Universidade de Aveiro
2022

David Manuel Teixeira Alves da Rocha

# Recolha de dados em veículos conectados para aplicações de segurança rodoviária

# Data collection on connected vehicles for road safety applications

Universidade de Aveiro
2022

**David Manuel Teixeira Alves da Rocha**

# Recolha de dados em veículos conectados para aplicações de segurança rodoviária

# Data collection on connected vehicles for road safety applications

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Joaquim José de Castro Ferreira, Professor Coordenador da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro, e do Doutor João Miguel Pereira de Almeida, Investigador do Instituto de Telecomunicações.

Este trabalho foi realizado no âmbito do projeto TRUST com o nº 037930 (POCI-01-0247-FEDER-037930), financiado pelo FEDER através do programa COMPETE 2020, do projeto europeu 5G-MOBIX com o nº 825496, financiado pelo programa de investigação e inovação Horizonte 2020 da União Europeia, e da FCT/MCTES através de fundos nacionais e, quando aplicável, cofinanciado por fundos comunitários no âmbito do projeto UIDB/50008/2020-UIDP/50008/2020.

Dedico este trabalho aos meus pais, Manuel Rocha e Beatriz Teixeira, ao meu irmão Simão e à minha namorada Maria.

**o júri / the jury**

| | |
|---|---|
| presidente / president | Prof. Doutora Iouliia Skliarova<br>Professora Auxiliar, Universidade de Aveiro |
| vogais / examiners committee | Prof. Doutor Jorge Manuel Alves Lopes<br>Diretor de Tecnologia Empresarial, Brisa |
| | Prof. Doutor Joaquim José de Castro Ferreira<br>Professor Coordenador, Universidade de Aveiro |

**agradecimentos / acknowledgements**

Aos meus orientadores, Prof. Dr. Joaquim Ferreira e Dr. João Almeida, pela orientação e partilha de conhecimento. Ao Emanuel Vieira, Mohannad Jooriah e Seyed Hosseini pela constante disponibilidade para ajudar, e a todo o grupo *Embedded Systems* do Instituto de Telecomunicações pelo ambiente de trabalho, sentido de grupo e colaboração.

**Palavras Chave**

Cidades Inteligentes, Comunicações Celulares, Comunicações Veículares, Infraestrutura Rodoviária, Perceção Coletiva, Sistemas de Transporte Inteligentes.

**Resumo**

O constante crescimento da indústria automóvel e a necessidade do sobreuso do veículo pessoal amplificam problemas diretamente relacionados com a segurança rodoviária, tais como a degradação da qualidade das estradas, o aumento do volume de fluxo automóvel e o acréscimo de eventos metereológicos perigosos causados pelas alterações climáticas. Como forma de atenuar estes problemas emergentes, surgem os sistemas inteligentes de comunicação cooperativos (C-ITS) e de internet das coisas (IoT), que permitem ultrapassar limitações humanas e de sistemas sensoriais locais através da recolha e distribuição de dados em veículos conectados, algo fundamental para encontrar soluções que transformem o conceito de Smart City em realidade. A presente dissertação implementa um sistema de recolha de dados sensoriais intra- e inter-veículares, começando pela aquisição de dados relavantes presentes no barramento CAN, coletados através da porta OBD-II do veículo e de sensores externos. É feito uso de comunicações de curto alcance tais como Bluetooth-Low-Energy (BLE), Veículo-a-Veículo (V2V), e Veículo-a-Infrastrutura (V2I) em conjunto com comunicações celulares de longo alcance (LTE/5G). São fornecido endpoints de acesso aos dados através duma API e de um broker MQTT. Por fim métodos de logging são desenvolvidos para permitir depuração consciente destes sistemas e avalição de requisitos temporais. Os resultados dos testes experimentais efetuados revelam a utilidade forte que os dados adquiridos contém, por permitirem a realização de análises longitudinais detalhadas a estradas de perigo, assim como para fornecimento, em quase tempo-real, de condições adversas da estrada a condutores. Deste modo, o sistema de recolha de dados desenvolvido revela-se como ferramenta potencialmente valiosa para o fornecimento de informação útil tanto a autoridades competentes como à população comum, como meio de melhoria da segurança rodoviária.

**Keywords** Cellular Communications, Collective Perception, Intelligent Transport Systems, Road Infrastructure, Smart Cities, Vehicular Communications.

## Abstract

The increasing growth of the automobile industry and the need of overusing personal vehicles amplifies problems directly related to road safety, such as the degradation of the quality of the roads, the increase in volume of the automobile flow, and through the addition of dangerous weather events caused by climate change. To alleviate these emerging problems, intelligent cooperative communication systems (C-ITS) and Internet of Things (IoT) solutions emerge, allowing the overcome of human and local sensory systems limitations through the collection and distribution of relevant data in connected vehicles, which is fundamental in finding solutions that transform the concept of Smart Cities into reality. This dissertation implements an intra- and inter-vehicle sensory data collection system, starting with the acquisition of relevant data present on the CAN bus, collected through the vehicle's OBD-II port and external sensors. Use is made of short-range communications such as Bluetooth-Low-Energy (BLE), Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) in conjunction with long-range cellular communications (LTE/5G). Data access endpoints are provided through an API and a MQTT broker. At last, logging methods are developed to allow conscious debugging of these systems, as well as to evaluate timing restrictions. The results of the experimental tests carried out reveal the usefulness of the acquired data, which allows the realization of detailed longitudinal analyzes of dangerous roads, as well as notifying, in near real-time, adverse road conditions to drivers. Therefore, the data collection system developed reveals itself as a potentially valuable tool for providing useful information both to competent authorities and to the common population, as a method to improve road safety.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| **3G** | Third Generation Mobile System |
| **3GPP** | Third Generation Partnership Project |
| **5G** | Fifth Generation Mobile System |
| **AFH** | Adaptive Frequency Hopping |
| **AI** | Artificial Intelligence |
| **API** | Application Programming Interface |
| **APN** | Access Point Name |
| **ACK** | ACKnowledge |
| **ATT** | Atribute Protocol |
| **BER** | Bit Error Rate |
| **BLE** | Bluetooth Low Energy |
| **BSS** | Basic Service Set |
| **BSSID** | Basic Service Set Identification |
| **CiA** | CAN in Automation |
| **C-ITS-S** | Central ITS Station |
| **CA** | Cooperative Awareness |
| **CAM** | Cooperative Awareness Message |
| **CAN** | Controller Area Network |
| **CAN DBC** | CAN Database |
| **CCAM** | Cooperative, Connected and Autonomous Mobility |
| **CDMA** | Code Division Multiple Access |
| **C-ITS** | Cooperative Intelligent Transport Systems |
| **CP** | Collective Perception |
| **CPM** | Collective Perception Messages |
| **CP-OFDM** | Orthogonal Frequency Division Multiplexing |
| **CPS** | Collective Perception Service |
| **CRC** | Cyclic Redundancy Check |
| **CSV** | Comma Separated Values |
| **CSMA/CA** | Carrier Sense Multiple Access with Collision Avoidance |
| **DFT-s-OFDM** | Discrete Fourier Transform spread Orthogonal Frequency Division Multiplexing |
| **DLC** | Data Length Code |
| **DEN** | Decentralized Environmental Notification |
| **DENM** | Decentralized Environmental Notification Message |
| **DTC** | Diagnostic Trouble Code |
| **E-UTRA** | Evolved Universal Terrestrial Radio Access |
| **E-UTRAN** | Evolved Universal Terrestrial Radio Access Network |
| **ECU** | Electronic Control Unit |
| **EDCA** | Enhanced Distributed Coordination Access |
| **eMBB** | enhanced Mobile Broadband |
| **EOF** | End of Frame |
| **EOBD** | European On-Board Diagnostics |
| **EPA** | Environmental Protection Agency |
| **NBR** | Nominal Bit Rate |
| **NBT** | Nominal Bit Time |
| **eNB** | Evolved Node B |
| **EPC** | Evolved Packet Core |
| **EPS** | Evolved Packet System |
| **ETSI** | European Telecommunications Standards Institute |
| **EU** | European Union |
| **FD** | Flexible Data |
| **GPS** | Global Positioning System |
| **GAP** | Generic Access Profile |
| **GATT** | Generic Atribute Protocol |
| **GFSK** | Gaussian Frequency-Shift Keying |
| **gNB** | generation Node-B |
| **HCI** | Host Controller Interface |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **HSPA** | High Speed Packet Access |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **IoT** | Internet of Things |
| **IP** | Internet Protocol |
| **IT** | Instituto de Telecomunicações - Aveiro |
| **ITS** | Intelligent Transport Systems |
| **ITS-S** | ITS Station |
| **ITU** | International Telecommunication Union |
| **ISM** | Industrial, Scientific, and Medical |

| | |
|---|---|
| **ISO** | International Organization for Standardization |
| **ISRG** | Internet Security Research Group |
| **JSON** | JavaScript Object Notation |
| **L2CAP** | Logical Link Control Adaptation Protocol |
| **LIN** | Local Interconnect Network |
| **LLC** | Logical Link Control |
| **LTE** | Long Term Evolution |
| **LTE-A** | Long Term Evolution Advanced |
| **M2M** | Machine to machine |
| **MAC** | Medium Access Control |
| **MCC** | Mobile Country Code |
| **MNC** | Mobile Network Code |
| **MBIM** | Mobile Broadband Interface Model |
| **MIB** | Management Information Base |
| **MIMO** | Multiple Input Multiple Output |
| **MQTT** | Message Queuing Telemetry Transport |
| **ML** | Machine Learning |
| **MEC** | Multi-Access Edge Computing |
| **NR** | New Radio |
| **NFV** | Network Function Virtualization |
| **NTP** | Network Time Protocol |
| **OASIS** | Organization for the Advancement of Structured Information Standards |
| **OBU** | On-Board Unit |
| **OBD** | On-Board-Diagnostics |
| **OCB** | Outside the Context of a BSS |
| **OFDM** | Orthogonal Frequency Division Multiplexing |
| **OFDMA** | Orthogonal Frequency Division Multiple Access |
| **OSI** | Open Systems Interconnection |
| **PAPR** | Peak-to-Average Power Ratio |
| **PASMO** | Plataforma Aberta para o desenvolvimento e experimentação de Soluções para MObilidade |
| **PCAPNG** | PCAP New Generation |
| **PLMN** | Public Land Mobile Network |
| **PDU** | Packet Data Unit |
| **PDR** | Packet Delivery Ratio |
| **PID** | On-board diagnostics Parameter ID |
| **PHY** | Physical Layer |
| **PPCP** | Peripheral Preferred Connection Parameters |
| **QoS** | Quality of Service |
| **RAN** | Radio Access Network |
| **RSRP** | Reference Signal Received Power |
| **REST** | REpresentational State Transfer |
| **RHW** | Road Hazard Warning |
| **R-ITS-S** | Roadside ITS Station |
| **RMSE** | Root-Mean-Square Error |
| **RNC** | Radio Network Controller |
| **RWM** | Road Weather Model |
| **RWS** | Road Weather Station |
| **RSU** | Roadside Unit |
| **RSSI** | Received Signal Strength Indication |
| **RTR** | Remote Transmission Request |
| **SAE** | System Architecture Evolution |
| **SARWS** | Secure and Accurate Road Weather Services |
| **SCoT** | Smart Cloud of Things |
| **SIG** | Special Interest Group |
| **SIM** | Subscriber Identity Module |
| **SoC** | System-on-Chip |
| **SOF** | Start of Frame |
| **SQL** | Structured Query Language |
| **TCP** | Transmission Control Protocol |
| **TLS** | Transport Layer Security |
| **TDMA** | Time Division Multiple Access |
| **TMC** | Traffic Management Center |
| **TQ** | Time Quanta |
| **TRUST** | Transportation and Road monitoring system for UbiquitouS real-Time information services |
| **UE** | User Equipment |
| **UMTS** | Universal Mobile Telecommunications System |
| **URLLC** | Ultra Reliable and Low Latency Communications |
| **USB** | Universal Serial Bus |
| **UUID** | Universal Unique IDentifier |
| **V2I** | Vehicle-to-Infrastructure |
| **V2N** | Vehicle-to-Network |
| **V2R** | Vehicle-to-Vulnerable Road User |
| **V2V** | Vehicle-to-Vehicle |
| **V2X** | Vehicle-to-Everything |
| **V-ITS-S** | Vehicle ITS Station |
| **XML** | Extensible Markup Language |
| **WV-ITS-S** | Weather-Pollution V-ITS-S Station |

CHAPTER 1

# Introduction

*The first chapter introduces the theme of the present dissertation. It starts with the motivation behind this dissertation, analysing the theme from an environmental, economical, social, and technological point of view and how it affects the road environment. The chapter continues with the contextualization behind this work, the defined objectives and ends with a brief overview of the following chapters in this dissertation.*

## 1.1 Motivation

Road traffic in every European country has been in constant growth in the past few years, and this is expected to continue [1]. According to European manufacturers, the automotive sector employs 13.8 million workers, which represents 6.1% of the total European Union (EU) employment, a substantial part of the EU employability [2].

As a result of the growth in the number of vehicles circulating, concerns have arisen about road safety. Despite the efforts by manufacturers in offering more advanced and effective safety triggers in their vehicles, more than twenty-three thousand people died in 2018 [3] in the EU, and if we consider a world scenario, it's more than a million deaths [4], also resulting in millions of expenses issued while dealing with road accidents.

Traffic congestion is also a problem in a big scale environment. Drivers spend a large amount of hours in traffic queues everyday, and its continuously increasing in big cities all over Europe [5]. This reflects consequences on public health, quality of life and economic costs [6].

Additionally, air pollution is now a key concern of automotive manufactures. In the 60s, vehicle pollution became a cause for concern due to smog problems in Los Angeles, so the state of California ordered them to create emission control systems to meet Environmental Protection Agency (EPA) emission standards. Companies started to produce their systems but in 1988, the System Architecture Evolution (SAE) established a connector and a set of standard tests so that they all have the same technology. In 1996 the OBD-II was created and became mandatory in the United States, later in 2003 also being adopted in the EU, referred as European On-Board Diagnostics (EOBD). Thanks to this protocol and with the boost in the cars' electronic development, it is possible to access and report emission related and status parameters in real-time, through the OBD-II port present in the car [7].

Combining all negative aspects of the road environment, a need for deployment of new technologies not only in vehicles but also in infrastructures arises, in order to mitigate driver's limitations such as slow reaction times and line of sight, giving him and his car the tools to become more aware of their surroundings and more capable of preventing dangerous situations. Making use of close-range communication technologies and long-range communication technologies between vehicles and between vehicles and the infrastructure, a connected environment can be achieved, creating a large network of vehicular communications and giving life to the term Intelligent Transport Systems (ITS) [8]. Currently, use cases are being tested where not only vehicles with communication capabilities are integrated, but also legacy vehicles which can not communicate with external nodes, creating a solid base for these technologies to emerge on real traffic scenarios.

Data collected from vehicles own sensors through OBD-II can be used to increase the reliability and completeness of vehicular networks by providing useful data about the road environment.

## 1.2 Contextualization

This dissertation is part the Transportation and Road monitoring system for UbiquitouS real-Time information services (TRUST) project, that aims to develop an environmental and weather monitoring system capable of identifying risk conditions for driving in road infrastructures, equipped with ITS technologies. Communication between vehicles and infrastructures is intended to alert drivers in transit and warn about the danger of accidents in specific locations, reducing the number of incidents and associated fatalities.

The work contributes to the collection and dissemination of parameters accessible through the CAN bus. Nowadays, a single ECU present in modern vehicles produces enormous amounts of data, which can be used to collect information regarding each vehicle's condition and usage. Making wireless communications available, this data offers significant potential for the development of ITS applications, in themes such as traffic management, traffic information, road safety and weather-related.

By making data from vehicle integrated sensor systems accessible, an improved perception of the surrounding environment can be provided. As a result, the TRUST project aims to integrate the CAN bus with the vehicular communications platforms on fleet vehicles, enabling sensor-data collection that allow to infer weather-related events and improving station's perception of road environment.

Additionally, the logging segment of this dissertation was designed in the context of the *5G-MOBIX* project. This project aims to develop and test automated vehicle functionalities using 5G core technological innovations along multiple cross-border corridors and urban trial sites [9].

## 1.3 Objectives

For the work developed in this dissertation, the following objectives were set:

- Acquire knowledge about serial, close-range and long-range communications and vehicular networks. Study the European standards that define the functions of the various protocol layers in the communications between vehicles;
- Research articles that have contributed to the development in the area of vehicular sensor data collection. Studying state-of-the-art is essential to understand what already exists and needs to be improved, offering value to this work;
- Design and implement several processes and methods to extract vehicular data, with data quality and reliability in consideration;
- Design a process for transmitting the collected data to an OBU platform and disseminating it to the vehicle network with the cooperative messages defined by Cooperative Intelligent Transport Systems (C-ITS) protocols;
- Design a process for transmitting the data collected on the OBU platform to various cloud services;
- Design a reliable logging method to accurately test and verify imposed time restrictions.

## 1.4 Document Organization

The remainder structure of this dissertation is divided into five more chapters organized as follows:

- **Chapter 2: Fundamental Concepts -** Provides detailed information about the main concepts covered in this dissertation as well as topics important to it's understanding. This chapter also presents a review of the literature directed to the theme of this dissertation;
- **Chapter 3: System Architecture -** Exposes the contextualization of this dissertation and the architecture of the proposed system;
- **Chapter 4: Implementation -** Provides a detailed description of the developed work;
- **Chapter 5: Tests and Results -** Contains the presentation, analysis and discussion of the tests performed to the implementation work and the subsequent results obtained;
- **Chapter 6: Conclusion and Future Work -** Presents the conclusion taken from the result analysis and future work.

