes based on the traffic situation ahead, avoiding situations that would result in traffic congestion and therefore increasing traffic flow, leading to smoother and more efficient routes.
- **Reduce Pollution:** AVs are designed to function as efficiently as possible, reducing the amount of energy or fuel consumption when possible and therefore having lower emissions, which contributes to reducing environmental pollution.

However, to enjoy these benefits, it is vital for the public to welcome and trust AVs. To guarantee a positive reception, it must be ensured that AVs are

safe and secure. Therefore, this project focuses on enhancing the security of AVs, particularly focusing on the AV network. This will be achieved by developing a neural network capable of identifying whether the network is being attacked, which will allow it to take measures against the attacker and protect its users.

4.3 Methodology

The project has been divided into different goals to create the environment needed to achieve the main objective; the development of a new defense solution for AV networks.

4.3.1 Simulation of Urban Mobility

Development of traffic simulations using real life scenarios and locations, focusing on urban environments. These simulations are used to reproduce the behavior and movement of autonomous vehicles following the traffic rules and status. This section is important to obtain vehicular movement data, their rate of acceleration and deceleration with traffic lights and signs, and its behavior in different situations.

4.3.2 Simulation of Wireless Network Environment

Once the mobility data is gathered, it is utilized as an input in Mininet-Wifi, the network simulator chosen to reproduce AV communications. The vehicular data will be utilized to replicate the position changes and movement of AVs in urban streets.

Antenna Placement

For AVs to communicate with the network and its data servers, it is needed to implement a cellular network in the simulation. AVs are mainly used in urban populations, where Quality of Service (QoS) is prioritized and there is full network coverage. To ensure that these two requisites are achieved, the simulation environment has been divided into cells, each covered by an Access Point connected to the network.

4.3.3 Modeling Vehicular Network Data Traffic

With the environment properly set, AVs are able to start transmitting and receiving information. MGEN has been the program chosen to produce data traffic, as it is able to develop scripts that allow AVs to send data packets using different transport protocols. These packets simulate the transmission and reception of information over the network with data servers.

Data Capture and Cleaning

The network traffic is needed to be captured to be further analyzed and cleaned. This task will be performed using Wireshark, which captures data packets sent over the network. The raw data is not suitable to be used for model training in its captured state, requiring a separate python code to clean it and gather the variables and elements valuable for the model. This data will play a vital role in the next section, as it will be used to train and test the machine learning algorithm.

4.3.4 Machine Learning Model for Denial of Service Attack Detection

The data previously mentioned data will be utilized to train a neural network to identify denial of service attacks, having 80% of the data utilized for training, and 20% to test its efficiency. Different configurations will be tested until a successful model is obtained.

4.3.5 Study of the performance results

To understand the value that the machine learning model created brings to the AV network, performance metrics have been obtained from the network while being in a normal scenario and under attack. Furthermore, performance metrics of the neural network are provided to analyze its positive performance in detecting hacking attacks.

| Tasks | September | October | November | December | January | February | March | April | May | June |
|---------------------------------------|-----------|---------|----------|----------|---------|----------|-------|-------|-----|------|
| State of the Art | | | | | | | | | | |
| Simulation of Urban Mobility | | | | | | | | | | |
| Vehicular movement in Mininet-Wifi | | | | | | | | | | |
| Cellular Network Development | | | | | | | | | | |
| Data Transfer Implementation | | | | | | | | | | |
| Development of malicious scenarios | | | | | | | | | | |
| Data Capture and Cleaning | | | | | | | | | | |
| Machine Learning Training and Testing | | | | | | | | | | |

Figure 4.1: Gantt Diagram with the project's planning.

4.4 Planning

To develop this project, it was first needed to understand how AVs work and the different articles and projects that have been developed relating their security, to ensure a new and innovative contribution is made. Afterwards, the project was organized following an Agile Methodology. This methodology has been done via sprints, with duration of one, two or four weeks, depending on the functionality that is being developed. The goal is to prioritize tasks in each sprint to have them completed by the end of the sprint. Figure 4.1 shows the Gantt Diagram performed to keep track of the objectives and timeline of the project, helpful to set realistic goals and tasks to be achieved in each sprint.

Chapter 5

Simulation of Urban Mobility

This project aims to solve hacking attacks that AVs may be object to in present scenarios, where they would be moving through traffic and following the traffic rules and signs. Therefore, the first objective is to obtain a representation of vehicular traffic and movement in real situations and geographical locations. This has been achieved using SUMO, a Simulator of Urban Mobility, mainly used to simulate and analyze traffic congestion, mobility and road efficiency.

5.1 Network Traffic Creation

SUMO is a flexible program, providing different options to create the road network and generate the paths and routes that vehicles will follow. First, it will be explained how does SUMO function by describing how to manually set up a traffic network, which is valuable to understand the other options available to create traffic simulations.

5.1.1 Manual Creation

Nodes

In SUMO, streets are formed by connecting nodes. Nodes represent the foundation of the simulation, as their placement and connection creates the streets, junctions, roundabouts, and other elements in the simulation. The first step to create a

network is therefore to create a file with these nodes, in XML format. The only requirement to create a node is to provide the position at which it will be placed, an ID, and if desired special parameters to inform of characteristics such as the addition of a traffic light in that place. A node creation example is given in Listing 5.1, creating three nodes with traffic lights in the positions of 90,90; 200,90; and 200,200; having the first number represent the X coordinate and the second the Y coordinate.

```
1 <nodes>
2 <node id="n1" x="90" y="90" type="traffic_light"/>
3 <node id="n2" x="200" y="90" type="traffic_light"/>
4 <node id="n3" x="200" y="200" type="traffic_light"/>
5 </nodes>
```

Listing 5.1: Traffic light node creation using XML.

Edges

An edge is the term utilized to join two nodes together, with the goal of creating a road between them. The creation of edges is done via an XML file, indicating the source node ID and destination node ID that will be joined together, and giving an ID to the edge created. There are additional parameters that could be included to indicate characteristics of the connection, which can be defined in another XML file, called type file, as it indicates the type of parameters and properties that the edge created will have, such as number of lanes and maximum speed. Listing 5.2 represents the XML code utilized in a type file to generate a 2 line road with a 50kmh maximum speed.

```
1 <types>
2 <type id="2L50" numLanes="2" speed="50"/>
3 </types>
```

Listing 5.2: Type file example to determine edges with two lanes and a maximum speed of 50kmh.

Using this type of edge created, listing 5.3 joins the previously defined nodes creating a bidirectional street of 2 lanes and a maximum speed of 50kmh.


```

1 <edges>
2   <!-- Join n1 to n2 with 2 lanes, max speed of 50km/h -->
3   <edge from="n1" to="n2" id="road1to2" type="2L50"/>
4
5   <!-- Now n2 to n3-->
6   <edge from="n2" to="n3" id="road2to3" type="2L50"/>
7
8   <!-- Same road, but for the traffic in the opposite direction -->
9   <edge from="n2" to="n1" id="road2to1" type="2L50"/>
10  <edge from="n3" to="n2" id="road3to2" type="2L50"/>
11
12 </edges>

```

Listing 5.3: Union of two traffic lights to create a bidirectional street in XML.

Vehicles and Routes

The final step to complete the traffic simulation is to create an XML file with the types of vehicles to be simulated and its routes. SUMO supports the creation of many different types of vehicles, characterized by properties such as vehicle length, acceleration and deceleration rate, and maximum speeds to model their behavior. Drivers and vehicles do not all behave in the same manner, having the sigma parameter to account for these differences. This parameter introduces some random behavior to the vehicle, hence if it is set to 0 it would result in a car that follows exactly the model defined for the vehicle.

With the vehicle model created, the last step needed is to provide routes for the vehicles to follow. Routes are designed by selecting the edges to be followed by the vehicle, in addition to giving it an ID. Listing 5.4 creates a car model, designs a route and assigns a new vehicle to execute this route once the network starts, in its first second of execution.

```

1 <routes>
2   <vType accel="1.0" decel="4.0" id="Car" length="2.0"
3   maxSpeed="120.0" sigma="0.1"/>
4   <!-- In this example, the car will go from n1 to n2,

```

```
5   then n2 to n3 and stop, as it is the last edge-->
6   <route id="route0" edges="road1to2 road2to3"/>
7   <vehicle depart="1" id="veh0" route="route0" type="Car"/>
8 </routes>
```

Listing 5.4: Vehicle configuration and creation of a route using XML

It is worth noting the attribute "depart", as it determines when will the vehicle appear in the simulation and start its route. Using these files, one can create a configuration file to indicate SUMO the path to the XML files pertaining the nodes, edges, routes and types created; in addition to indicating how much time will the simulation be running. This configuration file will be used to execute SUMO, loading the network and routes that have been determined.

5.1.2 Network with OpenStreetMaps

Manual creation of networks is a great option for projects relating to the creation and analysis of new road layouts and structures, needing to manually indicate how will it be formed. Meanwhile, this project aims to represent and use locations that already exist. Urban scenarios are very complex and detailed, making the task of manually replicating such environments an unreasonable task. To perform tasks in already existing locations, SUMO developers have included the option of inputting the location that one wants to simulate, and have the program create and generate the network automatically, taking the area data from Open Street Maps. Using this data, SUMO is able to understand how many lanes each road has, the maximum speed according to the traffic rules of such country, which intersection has traffic lights, and such details to creates the necessary nodes, edges, and characteristics of the network needed to replicate the location. Figure 5.1 shows the generation process of a SUMO simulation using OpenStreetMaps data to create a simulation of the colored area, which represents one of Madrid's urban areas, Argüelles.

With the nodes and edges created, the simulation is now only missing routes and vehicles to be simulated. Fortunately, another key feature of utilizing this type of network creation is the generation of random paths for the type of vehicles and road users, shown in the right side of figure 5.1. In this selection, the user

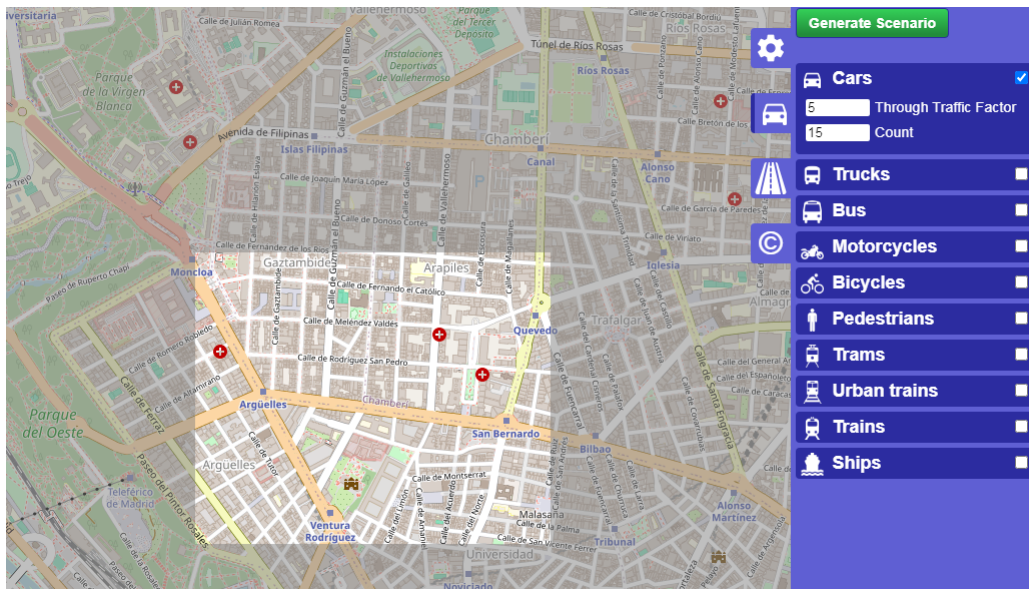


Figure 5.1: Creation process of a SUMO simulation in Madrid, Argüelles, using OpenStreetMaps data.

determines the properties desired for the routes that these vehicles will follow in the simulation, in addition to the number of vehicles and routes that are desired.

- **Through Traffic Factor** determines the likelihood of an edge in the boundaries of the simulation to be selected as the starting or ending point of the generated route. If this parameter is high, boundaries of the map will be the starting and ending point of the path to be followed, and therefore there would be a high amount of traffic running through the simulation to reach the other side of the network.
- **Count** defines the amount of vehicles that will be generated per hour and lane-kilometer. For instance, if the network has a total of 10km of lane paths, and the count parameter is set to 15, then 150 vehicles will be generated each hour of simulation.

To emulate the network working with AVs, only cars will be generated in these simulations, having as a result a simulation similar to figure 5.2, which shows the SUMO simulation running with different vehicles in the streets of Argüelles.

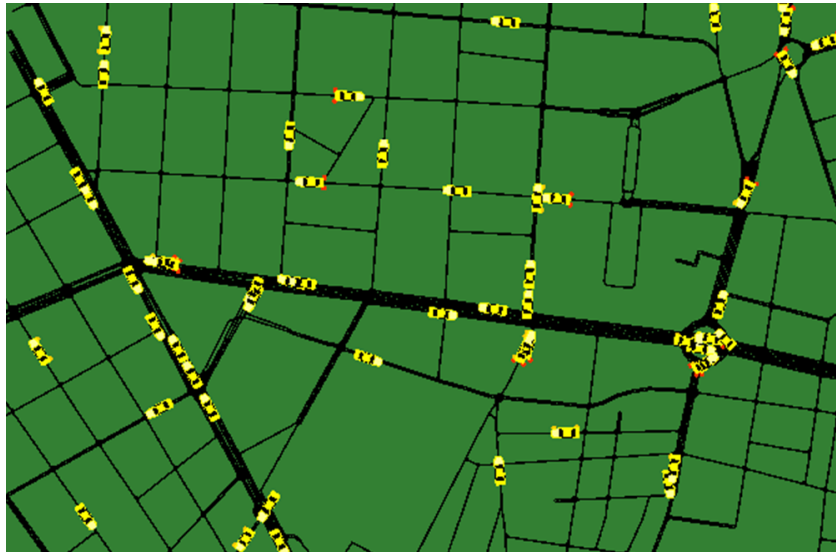


Figure 5.2: SUMO traffic simulation in Madrid, area of Argüelles.

5.2 SUMO Position Data Transfer

As previously mentioned, SUMO is mainly used to analyze vehicular networks, not communication networks, therefore it is needed to obtain the data desired from these simulations to feed the communication network simulations in Mininet-Wifi.

5.2.1 SUMO and Mininet-Wifi Simultaneous Execution

There is documentation about Mininet supporting the simultaneous usage of SUMO to directly feed the network simulations. This option was explored first, however, while it is possible to run both codes and programs simultaneously, having to send the information from SUMO to mininet's code was highly inefficient. This approach would often result in both programs and codes waiting for the other party to send the signal or information needed to continue the execution, freezing the execution and therefore unable to work.

Another issue that could have contributed to the programs not working was that in SUMO, when a vehicle finishes its route, it is eliminated from the simulation to save up resources for other executions. Mininet has no issue in creating dynamic scenarios with moving stations, which would be the AVs. Meanwhile, mininet

simulations are usually utilized with a defined number of users and access points, created before starting the simulation. Therefore it could have caused problems for the program to generate and eliminate stations, i.e. AV users of the network, in the midst of execution, possibly being one of the sources of the malfunctioning of this method.

5.2.2 SUMO Position Recording

Without the first option, data recording and further usage of it as input was explored. The first problem encountered was how to obtain the data from the SUMO simulations. To accomplish this task, TraCI library was used [25]. TraCI stands for Traffic Control Interface, and allows python code to run traffic simulation from SUMO, enabling the code to access and modify the behavior and values of the simulated objects in SUMO. Therefore, a python code using TraCI was created, in which this library would be used to execute and run the SUMO simulations. Each simulation was run exhaustively, using TraCI to analyze and extract the values of each vehicle at every second of the simulation. These values were stored in data files, one for each vehicle, to be used as inputs in Mininet-Wifi, which is further explained in the next chapter.

Chapter 6

Simulation of Wireless Network Environment

Mininet-WiFi [11] has been chosen as the network emulator for modeling the AV wireless network, as it offers support for dynamic scenarios featuring nodes in motion and network transitions. As described in Chapter 3, Mininet-WiFi is an extension of the Mininet network emulator, which is specifically designed to simulate wireless network environments. Its wireless nature, and its support for dynamic scenarios, makes Mininet-Wifi a great tool to simulate the the behavior of AVs moving through urban layouts while communicating with data servers. Before explaining the different steps and processes that have been done to simulate the AV network, it is important to understand the key concepts and vocabulary that mininet uses to design the different objects that will be simulated in the network.

6.1 Mininet-Wifi Concepts

To create simulations in mininet and mininet-wifi, one must generate them via code written in python. Coding the simulations allows the users to shape them and configure them to fit their needs and requirements. The network created for this project is comprised of three main elements: access points, stations, and end-devices.

6.1.1 Access Point

One of the most important elements that a wireless network needs are Access Points. They are used to give access to the network to all devices that are in range of connection with it, making them a great choice to simulate the functioning of cellular towers. Access Points are able to change the radio channel and frequency at which they function, in addition to being able to customize the maximum range at which they are able to offer connection.

6.1.2 Station

To simulate AVs, it was needed to use an object capable of moving freely in space and change connection between access points depending on their position. Stations fit in this category, and are able to perform all tasks previously mentioned, therefore making them a great choice to simulate AVs. Although this project focuses on long distance AV communications, it is worth noting that mininet-wifi could be also used to simulate short distance AV communications, such as V2V, as stations are able to communicate with other stations and devices that are in its range.

6.1.3 End Device

The last element that is needed to be incorporated are data servers, end devices of the network. End devices in mininet are static objects that are unable to move from their connection, usually connected via a physical link such as Ethernet. They will be used to simulate the role of data servers, performing data communications to the different users in the AV network.

6.2 Autonomous Vehicle Movement

To simulate an AV network, one must first represent the physical movement of the AVs, just as described and performed in Chapter 5. Therefore the first goal to accomplish in the network simulator is to use these position values as inputs for the stations to move. Mininet-Wifi, as a dynamic network simulator, has built in

functions to allow network stations to move and follow paths, which are further explored below.

6.2.1 Random Paths

One of the most common approaches in mininet-wifi's mobility scenarios is to configure the simulation to design random paths and movements for the stations. However, the random movements of the stations is not useful for this project, as the goal is to reproduce vehicle movement through a defined street and road network, stopping at red lights and other traffic signs; therefore this approach was not considered a viable option.

6.2.2 Path Programming via Events

Mininet-Wifi has built in functions to allow network stations to follow a path. This path is created by two events, a starting event with its starting position and time, and an ending event with the final position and time to arrive there. These events would cause the station to move at a uniform speed until it arrived at the destination. However, this path creation had two main issues to represent vehicular mobility, which are explained below.

Turns in the Path

The path created takes the shortest route to reach the destination, therefore it makes a straight line to arrive. Meanwhile, streets do not always follow straight lines, having elements such as roundabouts that make the vehicle turn and have a non-linear path.

Number of Executable Events in a Path

To establish a path, it is necessary to determine events for the station to start moving and finish moving at a desired position and time in the simulation. These paths require a starting event, and an ending event; and are unable to concatenate multiple starting and ending events.

The inability to concatenate multiple starting and ending events forces stations to only move once in the simulation, rendering this option unusable to satisfy the needs of the project.

6.2.3 Replaying Mobility Configuration

The option that was finally implemented was to move the vehicles from one position to another updating their positional value each second. This configuration is called "Replaying Mobility", as it allows the user to manually modify the movements that will be followed before the execution begins, therefore replaying the mobility and its values.

Utilizing the data files created in Chapter 5, this configuration allows the python code to read all these vehicular movement files and utilize those values and data to simulate the positional values of each vehicle. In addition to allowing mininet-wifi to replicate the movements of the vehicles, this solution brings the following positive qualities with its execution.

Reproducibility

An important factor in the scientific method is the replicability of the experiments. Using data files to generate the movements of the vehicles makes the experiments easy to recreate, as it is only necessary the data files to make the vehicles follow the same route generated by SUMO.

Efficiency

In both of the other mobility options, the stations would be moving in a uniform speed from point to point, having to update this value many times every second to make it seem as a uniform movement. Meanwhile, with this configuration, the program only needs to update the positional values of the vehicle once every second, making it more efficient and less resource demanding, enabling the computer to have more resources and memory to run the simulation.

Network Reality Representation

Vehicles are constantly moving, and the network is only able to know that the vehicle has moved when it communicates and transmits information. Consequently, if the network was able to generate a map of the positions where a vehicle has been by solely looking at its network transmissions, the map represented would be a collection of points where it transmitted, which would look like the representation built using Mininet-wifi.

6.3 Cellular Network

Once the movement and paths of vehicles are created, it is needed to design the cellular network in which they will be communicating.

AVs are mainly used in urban populations, where Quality of Service (QoS) is prioritized and there is full network coverage. To ensure that these two requisites are achieved, the simulation space has been divided into hexagonal cells, each covered by an Access Point connected to the network.

To ensure full coverage and QoS in a hexagonal cell, cellular towers must extend their coverage slightly beyond the cell boundaries, resulting in a circular area of coverage. This creates an overlap with adjacent cells, which is desirable for maintaining consistent coverage and full network access to the users. When users move through the network, they will be entering and exiting cells, connecting to their antennas with methodologies such as to connect to the strongest signal first, or to connect to the least loaded provider first.

Achieving this coverage typically involves using three antennas, each with a 120° main lobe to collectively cover 360° . Alternatively, a higher power isotropic antenna can be used, though this method is less efficient. Antenna coverage doesn't stop abruptly at a specific distance; instead, the signal strength diminishes with distance, providing high QoS when the user is inside the hexagonal cell, and lower QoS further away. These signals received outside the cell, although they carry less energy, can collide with other transmissions in the medium and generate interference, reducing the QoS. This phenomenon is called Radio Frequency Interference (RFI), and occurs when two transmitters are using the same Radio Frequency

(RF) [26].

Therefore, this approach would be incomplete without the usage of different RF channels in each tower. Fig. 6.1 shows a representation of two cell towers using the same RF channels, picturing how the signals reach the adjacent cells and would disturb the medium.