CHAPTER 2

# Fundamental Concepts and State-of-the-Art

*This chapter introduces related concepts of this dissertation and the State-of-the-Art of the area. Firstly, the protocols responsible for intra-vehicular networks are presented, from the physical layer to the higher layers. Next, localized (short-range) and networked (long-range) wireless communication technologies are explored, that can be used for inter-vehicular and cloud communications. Afterwards, the standards, messages and protocols related to the work developed in the European Union regarding the C-ITS environment. The chapter ends with a review of the current literature with work that contributed to this field.*

## 2.1 Controller Area Network

The CAN bus was designed by BOSCH as a multi-masting, message broadcast system that defines a maximum of 1 megabit per second as the baud rate [10].

Unlike other network protocols like Universal Serial Bus (USB) or Ethernet, CAN doesn't send big blocks of data in an end-to-end direction under the supervision of a bus master. Rather than that, many small messages are broadcasted to the entire network. It is a message-oriented protocol, where every message uses an unique identifier within the network and defines both the content and the message's priority. The protocol is very flexible, as it can be used both in systems that require low latency or that demand high bandwidth [11].

These are some characteristics of the protocol:

- Every node can act as a master or a slave in separate moments (Multi-Master);
- All nodes can receive simultaneously any message transmitted (Multi-Cast);
- Messages with smaller identifiers have priority in the bus over messages with bigger identifiers;
- Intrinsic error detection and automatic re-transmission of corrupted messages is directly supported;

CAN defined by ISO as a serial communication bus, that was originally developed for the automotive industry to replace the previous complex wiring in vehicular networks. The specification describes high immunity to electrical interference and the ability to self-diagnose and fix data errors. This led to CAN popularity in a variety of industries including automation, medical and manufacturing [12].

The bus enables ECU to communicate with all other ECUs without any complex dedicated wiring. An ECU can prepare and broadcast data via the CAN bus (which consists of two wires, CAN low and CAN high), and then, the broadcasted data can be accepted by all other ECUs on the network [13]. Figure 2.1 represents this organization.

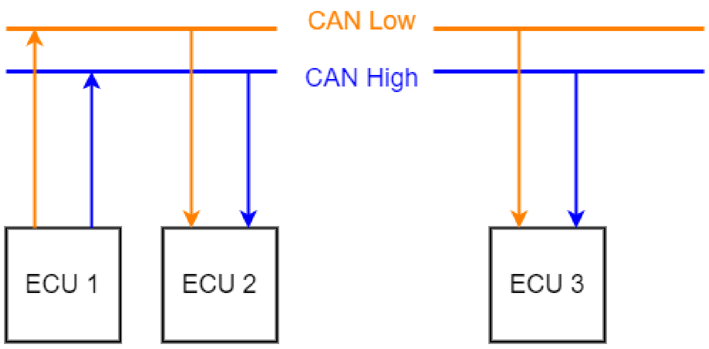


**Figure 2.1:** ECUs organization in the bus.

### 2.1.1 Protocol Stack

The CAN communications protocol, ISO 11898, details how data is exchanged between devices on the network, and conforms to the existing Open Systems Interconnection (OSI) (Open Systems Interconnection) model. The OSI reference model is represented by seven abstraction layers where data passes through during communications. It is a reliable model and is widely used in several communication protocols. The ISO 11898 architecture defines the lowest two layers of the OSI/ISO model as the physical and data link layers. A representation of the OSI model is represented in figure 2.2 along with correspondent CAN entities.

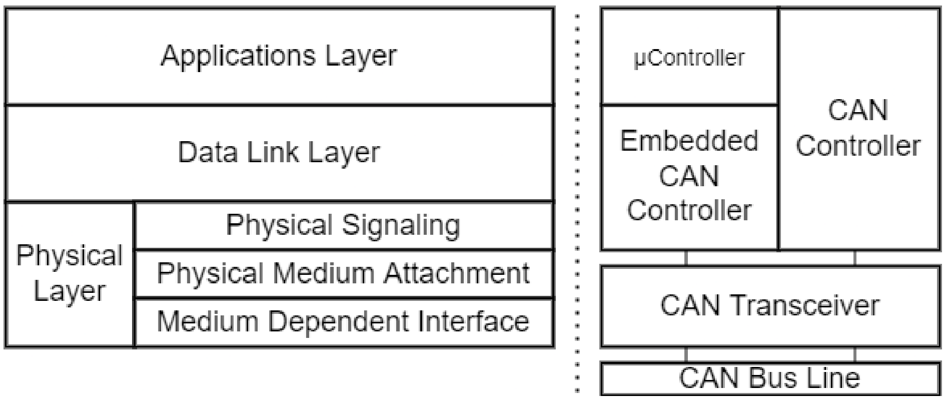


**Figure 2.2:** Layered ISO 11898 standard architecture (left) compared to CAN bus entities (right).

The details of the CAN protocol are defined in the standard ISO 11898, and is divided in three parts:

- ISO 11898-1: Contains the details of the Data-link layer and physical signal link [14];
- ISO 11898-2: Specifies the CAN physical layer for high speed CAN. The high-speed CAN allows a baud rate up to 1 Mbps. [15];
- ISO 11898-3: Specifies the CAN physical layer for low-speed CAN. It allows baud rate up to 125 kbps and is used when the speed of the communications is not a critical factor [16].

*Physical Layer*

The physical layer details the bit timing, synchronization, and physical signaling. Regarding bit timing, the protocol defines the Nominal Bit Rate (NBR) as the number of bits per second transmitted in ideal conditions, and can be described with the equation:

$$NBR = 1/t_{bit}$$

The Nominal Bit Time (NBT), or $t_{bit}$, is made up of non-overlapping segments (Figure 2.3), each made up of integer units called Time Quanta (TQ). The NBT is the sum of the following segments:

$$t_{bit} = t_{SyncSec} + t_{PropSeg} + t_{PS1} + t_{PS2}$$

To synchronize every node in the bus, the Synchronization Segment (SyncSeg), and thus the bit edges are expected to occur within it. The Propagation Segment (PropSeg) exists to compensate for physical latencies between nodes, which are defined as twice the sum of the signals propagation time on the bus, including latencies associated with the driver. Finally, phase segments (PS1 and PS2) are used to compensate for edge phase errors, and both can be adjusted to synchronize the receiver and the transmitter [17].

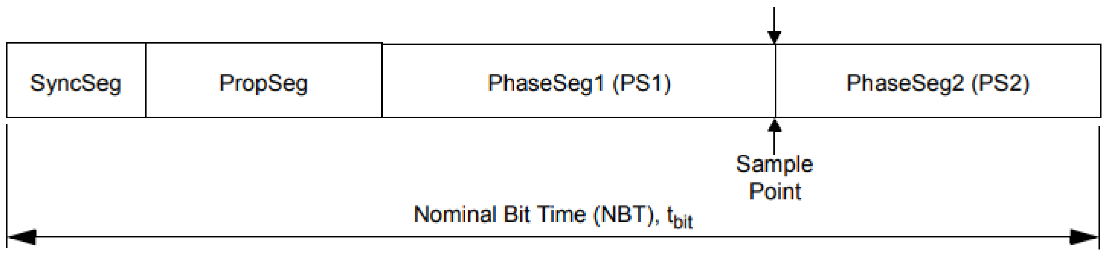


**Figure 2.3:** Bit time segments.

Using two resistors at the termination of each side of the bus is a critical factor for efficient communication. Only one resistor or none would increase the latency of the transition between a dominant and a recessive state. In some cases, the transition may not even happen, causing a bit-error. Figure 2.4 shows a recommended termination model for high-speed CAN, with the traditional 120 Ohm resistors on both ends of the bus [18].

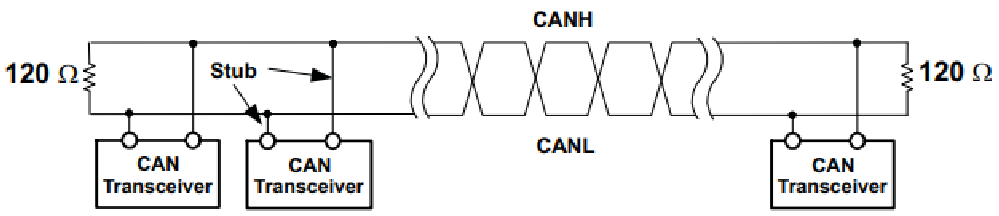


**Figure 2.4:** CAN standard termination.

The CAN physical layer protocol also specifies cabling types, electrical signal levels and the bus termination values to suppress reflections [19].

*Data Link Layer*

This layer specifies the format of the exchanged messages. A CAN frame begins with the Start of Frame (SOF) bit, that indicates the start of the message. This is followed by an Arbitration Field, which consists of an 11-bit identifier, following is a Remote Transmission Request (RTR) bit which is used to distinguish between a data frame and a remote frame [13].

A control bit that follows the RTR makes the distinction between the format of the frame, which can be either a standard frame or an extended frame. In the case of an extended frame, this bit is in the Arbitration Field, and more bits are added to the identifier, changing to 29 bits. In the case of a standard frame, the this bit is located in the Control Field. The Data Length Code (DLC) field is used to indicate the number of bytes in the Data Field, which in turn contains the message content, up to 8 bytes. As for the error detection, the messages contain a Cyclic Redundancy Check (CRC) Field with a CRC sum and delimiter. The ACKnowledge (ACK) also consists of a ACK slot and delimiter. Transmissions are sent with the ACK slot as a recessive bit and, if the message was received correctly, the receivers write it as a dominant bit. Finaly, the end of the frame is indicated by the End of Frame (EOF) bit [13]. Figure 2.5 represents the structure of a standard data/remote frames.

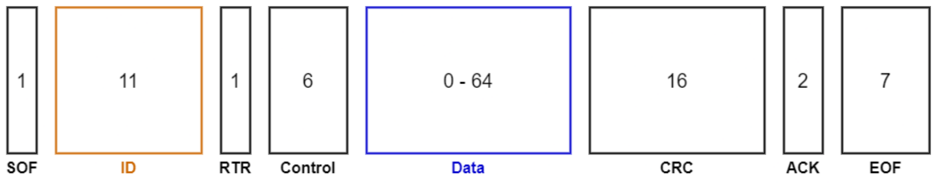


**Figure 2.5:** CAN standard frame format.

In total, four types of messages are specified: Data, Remote, Error, and Overload frames [20]. The Data Frame carries data up to 8 bytes between a transmitter and the receiver . In this type of frame, the RTR bit is dominant. The Remote Frame is used when a given node wants to request data from another node. This type of message is identified by the RTR bit, in which it is recessive. The Error Frame is used when a detection of an error in the bus happens. Finally, the Overload Frame uses two fields, the overload flag and overload delimiter and there are two conditions for an overload frame to be transmitted:

- A receiver requiring a specific delay for the next data or remote frame;
- The detection of a "dominant" bit during intermission.

Until the existence of the CAN 2.0 specification, CAN messages consisted of only 11-bit identifiers. With CAN 2.0 released came the possibility of using 29-bit identifiers, increasing the number of messages available. According to the specification, a message with an 11-bit identifier is denominated a Standard Frame while a message with 29-bit identifiers is an Extended Frame [20]. Figure 2.6 represents the structure of a extended message, referring to data/remote frames, respectively.

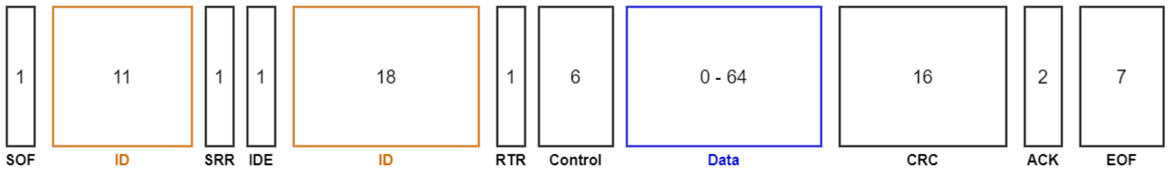


**Figure 2.6:** CAN extended frame format.

### 2.1.2 Security

CAN is a low-level protocol and intrinsically doesn't have any security features. There is no encryption, which makes these networks open to man-in-the-middle interception attacks. So, applications from nodes in the bus are expected to implement their own security mechanisms (e.g., authenticating incoming commands, addition of specific devices on the network).Not implementing proper security measures may result in various types of attacks if a way to inject messages on the bus is found. While solutions based on passwords exist for critical functions like modifying firmware or controlling the Anti-lock Braking System (ABS), these systems are not implemented universally and have a limited number of key pairs [21].

### 2.1.3 CAN FD and CAN XL

The vehicles constant modernization requires a greater capacity of CAN communication to follow this progress by the manufacturers. Therefore, CAN in Automation (CiA) has designed new protocols with the requirements of future in-vehicle networks.

CAN Flexible Data (Flexible Data (FD)) was released in 2015 and provides baud rates up to 5 megabits per second, with a payload of 64 bytes [14]. This growth in the amount of data reduces the overhead and improves the efficiency of the protocol [22]. Additionally, it presents an improvement in error detection with an improved CRC field [23]. This protocol operates simultaneously with classical CAN, by allowing each node in the bus to transmit messages of both types. When only one node is transmitting, the data rate can increase because the is no need to synchronize any nodes, however, when several nodes are transmitting, they have the flexibility to change the type of messages and move to the classic CAN [24].

CAN XL is a protocol still in development, and its specified as the third generation of the CAN data link layer protocol by the CiA members. CAN XL can transmit messages up to 10 megabits per second, with a payload of 2048 bytes. It is protected with a cascaded CRC, which features a Hamming Distance of 6, meaning that up to five randomly distributed bit-errors are detected. This protocol forces an upgrade on the physical layer, producing new transceivers that can operate with all the other transceivers that support CAN and CAN FD [25] [26].

## 2.2 Local Interconnect Network

The LIN protocol standard has been introduced to complement the CAN bus for non-critical system in vehicles such as air-conditioning and dashboard control, where transmission speed and reliability are less important.

It is a serial network protocol used for communication between components in vehicles, that uses a single wire and supports communications up to 19.2 kilobits per second, at a maximum length of 40 meters [27]. The need for a cheaper serial network came as the technologies implemented in the each car grew, and the CAN bus was too expensive to implement for every single component in a car. European car manufacturers started using different serial communication technologies, but this led to compatibility problems.

The LIN protocol has been serving increasingly important roles in providing low cost functionalities expansion in modern vehicles. As such, LIN bus has exploded in popularity in the last decade, and it is expected to exists more than 700 million nodes in cars by 2020 [28]. Figure 2.7 shows a comparison between protocols used in automotive networks [28].

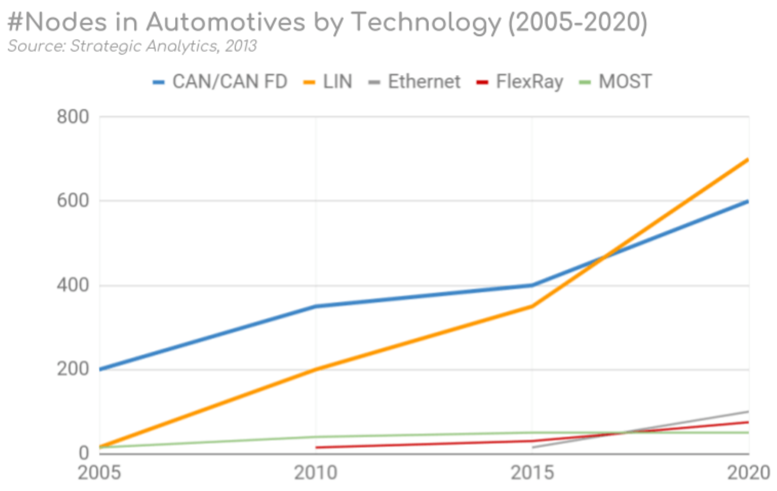


**Figure 2.7:** LIN comparison to other network protocols.

LIN bus acts as a supplement to CAN, offering worse performance and reliability, but substantially reducing costs. Its nodes are typically bundled in clusters, each with a master that interfaces with the backbone CAN bus. Figure 2.8 shows an example of it's implementation in junction with the CAN bus [29].

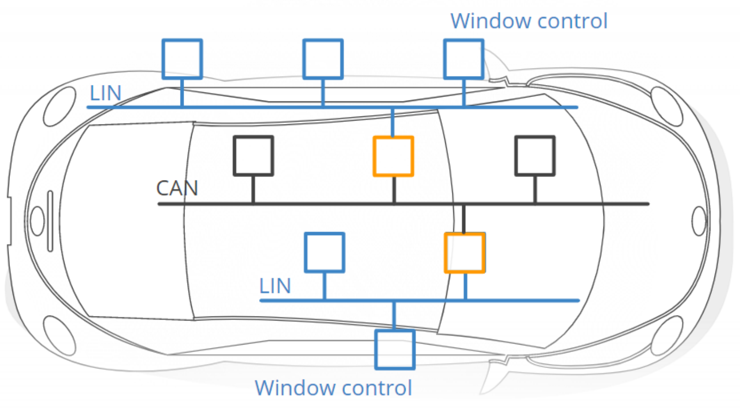


**Figure 2.8:** LIN interconnections with the CAN bus.

These are some examples of automotive use cases in which this protocol is used:

- Steering wheel: Cruise control, wiper, climate control, radio;
- Door: Side mirrors, windows, seat control, locks;
- Other: Window wipers, rain sensors, headlights, airflow.