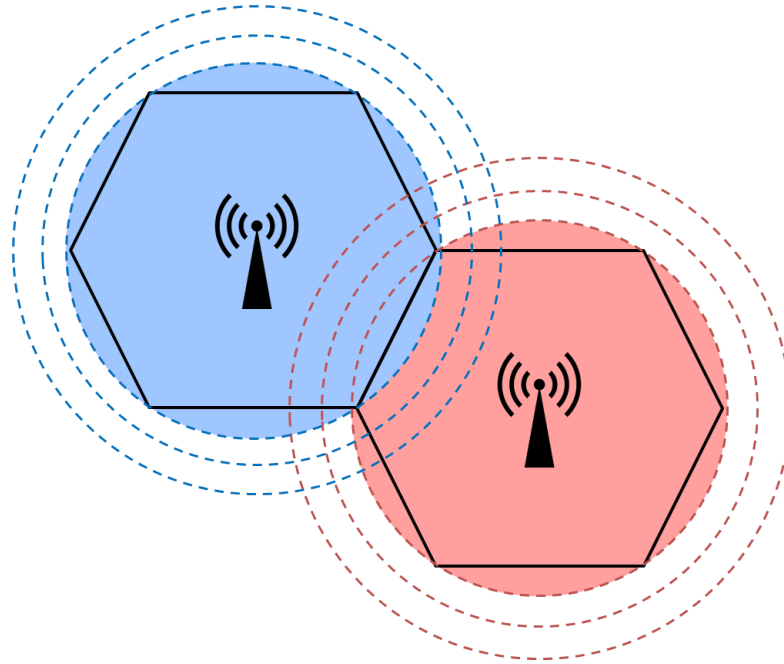


Figure 6.1: Radio frequency interference representation between cellular nodes.

Bandwidth is limited, and there aren't enough RF channels to use a different channel in all cell towers, it is needed to reuse the RF channels. Consequently, all scenarios have been designed to position the access point antennas in the network utilizing the co-channel separation and frequency reuse strategy.

This design ensures that there are no adjacent cells transmitting at the same RF channel, ensuring low RFI and thereby enhancing QoS in the network. This method divides the scenario into clusters, with each further divided into a set of hexagonal cells, each one inside the cluster operating on a different RF channel. Equation 6.1 describes the relationship between the number of cells per cluster, N , also referred as the co-channel reuse ratio; the distance between co-channel cells, D_i ; and cell radius, R_i [26].

$$\frac{D_l}{R_l} = \sqrt{3N} \quad (6.1)$$

The co-channel reuse ratio is deduced from equation 6.2, where i and j must be integer values [26].

$$i^2 + ij + j^2 = N \quad (6.2)$$

Depending on the environment to be used on, different number of cells per cluster can be chosen. For this project, seven cells within each cluster reaches a balance between minimizing RFI and maintaining practical cell dimensions, reducing the frequency of user cell transition. Figure 6.2 provides a representation of how co-channel separation would divide the physical space and allocate the resource of RF channels, for a scenario with a co-channel reuse ratio of seven, representing each RF channel with a different letter.

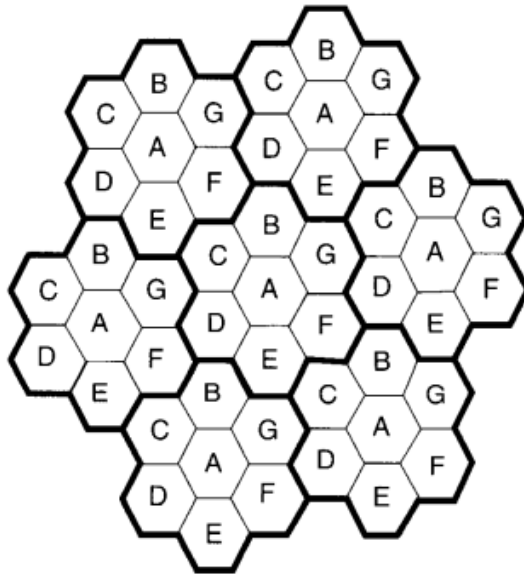


Figure 6.2: Hexagonal cellular architecture with a cluster size of seven cells, picture from [26].

With the co-channel reuse ratio (N) decided, equation 6.2 is also useful to determine where are located the cells sharing the same channel. Utilizing the previous equation, to obtain $N=7$, possible values for i and j would be $i=1$ and $j=2$. These values would be used in this case by moving one unit, i.e. the size

of your cell's radius, along one face of the hexagon because $i=1$; and two units, because $j=2$, 60° or 120° to this direction.

With the network topology defined, a python program was designed to build the desired cellular network, with a co-channel reuse ratio of seven. This program generated the network structure seen in figure 6.3, where each cell cluster is highlighted in grey and representing each access point with a different color to show the RF channel allocation that would be used.

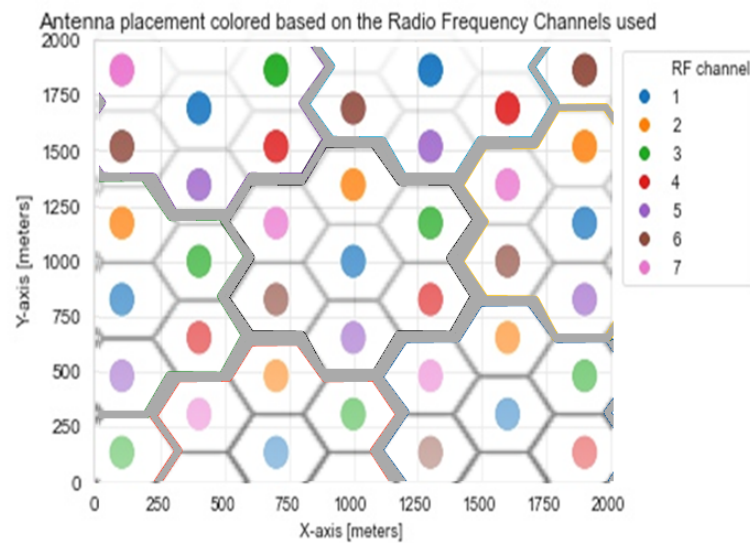


Figure 6.3: Network topology structure and channel allocation used in the simulations.

6.4 Mininet-Wifi Programming

6.4.1 Vehicular Configurations

With the vehicular movement being able to be inputted and a network topology decided to implement in the simulations, the last part needed is to implement all these pieces into a code and use it among the mininet-wifi libraries to generate the network simulation.

Each vehicle will be represented by a Station. These stations need an IP and MAC address to participate in the network, in addition to an ID and name to represent them in the visual simulation. In addition to this, the simulation is configured to connect the stations to the strongest signal received first, hence using the access point closer to them. Finally, as previously described, to move the vehicles in the decided routes, the simulator is configured to replay mobility scenarios, reading the data files with the position values saved from SUMO.

In reality, vehicular networks are constantly changing, particularly the participant users of the network, AVs. AVs join the network when they are turned on and programmed to get its users to their destination; and AVs leave the network once their route is completed and the vehicle is turned off until its next usage. For this reason, the AVs are simulated to follow the same behavior, appearing in the network when their route begins, and leaving it once their destination is reached.

It is worth noting that, as the simulations become more complex with vehicles and actions that have to be made, such as moving the vehicles and generating its network traffic, more resources are needed and the execution speed of the simulation gets reduced. The reduction of speed has been measured experimentally, particularly measuring the speed at which movement instructions were executed, resulting in the execution of these instructions every two to three seconds instead of every second. This information means that each vehicle would move one position approximately every two seconds, which is a problem as they should be moving one position every second. Therefore, to accurately represent the vehicle's movement and account for the execution time and processing slowing down, the vehicular movement speed is increased, reading two positions of movement per instruction instead of only one.

6.4.2 Network Configurations

Using the network topology and python code previously defined, the desired cellular network is built, with a co-channel reuse ratio of seven.

Each access point requires a Service Set Identifier (SSID), which is defined

using the cluster and channel that each access point is in. Each cluster is therefore numbered, having as a results SSIDs such as "ssid14", which would match with the access point using the fourth channel in the first cluster. The same technique was also used to name the access points, needed to identify them in the visual representation of the simulation. The same access point used in the previous example would be given the following name: "ap1_4". Naming the access points is also useful to visually ensure in the simulations that there are no antennas with the same RF channel adjacent to each other.

Antenna Coverage Distance

To provide a high QoS and stable connection in urban environments with 5GHz usage, the distance between the access point and the user should be less than 300 meters [27]. Therefore, the access points in the simulation are configured to provide a coverage area of 300 meters.

Cellular Network Connection

With the access points created and distributed following the desired structure, it is now needed to connect them to allow communication through all the simulation.

Experimental testings showed that the access points would only provide wireless access to the network to the AVs if they had two or less connections with other access points, therefore reducing the possible connection structures. This constraint led to the decision of performing the connection of the network by joining the antennas taking usage of the previously designed clustering pattern. This involved linking first the antennas with two adjacent antennas from the cluster, and leaving one of the outer cell nodes of the cluster to have one link connecting it to its cluster, and another link to connect it to the center antenna of a near cluster. The topology resulting from this approach is shown in figure 6.4, where the links between access points is denoted with a blue line and the access points are named following the previously explained method.

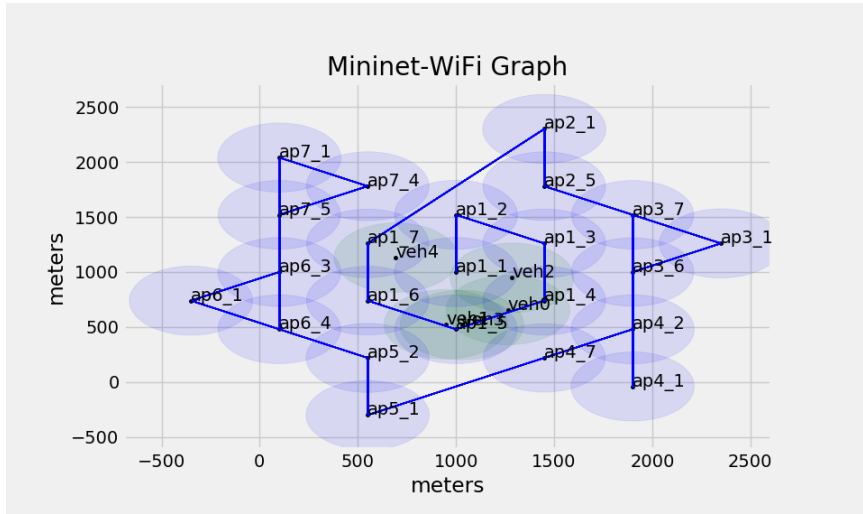


Figure 6.4: Cellular network connection in the simulation.

6.4.3 Attenuation in Wireless Communications

As previously explored when designing the cellular network for the simulations, the farther away the receptor is from an antenna, the less energy it receives from it. This phenomenon is called attenuation, and it refers to the reduction in power density of an electromagnetic wave as it travels through a medium. The power loss is mainly due to the spreading of the wave as the distance from the source increases, having to cover more space with the same energy.

This concept is applied and explained in the Friss transmission equation, equation 6.3, where it accounts for the transmission's spread in all directions by dividing the power received with the area of the sphere radiated, which is $(4\pi d)^2$. In addition to this element, to obtain the value of the power received, P_r , it utilizes the power transmitted P_t , the gain of the transmitter and receiver antennas respectively, G_t, G_r , and the wavelength of the carrier wave, $\lambda = c/f$, where "c" is the speed of light and "f" is the frequency of the carrier signal. This parameter is also important, as depending on the frequency of the signal, it will require a different quantity of energy to reach its destination [26].

$$P_r = P_t G_t G_r * \left(\frac{\lambda}{4\pi d}\right)^2 \quad (6.3)$$

This equation is mainly used in free space, where it is considered that there are no elements that could block the transmission. However, depending on the environment of communications, the attenuation and path loss could be different. Particularly, in an urban area, there are multitude of buildings, trees, signs, vehicles, and other elements that could block the signal, causing additional losses through reflection, diffraction and scattering.