## 2.3 On Board Diagnostics

The reporting and self-diagnostic abilities of a vehicle is refered by OBD. These systems give the vehicle owner or automotive technicians access to the status of the various vehicle parameters [30].

Nowadays, the amount of diagnostic information available through OBD is very different from its introduction in the eighties. Primitive versions of OBD would simply irradiate a Malfunction Indicator Light (MIL) when a problem was detected, but without providing any hints regarding the cause of the anomalies. Present-day OBD implementations use standardized communication ports to provide real time data collection as well as a standardized Diagnostic Trouble Code (DTC) list, which allows to rapidly identify specific problems within a vehicle.

OBD-I implemented the first standerdized interface, and its goal was to incentivize automobile manufacturers to design efficient emission control systems, forcing annual emissions testing for California, and denying registrations to vehicles that did not fulfill requirements in these tests [31]. However, as reporting specific parameters was not standardized between different manufacturers, OBD-I was not successful.

OBD-II is an expansion over this first iteration in capacities and more importantly, standardization. It specifies a type of diagnostic connector and its pinout, the available electrical signalling protocols, and the messaging format [32].

It also provides a candidate a On-board diagnostics Parameter ID (PID) list to monitor, along with how to encode and decode their data [33]. The presence of OBD-II allows devices to acquire and process parameters such as the engine speed and load, vehicle speed, fault codes, fuel usage, etc. Devices can then use this information to determine other variables like over revving, speeding, fuel consumption, etc.

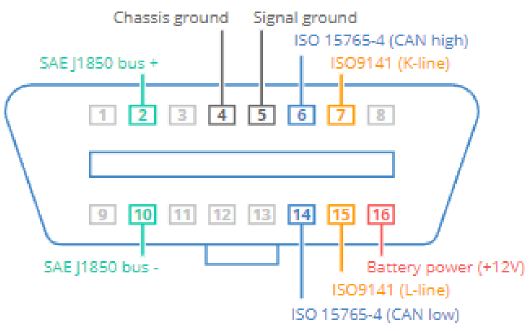


**Figure 2.9:** J1962 Connector.

**Figure 2.10:** J1962 Connector Location.

### 2.3.1 Connector

The system is accessed with a 16-pin connector, named J1962 connector [34]. This connector is used for OBD communications and uses pins 6 and 14 to allow access to the CAN bus, while pin 16 supplies battery power, making it possible to power devices while the ignition is off. All the other pins represented in figure 2.9 belong to other communication protocols that can be accessed through this port. This connector is usually located near the steering wheel or under the dashboard, behind covers, as represented in 2.10.

### 2.3.2 Message

Simplifying,OBD-II messages are made of identifiers and data content. The identifier corresponds to the 11-bit CAN ID field, and the data is is composed by a mode, PID and four data bytes [34]. In figure 2.11 it's fields are shown, and then an explanation for each of them is itemized.

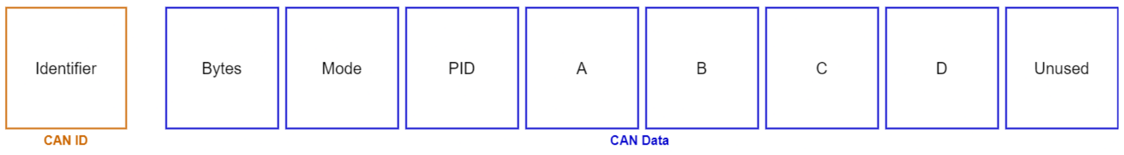


**Figure 2.11:** OBD-II message format.

- **Identifier** - Used to distinguish between requests and responses;
- **Bytes** - Length in number of bytes of the remaining data in the message;
- **Mode** - Used to distinguished between variosu modes specified in the standard [33] such as live or freeze frame date;
- **PID** - For each mode, a list of standard OBD-II PIDs exist;
- **A, B, C, D** - Data bytes, which need to be decoded according to the PID formula calculations.

### 2.3.3 Security

Keeping up with the increasing popularity of Internet of Things (IoT), various wireless OBD-II devices are developed, which can be simply plugged in cars, enabling remote control of many vehicle functionalities. However, since these devices are directly connected to vehicular networks, they also open over-the-air attack surfaces for vehicles.

Haohuang Wen et. al. [35], finds that out of 77 commercially available OBD-II dongles, the majority of them are vulnerable to attacks such as Data Leakages, Denial of Service through fuzzing, and Property Theft. Because of the lack of authentication on the CAN bus, any device connected through the J1962 connector can act as any other node in the CAN bus, being able to receive all other broadcasted messages or impersonating other ECU.

## 2.4 Bluetooth Low Energy

BLE was designed as a wireless personal area network technology. It acts independent of classic Bluetooth and implements no compatibility, but support for both often coexist in the same module. It was introduced in 2010 to reduce energy costs between data transfers, resulting in longer battery life for devices [36].

BLE is mostly used in fields where the transmission of data doesn't need to be fast and battery life is a priority, like Machine to machine (M2M) and IoT applications. It is also adopted by major brands such as Google and Apple, that provide BLE programming interfaces as part of their respective mobile development software development kits [37].

### 2.4.1 Protocol Stack

The protocol is divided in two parts. The Controller which consists of the Physical Layer and the Link Layer, typically implemented in a System-on-Chip (SoC) with an integrated radio, and the Host, that comprises Logical Link Control Adaptation Protocol (L2CAP), ATT, GATT, Generic Access Profile (GAP). The communication between these entities is made through the Host Controller Interface (HCI) [38].

Although BLE and classic Bluetooth implement the same architecture, a device that supports only BLE cannot exchange messages with a device that only implements classic Bluetooth. This type of device is called a single-mode device. When both implementations are present, it is called a dual-mode device [39].

Figure 2.12 shows the BLE protocol stack organization.

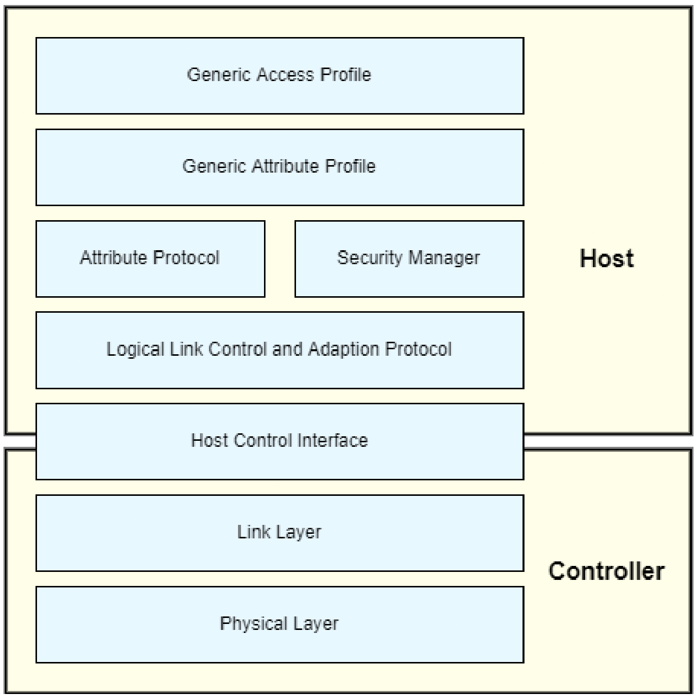


**Figure 2.12:** BLE protocol stack.

*Physical Layer*

This BLE layer describes the analog communications circuitry responsible for translation of digital symbols over the air. BLE operates in the Industrial, Scientific, and Medical (ISM) band at 2.4GHz and is divided into 40 radio frequency channels on 2 MHz spacing . There are two types within these channels: advertising channels (used to discover devices, establish connections, and broadcast transmissions) and data channels (used for communication between two devices) [40].

The Adaptive Frequency Hopping (AFH) technique is used on all channels to make the transmission signal more robust and reliable, adapting to interference that may arise, mainly from Wi-Fi. Channels also use Gaussian Frequency-Shift Keying (GFSK) modulation, where data pulses are filtered before being applied to alter the carrier frequency. The transmission rate of data is 1Mbps, with 1 bit per symbol[41].

For there to be a connection between two devices, it is necessary to share the same channel of the piconet, tuning in to the same frequency at the same time. In BLE there is no maximum number of devices that can connect to another. However, in Classic Bluetooth, that number was limited to seven devices [38].

*Link Layer*

This BLE layer controls the connections between devices, it is responsible for advertising, scanning, creating and maintaining connections.

When a device intends to broadcast data, it transmits the data in advertising packets through the advertising channels. Devices that are only receiving data through advertising channels are called scanners. When two devices establish a connection, one becomes the master and the other the slave, and the master may have more than one slave. The slave is in sleep mode and wakes up periodically to check if the master intends to send any data packets. The master determines this period and coordinates the average access using a Time Division Multiple Access (TDMA) [39].

The following diagram depicts two BLE hosts, initially in a Standby state. They enter a Discovery state whereby the device wishing to be discovered becomes the Advertiser and the host wishing to connect becomes a Scanner. The Advertiser sends advertising packets containing basic information about the host. All Scanners receive these packets [42].

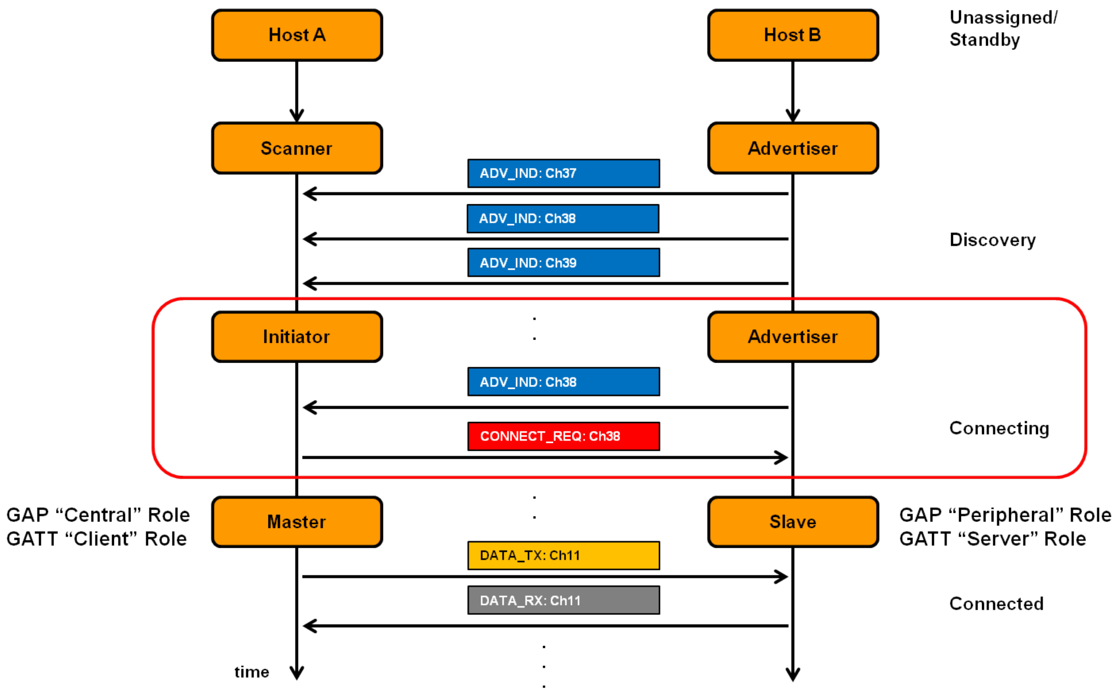


**Figure 2.13:** BLE host connection diagram.

At some point, the Scanner becomes an Initiator and decides to initiate a connection with a specific advertiser. This is known as the Connecting phase and is highlighted by the Initiator sending a connection request advertising packet to the Advertiser. Finally, the Advertiser accepts the connection request, thus becoming the Slave while the Initiator becomes the Master. This is known as the Connected phase [42].

Note that the Link Layer Master is also called as GAP Central and GATT Client, while the Link Layer Slave is also defined as GAP Peripheral and GATT Server.

The state machine of the different roles is described in figure 2.14.

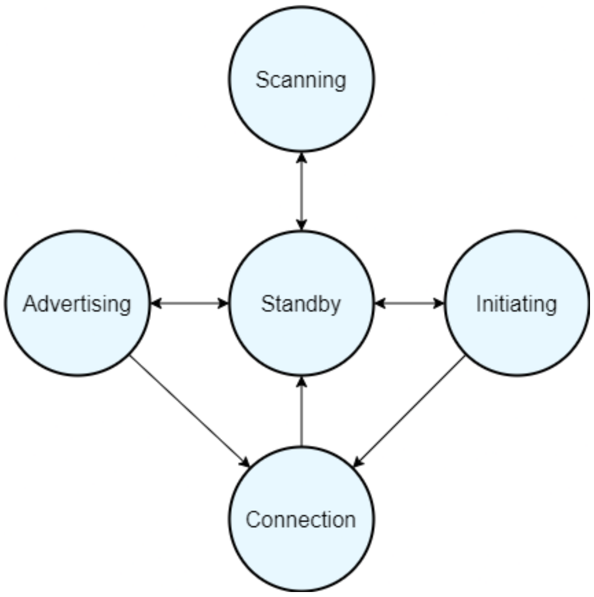


**Figure 2.14:** BLE link layer sate machine.

Only one packet format is defined in BLE Link Layer, used for both advertising channel packets and data channel packets, with a length that can vary between 80 bits and 376 bits. Figure 2.15 represents the package structure [43].

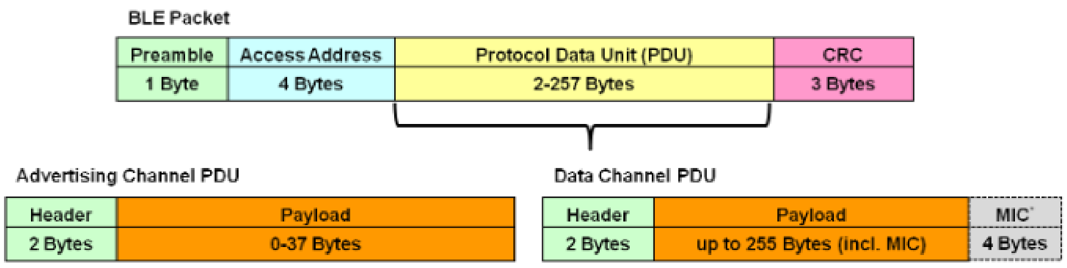


**Figure 2.15:** Data and Advertisement channels packets format and PDU field structure.

The PDU field is the only field with variable size, varying its structure through the physical channel through which it will pass.

Advertising channel PDUs serve two purposes, broadcasting data for applications that do not require a full connection, discovering slaves and connecting to them. Data channel PDUs are used so devices can send data to one another. This is achieved by exchanging data during regularly scheduled connection events. Due to protocol overhead of the higher layers, the maximum data payload is 246 bytes [43].

*Host Controller Interface*

The HCI allows and controls the communication between the host and the controller through a serial interface. Due to the controller's time requirements and the contact with the physical layer, it's separation from the host is important. Although the Host is responsible for more complex implementations, it is not demanding in time [44].

*Logical Link Control and Adaptation Protocol*

This protocol allows the transmission and reception of data packets from the upper layers. L2CAP is responsible for protocol multiplexing, segmentation. and reassembly operations, ensuring the quality of information in the upper layers [45].

*Attribute Protocol*

In the Attribute Protocol, two roles are specified: a server role and a client role. A server can expose a group of attributes to a client that is available using the ATT protocol [46].

The attributes are discrete values with the following properties:

- Attribute type, defined by a Universal Unique IDentifier (UUID);
- Attribute handle;
- Group of permissions for higher layers that utilize the attribute (read/write);
- Attribute value.

*Generic Access Profile*

GAP objective is to manage connections and advertisements, using rules described by in profiles. It is the GAP that controls functions such as visibility, connection establishment and security procedures.

Four roles are defined at this layer, Broadcaster, Observer, Peripheral and Central. These can be divided into two groups, those that allow connection establishments (central and peripheral), and those that do not allow (broadcaster and observer).

Devices operating in the Broadcaster role only send advertising events and, in the Observer role, only receive advertising events. Any device in the Peripheral role accepts a connection request and will have the Slave role in the Link Layer Connection state. A device in the Central role initiates a connection request and will have the Master role in the Link Layer Connection state. Unlike Peripherals, Centrals support multiple connections [47].

*Generic Attribute Profile*

GATT describes how BLE devices transfer data back and forth using nested objects called Profiles, Services and Characteristics, their relationship can be visualized in figure 2.16. ATT is used to store these objects in a clear lookup table. GATT arises as soon as a dedicated connection is established between two devices, being present during the GATT advertising process.

It is important to remember that GATT and its connections are exclusive, this means that when a BLE peripheral connects to a central device, it will stop advertising, and it will be impossible to other devices to connect until the prevailing connection is broken. It is essential to highlight that this connection is the only way to enable back and forth communication. The central device sends data to the peripheral and vice versa. If there is a need for data exchange between two peripherals, a mailbox system must be used, and consequently, all messages go through the central device. Furthermore, the communication can occur in both directions, contrary to the one-way broadcasting approach, which uses only advertising data and GAP.

The GATT profile can take on two roles: client and server. However, a device does not need to play a role exclusively, as the same device can be a server and a client simultaneousl. The GATT server stores the data that will be transported by the ATT, receives commands, requests and sends responses, notifications and indications to the GATT client. The GATT client sends commands and requests to the server, receiving the respective responses, notifications, and indications sent to it about the status of the server [48].