To account for these extra losses, a new concept is introduced in the equation, the path loss gradient, represented with α [26]. This concept is applied to the power of the distance that the wave needs to travel to reach its destination. The equation shaped to fit this new parameter is shown in equation 6.4.

$$P_r = P_t G_t G_r * \left(\frac{\lambda}{4\pi}\right)^2 * \frac{1}{d^\alpha} \quad (6.4)$$

Depending on the environment of communications, the path loss gradient value is different. To model free space communications, the path loss gradient has a value of two, representing only the energy lost due only to the energy being spread in the medium to reach its destination. Meanwhile, to accurately model the signal propagation in urban areas, with its additional losses through reflection, diffraction and scattering, a path loss exponent between 2.7 and 3.5 is used [28]. Hence, mininet-wifi is configured to simulate its communications using a path loss gradient value of three.

6.4.4 Data Server Configuration

To finalize generating the environment of the AV network, a data server was configured. This data server was generated as an end device, having a direct connection to the network to enable it to transmit and receive information from the AVs. The data server, as any device connected to the network, is configured with an IP address and MAC address, and is able to open and close its ports and exchange information through them, which will be further explored in Chapter 7.

6.4.5 Network Simulation Representation

A key element that Mininet-Wifi offers is its ability to represent the network in real time. This allows a visual monitoring of the network, vital to ensure that vehicles are moving, that once their route is finished they leave the network, and also to verify that co-channel access points are not set adjacent.

The result of the Mininet-Wifi simulations is represented in figure 6.5, where blue areas represent the transmission coverage area for each Access Point, while the green areas represent the transmission coverage area of the simulated AVs. This first image was taken at the beginning of the simulation, when not many AVs have joined the network yet.

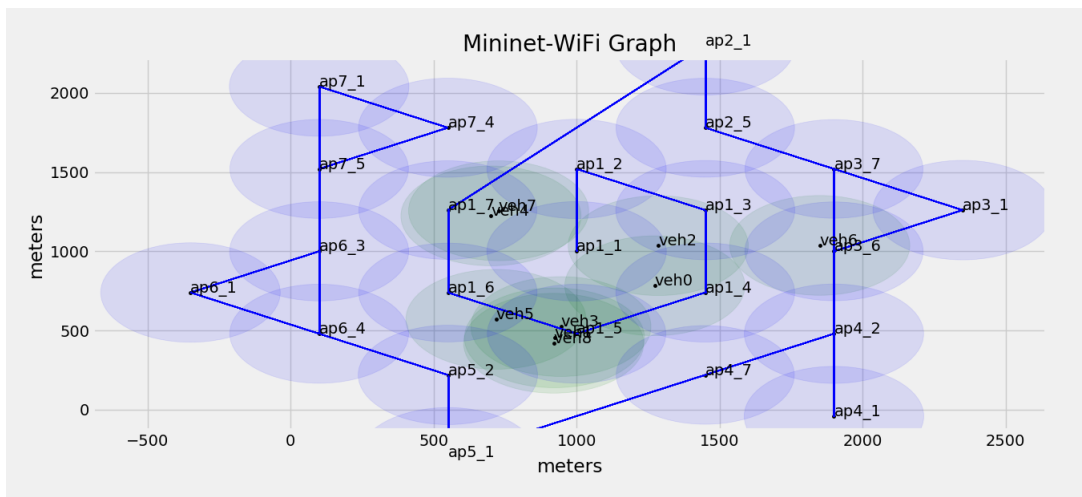


Figure 6.5: Network simulation start.

The following image, figure 6.6, shows the network time after the first image, showing how the network has evolved, vehicles have moved, and more AVs have joined the network.

From these images, it is also noticeable that there are instances where many AVs are close to each other, overlapping their labels. This occurs in situations such as AVs waiting in a traffic light. All these AVs waiting in traffic are connected to the access point with the strongest signal, therefore they are all connected to the same access point, placing it under a lot of traffic as these vehicles use its resources to send information to the network.

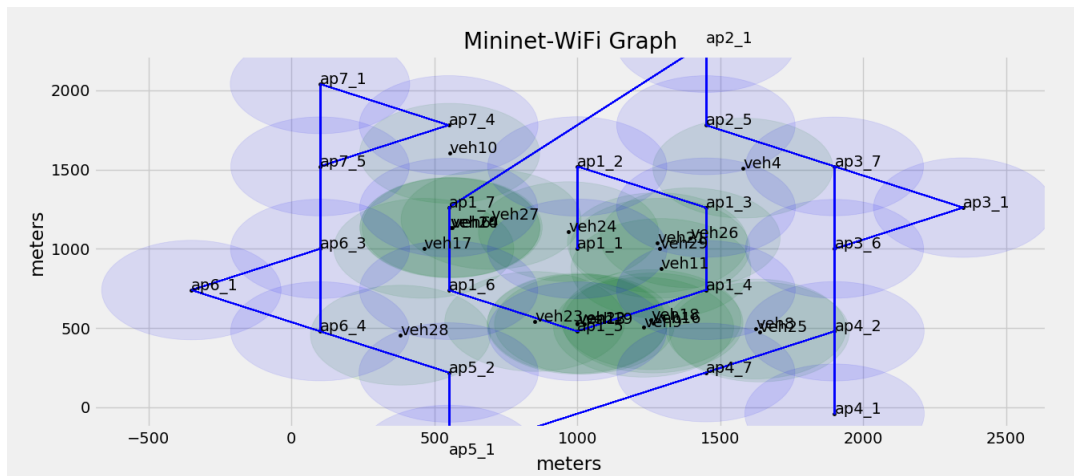


Figure 6.6: Network simulation with more participants in the network.

The last image, figure 6.7, shows the speed of movement in the network, as even if it was taken only a few instances after the previous image, figure 6.6, the position of the vehicles is noticeably different.

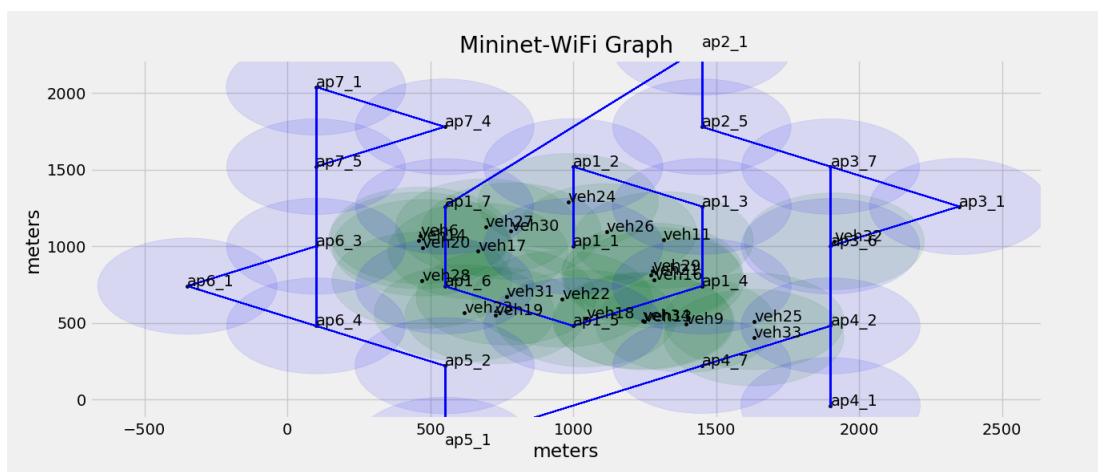


Figure 6.7: Network simulation representation to compare the position of the AVs with the previous image.

The network simulation keeps running until all AVs programmed have fulfilled their routes and exit the network.

6.4.6 Connectivity Performance

To analyze the connectivity performance of AVs with the cellular network, the Received Signal Strength Indicator (RSSI) metric has been utilized.

RSSI is a valuable metric for assessing the strength of radio signals received and the quality of communications, representing the connectivity and strength of the signal received from the cellular network to the AVs. Figure 6.8 shows two plots of the RSSI received by two vehicles of the network. When the AV is stopped by a red light or a stop sign, the RSSI plot represents this pause as a straight line, showing that while the vehicle is stopped the signal received does not change. Peaks in the figure represent instances where the vehicle transitioned between base towers due to its movement, shifting its connection to the signal of the new base tower, which has a stronger signal received.

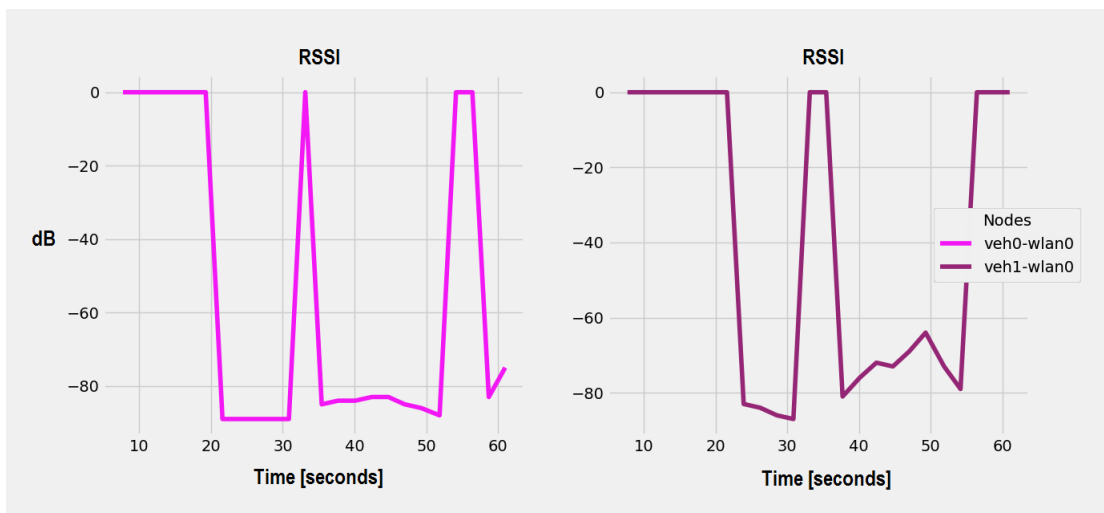


Figure 6.8: RSSI measurement of two AVs in the network.

These results are significant, particularly considering the high mobility in vehicular networks and therefore their frequent cell transitions. Each transition between cells momentarily disrupts communication of the AV with the network via the Access Points, rendering the vehicles more susceptible to attacks during these intervals.

A larger co-channel subdivision factor would result in more subcells with reduced coverage, potentially enhancing QoS but also leading to increased offline

periods for AVs as they move between them with their high mobility profiles.

Chapter 7

Modeling Vehicular Network Data Traffic

Once the network environment is properly configured, the next task is to ensure that the network functions and that AVs are able to communicate to the data server, which was confirmed by performing a series of ping operations, mainly to verify that the AVs had connection with the server. As it has been explained thorough this project, AVs need to communicate with its network in order to function as optimally as possible. To perform long range transmissions with servers, there are two main transport layer protocols used by AVs, depending on the type of transmission to be performed.

Transmission Control Protocol

Transmission Control Protocol (TCP) is a transport layer protocol used with the Internet Protocol (IP). It is a connection-oriented protocol, which means that before the data is transmitted, it establishes a connection between source and destination, which allows both users to transmit key packets such as acknowledgment (ACK) and synchronization (SYN) packets. ACK packets are specially important in TCP, as they ensure that the information sent arrives accurately and in the correct order, using these ACK packets to confirm the proper reception or to ask for the re-transmission of any lost or corrupted packets. TCP is also able to detect and correct errors thanks to the checksum functions that are incorporated in its

packets, in addition to having flow and congestion mechanisms to prevent a sender from overwhelming a receiver and to manage the transmission rate based on the network conditions to not congest the network.

All these properties make TCP a suitable protocol to use in situations where reliability or ordered delivery of packets is necessary. For AVs, these situations could be the exchange of sensible data with servers, data that needs to be correct such as updates in the path that the vehicle should take. AVs may also create communication traffic because of the user that is transporting. This user may want to utilize applications from the AV such as Spotify to listen to music, which utilizes the TCP protocol to send its data [29]; or its passengers may want to watch Netflix, which also utilizes TCP to transmit its information [30].

User Datagram Protocol

User Datagram Protocol (UDP) is another transport layer protocol used with IP, which unlike TCP, does not require a previous connection to transmit data, making UDP a connectionless protocol. This means that UDP does not need to establish a connection with its receptor to transmit data, it directly transmits it, requiring less amount of packets sent through the network transmit information as it does not require the sending of request and ACK packets to start the transmission. Its lack of a previous connection, added to its minimal overhead, makes it a fast protocol to transmit data, although this approach has some disadvantages. UDP is unable to know if the data has arrived to its destination, and if it has arrived in order, which is why it transmits its data as independent packets, called datagrams. In addition, applications that utilize UDP must be able to tolerate packet loss, duplication, and no order in reception. These characteristics make UDP a great protocol to use for applications where speed is more critical than reliability, being used in applications such as video transmissions or voice communications, where the user requires that the frames of the video or the voice of the receptor arrive as fast as possible. In AV networks, the UDP protocol is used to send metrics and data to its data servers to keep them informed about its status, position, current route, and other aspects to safeguard its performance [31].

Its lack of previous connection to transmit information makes UDP a popular

protocol to be used in DoS attacks, being used to flood the server with false UDP packets utilizing all its resources and making it unable to work properly for its users. Therefore, this project will focus on analyzing the case of AVs transmitting UDP packets to a data server, and scenarios where there are malicious users in the network flooding the server with UDP packets.

7.1 Generation of Data Traffic

MGEN [12] facilitates the creation of communications and network traffic among the AVs. Using this tool, AVs will engage in communication with a server, simulating the exchange of vital information such as location, tire pressure, direction, traffic conditions, and other data. This software is open source, and has been developed by the U.S. Naval Research Laboratory to generate and load traffic patterns using scripts. It can emulate the transmission of UDP/IP and TCP/IP packets, simulating applications that utilize them. For the purpose of this project, it will be used to simulate UDP/IP data transmissions between the AVs and a data server. MGEN works by creating scripts that are executed in the devices that will be participating in the network. Two types of scripts are needed, scripts to allow the reception of data and scripts to transmit data

7.1.1 Reception Scripts

The first step for transmission of data is to have the receptor ready to receive data. The scripts used to configure the nodes to be able to receive data is shown in listing 7.1.

Listing 7.1: Scheme followed to configure a node to receive data in MGEN

```
<eventTime> LISTEN <protocol> <portList>
```

Each field of the script is further explained bellow:

- "EventTime" represents the time (in seconds) at which the action should be performed, relative to the time at which the script was executed.

- "Protocol" configures the transport protocol to be listened, such as UDP and TCP.
- "PortList" is a list of port numbers, from the host that will be executing this script, that will begin monitoring and receiving the data traffic selected.

With this information, the reception script to be executed in the data server is created, listening to UDP traffic from the beginning until the end of the execution. Listing 7.2 shows a possible reception script to be used in the data server to accept UDP traffic in two ports.

Listing 7.2: Example of reception script in MGEN

```
0.0 LISTEN UDP 5000-5001
```

7.1.2 Transmission Scripts

To transmit information, one must configure the transmission script that will be executed in the AVs. The scheme followed to configure the transmission scripts is represented in listing 7.3, where three different type of events are shown: ON, MOD and OFF.

Listing 7.3: Scheme and events used to design a transmission script in MGEN

```
<eventTime> ON <flowId> <protocol> [connect] DST <addr>/<port>  
<pattern [params]> [<options ...>] [DATA [<hex><hex>]]
```

```
<eventTime> MOD <flowId> [<options ...>]
```

```
<eventTime> OFF <flowId>
```

There are three type of events in transmission scripts:

- ON events are utilized to start a new transmission flow at the time given by the "EventTime" parameter. To create the event, a "FlowID" is required, as it is utilized to identify the flow that has been created in the script, to allow future modifications to this flow by using the MOD or OFF events.

- MOD events are used to modify the flow given by the "FlowID" at the time selected in "EventTime".
- OFF events functionality is to disconnect the traffic flow created at the time configured in "EventTime".

To create a transmission flow, there are many parameters that can be configured. "EvenTime" and "FlowID" have already been explained, but there are more configurations available. The "protocol" field, just like in the reception script, represents the type of transport protocol that will be implemented, supporting as options "TCP", "UDP" and "SINK". The optional parameter "[connect]" is used to further configure different options for the connection, such as to instate from which port will the transmission be executed, which would be indicated with "SRC PortNumber".

To select a destination for the communications, it is used the label "DST" followed by the IP Address of the receiver in addition to the port that will receive the transmission. "Pattern" is one of the most important configurations of the flow transmission, modifying the amount of data to be sent in each transmission in addition to the transmission pattern that is followed, such as Periodic communications, communications that follow a Poisson distribution, and others.

The data flow has been configured to follow a Poisson distribution, which means that while there is an average rate of data arrival, there can be variability around this average. This approach is useful to simulate how communications, even if programmed to be sent periodically, could arrive at a different time than expected. This variability allows the machine learning model to learn that, while there is an expected time of arrival, packets arriving slightly earlier or later are still valid. To perform a Poisson pattern, one must follow the configuration scheme shown in listing 7.4.

Listing 7.4: Scheme configuration of Poisson transmission pattern

```
... POISSON [<aveRate (msg/sec)> <size (bytes)>] ...
```

Using this pattern, messages will generate at intervals varying following the Poisson distribution at an average of "aveRate", in messages per second. These

messages will carry as much data as indicated in "Size", in bytes. If UDP is being used as protocol, the maximum quantity of data that one can send is 8192 bytes.

As options, one can change different aspects of the flow such as sending only a limited number of packets and stopping afterwards, changing the Time To Live (TTL), and other properties. For the creation of the flows used in the simulation, a TTL of 30 hops was configured, as the simulator had by default a TTL of only 3 hops and the network created could require more hops to arrive to the data server. With 30 hops it is ensured that the data packets can arrive to its destination, and that the packets do not remain in the network for too long if they get lost. Data is the last option available, in which it is possible to specify the data to be transferred in hexadecimal form. This option was not configured

Knowing how to create transmission flows using scripts, it is now possible to design a transmission flow per each AV. As previously explained, each vehicle joins the network once their route begins, and leaves the network when they have reached its destination. To mimic this behavior, AVs will begin their transmission of UDP packets when they join the network, time denoted in the EventTime of the ON event; and will stop transmitting once they turn off and leave the network, configuring this time in the EventTime attribute of the OFF event. Using the previous reception example script, listing 7.5 shows a possible transmission script to be used in an AV that uses the configurations described, joins the network at the second 101 of execution, and leaves it at the second 239. It will use the port 5001 as source of its transmissions and will send it to 10.0.1.100 port 5001, following a Poisson distribution of 2.1 packets per second each with 1500 bytes of data, having as options the configuration of a TTL of 30 hops.

Listing 7.5: Example of a transmission script in MGEN

```
101 ON 1 UDP SRC 5001 DST 10.0.1.100/5001 POISSON [2.1 1500] TTL 30  
  
239 OFF 1
```

7.1.3 Denial of Service Attack Design

Denial of Service (DoS) attacks, as mentioned in Chapter 2, are attacks whose goal is to render a service unavailable to its users by overwhelming the target with a flood of packets and requests to consume all the target's resources, making it unable to provide a service to its true users.

DoS attacks could be accomplished in many different ways, such as transmitting packets to different ports in the server to make it generate an Internet Control Message Protocol (ICMP) per transmission to inform the transmitter that the port it is trying to access is not available, consuming the server's resources. Meanwhile, this project aims to cover attacks that are specifically targeted to the data server, attacks that could be performed by flooding the ports that are available for UDP transmissions. The amount of packets fills the buffers of the data server, making it unable to process legitimate communications.

To simulate this attack, the hacking scenario has been created to simulate an attack from multitude of sources, configuring the AVs to have a twenty percent chance of being attackers and therefore transmitting fifty times more packets than a true vehicle. Figure 7.1 shows the data traffic that would be seen from the data server, being flooded by a malicious user with the IP 10.0.0.4. As seen in the columns "Time", it barely passes time before another UDP packet from this user arrives, filling up the buffer of the server and using its resources.

7.2 Data Capture from the Network

Once the normal and malicious simulations are set up, it is now important to manage the network and capture the network traffic from the AV network. The tool utilized to capture the network activity has been Wireshark, which as explained in Chapter 2, it allows the user to capture network traffic and get detailed information of the packets transmitted over the network.

The network traffic is generated inside mininet-wifi's environment, therefore it is needed to configure the network simulator to allow external programs to view its traffic. This was accomplished by opening and allowing an interface in the virtual machine to access to the traffic exchanged inside the network simulator, enabling

| udp | | | | | | |
|------|--------------|----------|-------------|----------|--------|----------------------|
| No. | Time | Source | Destination | Protocol | Length | Info |
| 5372 | 20.601446700 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5374 | 20.601461517 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5376 | 20.601482858 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5378 | 20.601562657 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5380 | 20.601585331 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5382 | 20.601605412 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5384 | 20.601627159 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5386 | 20.601647390 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5388 | 20.601667148 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5390 | 20.601684947 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5392 | 20.601712418 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5394 | 20.601735163 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5396 | 20.601756806 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5398 | 20.601852612 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5400 | 20.601872406 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5402 | 20.601891048 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5404 | 20.601908277 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5406 | 20.601919236 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5408 | 20.601930407 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5410 | 20.601941251 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5412 | 20.601952043 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5414 | 20.601997519 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5416 | 20.602012714 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |
| 5418 | 20.602024036 | 10.0.0.4 | 10.0.1.100 | UDP | 102 | 5001 → 5001 Len=1500 |

Figure 7.1: Network capture of the server being attacked.

Wireshark to use this interface to capture data sent through the AV network.