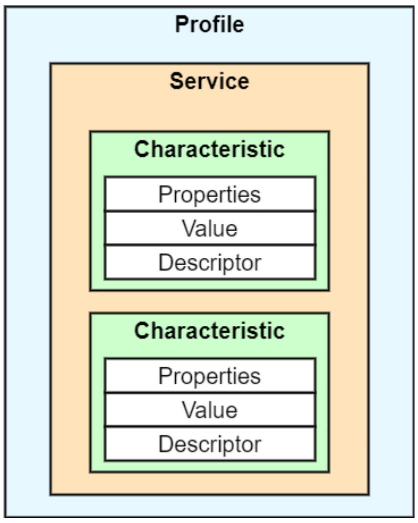


**Figure 2.16:** Hierarchy of GATT profiles.

### 2.4.2 Security

In this article [49], the authors present various techniques for listening to BLE conversations, showing how packets can be picked up and reassembled in connection streams. However, the Bluetooth Special Interest Group (SIG) responded to most threats and provided ways to mitigate them. It is concluded that building a secure BLE device can be achieved by making use of all the security mechanisms provided by the specification.

### 2.4.3 Performance

In order to understand this technology's limits, is is important to ascertain the performance that a BLE device can have [50] [51]. In [39], three researchers performed an overview and evaluation of BLE performance based on four factors: energy consumption, latency, piconet size, and throughput.

*Energy Consumption*

Figure 2.17 illustrates a slave's lifetime for one-way and round-trip approaches and different parameter configurations, based on a CC2540 SoC current measurements. The slave was expected to consume less energy when the *connSlaveLatency* is at its maximum. The maximum value obtained was 14.1 and 12.4 years for the one-way and round trip method, respectively, resulting from a *connInterval* of 86.25ms and *connSlaveLatency* of 370 (the master obtained readings every 32 seconds). The smallest value obtained had a *connInterval* of 7.5ms and a *connSlaveLatency* of 0, with a duration of 2.6 and 2.0 days for each method.

Note that the slave's lifetime, when the *connSlaveLatency* has the maximum value, does not differ with the increase in the *connSlaveLatency* due to the value of the *connInterval*, which cannot exceed 499 [39].

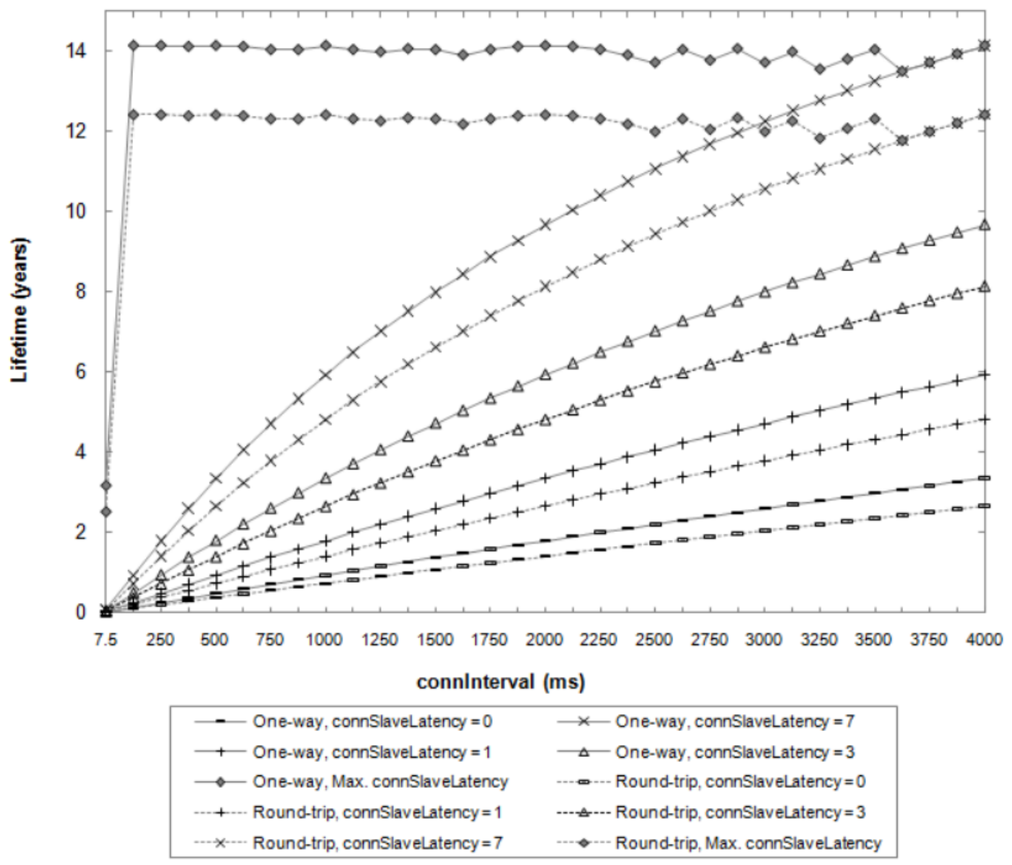


**Figure 2.17:** Theoretical lifetime of a slave for one-way and round-trip ATT message exchanges.

*Latency*

Figure 2.18 shows the average latency for one-way and round-trip message exchanges, for various *connInterval* and Bit Error Rate (BER) values. Latency was measured between the first message transmitted and the correct reception of the last message. The figure represents an average of more than ten million simulations. The average latency for smaller BER values for one-way and round-trip presents values were smaller than 1ms and 2ms, respectively, for different *connInterval* values. For higher BER values, the value of *connInterval* influences latency levels. The correct data transmissions are slower due to the tendency to have more than one connection for a single transmission to be successful [39].

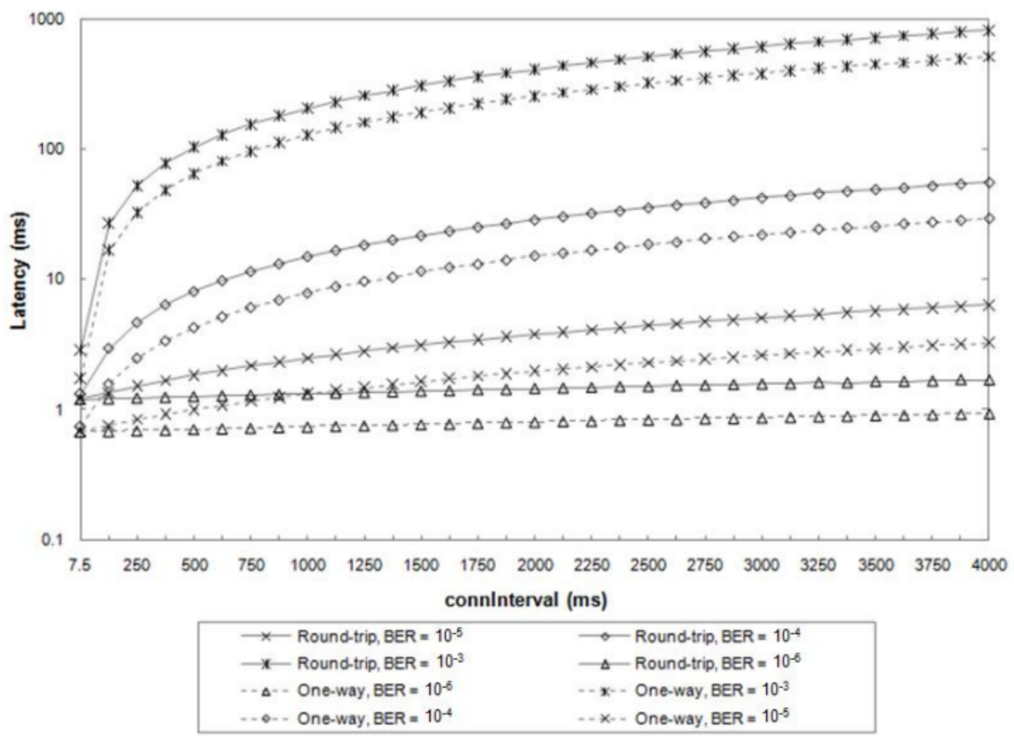


**Figure 2.18:** Average latency for one-way and round-trip message exchanges.

## 2.5 Cellular Networks

Cellular networks provide internet access through broad areas as well as throughput rates that satisfy generic applications requirements.

The success of 3G (also known as Universal Mobile Telecommunications System (UMTS)) and it's growth in mobile services resulted in big amounts of data exchanges, which triggered the necessity of newtechnology deployments. At first, High Speed Packet Access (HSPA) was introduced, an overall upgrade to the performance of the UMTS system [52]. Although HSPA was able to achieve higher throughput values and lower latencies, various improvements were under analysis with the objective to reach greater specifications for 3G systems, which led to the initiated the search for a long-term evolution of the radio access technology, that resulted in Release 7[1] Technical Reports Third Generation Partnership Project (3GPP) TR 25.912 [53], and 3GPP TR 25.913 [54] conveying the design framework for Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN), evolutions for the third generation Radio Access Network (RAN) [55].

In 2007, the project known as Evolved Packet System (EPS) had two work items being developed simultaneously, SAE, focusing in designing a new core network architecture, and LTE[2] covering the air interface, radio access network as well as the User Equipment (UE).

[1]Each release incorporates hundreds of individual Technical Specification and Technical Report documents, each of which may have been through many revisions.

[2]the name LTE was later adopted to refer to the whole project, and will be adopted in the rest of the document when referring to the EPS.

In December 2008, all the specifications for a fully fledged LTE system implementation were stable and ready to be deployed [56]. Besides the low latency and high data rates requirements, new features like improved coverage, high mobility support, spectrum efficiency and interoperability among other networks were introduced. Table 2.1 resumes some of the specifications between these releases.

| **Specification** | **UMTS (Rel.99-4)** | **HSPA+ (Rel.7-10)** | **LTE (Rel.8-9)** |
|---|---|---|---|
| Downlink Peak Rate (Mbps) | 0.384 | 42.2 | 100 |
| Uplink Peak Rate (Mbps) | 0.128 | 11 | 50 |
| User Plane Latency (ms) | 100 | 25 | 5 |
| Cell Spectrum efficiency (bps/Hz) | 0.53 | 1.28 | 1.5 |
| Carrier Bandwidths (Mhz) | 5 | 5 | 1.4,3,5,10,15,20 |
| Connect Setup time (ms) | 1000 | 100 | 100 |

**Table 2.1:** Performance specifications of different cellular technologies.

Making the transition to LTE, changes to the architecture were made both on the access network and core network, becoming less complex by introducing more advanced elements with increased capabilities [57] [58]. Figure 2.19 illustrates these changes.

The Access Network is mainly formed by base stations named Evolved Node B (eNB), which are enhanced versions of the UMTS Base Station Node B. Node B uses a centralized unit, Radio Network Controller (RNC), that manages the radio resources and the handovers among the centralized units and the stations, as well as the connection to the core network. The eNB in E-UTRAN, eliminates the centralized unit by supporting all the radio resource control, admission control and mobility management that were previously contained by the RNC. Each eNB is connected to the core network by means of an S1 interface, and all the packet and signal forwarding are made throughout the X2 interface, that inter-connects the base stations. In terms of access technology, UMTS used Code Division Multiple Access (CDMA), which performs well for low data rate communications such as voice, but is very unstable over high-speed transmissions, therefore, Orthogonal Frequency Division Multiple Access (OFDMA) is adopted in LTE, which in turn is more spectral efficient and brings high data rate support as well as solutions for other problems such multipath interference and narrowband interference.

The Core Network changed from the alternating circuit-switched/packet-switched approach, to a fully packet switched approach, optimizing the data delivery in LTE and all the costs associated with infrastructure. The architecture is split into the user plane and the control plane. The former is responsible for transportation of the user data traffic, while the latter handles management tasks such as connection or mobility. Another improvement is giving the ability for the user to maintain an Internet Protocol (IP) address whenever the device is switched on and released when switched off, something that did not happen in UMTS, where an IP address was only set on request and disposed when was no longer required.

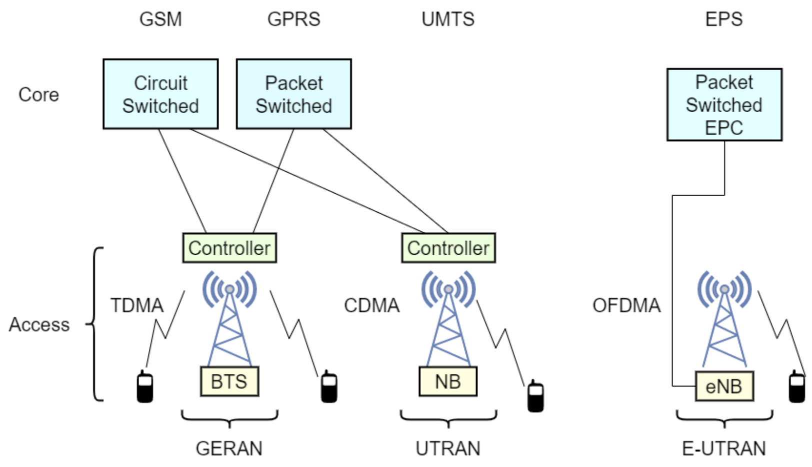


**Figure 2.19:** Network solutions from 3G technologies to LTE.

Release 10 was the introduction to Long Term Evolution Advanced (LTE-A), which brought up some additional features such as support to wider bandwidth through carrier aggregation, the use of advanced Multiple Input Multiple Output (MIMO) techniques and the introduction of relay nodes, improving the overall peak data rate, capacity, coverage and flexibility. In October 2010 LTE-A was accepted as a 4G technology, satisfying or exceeding all the requirements proposed by International Telecommunication Union (ITU) [59].

3GPP Release 15 brought a new 5G RAN, known as 5G New Radio (NR). The 5G requirements that led to focusing on areas such as enhanced Mobile Broadband (eMBB) or Ultra Reliable and Low Latency Communications (URLLC) were defined considering new services already implemented in the market and the necessities by different industry sectors. The first 5G NR specification was also part of this Release, that introduced the Phase 1 of the 5G system deployment where two architectures were distinguished: the non-standalone version that relies on some of the elements in the LTE system, and the standalone version that is a fully 5G system architecture. Both architectures are shown in Figure 2.20. In a non-standalone version only 4G services are supported but provided by the 5G NR, this was seen as a middle step and many current projects have been deployed in a non-standalone architecture. This version was highly motivated due to the market pressure to release 5G system parts as soon as possible [60].

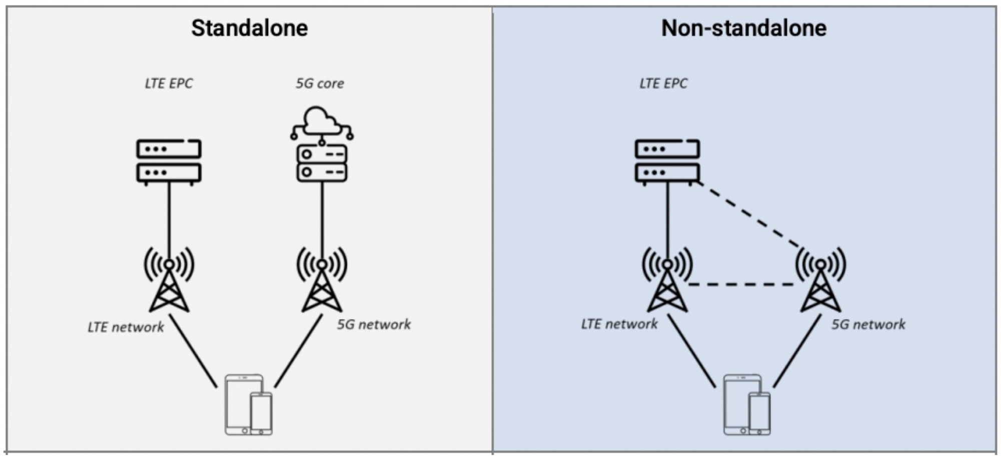


**Figure 2.20:** Comparison between standalone and non-standalone 5G.

Regarding the 5G standalone architecture, The Access Network is mainly formed by the generation Node-B (gNB), a base station formed by several units that manages and controls several operations in the user and control plane. The gNBs are connected among each other and to the core network. The connection to the UE uses Orthogonal Frequency Division Multiplexing (CP-OFDM) both in the downlink and uplink. However, an alternative can be used in the uplink plane referred as Discrete Fourier Transform spread Orthogonal Frequency Division Multiplexing (DFT-s-OFDM), that provides low Peak-to-Average Power Ratio (PAPR) and could improve the uplink coverage.

The Core Network is also split into the user plane and the control plane similar to the Evolved Packet Core (EPC). Although, the implementation of the 5G core elements is based on network functions that communicate with each other by sharing different services and resources. This creates a service-based approach architecture, that through virtualization enables modularity, reusability and introduces concepts such as network slicing and mobile edge computing.

5G system enhancements have already been defined in 3GPP Releases 16 and 17 with several areas of interest [61] [62]. 5G-Advanced is now moving into focus by Release 18, introducing further machine-learning based techniques at different levels of the wireless network [63].

## 2.6 MQTT

MQTT is a messaging protocol standardized by Organization for the Advancement of Structured Information Standards (OASIS), designed to implement and mechanism for M2M communications. It is classified as publish-and-subscribe, which means that instead of communicating with a server, client devices and applications publish and subscribe to topics handled by a proxy (broker). IP is typically used as its transport but other bi-directional transport protocols can also bed used [64].

Two entities are defined,MQTT broker client. The MQTT broker acts as a server, receiving all messages from clients and then routing the messages to the subscribed clients. The MQTT client is any other device that connects and subscribes to the MQTT broker. The broker organizes data in various topics, When a client wants to publish data, it sends a message to the specific topic in the broker, and the broker then distributes it to any clients subscribed to that topic [65]. The following message types are described by the protocol:

- **Connect** - Used by the MQTT client to establish a connection with the MQTT broker;
- **Disconnect** - Used by the MQTT client to close a connection with the MQTT broker.
- **Subscribe** - Used by the MQTT client to inform the intention of receiving messages published on a topic;
- **Publish** - Used by the MQTT client and the broker to exchange data between the nodes.

Figure 2.21 shows the exchanged messages between two clients and a broker during a connection establishment.

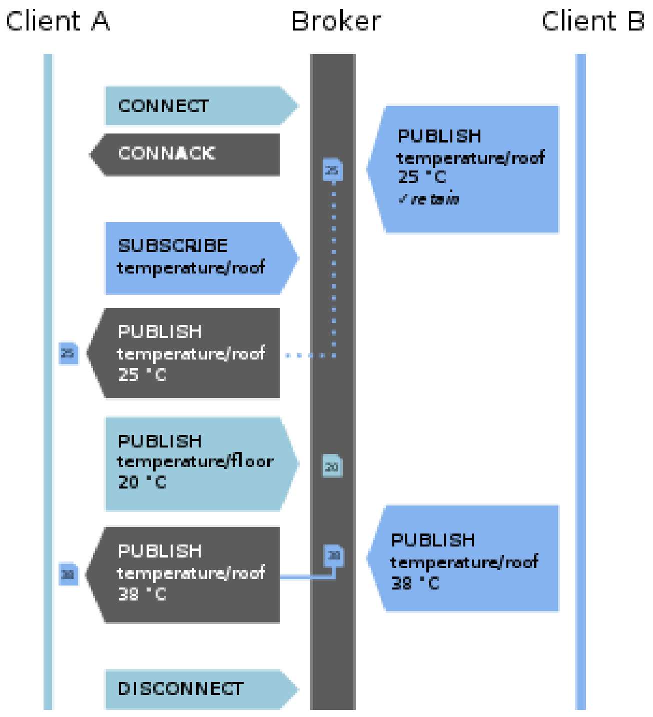


**Figure 2.21:** Example of an MQTT connection and message exchange.

While message persistence isn't directly supported, it is possible to retain messages. The broker stores the last retained message for the selected topic, and each client that subscribes to that topic receives the retained message immediately after they subscribe. It is only possible for the broker to retain one message per topic.

Regarding security, MQTT sends connection credentials in plain text format and does not include any measures for security or authentication [64]. This can be overcome by running Transport Layer Security (TLS) over MQTT to encrypt and protect the transferred data.

## 2.7 Cooperative Intelligent Transportation Systems

ITS is defined as a combination of state-of-the-art information and communication technologies, and its objective is improve the efficiency and safety of transportation networks, minimizing congestions and environmental impacts.

While ITS focus is providing intelligence in roadside infrastructure or in vehicles, C-ITS focuses on the Vehicle-to-Everything (V2X) communications between entities. These communications are implemented in dedicated devices able to provice real time ITS services to road users throughout Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I) Vehicle-to-Network (V2N), or Vehicle-to-Vulnerable Road User (V2R).

### 2.7.1 ITS Station

The European Telecommunications Standards Institute (ETSI) defines an ITS-S with a reference architecture which refers to various functionalities. Based on the OSI model, this reference architecture is expanded in order to also describe access technologies and protocols to handle the ITS services. It is is organized by four horizontal layers and two vertical layers [66] [67] [68]. Figure 2.22 represents this protocol stack.

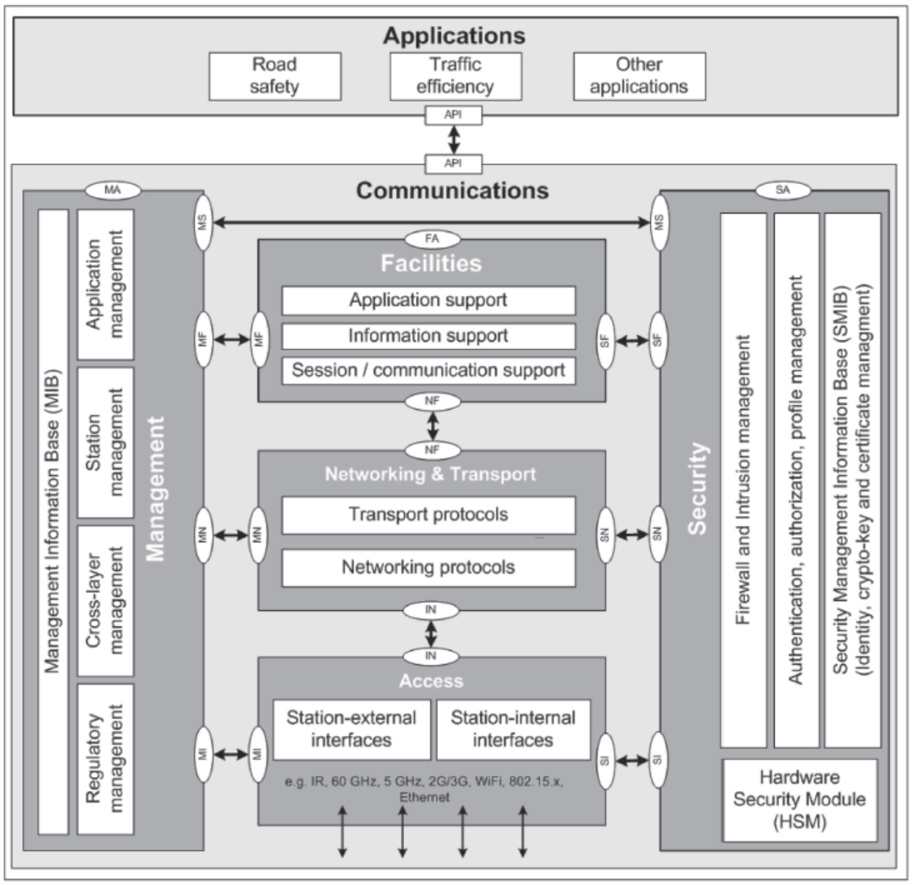

Applications
Road safety
Traffic efficiency
Other applications
API
API
Communications
MA
SA
MS
MS
FA
Facilities
Application support
Information support
Session / communication support
MF
MF
SF
SF
NF
NF
Networking & Transport
Transport protocols
Networking protocols
MN
MN
SN
SN
IN
IN
Access
Station-external interfaces
Station-internal interfaces
e.g. IR, 60 GHz, 5 GHz, 2G/3G, WiFi, 802.15.x, Ethernet
MI
MI
SI
SI
Management Information Base (MIB)
Application management
Station management
Cross-layer management
Regulatory management
Management
Security
Firewall and Intrusion management
Authentication, authorization, profile management
Security Management Information Base (SMIB) (Identity, crypto-key and certificate managment)
Hardware Security Module (HSM)

**Figure 2.22:** ITS-S reference architecture.

The Access Layer represents the layer 1 and 2 of the OSI model. It supports the various protocols, interfaces and types of media responsible for external and internal communications.

For external type communications some of the technologies could be non-ITS specific, such as LTE, or ITS exclusive, like ITS-G5.

The Network and Transport layer represents the layer 3 and 4 of the OSI model. It specifies the networking protocols, the addressing, data routing from source to destination, and geographical-based data dissemination. The transport protocols provide end-to-end reliable data exchange and additional features such as flow control and congestion avoidance.

The Facilities layer represents the layer 5, 6 and 7 of the OSI model and the main purpose is to assist the ITS applications that require shared data and offering access to common functionalities.

The Applications layer provides ITS service through ITS applications, relying on common functionalities and data provided by the facilities layer.

The Management layer is responsible for cross layer functionalities between different layers, transmission and synchronization management, security and privacy functions management as well as the configuration of the ITS-S itself. It also grants access to the Management Information Base (MIB), an entity which is responsible to define important data variables and data sets.

The Security layer provides different security and privacy functionalities that are responsible for performing security operations and storing credentials used by security protocols and mechanisms used in the different horizontal layers. Such functionalities include key management, hardware security, firewalls, encryption or decryption, and identity authentication.

### 2.7.2 Messages

The traffic load that mainly flows by means of the ITS network is based on different types of messages generated by the different ITS-Ss present in the roadside environment. Following, some of these message types will be explained.

*Cooperative Awareness Message*

CAMs are messages shared by an ITS-S that periodically provides information to neighboring nodes. Therefore, each ITS-S maintains awareness of the environment surrounding it through status information such as time, position, motion state, and attribute information, including data of dimensions, type of vehicles, and function in the road traffic. The construction, encoding, management and processing of CAMs are responsible by the Cooperative Awareness (CA) basic service. This service interfaces with the ITS applications layer and the facilities layer to forward the received CAM content for further processing [69]. Figure 2.23 represents the CAM general structure.

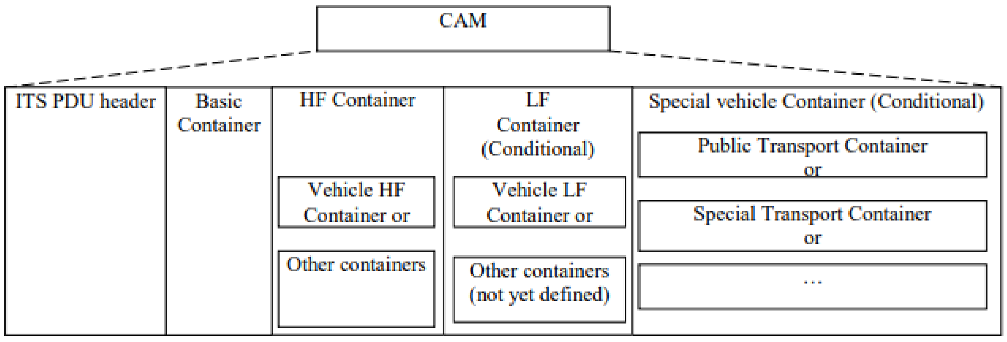

CAM
ITS PDU header
Basic Container
HF Container
Vehicle HF Container or
Other containers
LF Container (Conditional)
Vehicle LF Container or
Other containers (not yet defined)
Special vehicle Container (Conditional)
Public Transport Container or
Special Transport Container or
...

**Figure 2.23:** CAM general structure.

Its different containers can be described as:

- **ITS PDU Header** - Contains information about the protocol version, message type and the source ITS-S ID;
- **Basic Container** - Contains information regarding the source ITS-Sm such as type of ITS-S and latest location;
- **High Frequency container** - Contains highly dynamic data about the source ITS-S, such as the speed;
- **Low Frequency Container** - Contains mostly static data from the source ITS-S, such as vehicle dimensions and various sensor status;
- **Special Vehicle Container** - Contains specific information about the vehicle role of the source ITS-S.

According to the use cases, there are different transmission frequency requirements, that can vary between 1 and 10 Hz. Although having a big load in the ITS network, CAMs are transmitted only from a ITS-S in a single hop to the receiving ITS-Ss, and a received CAM shall not be forwarded to other ITS-Ss [69].

*Decentralized Environmental Notification Message*

DENMs are messages only transmitted when the ITS station detects an event on the road. They are used in Road Hazard Warning (RHW), and these applications are event-based and composed of multiple use cases that will define the final message structure. The Decentralized Environmental Notification (DEN) basic service interfaces with the ITS applications layer to receive the application request for DENM transmission or to provide the received DENM content directly to the application[70]. Figure 2.24 represents the DENM general structure.

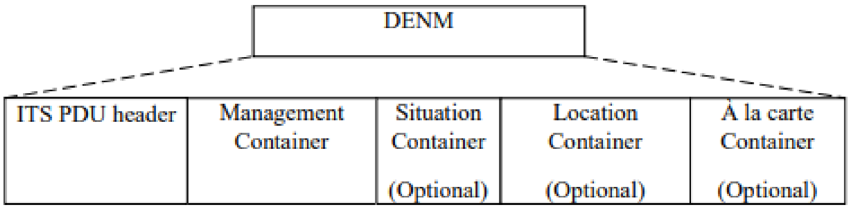

DENM
ITS PDU header
Management Container
Situation Container (Optional)
Location Container (Optional)
À la carte Container (Optional)

**Figure 2.24:** DENM general structure.

Its different containers can be described as:

- **ITS PDU Header** - Contains information about the protocol version, message type and the source ITS-S ID;
- **Management Container** - Contains information regarding the DENM protocol;
- **Situation Container** - Contains information that describes the event detected;
- **Location Container** - Contains information of the event location and relevance area;
- **À la Carte Container** - Contains additional information that could be usefull for transmission and is not present in other containers.

Some of its use cases include detection of low visibility conditions, heavy rain, traffic jam and collision detection. The time interval regarding DENM transmission repetition is not defined, although the ITS application responsible for the DENM generation event should provide a repetition interval parameter and a repetition duration parameter. The DENM repetition should be applied to the most updated DENM regarding specific identifier, and if no repetition parameters are provided, the DENM repetition will not be executed. Another important concept is the relevance area, that is set by the ITS application of the originating ITS-S, with the objective of defining the area that the event could have a direct or indirect impact. Contrarily to the CAM dissemination, DENMs could be forwarded by different stations in order to get to as many ITS-S as possible within the area [70].

*Collective Perception Message*

CPMs are messages that rather than disseminate it's station current state, shares information regarding several entities that are detected by the ITS-S local sensorial system. The Collective Perception (CP) aims to exchange the locally perceived objects in the network, reducing the uncertainty in the road environment for other ITS-Ss by contributing with information. Together with CAMs, this concept enhances and enables safety applications based on received information about objects located outside the range of the ITS-S's perception sensors. The Collective Perception Service (CPS) is an entity located at the facilities layer responsible to generate, receive and process CPMss, which contains information concerning the detected objects, as well as status information regarding the originating ITS-S and its sensory capabilities [71]. Figure 2.25 represents the CPM general structure.

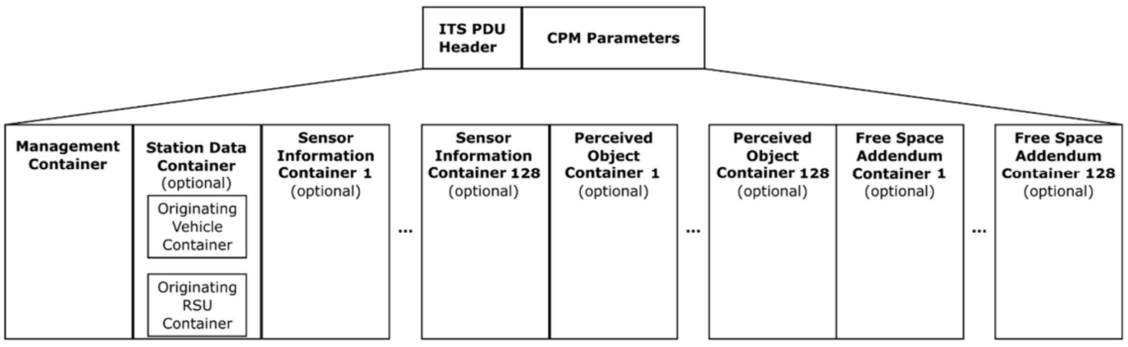

ITS PDU Header
CPM Parameters
Management Container
Station Data Container (optional)
Originating Vehicle Container
Originating RSU Container
Sensor Information Container 1 (optional)
...
Sensor Information Container 128 (optional)
Perceived Object Container 1 (optional)
...
Perceived Object Container 128 (optional)
Free Space Addendum Container 1 (optional)
...
Free Space Addendum Container 128 (optional)

**Figure 2.25:** CPM general structure.

Its different containers can be described as:

- **ITS PDU Header** - Contains information about the protocol version, message type and the source ITS-S ID;
- **Management Container** - Contains information about the CPM protocol;
- **Station Data Container** - Contains information that describes the event detected;
- **Sensor Information Container** - Contains information of the event location and relevance area;
- **Perceived Object Container** - Contains information about every object that has been perceived by the source ITS-S;
- **Free Space Addendum Container** - Used to describe changes to a computed free space description.

The frequency for CPMs should be set between 1Hz and 10Hz for both vehicle and roadside ITS-Ss. Even if no objects are detected, these messages should be generated anyway to report that detection is enabled. In a situation where CPMs are dropped because of, for example, channel congestion the objects present in the dropped CPM could be included in the next CPM generation event [71].

### 2.7.3 ETSI ITS-G5

ITS-G5 is one of the wireless technologies specified for the ITS-S architecture, and is based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11p standard. As stated in the ITS-S section, the access layer hosts the two lowest layers of the OSI protocol stack known as the data link layer and physical layer, where the first is subdivided into the Logical Link Control (LLC), which provides the capability of distinguish between different network layer protocols and the Medium Access Control (MAC) that is responsible for scheduling transmissions to prevent interference among ITS-Ss [72]. Figure 2.26 represents the ITS-G5 access layer architecture.