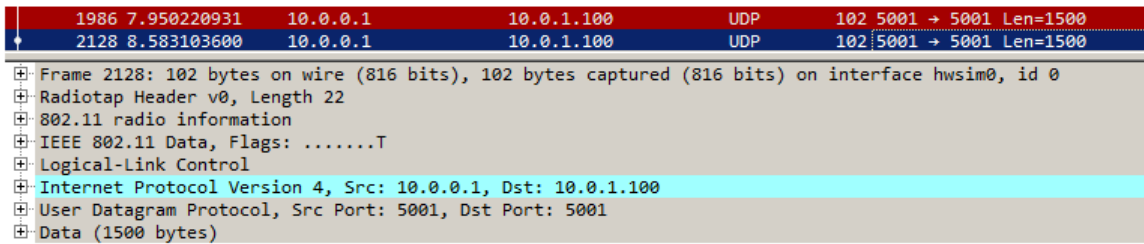
7.3 Data Cleaning

The data captured by Wireshark is saved as a binary file with the suffix ".pcapng", being illegible to applications that are not Wireshark. To start with the data cleaning, it is first needed to be able to access to the data. To accomplish this task, the library Pyshark was used, which allows Python to read and analyze the contents of the packets captured [32].

Wireshark saves all data transmitted through the network being monitored, and the interface utilized is able to see all traffic of the network, therefore it captures packets that are not directly related to the AV network such as broadcast packets and control information that Access Points send to each other. The first task is therefore to filter out these packets and visualize only the traffic important for this project, UDP packets.

The next step is to select the data that could be useful for the model. Each data packet has a great amount of information saved, structured in layers, but not all of this data is relevant to the model. Figure 7.2 shows the different layers of

information that are available to analyze from a regular UDP packet.



| No. | Time | Source | Destination | Protocol | Length | Info |
|------|-------------|----------|-------------|----------|--------|--------------------------|
| 1986 | 7.950220931 | 10.0.0.1 | 10.0.1.100 | UDP | 1500 | 102 5001 → 5001 Len=1500 |
| 2128 | 8.583103600 | 10.0.0.1 | 10.0.1.100 | UDP | 1500 | 102 5001 → 5001 Len=1500 |

```

Frame 2128: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface hwsim0, id 0
  Radiotap Header v0, Length 22
  802.11 radio information
  IEEE 802.11 Data, Flags: .....T
  Logical-Link Control
  Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.1.100
  User Datagram Protocol, Src Port: 5001, Dst Port: 5001
  Data (1500 bytes)

```

Figure 7.2: Normal UDP packet capture showing the different layers of information available in Wireshark.

Each packet has its information stored in different layers, hence the python code needs to be able to search in each layer the information that is used to train the model.

The primary layers used to obtain information are the network and transport layers. From these layers, it is possible to obtain details such as the IP address and port numbers of the source and destination, data length, and the time required for transmission. Additionally, information on the channel frequency has been collected, which could be particularly relevant as vehicles frequently switch between cellular networks, changing also the frequency used. In contrast, hackers might be stationary, making a constant frequency a potential detection metric. By correlating all these parameters with other metrics, such as the time elapsed since the last packet received from the same user and the total number of packets sent by each user, it is possible to train a model able to detect DoS attacks.

The last step needed to make the neural network able to determine whether a packet is part of an attack or not, is to label the information. Labeling the information means adding extra parameters to each data point given, to provide the model with the output and correct result. Using these labels, it can learn and identify the inner relations of the datam useful to determine the outcome for unseen data points.

Hackers are randomly selected on each simulation, therefore to know which users are hackers, after generating the MGEN script to be followed, a comma-separated values (csv) file is created with the IP of the hackers, labeling traffic coming from those IPs as hackers.

7.4 Data Format

7.4.1 Numerical Formatting

To train a neural network, it is desirable to have all information in a numerical format, either as integers or as decimal numbers (floats). The majority of the parameters included are already in a numerical format, except IP addresses. Values such as "10.0.0.1" would be saved as a String, making it unable to be processed. Instead, the value of each byte of the IP address has been linked together, creating thus an unrepeatable ID for each IP. The IP given in the previous example, 10.0.0.1, would have the ID of 10001. These IDs would be temporarily stored, with the purpose of being able to trace which IP address is a malicious user and therefore have possible responses such as to block the traffic incoming from such user. Another option to format IP addresses is to add a padding number in between each byte of the IP address. A possible padding option could be adding the number "999", which would make the previous IP address (10.0.0.1) be saved as 10999099909991. This option would allow tracing back the IP address by only using the ID and filtering the padding value (999), but would also need to utilize more memory to store each parameter.

Another parameter easily formatted to fit the new numerical request are the labels set to each data point. Instead of saving the label as a True or False values to indicate if it is a hacker or not, it is saved as a binary number, 0 or 1, being 0 a legitimate user and a 1 an infected or malicious user.

7.4.2 Data Normalization

Data normalization and standardization are techniques whose goal is to modify the range of the values in a dataset in order to give the input features a similar scale, as values without normalization may lead to having features with very large ranges that could influence the learning process, leading to a sub-optimal model [33]. Additionally, normalized data is also useful to help models converge faster in training and to obtain a higher performance rate, being therefore a desirable action to be taken with the data. The normalization of the data has been accomplished

using the library sklearn, concretely the StandardScaler function [34].

As normalization makes the data have a different representation than originally, data parameters such as the IP of the sender will not have the same structure as before. As it is desirable to obtain the IP of the sender that is attacking the network to take actions against it, the original representation could be obtained saving the normalized value of the ID of the sender and the IP to which it relates to. Another solution would be to save the mean, standard deviation, minimum and maximum value and undo the process manually.

7.5 Resulting Database for Model Training and Testing

Once the data has been captured, filtered, labeled and formatted, the resulting data is saved to train and test the model. The distribution of the data is represented in table 7.1, where the quantity and percentage of packets of legitimate users and DoS attackers are shown.

| | DoS Attackers | Legitimate Users | Total |
|-------------------|----------------------|-------------------------|--------------|
| Quantity | 91521 | 4051 | 95572 |
| Percentage | 0.957 | 0.043 | 1.00 |

Table 7.1: Distribution of labeled data in terms of packets from attackers and legitimate users.

It is worth noting the large amount of packets belonging to DoS attackers. This is to be expected, as DoS attacks function by generating a significant amount of data traffic to exhaust the server's resources. Consequently, the simulator and its attacks have been limited to four minutes of execution time. This limitation is necessary because DoS attacks generate much more data than legitimate users, and therefore a longer execution would result in an excessively imbalanced dataset, which would result in a less efficient model.

With the data properly captured, cleaned and configured, it is now ready to train the model and obtain results from it.

Chapter 8

Machine Learning Model for Denial of Service Attack Detection

As discussed in Section 2.5, machine learning (ML) is a powerful tool that allows computers to learn patterns from data without being explicitly programmed. By processing and analyzing data, these algorithms are able to understand the data and use it to develop rules and calculations with the data to be able to fulfill its task. Neural networks have been the model chosen to fulfill the defense of the AV network against DoS attacks, as their capacity to understand the characteristics and complexity of the data makes them a great candidate to achieve complex applications, such as predicting attacks in an AV network.

8.1 Model Development for Vehicular Network Data Traffic

The type of neural network model that has been chosen for this project is the feedforward neural network (FNN). In these networks, information only moves forward, from the first layer to the following, until reaching the output layer. FNNs are the simplest type of neural networks, therefore providing the fastest and most resource efficient models; features which are desirable for data servers to provide

a quick response against hackers, and to not use their resources on executing the model and instead focus on managing their communications and data operations. The model developed has been done utilizing python and the pytorch library [15], and utilizing concepts explained in the book [33].

8.1.1 Number of Layers and Neurons

To start developing a neural network model, it is needed to decide the number of hidden layers and neurons to be utilized. The input layer must have as many input neurons as properties and values of the data to be analyzed, therefore in this project the model trained has eleven input neurons.

Depending on the type of application and data used, the optimal number of neurons and hidden layers can vary. The process of selecting the number of layers and neurons per layer is performed by testing the performance with each layout until obtaining a desirable outcome. For complex applications, such as image recognition, a high number of neurons and layers is often chosen, as it enables the model to capture and comprehend patterns and features of the data to recognize the images given [35]. However, using too many neurons and layers for simpler tasks may lead to overfitting, having a model perform well with training data but poorly on new data. Contrary, few hidden layers and neurons might result in underfitting, where the model does not learn the underlying patterns and complexity of the data, therefore performing poorly.

To determine the amount of hidden layers to be used, several iterations have been made in the model, changing the amount of layers and neurons in each iteration until reaching a desirable equilibrium between complexity and efficiency. The first layout explored had one hidden layer. While these models were able to detect attacks and have a low complexity, its results were not as good as those with two hidden layers. This project aims to provide a critical service, the defense of the AV network and its servers, therefore it has been prioritized to obtain the best possible prediction results rather than having a less complex model with one less hidden layer. Using three hidden layers was also tested, however, the model showed signs of overfitting and performed worse than with two hidden layers, hence being discarded.

The number of neurons per layer was also tested and adjusted, resulting in the best results obtained with a first hidden layer with four times the amount of input neurons, having forty four nodes, and a second hidden layer with twice the amount of input nodes, having twenty two neurons.

Thanks to having more neurons than input nodes in the first hidden layer, it is able to capture and understand a large range of features and patterns from the input data. The following hidden layer, by having less neurons than its predecessor, forces the model to refine these features and promotes generalization, preventing overfitting. In addition, having less neurons involves utilizing less resources to run the model, improving its resource utilization and reducing its complexity. Finally, the output layer has only one neuron, as the desired outcome is only one number, which is the probability of the data of being from a hacker.

8.1.2 Model Training

The model needs to learn from the data and distribute weights to the connection between neurons to allow it to correctly predict the data. This process is performed during training, and is based on the error of its prediction using backpropagation.

Backpropagation calculates the gradient of the loss function with respect to each weight in the network, updating the weights to minimize the prediction loss. The process is performed after every prediction performed in training, to optimize the prediction outcome after every iteration and gradually improve its results.

It starts with a forward pass, having the data pass through the network and performing its actual weighted sums and activation functions, until reaching its output. The difference between the predicted output and the actual target is the loss. This is where backpropagation starts, performing a backward pass from the output layer to the previous hidden layer until reaching the input layer, to calculate the gradient of the loss function. The derivative of the loss function is performed, with the goal of finding the minimum of the function to update each weight in the direction obtained by its function, and therefore minimize the loss for the next iteration [33].

8.1.3 Loss Function

In this project, the final output is a binary classification of either 0 or 1, obtained by the output probability of the data belonging to the positive class (attacker). To calculate the loss function for binary results, it has been utilized the binary cross-entropy loss function (BCELoss), which calculates the difference between the predicted probability and the actual label [33]. Thanks to performing the gradient to each weight, all parameters in the network are updated proportionally to their contribution to the error.

8.1.4 Gradient Optimizer

To perform the gradient operation, the Adam optimizer has been utilized, which stands for Adaptive Moment Estimation. This gradient function has per-parameter learning rates adapted on the first and second moment of the gradients, resulting in a faster convergence. It also allows the user to modify its learning rate, which has been set to 0.001, to control the step size done after each iteration while moving towards a minimum of the loss function [33].

8.1.5 Learning Rate

The learning rate parameter is very important, as a high learning rate results in an inefficient model, due to its inability to converge and have proper weights for its neurons, and a too small learning rate could be too slow and result in the model not reaching an optimal weight distribution. This was tested by having a learning rate of 0.01, which produced a model that, while had good values of accuracy, was unable to distinguish legitimate traffic from attackers and would label all as hackers. This example showed the importance of the learning rate and specially the value of having multiple metrics to measure performance, as this model had an accuracy of 0.97, but when looked at the confusion matrix it was clear the model was not properly working.