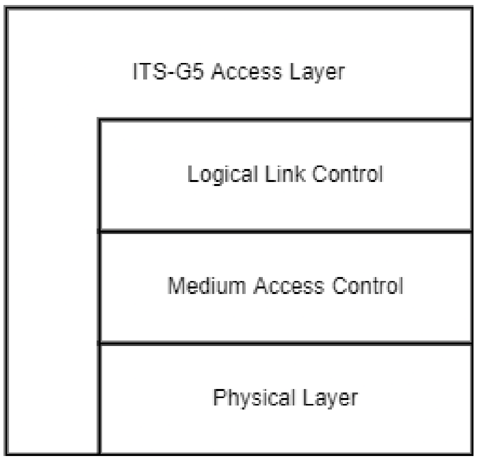


**Figure 2.26:** ITS-G5 access layer architecture.

The smallest block of an IEEE 802.11 network is the Basic Service Set (BSS), which consists of a group of stations that can communicate with each other independently or through infrastructure. Although Independent BSSs could be called of ad hoc, they still carry a magnitude of complexity and overhead that cannot meet the vehicular communications requirements. Every time a stations wants to join a BSS, the association, synchronization and authentication procedures are executed, resulting in a time intensive process where transactions between the nodes in the road environment cannot be completely accomplished if the brief time window of connection is spent performing these procedures [73]. Communications Outside the Context of a BSS (OCB) are achieved by setting the Basic Service Set Identification (BSSID) to a wildcard in every frame transmitted in an 802.11p communication, which means that the vehicular environment needs to contain predetermined channels, and so channel scanning is no longer necessary in this scenario [72].

For IEEE 802.11p, changes are mad to the Physical Layer (PHY) and MAC layers, changing features already implemented in previous versions [74] [75]. In the PHY, the IEEE 802.11p deploys the half-clocked mode of Orthogonal Frequency Division Multiplexing (OFDM) to achieve 10MHz frequency channel bandwidths, this way it is possible to address problems such as interference due to multi-path propagation and the Doppler shift effect [76]. In the MAC layer, the medium access algorithm deployed is named Enhanced Distributed Coordination Access (EDCA) which is based on the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) algorithm principles but adding Quality of Service (QoS). This means that priority data traffic will have a higher probability of being transmitted by shortening the nodes listening period of the medium [72].

## 2.8 State-of-the-Art

Different topics are studied in order to have a clear vision of the current technological developments in the area, starting with the extraction of data from the CAN bus, the usage of the vehicular sensor data and finally the importance of cellular networks for C-ITS.

### *CAN Data Extraction*

Correlating responses from the OBD-II diagnostic system with the proprietary signals of each vehicle is a straightforward approach for analyzing the raw CAN traffic [77]. The SAE J1969 standard defines a list of PID that must be the same in all vehicle models, allowing access to different types of parameters without knowing the vehicle's CAN specification. However, this does not allow identification of parameters beyond the PID list. Therefore, different methods are needed in order to locate different functions in the bus.

Daniel Frassinelli et. al. [78] presents *AutoCAN*. This tool extracts unknown CAN data from vehicles and tries to discover parameters by establishing relationships based on physics laws. It specifies four signal categories:

- **Continuous** - Signals like speed or temperature and are generally generated by sensors, or are derived from them;
- **Pseudo-random** - Values such as checksums, which are generally contained in frames carrying safety sensitive information;

- **Enumeration** - Signals like gear, break pressed, door open and are generally used to encode a certain state of the ca;.
- **Cyclic** - Signals that iterate continuously like clocks or counters.

After the classification through various steps, the algorithm identifies the signals using the correlation between them. Unlike other algorithms that correlate the segmented signals with PID signals, *AutoCAN* applies physical and mathematical properties relating all signals simultaneously, for example, integrating vehicle's speed and the odometer data. The correlation is classified using Pearson's coefficient.

*AutoCAN* was tested in four different vehicles, resulting in a percentage of correct segmentation between 70% and 89% of the known signals and 6% to 11% of average bit-error, based on a ground-truth with several documents, OBD analysis and manual reverse engineering.

Alessio Buscemi et. al. [79] introduces *CANMatch*, an automatic CAN reverse engineering algorithm that eliminates the need of manual effort by re-utilizing similar CAN frames in different vehicle models. *CANMatch* is divided into the three steps:

- **Frame Matching** - Extracts CAN messages, along with a timestamp. *CANMatch* matches identifiers found in a trace file with the same identifiers present in previously decoded signals. If a match occurs, the frame is decoded according to the ground truth dataset;
- **Tokenization** - After the matching, portions of the trace may still not be decoded. This step is performed on this unknown areas, resulting in a set of tokens, represented by their borders and endianness;
- **Redundancy Resolution** - The algorithm looks for correlations between the decoded signals from the first phase and the unknown tokens. If a match occurs, the sensor function of the decoded signal is attributed to the token.

The application of *CANMatch* resulted in a mean recall of 83.3% and mean precision of 97.7% by the first phase. However, *CANMatch* makes intensive use of ground truth CAN specification files or CAN Database (CAN DBC) files, which if not acquired from third parties, still implies intensive manual reverse engineering processes.

Ricardo Correia et. al. [80] proposes a solution based on a device connected to the OBD-II port of the vehicle, providing methods to decode CAN data in less than 30 seconds of manual work, without the need for CAN DBC files with millions of entries. Implementation is separate for binary and non-binary parameters.

For binary parameters, such as headlight status, windshield wiper state, or pedal status, the following phases were defined:

- **Reference** - In this phase, the user needs to not perform any action inside the vehicle for a while. The algorithm searches for bits that did not show any transition, saving the reference for the next phase;
- **Monitoring** - Here, the user will have an order to make a specific number of transitions in the parameter he wants to identify (e. g., switching on and off the minimum headlights). The algorithm then filters the bits with a bit-flip count twice the number of triggers

and deletes those that present different results. This filtered parameters are the only ones that the algorithm will be aware of in the next phase;
- **Validation** - This phase serves as confirmation to ensure that identification is made correctly.

The identification of non-binary parameters includes two processes, one based on parameters that can be decoded when the car is stopped (e. g., headlight status speed, steering wheel angle). and the other for parameters that require the car to be moving (e. g., throttle position, wheels speed).

For the former, the following phases were defined:

- **Reference** - In this phase, the user needs to not perform any action inside the vehicle for a while. Here, the algorithm is not scanning the messages bit-by-bit, but byte-by-byte, so the number of candidates reduces. Any transition of a bit in a determined byte excludes the candidate;
- **Monitoring** - When this phase begins with the driver being subjected to several instructions to execute inside the car. The type of instructions depends on the type of parameter to be identified. However, the processing and analysis of the data are identical, as the controller detects the messages via the OBD-II controller, the driver carries out the instructions, and the algorithm records the increments and decrements of the respective bytes that are being processed. These counters permit are the used to calculate derivative of the signals and conclude about if their behaviour are similar to the driver's instructions.

Regarding the other functions that require the car to be moving, a solution based on correlation to other OBD-II parameters was implemented, without the need for the driver to execute specific instructions. This was achieved by calculating the Root-Mean-Square Error (RMSE) of some defined features of a specific OBD-II response and all the other signals. The lower the RMSE value, the lower the distance between them, and so, higher is the probability of a given candidate to be chosen.

*Vehicular Sensor Data*

Concerning the use of vehicular sensor data, the success of C-ITS largely depends on the platform used to access, collect, and process accurate data from the environment [81].

Toon Bogaerts et. al. [82], proposes a model to enhance the current Road Weather Model (RWM) by making use of vehicular sensor data, enabling real-time road weather warnings for local weather phenomena and dangerous road conditions.

Vehicles are equipped with a CAN reader and external sensors, and the data collected by the fleet is then sent to a cloud back-end using a data distribution framework. Although all the CAN data was collected, within this study only data from external sensors was processed for investigation of the relationship between the measurements and the road conditions, due to the fact that CAN data is not always available and can be erroneous due to the reverse engineering processes. Table 2.2 overviews the external sensors used.

| In Car | Sensor Box |
|---|---|
| GPS Module | Gyroscope |
| Camera | Accelerometer |
| Thermal Imaging Sensor | Temperature Sensor |
|  | Humidity Sensor |

**Table 2.2:** Overview of external sensors placed on test vehicles used in [82].

Every 3 seconds, when the GPS module generated a new position, the sensor measurements are collected at this timestamp as well as all CAN messages within this time interval and were sent to a data distribution platform. With data from around 2000 hours of driving data collected from 15 vehicles, the authors findings indicated the potential value of vehicle sensors in relation to the road weather.

With these sensor measurements, trends with the temperature values were noticeable that occurred when rain started or stopped. The results showed a median RMSE close to 4 °C between the temperature forecasts performed at a Road Weather Station (RWS) and the ones performed at close observed road segments.

Toon Bogaerts et. al. [83], using the same equipment as in [82], demonstrated two examples of how Artificial Intelligence (AI) and Machine Learning (ML) can be leveraged to process data from a fleet of vehicles and obtain relevant weather information with an improved data quality compared to the raw measurements, using convolutions neural networks to extract visibility and precipitation data from raw camera images. With this, a new RWM was presented, to enrich the modeling of road weather with the cars data at a high spatio-temporal resolution.

The results showed a median RMSE below 2º C between the temperature forecasts performed at a RWS and the ones performed at close observed road segments.

*Cellular Networks and C-ITS*

Gorka Velez et. al. [84], surveys the 5G features and standards that would be required by a data platform that aims to capture data from vehicles to make it available to other parties in the cloud.

Considering the huge amount of data that a single connected car produces, the authors do not consider feasible to deploy a centralized Cloud-based solution that stores all the captured data. A more sustainable and realistic approach would involve a platform for live-data delivery. In this case, stakeholders would be subscribed to certain data flows, and decide what to do with the incoming data, either processing, storing or discarding.

The following 5G features that would enable such a platform were identified:

- **Network Function Virtualization (NFV)** - The purpose of NFVs is to create a network architecture that uses technologies widely adopted in virtualization. This improves load-balance, scalability, and the ability to relocate functions across different hardware resources. By continually updating virtualized functions, operators can keep things running on the latest software without interruption;

- **Network slicing** - 5G networks need to support a diverse list of verticals, with an exhaustive number of use cases each. These use cases demand very different requirements in terms of network performance. A logical network, named as a 5G network slice, is composed of several elements of the network, and each of these elements may be used in a dedicated way or be shared with other network slices. Each 5G network slice is isolated in a way that the use of the other slices will not affect the QoS provided by another slice. This is pivotal to achieve the performances needed for most of the relevant use cases of different Cooperative, Connected and Autonomous Mobility (CCAM) applications;
- **Multi-Access Edge Computing (MEC)** - MEC is a new paradigm that provides computing, storage, and general networking resources within the edge of the network, this is relevant for all the use cases that need scalability and low latency;
- **Cybersecurity** - This is important for every possible use case. [85] defines different sets of security measures organised by different domains, like network access, user domain, application domain, etc.

The authors conclude that 5G is a key enabling technology of CCAM and C-ITS systems, that includes several features that can contribute to create a scalable, flexible, reliable and secure data platform.

The work developed during the dissertation, described in detail in the following sections, makes use of some of this methods and the knowledge shared by the authors to build a reliable mechanism capable of extracting vehicular sensor data, disseminating it to other vehicles/infrastructure, and to the cloud.

CHAPTER 3

# System Architecture

*This chapter presents the system's global architecture where this work is inserted, followed by the detailed architecture of the proposed solution, describing the functionality of the different elements involved.*

## 3.1 Global Architecture

As mentioned in the first chapter, the work developed in this dissertation is part of the TRUST project, which belongs the SARWS project.

SARWS is a project that longs to expand local data collection mechanisms from traditional road weather data sources to completely new ones, exploiting elements from the C-ITS environment, proposing the gathering and sharing of meteorological data using units such as vehicles, road-weather sensors and smartphones. Figure 3.1 and table 3.1 details the SARWS architecture.

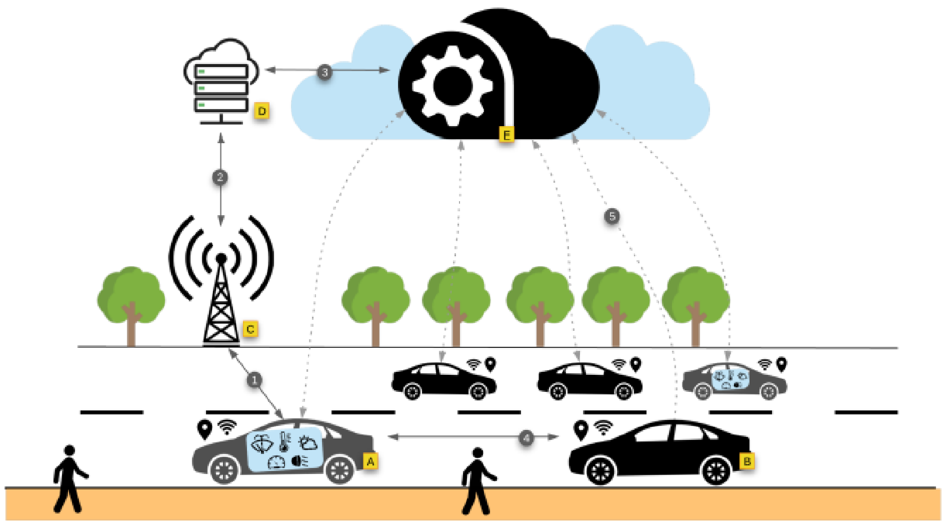


**Figure 3.1:** Scheme of the global SARWS architecture.

| **Item** | **Description** |
|---|---|
| A | Weather-Pollution V-ITS-S Station (WV-ITS-S) |
| B | Vehicle ITS Station (V-ITS-S) |
| C | Roadside ITS Station (R-ITS-S) |
| D | Central ITS Station (C-ITS-S), Traffic Management Center (TMC) |
| E | Broker, Cloud |
| 1 | V2I Communication |
| 2 | C-ITS-S to R-ITS-S |
| 3 | Broker to C-ITS-S |
| 4 | V2V Communication |
| 5 | V-ITS-S to Cloud |

**Table 3.1:** SARWS architecture components description.

This architecture is divided by four functionality layers:

- **Acquisition Layer** - Composed by all the sensors that notify about weather/pollution and traffic information;
- **Communication Layer** - Uses all communication types (1 to 5), as shown in 3.1, assuring the connection between ITS-S and providing data to both the C-ITS-S and the broker in the cloud (D, E);
- **Management Layer** - Responsible for handling, storing, and analysing the data obtained from the installed ITS-S. This management process is restricted to the C-ITS-S with limited interaction with the cloud broker;
- **Data Diffusion Layer** - This is where the information collection and distribution with the end-users and brokers occurs.

With SARWS, all the vehicles can benefit from the proposed RWM, even if they are not equipped with a OBU or Internet access, because SARWS notifies drivers about any relevant information via smartphones. Furthermore, this project can evaluate the traffic pollution by tracking the air quality from the vehicles sensors and reporting these to the public authorities.

The main goal of the project is to provide weather services in real-time that ensure safety, efficiency, and energy sustainable mobility that cannot be seen in traditional approaches, and that is where a fully fledged ITS environment plays a part, contributing to human comfort and safety, providing substantial results in an efficient way.

## 3.2 Proposed Solution

As mentioned, this work aims to implement a reliable mechanism capable of extracting vehicular sensor data, and forward it to other vehicles, infrastructure or to the cloud. To accomplish this, two sub-systems are designed.

*Intra-Vehicular*

This subsystem is responsible for requesting, extracting and decoding relevant sensor data from the CAN bus of the vehicle, and sending it to the local OBU. Figure 3.2 illustrates architecture scheme.

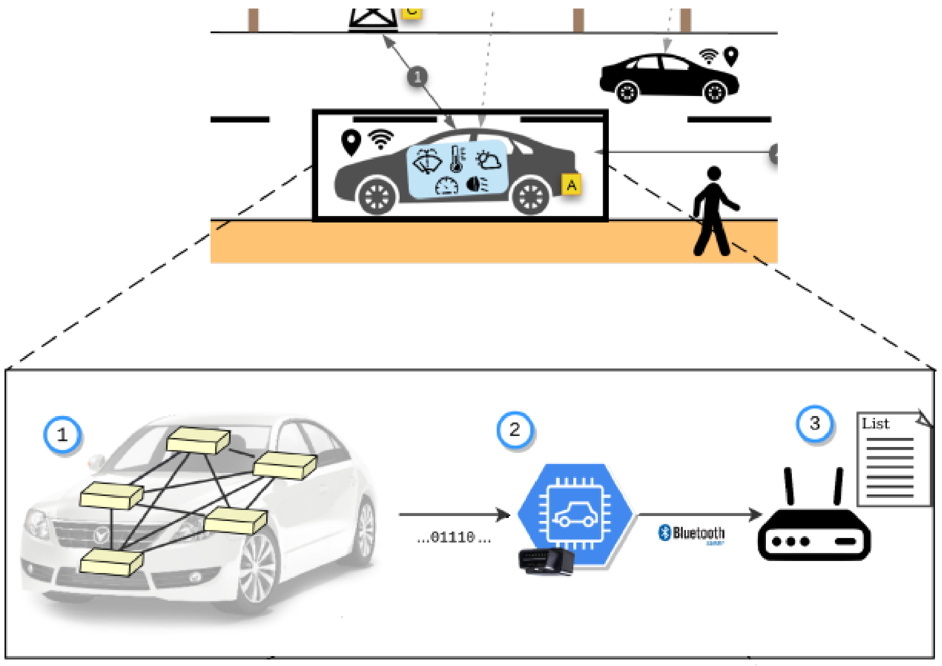


**Figure 3.2:** Proposed intra-vehicular architecture.

Detailing this architecture, the following entities are specified:

**Vehicle:** Represented by 1 in figure 3.2, this element consists of a CAN network responsible for controlling the various parts of the vehicle through ECUs. The OBD-II port acts as an input/output port, which allows monitoring the data being communicated between the units, that will be used to request and collect data;

**CAN Transceiver and Controller:** Represented by 2 in figure 3.2. This element consists of a CAN controller and a algorithm capable of identifying and decoding the raw CAN messages. A CAN transceiver needs to exist to enable the communication between the bus nodes and the controller. A *Carloop* with a *Redbear Duo* combo was chosen for this role because it implements both functions, while also having a BLE module for short-range wireless communication. Figure 3.3 shows a picture of the kit;

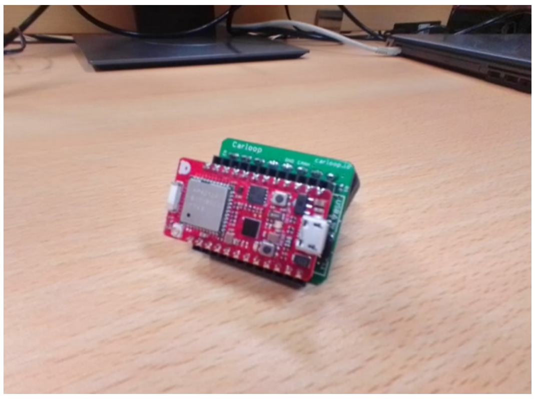


**Figure 3.3:** CAN transceiver and controller.

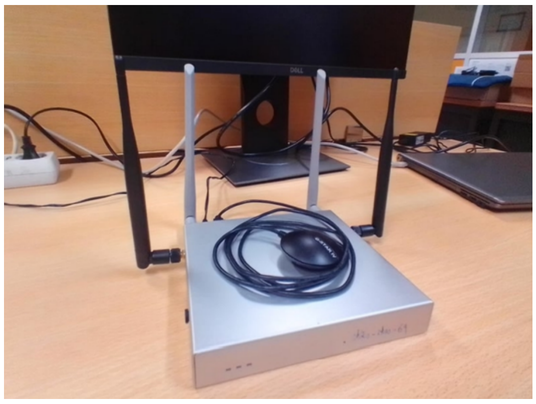


**Figure 3.4:** OBU with GPS receiver.

**OBU:** Represented by 3 in figure 3.2, this element is built on top of a system board from *PC Engines* with several features and a set of extensions, including a GPS receiver, a WiFi module, an LTE or module, and a pair of appropriate antennas for each module. A Dual-Mode Bluetooth 4.0 USB adapter is also used to allow communication with the CAN controller. Figure 3.4 shows a picture an example platform.

After the local OBU has possession of the vehicular sensor data, it is ready to start disseminating it to other connected cars via ITS-G5 (implementing V2I and V2V communications), and to the cloud via LTE or 5G.

*Cloud*

This subsystem defines the architecture of the services available in the Cloud, their relationships, the organization of the shared messages and how stakeholders should interpret the data. Figure 3.5 illustrates the architecture scheme.

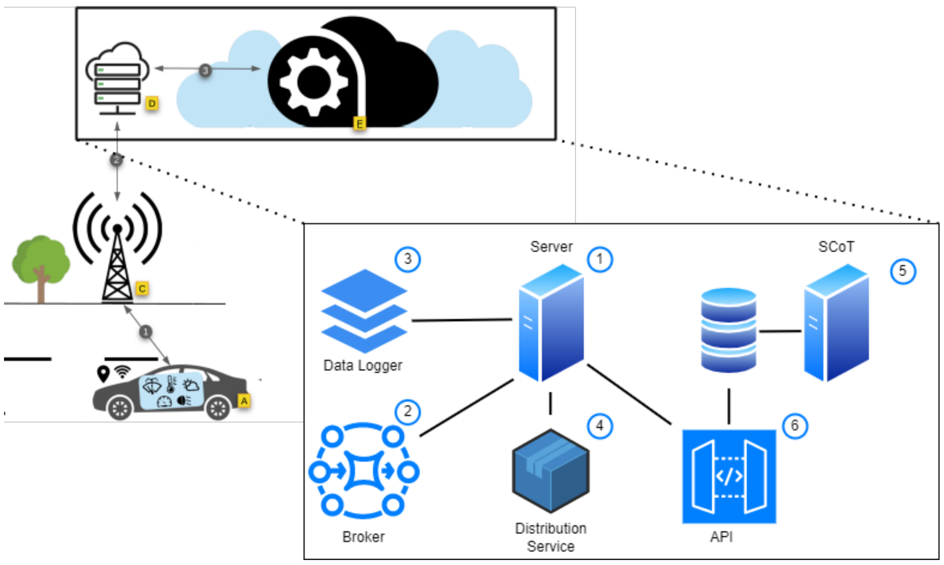


**Figure 3.5:** Proposed architecture for the cloud services.

Detailing this architecture, the following entities are described:

**Server:** Represented by 1 in figure 3.5, this is defined by one or more Linux machines that accommodate other services like brokers, loggers, databases, public key certificates for secure connections, etc. In order to take advantage of 5G, this should be deployed at the edge of the access network, to fully take advantage of the MEC concept;

**MQTT Broker:** Represented by 2 in figure 3.5, the broker provides an endpoint for other stakeholders such as vehicles to publish and subscribe to different topics in order to share live data to other vehicles in motion.

Figure 3.6 shows an example of the structure of a topic:

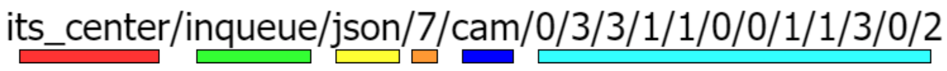


**Figure 3.6:** Defined MQTT topic structure.

- **Queue type** Represented by green, this field indicates the message destination agent. This can take the value of *inqueue* when the message is received, *outqueue* for messages that should be published to other stations, or *inter_mecs* when publishes should be made to a different 5G MEC platform;
- **Message format** Represented by yellow, this field indicates the message encoding type (e.g.: Extensible Markup Language (XML), JavaScript Object Notation (JSON));
- **Station ID** Represented by orange, this field indicates the ITS-S identification number.
- **Message type** Represented by blue, indicates the ITS messages types (e.g.: CAM, CPM);
- **Location** Represented by cyan, indicates the geographical location of the sender ITS-S, following a quad-tree map system representation.

This location representation follows a tree data structure in which children and parent nodes represent regions of the world map. The map is recursively subdivided into four quadrants, allowing different precision/zoom levels depending on the number of times the map has been divided. This approach fits well for topic creation on a MQTT broker because quadrants can be enumerated which allows a certain region to be specified by a sequence of numbers;

**Logger:** Represented by 3 in figure 3.5, this service is implemented in order to assert the strict CCAM timing requirements, as well as for facilitating debugging;

**Message Distribution Service:** Represented by 4 in figure 3.5, this service function is to subscribe to the local MQTT broker input queue and publish relevant messages to other output queues or to other brokers;

**Smart Cloud of Things (SCoT):** Represented by 5 in figure 3.5, SCoT is a local platform developed with IoT in mind, that provides a Hypertext Transfer Protocol Secure (HTTPS) endpoint for storing sensor data in a database [86];

**API:** Represented by 6 in figure 3.5, this offers an endpoint for users to access historical vehicular sensor data, providing an easy way to get relevant road statistics.

Figure 3.7 illustrates how the different cloud services would be distributed in a 5G network. The MQTT broker, the message distribution service and the logger should deployed at the edge of the network (on a MEC platform), because they need to be able to provide results as soon as possible, being subjected to strict time restrictions. The API and the SCoT platform don't need to fulfill any strict timing requirements because they are only used to provide historical data, so, there's no need to deploy them at the edge of the network.

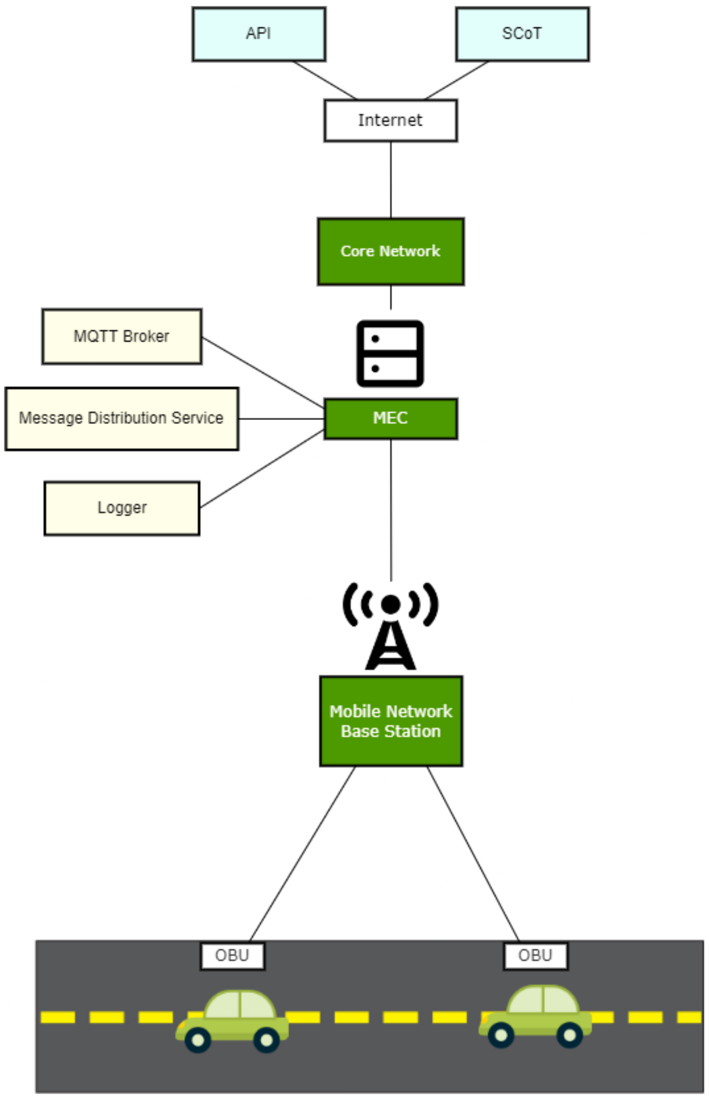


**Figure 3.7:** Defined cloud services architecture on a 5G network.

CHAPTER 4

# Implementation

*This chapter details the hardware specifications and the software development of the implemented solution, starting with the vehicular sensor data identification and extraction, followed by the configuration and implementation of the communication mechanisms, the organization of the different services installed in the OBU, and lastly the framework details of the cloud services.*

## 4.1 Vehicular Sensor Data

The hardware used in order to be able to identify and extract this data is composed by a *Carloop* acting as a CAN transceiver, with a integrated *RedBear Duo* board functioning as a controller. Their roles are:

- **CAN Transceiver** - To provide an interface between the physical layer and a CAN controller. Table 4.1 details its specifications;
- **CAN Controller** - To run applications that request, process and make responses to the bus via the CAN transceiver. Table 4.2 details its specifications.

| Model | Carloop |
|---|---|
| Dimensions | 10.2 x 7.6 x 2.5 cm |
| Weight | 23 g |
| CAN Mode | CAN2.0A/B |
| CAN Speed | Up to 1Mbit/s |

**Table 4.1:** CAN transceiver specifications.

| | |
|---|---|
| Model | RedBear Duo |
| Dimensions | 2.1 x 3.9 cm |
| Weight | 17 g |
| Microcontroller Unit | STMicroelectronics STM32F205 |
| Input/Output Unit | 18 General Purpose Input/Output (GPIO) |
| Clock Speed | 120 MHz |
| Memory | 128KB SRAM and 1MB Flash |
| WiFi + BLE Module | Broadcom BCM43438 |
| Temperature Range | 0 $\sim$ 70º C (Operating) / -20 $\sim$ 70º C (Storing) |

**Table 4.2:** CAN controller specifications.

### 4.1.1 Identification

Different manufacturers develop distinct ways of representing content in CAN messages (e.g., data may be in signed or unsigned, little-endian or big-endian, conversion to integers may be have different scales and offsets), making it difficult to interpret the data. Furthermore, CAN DBC files are not easily accessible, and the ones that can be found on online forums such as *opengarages* are often incomplete and not available for all car models, making it impossible to have a ground truth set of CAN IDs. As a consequence, manual reserve engineering methods must be applied in order to identify relevant data in the bus.

The implementation of the reverse engineering methods in this dissertation follows the work done by Ricardo Correia et. al. [80], explained in section 2.8, using the developed algorithms for detection of binary and non-binary parameters. However, since this algorithms only allow for online identification, they are limited by the capabilities of the *Redbear Duo*, as mentioned by the author.

With this in mind, an offline algorithm was developed that uses CAN traces previously acquired, and calculates the Pearson's correlation coefficient between different CAN messages with data requested by OBD. Given a pair of variables $(x, y)$, the correlation $\rho$ is given by:

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{4.1}$$

Different methods exist for calculating correlations between different data sources, but the one presented takes factors like different offsets and scales into account, being a suitable method for correlating different CAN data.

Nevertheless, because different CAN messages do not have the same sampling frequency, data needs to be interpolated in order to achieve the same number of samples in both variables. Given two points $(x_1, y_1)$ and $(x_2, y_2)$, a interpolated value $(x, y)$ can be obtained by the linear interpolation method:

$$y = y_1 + (x - x_1)\frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{4.2}$$

For acquiring a trace, a trip must be made while always making different OBD-II requests through the CAN controller, sending every CAN message that is captured in the bus to a application that is listening and logging all the messages. After acquiring the trace, the following steps are carried out to identify parameters:

- **Pre-Processing** - In this step, every byte from different CAN IDs is grouped in a data structure. This results in various arrays containing the evolution of different bytes throughout the trace. Every pair $(ID, Byte_{index})$ gets assigned one of these arrays;
- **Processing** - In this step, the algorithm will take a reference array from an input, and calculate it's correlation to every other $(ID, Byte_{index})$ pair;
- **Validation** - Finally, the algorithm outputs the pair $(ID, Byte_{index})$ that most resembles the input given, with it's correlation coefficient.

Figure 4.1 illustrates this process.

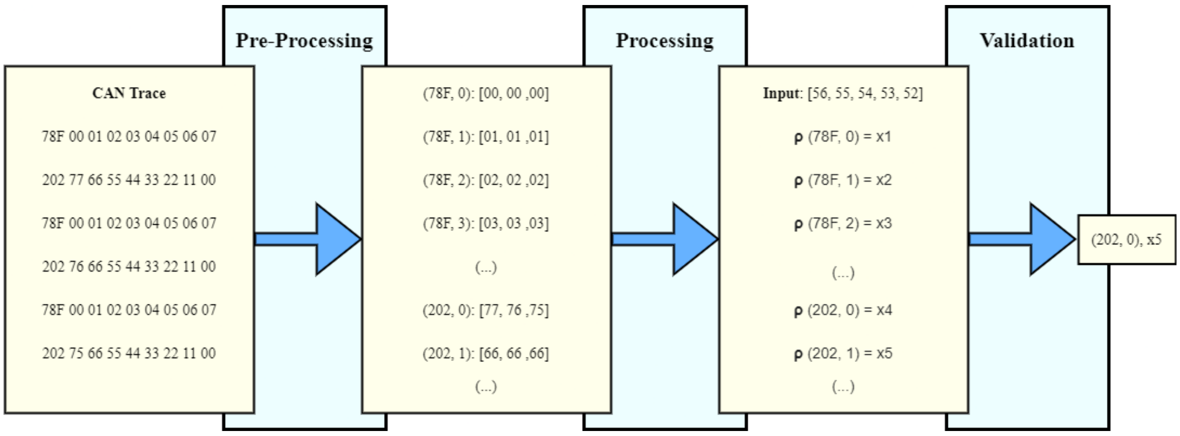


**Figure 4.1:** Representation of the proposed algorithm to identify CAN parameters.

### 4.1.2 Extraction

After identifying the CAN IDs that are relevant, a process is designed that listens to the bus for this messages while also making requests according to the OBD-II standard. A list of the OBD-II PIDs, as well as the relevant CAN IDs should be pre-configured, and if no configuration is made at this level, only requests to the OBD-II are done. The following algorithm represents the process in pseudo-code, where *pidList* is the list of OBD PIDs to request, *canIdList* the list of relevant IDs, and *loopFunction* the main event loop function.

```
$pidIndex \leftarrow 0$
$totalNumberOfPids \leftarrow$ GETLENGTH($pidList$)
$loopFunction \leftarrow sendRequest$

function SENDREQUEST()
    $pidIndex \leftarrow (pidIndex + 1) \% totalNumberOfPids$          ▷ Iterate through PIDs
    $message.id \leftarrow 0x7df$                        ▷ 0x7DF is the ID used for OBD-II requests
    $message.data[0] \leftarrow 0x02$                         ▷ Number of remaining data bytes
    $message.data[1] \leftarrow 0x01$                  ▷ 0x01 is the mode specified for current data
    $message.data[2] \leftarrow pidList[pidIndex]$                          ▷ Set PID to request
    BUSSENDMESSAGE($message$)                     ▷ Sends request message to the CAN bus
    $loopFunction \leftarrow waitForResponse$
end function

function WAITFORRESPONSE()
    if $elapsedTime > 500$ then                              ▷ Check if request timed out
        $loopFunction \leftarrow sendRequest$
    end if
    while BUSRECEIVEMESSAGE($message$) do         ▷ While the bus buffer has messages
        if $message.id == 0x7e8$ then         ▷ 0x7e8 is the ID used for OBD-II responses
            SAVEMESSAGE($message$)                      ▷ Saves message to local register
            $loopFunction \leftarrow sendRequest$
        else if $message.id$ in $canIdList$ then
            SAVEMESSAGE($message$)
        end if
    end while
end function
```

Because OBD-II requests can be lost in the bus or responses could be missed, a timeout mechanism is applied. If 500 milliseconds go by while waiting for a response, another request is made instead.