8.1.6 Epochs and Batches

As previously explained, the model updates its weights and learns how to better predict the outcome after every iteration. In this learning process, feeding the model with data is key, as it allows the model to further upgrade itself until having optimal weights. Training is one of the most important parts of model development, which is why the training process is sometimes repeated to ensure that the model has properly learned. Each time the model executes the whole training data is called an epoch. During training with multiple epochs, data is given to the model in small groups, called batches. In FNNs, it is common to process data in batches instead of one at a time, as these groups of data allow the model to update the weight based on the average of the gradient of the loss function over all data in the batch, which makes the training more stable and spends less computational resources in training.

The amount of data per batch, also known as batch size, is an important parameter. A batch size of 32 data elements per batch was chosen. A too small batch may cause the model to upgrade its weights based on the noise of the data and therefore have trouble converging into the solution. Meanwhile, large batches may slow down training and, depending on the size chosen, might not be able to be executed due to a lack of memory to temporary save all those data points before being updated. These batches of data get their data ordered randomly to ensure that the model truly learns how to analyze the data structure and use it to predict the outcome, and does not learn instead the order of the training data to predict it. Forty epochs have been utilized to train this model, utilizing a scheduler to diminish the training rate every ten epochs, multiplying the learning rate by a factor of 0.1. This allows the model to avoid issues such as overshooting the minimum of the loss function, becoming more precise and converging better into the most efficient value to have better final model results [33].

8.1.7 Overfitting Prevention

The neural network includes dropout layers to help preventing overfitting, reducing the reliance on specific neurons. Dropout randomly zeros some inputs of the neurons with a specific probability, 50% of the time being zeroed for the case of

this model. This forces the model to not rely on single neurons and learn more robust features that are not dependent on particular nodes, ensuring that the model generalizes better to unseen data and preventing overfitting [33].

8.1.8 Backpropagation Issues and Solutions

Backpropagation is not a perfect technique, and can have issues such as the vanishing gradient problem, where gradients become too small as they are propagated backwards through the network, making it difficult for the network to learn as the weight updates are minimal. The opposite problem could also occur, called the exploding gradient problem, resulting in large gradients that lead to unstable updates. To mitigate the vanishing problem and ensure that the model learns appropriately, this model utilizes the ReLU function.

ReLU passes the input directly if it is a positive value, and outputs zero for any negative one. This means that ReLU has a consistent gradient of 1 for positive inputs, which prevents the gradient from diminishing as they propagate backwards through the network and layers. Therefore, ReLU allows gradients to remain significant therefore the model is updated and learns to produce less error in future iterations [33].

Data normalization has also been included, which improves the learning process and makes it more stable.

Once all these configurations are applied to the neural network, and that it has been thoroughly trained, it is tested to analyze its performance. Model results are shown in chapter 9.

8.2 Metrics to Analyze Performance

Once a model is created and trained, it is necessary to test its efficiency with data it has not been trained on and obtain various metrics to analyze its performance. Many of these metrics rely on the concepts of true positives, false positives, true negatives and false negatives, which are explained below. Data is labeled as 1 or 0, with 1 being data from an attacker and 0 indicating legitimate users. Data labeled as one is called positive, and data labeled as zero is called negative.

- **True Positives (TP)** are instances correctly predicted as 1 (attackers).
- **True Negatives (TN)** are instances correctly predicted as 0 (legitimate users).
- **False Positives (FP)** are instances predicted as 1 (attackers) which are actually 0 (legitimate users). False positives represent legitimate users who have been misclassified as attackers.
- **False Negatives (FN)** are instances predicted as 0 (legitimate users) which are actually 1 (attackers). False negatives are particularly important to measure, as it is not desirable to classify hacker traffic as legitimate.

The metrics used to analyze performance are accuracy, precision, recall, F1 Score, receiver operating characteristic area under curve (ROC AUC), confusion matrix and execution time.

8.2.1 Accuracy

Accuracy measures the ratio of correct predictions, both TP and TN, to the total of instances predicted. Its formula is shown in equation 8.1.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (8.1)$$

8.2.2 Precision

Precision measures the ratio of true positive predictions among all positive predictions. It is useful to analyze out of all positive predictions, how many of them were actually positive. Its formula is represented in equation 8.2.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (8.2)$$

8.2.3 Recall

Recall, also called sensitivity, measures the proportion of true positives predicted out of all actual positives, including those that have been misidentified as negatives

(False Negatives). It indicates how well the model captures positive instances (attackers), and is specially useful in this project to analyze the amount of hackers that are misclassified as legitimate users, therefore being one of the most important metrics for this project. Its formula is represented in equation 8.3.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (8.3)$$

8.2.4 F1 Score

F1 Score is the harmonic mean of precision and recall. It is useful to unify both metrics into a single metric, utilizing both of them to obtain a balanced metric, making it a useful metric for imbalanced datasets such as the one that is being used. Its formula is shown in equation 8.4.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (8.4)$$

8.2.5 ROC AUC

The ROC AUC (Receiver Operating Characteristic Area Under Curve) measures the ability of the model to distinguish between classes. It is obtained by calculating the area under the curve generated by plotting the true positive rate against the false positive rate.

8.2.6 Confusion Matrix

The confusion matrix is used to show the data classified by the model, representing true positives, true negatives, false positives, and false negatives. The format of the matrix used is represented in table 8.1.

| | Predicted Negative | Predicted Positive |
|------------------------|---------------------------|---------------------------|
| Actual Negative | True Negative (TN) | False Positive (FP) |
| Actual Positive | False Negative (FN) | True Positive (TP) |

Table 8.1: Confusion matrix scheme.

8.2.7 Execution Time

Execution time measures the time it has taken the model to predict data. It is crucial to evaluate the prediction time of the model, as the objective is to have it proportionate a quick and reliable response against attacks in the AV network.

Chapter 9

Performance Analysis and Numerical Results

This project has successfully achieved its objective of creating an environment to simulate AV networks under different conditions such as under DoS attacks, and provide a neural network model to defend the network against them. This environment is capable of representing vehicular movement, characteristic from AV networks, utilizing SUMO to obtain mobility data from real locations, choosing Madrid as the urban area to be simulated. This movement data is inputted to the network simulator, Mininet-Wifi, where a cellular network was designed to provide connectivity between the AVs and a data server to communicate with. Data communications were implemented with MGEN, generating data traffic exchange between the AVs and the data server, transmitting UDP packets to simulate the transmission of metrics and other parameters. The same tool was used to create hacking scenarios, particularly DoS. To understand the value provided by the neural network defense, it is important to analyze the performance and impact on the network that DoS attacks have. Therefore, the network has been analyzed in a regular scenario and while being under attack, utilizing the network tools ping and iperf.

9.1 Autonomous Vehicle Network Performance

9.1.1 Ping Performance Results

Ping is a tool used to test connectivity between users on an IP network, measuring the round-trip time (RTT) of packets sent, which is used to measure network latency. It works by sending an Internet Control Message Protocol (ICMP) echo request to the target, and waiting for its echo reply. If the echo reply does not arrive, the packet is counted as lost, being therefore able to also detect packet loss in the network. The metrics measured by ping are the minimum RTT (Min RTT), average RTT (Avg RTT), maximum RTT (Max RTT) and mean deviation of RTT (Mdev RTT), all which have been measured in milliseconds (ms). The result of these metrics for both situations is represented in table 9.1, showing also the percentage increase between scenarios.

| Situation | Min RTT (ms) | Avg RTT (ms) | Max RTT (ms) | Mdev RTT (ms) |
|---------------------|--------------|--------------|--------------|---------------|
| No Attack | 0.850 | 6.510 | 26.752 | 8.028 |
| Attack | 0.865 | 17.732 | 292.126 | 47.792 |
| Percentage Increase | 1.76% | 172.37% | 991.77% | 495.29% |

Table 9.1: Ping metrics comparison between normal and DoS attack scenario.

RTT values, specially the average RTT, show that during an attack the network suffers a great decrease in performance, increasing the time needed to transmit simple ping packets from 6.51ms to 17.732ms on average, being a 172.37% increase in response time. This affects the network latency, making it less fast and therefore less efficient, both undesirable characteristics of an AV network that needs quick and reliable packet transmission. Additionally, packet loss and packet errors have also been explored, having the normal scenario with no packets dropped or arrived with errors. However, the DoS scenario had 33.3% packets dropped, receiving 36 packets out of 54 transmitted, and out of those packets received, one arrived with errors; further showing the damage that DoS attacks would have in the AV network.

9.1.2 Iperf Performance Results

Iperf is a network tool used to measure the bandwidth between two nodes in a network. It works by generating TCP and UDP traffic, and testing the maximum bandwidth that can be achieved between the users selected. The bandwidth available for TCP and UDP communications in both situations is represented in table 9.2, showing a great decrease in the rate of communications, specially for UDP transmissions.

| Situation | TCP Bandwidth (Mbits/sec) | UDP Bandwidth (Mbits/sec) |
|---------------------|---------------------------|---------------------------|
| No Attack | 17.7 | 18.200 |
| Attack | 9.38 | 0.334 |
| Percentage Decrease | 46.95% | 98.16% |

Table 9.2: Iperf metrics comparison between normal and DoS attack scenario.

Attackers are utilizing the UDP protocol to overflow the server and use all its resources, noticeably reducing the network's bandwidth for UDP transmissions, having a bandwidth decrease of 98.16%. TCP transmissions are also affected, having a 46.95% decrease in transmission rate.

Speed and reliability are key elements in AV networks, and DoS attacks harm both properties, reducing communication rates and therefore damaging the performance and security of autonomous vehicles. The performance metrics obtained from the network are key to understanding the importance of detecting these attacks to defend against them, further strengthening the need and importance of the neural network created to defend AV networks against these attacks.

9.2 Model Performance

Data capture from the network has been obtained using Wireshark. This data has been filtered, cleaned, configured and labeled to be utilized as a database to train and test the neural network, with the goal of teaching it how to detect DoS attacks and therefore enabling the AV network to take measures against them.

The neural network has been configured and tested until obtained a model with excellent performance in both prediction metrics and speed of performance, successfully obtaining a model capable of being implemented in the AV network

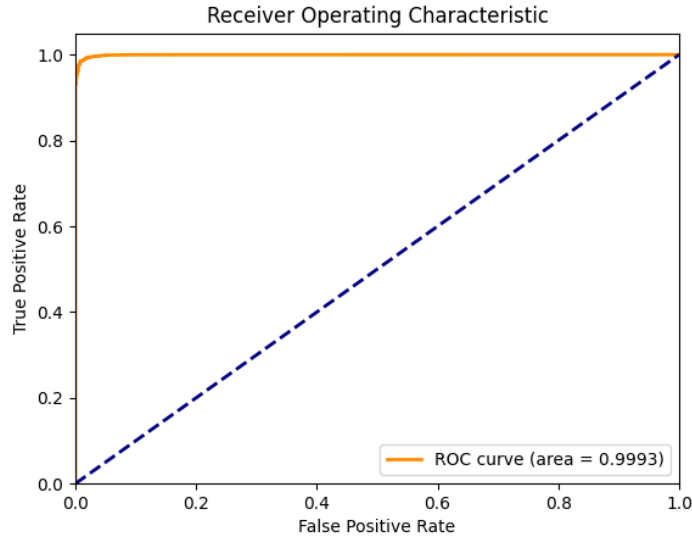


Figure 9.1: ROC curve plot.

and its data servers to defend them against DoS attacks. The definitive model has been trained using eighty percent of the data over multiple iterations, spending approximately two minutes to be trained over forty epochs, making it a quick model to be trained and updated.

The model has been tested utilizing the remaining twenty percent of the data to analyze its performance against new and unseen data, showing its results in table 9.3, having all performance metric achieve values over 99.63%, which represents an excellent capability in data classification between legitimate and attacker traffic. Figure 9.1 is included to represent the ROC curve and its area, proving the value obtained for the ROC AUC metric.

| Metric | Value |
|-----------|--------|
| Accuracy | 0.9963 |
| Precision | 0.9969 |
| Recall | 0.9992 |
| F1 Score | 0.9981 |
| ROC AUC | 0.9993 |

Table 9.3: Performance metrics of the model against unseen data.

These metrics indicate an excellent success in model prediction. To further

show its efficiency predicting attacks, table 9.4 is provided, where the confusion matrix of the predictions is given, further proving its excellent capability to detect DoS attacks, having only 14 data packets from attackers misclassified as legitimate traffic out of a total of 18316 packets from attackers.

| | Predicted Legitimate User | Predicted Attacker |
|-------------------------------|----------------------------------|---------------------------|
| Actual Legitimate User | 742 | 57 |
| Actual Attacker | 14 | 18302 |

Table 9.4: Confusion matrix obtained with the model.

To conclude, AV network applications need great performance results in addition to having a quick execution time. The time needed for the model to predict the unseen data was 31.20 milliseconds, analyzing a total of 19115 data points, therefore spending 1.63 microseconds predicting each data packet, ensuring quick response from the data server against attacks, preventing it from further damaging the network.

Chapter 10

Conclusion and Future Works

10.1 Conclusion

In this paper, an environment to simulate AV networks during urban mobility scenarios has been developed and used to analyze and perform the network under DoS attacks, with the goal of gathering data and train a neural network to defend the network against such attacks.

The creation of the environment involves the simulation of vehicular movement in real locations, using Madrid as a case study, and gathering data from these mobility scenarios to represent AV movement in the network.

Integration of vehicular mobility data in the network simulator was performed, alongside the development of a cellular network to provide network connection to the AVs simulated.

The network was used to simulate two scenarios, a regular scenario with data transfers between AVs and data server, and a DoS scenario where UDP packets overflow the server. Analysis of the impact of DoS attacks in the network was performed, showing the importance of developing detection measures to defend the AV network.

Data was gathered from both simulations to train a neural network, successfully obtaining a model capable of detecting DoS attacks with great performance results in both prediction metrics and execution speed, therefore making it a suitable addition for vehicular network detection of DoS attacks.

10.2 Future Works

To further improve this project, it would be beneficial to simulate more types of communications in AV networks, such as V2V communication and the usage of other transport protocols such as TCP/IP.

With these new communications included, more types of attacks could be simulated and therefore the neural network model could be improved to be able to detect more types of attacks.

Additionally, improvement to this project could be made by designing a defense reaction to be taken against hackers once detected by the neural network.

Other improvements could be to perform mobility simulations in new locations and with different traffic situations, to analyze the performance of the network under different traffic conditions and layouts. Different cellular network designs could also be explored, such as utilizing a different cell size or number of servers in the simulation.

Appendix A

Project Alignment with the Sustainable Development Goals

This project aims to help and contribute with the following Sustainable Development Goals (SDG).

SDG 3: Good health and well-being

By reducing the risk of accidents product of hacking attacks, this project helps improve the well-being and health of AV riders and the rest of users of the road, as it reduces the risk of hacking accidents.

SDG 9: Industry, Innovation and infrastructure

Improving the AV network and making a machine learning model which allows their communication network to be more resilient against attacks completely aligns with the innovation side of the objective and with improving the resilience of their infrastructure.

SDG 11: Sustainable cities and Communities

Making AVs safer has direct consequences in the development of sustainable communities, as they consume energy much more efficiently than a human driven car and are much safer than them, making communities a safer and more sustainable place.

SDG 16: Peace, justice and strong institutions

AV hacking could be used as a form of terrorism and start of conflicts, as once a hacker has gained control over the AV it could crash the car on purpose or not let the riders get out of the car. By protecting AV against hackers, this project is contributing to keeping a peaceful community and to maintaining the integrity of technological infrastructure.

Bibliography

- [1] J. Shuttleworth. *SAE Standard News: J3016 Automated-Driving Graphic Update*. Available online: <https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic> (accessed on [insert date of access]).
- [2] D.J. Yeong et al. “Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review”. In: *Sensors* 21 (2021), p. 2140. DOI: 10.3390/s21062140. URL: <https://doi.org/10.3390/s21062140>.
- [3] Md. Noor-A-Rahim et al. “A Survey on Resource Allocation in Vehicular Networks”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.2 (2022), pp. 701–721. DOI: 10.1109/TITS.2020.3019322.
- [4] Manzoor Ahmed et al. “Vehicular Communication Network Enabled CAV Data Offloading: A Review”. In: *IEEE Transactions on Intelligent Transportation Systems* 24.8 (2023), pp. 7869–7897. DOI: 10.1109/TITS.2023.3263643.
- [5] Alessandro Bazzi et al. “On the Performance of IEEE 802.11p and LTE-V2V for the Cooperative Awareness of Connected Vehicles”. In: *IEEE Transactions on Vehicular Technology* 66.11 (), pp. 10419–10432. DOI: 10.1109/TVT.2017.2709783.
- [6] R. Hasegawa and E. Okamoto. “Adaptive Transmission Suspension of V2N Uplink Communication Based on In-Advanced Quality of Service Notification”. In: *Vehicles* 5.1 (2023), p. 1. DOI: 10.3390/vehicles5010012.
- [7] Abdullah Algarni and Vijey Thayananthan. “Autonomous Vehicles: The Cybersecurity Vulnerabilities and Countermeasures for Big Data Communication”. In: *Symmetry* 14.12 (2022). ISSN: 2073-8994. DOI: 10.3390/sym14122494. URL: <https://www.mdpi.com/2073-8994/14/12/2494>.
- [8] Anastasios Giannaros et al. “Autonomous Vehicles: Sophisticated Attacks, Safety Issues, Challenges, Open Topics, Blockchain, and Future Directions”. In: *Journal of Cybersecurity and Privacy* 3.3 (2023), pp. 493–543. ISSN: 2624-

- 800X. DOI: 10.3390/jcp3030025. URL: <https://www.mdpi.com/2624-800X/3/3/25>.
- [9] Pablo Alvarez Lopez et al. “Microscopic Traffic Simulation using SUMO”. In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: <https://elib.dlr.de/124092/>.
- [10] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. <https://www.openstreetmap.org>. 2017.
- [11] R. R. Fontes et al. “Mininet-WiFi: Emulating Software-Defined Wireless Networks”. In: *2015 11th International Conference on Network and Service Management (CNSM)*. 2015, pp. 384–389. DOI: 10.1109/CNSM.2015.7367387.
- [12] *Multi-Generator Network*. Available: <https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN/>.
- [13] Wireshark Foundation. *Wireshark: Network Protocol Analyzer*. <https://www.wireshark.org/>. 2024.
- [14] Yılmaz Koçak and Gülesen Üstündağ Şiray. “New activation functions for single layer feedforward neural network”. In: *Expert Systems with Applications* 164 (2021), p. 113977. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2020.113977>. URL: <https://www.sciencedirect.com/science/article/pii/S0957417420307557>.
- [15] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [16] D.J. Yeong et al. “Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review”. In: *Sensors* 21.6 (2021), p. 2140. DOI: 10.3390/s21062140. URL: <https://doi.org/10.3390/s21062140>.
- [17] Minh Pham and Kaiqi Xiong. “A survey on security attacks and defense techniques for connected and autonomous vehicles”. In: *Computers & Security* 109 (2021), p. 102269. DOI: 10.1016/j.cose.2021.102269.
- [18] Electronics360. *System redundancy in self-driving cars*. <https://electronics360.globalspec.com/article/17200/system-redundancy-in-self-driving-cars>.
- [19] Nidhee Kamble et al. “Using Blockchain in Autonomous Vehicles”. In: *Artificial Intelligence and Blockchain for Future Cybersecurity Applications*. Ed. by Yassine Maleh et al. Cham: Springer International Publishing, 2021, pp. 285–305. DOI: 10.1007/978-3-030-74575-2_15.

-
- [20] S. Kumar et al. “Delimitated Anti Jammer Scheme for Internet of Vehicle: Machine Learning based Security Approach”. In: *IEEE Access* (), pp. 113311–113323. DOI: 10.1109/ACCESS.2019.2933850.
- [21] Qiyi He et al. “Machine Learning-Based Detection for Cyber Security Attacks on Connected and Autonomous Vehicles”. en. In: *Mathematics* 8.8 (2020). Number: 8 Publisher: Multidisciplinary Digital Publishing Institute, p. 1311. ISSN: 2227-7390. DOI: 10.3390/math8081311. URL: <https://www.mdpi.com/2227-7390/8/8/1311>.
- [22] *UCI KDD Cup 1999 Data Data Set*. 1999.
- [23] LawInfo. *How Many Car Accidents Are Caused by Human Error?* <https://www.lawinfo.com/resources/car-accident/how-many-car-accidents-are-caused-by-human-er.html>.
- [24] MarketsandMarkets. *Autonomous Passenger Car Market*. <https://www.marketsandmarkets.com/Market-Reports/near-autonomous-passenger-car-market-1220.html>.
- [25] SUMO. *TraCI*. <https://sumo.dlr.de/docs/TraCI.html>.
- [26] Kaveh Pahlavan and Prashant Krishnamurthy. *Principles of Wireless Networks: A Unified Approach*.
- [27] Gabriele Gemmi, Renato Lo Cigno, and Leonardo Maccari. “On Cost-Effective, Reliable Coverage for LoS Communications in Urban Areas”. In: *IEEE Transactions on Network and Service Management* 19.3 (2022), pp. 2767–2779. DOI: 10.1109/TNSM.2022.3190634.
- [28] Andrea Goldsmith. *Wireless Communications*. Draft of Second Edition. 2020.
- [29] G. Kreitz and F. Niemela. “Spotify – Large Scale, Low Latency, P2P Music-on-Demand Streaming”. In: *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*. 2010, pp. 1–10. DOI: 10.1109/P2P.2010.5569963.
- [30] Trinh Viet Doan, Vaibhav Bajpai, and Sam Crawford. “A Longitudinal View of Netflix: Content Delivery over IPv6 and Content Cache Deployments”. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 2020, pp. 1073–1082. DOI: 10.1109/INFOCOM41043.2020.9155367.
- [31] “IT Based Vehicle Tracking System for Effective Management in Public Organizations”. In: *Procedia Economics and Finance* 33 (2015), pp. 506–517. DOI: 10.1016/S2212-5671(15)01733-5.
- [32] KimiNewt. *PyShark: Python Packet Parser using Wireshark’s TShark*. GitHub repository. 2024. URL: <https://github.com/KimiNewt/pyshark>.

BIBLIOGRAPHY

- [33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [34] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [35] G. Bebis and M. Georgiopoulos. “Feed-forward neural networks”. In: *IEEE Potentials* 13.4 (1994), pp. 27–31. DOI: 10.1109/45.329294.