After some seconds of having the process running, all the data for the pre-configured parameters should be saved in local registers, and the *RedBear Duo* is ready to start sending the vehicular sensor data to local the OBU.

## 4.2 Dissemination

Multiple hardware modules are needed for the different communication technologies that are required by the different flow directions:

- For communications between the *RedBear Duo* and the OBU, BLE is used. Table 4.3 details the specifications of the Bluetooth Dual-Mode USB adapter used in the OBU;
- For communication between the OBU and the Cloud, cellular networks are the best option, a LTE module was configured for this purpose. Table 4.4 details its specifications;
- For V2V and V2I communications, these are enabled by the ITS-G5 technology. A WiFi module was used because as ITS-G5, it is also based on the IEEE 802.11 technology. Table 4.5 details the specifications of the WiFi module used;

| | |
|---|---|
| Model | Asus USB-BT500 |
| Standard | BluetoothR 5.0+EDR BLE |
| Data Rate (BLE) | 2 Mbps |
| Range (BLE) | Up to 40 Meters (open space) |
| Frequency Band (BLE) | 2402   2480 MHz |
| Spectrum | FHSS |
| Temperature Range | 0 ~ 70º C (Operating) / -20 ~ 70º C (Storing) |

**Table 4.3:** Bluetooth dual-mode adapter specifications.

| | |
|---|---|
| Model | Huawei ME909s-120 |
| Technology | LTE - cat 4 |
| Maximum Speed | 150 Mbps (Downlink) / 50 Mbps (Uplink) |
| SIM Interface | Through miniPCIe connector |
| Temperature Range | -40 ~ 85º C (Operating) |

**Table 4.4:** LTE module specifications.

| | |
|---|---|
| Model | Atheros Compex WLE200NX |
| Technology | 802.11a/b/g/b |
| Frequency | 2.4 / 5 GHz |
| Maximum Output Power | 18 dBm |
| Temperature Range | -20 ~ 70º C (Operating) |

**Table 4.5:** WiFi module specifications.

### 4.2.1 BLE

The OBU and the CAN controller will both take roles that allows connection establishment, according to the GAP specification:

- **Peripheral** - This role will be taken by the CAN controller, waiting for connections and acting as a slave;
- **Central** - This role is filled by the OBU, scanning for the peripheral, requesting a connection to it, and then acting as the master.

*Peripheral*

This device needs to act as a server to provide the sensor data to other BLE devices, so it implements a GATT standard profile, composed by four different services. Figure 4.2 illustrates its layout:

- **GAP Service** - This is a mandatory service for servers that contains three read-only characteristics, one that specifies the device name, another for the appearance (type of device), and lastly for Peripheral Preferred Connection Parameters (PPCP) values (such as connection intervals and timeouts;
- **GATT Service** - This is a mandatory service for servers that contains one indication characteristic, to enable the server to change its structure during a connection while still maintaining synchronicity with the client;
- **Data Service** - This is a custom service that sends the vehicular sensor data to the OBU. It's composed by a single notification characteristic that is configured to send the most recent raw CAN messages every second to the OBU;
- **Battery Service** - This is a optional service that provides a read-only characteristic for reading the battery level.

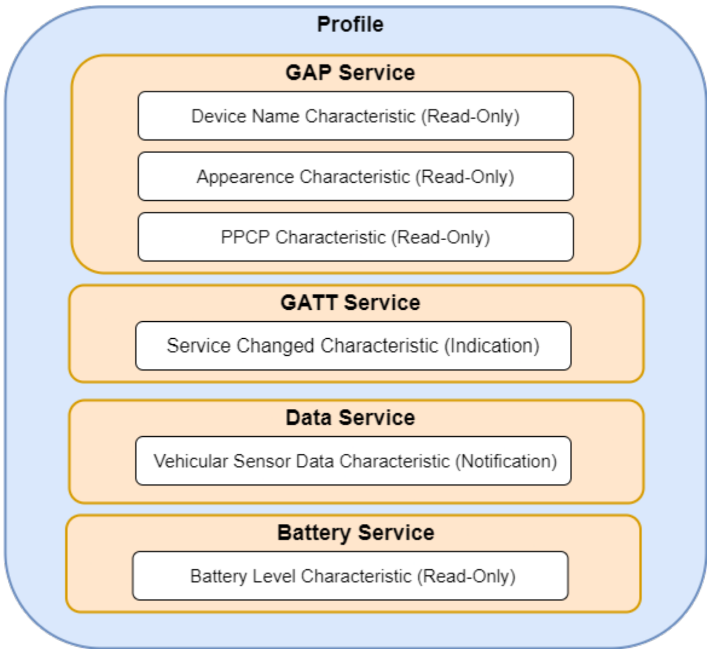

Profile
GAP Service
Device Name Characteristic (Read-Only)
Appearence Characteristic (Read-Only)
PPCP Characteristic (Read-Only)
GATT Service
Service Changed Characteristic (Indication)
Data Service
Vehicular Sensor Data Characteristic (Notification)
Battery Service
Battery Level Characteristic (Read-Only)

**Figure 4.2:** Organization of the peripheral services.

*Central*

This device needs to establish a connection with the Peripheral/Server, receiving its notifications without needing to provide any data. For this purpose, an application is designed that is able to access GATT standard profiles suffices, not requiring the implementation of a GATT profile.

This was implemented with an open-source library called *gattlib* [87], based on the linux *bluez* [88]. The following steps are taken by the application:

- **Scan** - In the first step, the central starts scanning for other BLE advertising devices until the peripheral is found (via MAC address, specified in a configuration file);
- **Connection and Registration** - After locating the peripheral, the central makes a connection request, registers and subscribes the Vehicular Sensor Data Characteristic in the Data Service, setting a callback function for processing each notification;
- **Main Loop** - This is where application keeps waiting for notifications, processing and writing them to a shared memory. More on this will be detailed in section 4.3. The loop keeps waiting for a *SIGINT* signal to stop the notification subscription and then disconnecting the device.

For managing the main event loop, another open-source library named *glib* [89] is used, that provides a main loop abstraction that polls event sources and invokes event handlers, avoiding busy-waiting periods when no notifications are sent.

### 4.2.2 Cellular Networks

To support the LTE cellular module, the Linux *cdc_mbim* driver is used. This driver is intended to support USB devices that are in compliance with the “Universal Serial Bus Communications Class Subclass Specification for Mobile Broadband Interface Model (MBIM)” protocol [90], optimized for mobile broadband devices.

This driver provides a interface (through the character device */dev/cdc_wdmX*) so users can have access to the device. Nevertheless, it does not participate in the device control and management, so tools such as *mbim-network* [91] and *mbimcli* [92] are required. The former is a simple network management tool of MBIM devices, the latter provides control over MBIM devices which includes triggering the connection overa cellular network and getting module information.

To start a connection, a *mbim-network* configuration file is required, to specify the Access Point Name (APN) details, as well a proxy option, to enable parallel commands to be sent to the module even if the interface already in use by a data connection. After this initial setup a connection can be established with the same tool. Code 1 shows the output of the connection status after its establishment.

```
it2s@admin@it2s-itss-03:/ mbim-network /dev/cdc-wdm0 status
Loading profile at /etc/mbim-network.conf
        APN: internetm2m
        APN auth protocol: unset
        APN user: unset
        APN password: unset
        mbim-proxy: yes
Getting status with 'mbimcli -d /dev/cdc-wdm0 --query-connection-state --device-open-proxy'...
Status: activated
```

**Code 1:** LTE connection status query.

A 5G module was also used, but its configuration was previously set up in Instituto de Telecomunicações - Aveiro (IT), which was utilized in the tests described in the next section.

### 4.2.3 ITS-G5

To fully support vehicular communications, the ITS-S platform should be able not only to operate within the ITS-G5 channel and power requirements, but also to transmit and receive OCB packets.

From version 3.19, the Linux kernel started to implement some changes in order to support IEEE 802.11p communications and OCB mode, making it possible to allow IEEE 802.11p communications with wireless cards that by default do not support IEEE 802.11p channel and power requirements, as it is with the Atheros Compex WLE200NX wireless card used.

To support the module, the Linux driver *ath9k* is used, but some changes must be applied to the driver in order to operate within the 5.9 GHz frequency band with 10 MHz channels and OCB mode. This are implemented via a custom kernel patch that specifies each central frequency of the ITS-G5 channels and the OCB mode of operation to the ath9k driver. This patch was previously developed at IT [93].

After the module is correctly working and able to communicate, a library was developed for employment in the access layer of the ITS-S reference architecture. This library is developed with raw sockets[1], implementing the following public methods for sending and receiving 802.11p frames:

- **it2s_ath_tx_packet**: Sends a frame. As argument, the frame contents and length are specified;
- **it2s_ath_rx_packet**: Listens for a frame;
- **it2s_ath_rx_packet_callback_set**: Sets a callback function to be called whenever a frame is received. As argument, a pointer to the callback function should be given;

[1]A raw socket bypasses the normal Transmission Control Protocol (TCP)/IP stack and sends the Ethernet frames directly to the application.

- **it2s__ath__loop**: Implements a main event loop, always listening for frames. As argument, a pointer to the object is should be given;
- **it2s__ath__user__data__set**: Sets a arbitrary value to a data structure to be used by other methods such as the callback function;
- **it2s__ath__free**: Used to delete and free the memory of internal variables used by the library.

In addition the arguments detailed above, all methods should also be given a pointer to a configuration data structure that contains fields such as socket file descriptors and *ifreq* structures, used to specify different interface parameters.

## 4.3 On-Board Unit Configuration

The protocol stack of the this ITS-S element was previously developed in IT [94], and it is used for the work described in this dissertation to implement, test and validate new features.

Besides the set of extensions described in the previous sections, it also includes a GPS receiver for location tracking. Their main specifications can be found in tables 4.6 and 4.7.

| Model | Apu3d4 |
|---|---|
| Dimensions | 16.8 x 16.7 x 3 cm |
| Weight | 245 g (Board) + 250 g (Chassis) |
| SoC | AMD Embedded G series GX-412TC |
| DRAM | 2 GB DDR3-1333 |
| Storage | -20 Internal SDHCI Controller, 1 SATA + power connector |
| Connector Type | 2.5 mm Jack, center positive, 12V DC |
| Input/Output | DB9 serial port, 2 USB 3.0 external, 4 USB 2.0 internal |
| Temperature Range | 0 $\sim$ 90º C (Operating) |

**Table 4.6:** OBU platform specifications.

| Model | G-Star IV |
|---|---|
| Dimensions | 10.6 x 9.7 x 5.1 cm |
| Weight | 99.8 g |
| Operating Voltage | 24 V |
| Sensivity | 163 dBm |
| Frequency | L1 1575.42 MHz |
| Number of Channels | 48 |
| Rate | 0.1 m/s |
| Temperature Range | -40 $\sim$ 90º C (Operating) |

**Table 4.7:** GPS receiver specifications.

### 4.3.1 Development Environment

The Linux distribution used by the OBU platform is *Arch Linux*. Although the patch referred to in the previous section works well for kernel versions lower or equal than *5.13.13-arch1* [95], it stopped working in following versions.

In order to have a stable kernel version that would not bring problems, a custom kernel was manually compiled and installed for the ITS-S platforms that used the WiFi module. These were the steps followed to attain this:

- **Linux kernel source retrieval**: This is achieved with the *svn* tool [96], allowing the retrieval of the specific 5.13.13 kernel source code;
- **PKGBUILD modification**: The intention of applying the patch should be declared here, as well as avoiding the compilation of documentation to save time;
- **Compilation and installation**: This is achieved with the *makepkg* tool [97], automating the compilation and installation of the *pacman*[2] package;
- **Initial ramdisk generation**: This is achieved with the *mkinitcpio* tool [98], which creates an initial ramdisk[3] environment for booting the custom linux kernel;
- **Bootloader configuration file generation**: The bootloader installed in the platforms is *GNU GRUB* [99]. It provides the tool *grub-mkconfig* that automatically detects the installed kernels and lists them in an array ordered by sequence of installation.

With the specification of a *pacman* hook file to run post-transaction commands in addition to PKGBUILD and patch files, this process was condensed into a single package added to an unofficial user repository, allowing the installation of the custom kernel to be done with one *pacman* command.

It is also important to note that the *Arch Linux* distribution follows a rolling release model, which means that not updating the kernel for a long time could bring a lot of issues to other drivers and applications. Therefore, a version of the newest kernel can always be kept and updated in addition to the custom kernel, using the *modprobe* tool (part of the Linux kernel) to specify which modules to use from either kernel.

### 4.3.2 Services

Besides the protocol stack of the ITS-S reference architecture running in the OBU, other services integrated with *systemd* [100] are developed that accommodate the need to process the vehicular sensor data received from the CAN controller and disseminate it using the technologies described in the previous sections:

- **it2s-obd-manager**: This process implements the BLE central described in section 4.2.1. It keeps listening for new messages from the CAN controller and always keeps the most recent ones updated, writing them in raw form into a shared memory file used for inter-process communications;

[2]*pacman* is the package manager used by Arch Linux.
[3]An initial ramdisk is used to load a temporary root file system into memory.

- **it2s-obd-reader**: At 1 Hz, this process gets the vehicle position from the GPS receiver via *gpsd*[4] [101], and the values in the previous shared memory file, decodes the raw messages into their corresponding values, and sends them to the cloud. One thread sends data to the MQTT broker (developed with *libmosquitto* [102]), and another to the SCoT platform (developed with *libcurl* [103]). The decoded messages are also kept updated in another shared memory file, used to share data with the ITS-S protocol stack.

In addition to these two services, the *facilities* layer of the ITS-S reference architecture is modified in order include relevant data to other ITS-S elements via C-ITS messages. It reads the values from the decoded shared memory whenever it exists, and include them in CAM and DENM messages upon their creation. The *access* layer was also adapted to include the option to use the *Atheros* WiFi Module, with the library described in section 4.2.3.

Figure 4.3 illustrates the interactions between the entities in the OBU.

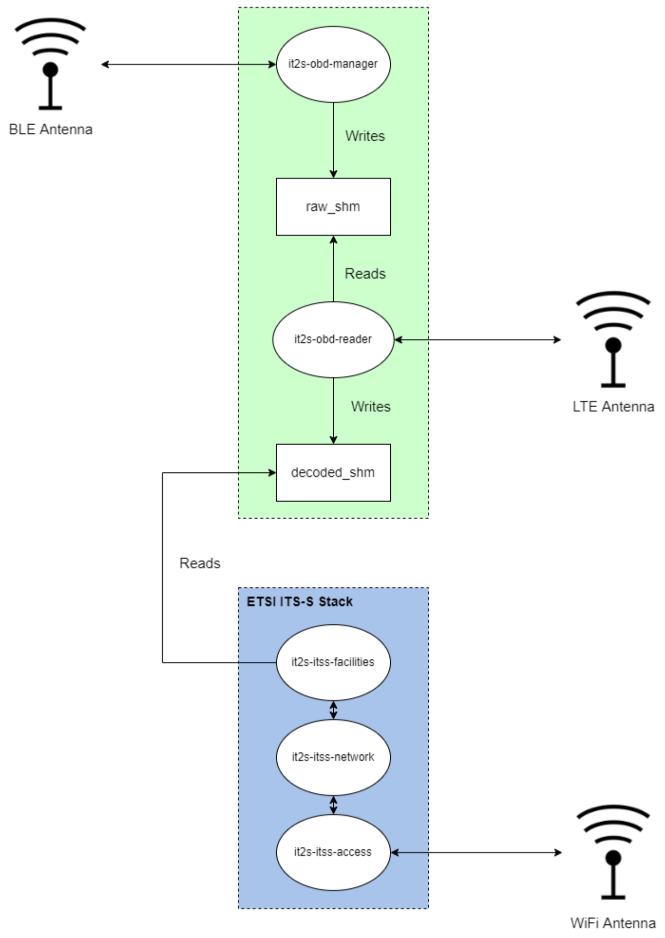


**Figure 4.3:** Organization of the OBU developed services.

[4]*gpsd* is a service daemon that monitors GPS receivers attached to a host computer through serial or USB ports.

In order to automate the execution of these services, *udev*[5] [100] rules were specified to start the services whenever the Linux kernel detects the insertion or removal of the USB Blutooth adapter. Code 2 shows the rules specified.

```
#ASUS USB_BT400
ACTION=="add", ATTRS{idVendor}=="0b05", ATTRS{idProduct}=="17cb", \
    MODE="0666", TAG+="systemd", ENV{SYSTEMD_WANTS}+="it2s-obd-manager.service"
ACTION=="remove", ATTRS{idVendor}=="0b05", ATTRS{idProduct}=="17cb", \
    RUN+="/bin/systemctl --no-block stop it2s-obd-manager.service"
```

**Code 2:** Udev rules specified for the Bluetooth adaptor.

The messages sent to the cloud are in JSON format and include the following vehicular sensor parameters:

- Vehicle Speed;
- Engine Speed;
- Engine Load;
- Wheels Speed;
- Steering Wheel Angle;
- Headlight Status;
- Barometric Pressure;
- Speed Pedal Position;
- Brake Pedal Position;
- Ambient Temperature;
- Anti-lock Braking System Status;
- Wiper Status.

In addition these parameters, location data is also attached to the messages, including latitude, longitude, altitude, speed, track and confidence values for each of them. Timestamps are assigned to to each distinct vehicular sensor parameter, and are in accordance to the ETSI standards [69], including leap seconds[6].

A configuration file must be present that specifies the correspondent IDs of each of these values in order to properly decoded them. These can be determined by the reverse engineering methods described in section 4.1.1 or through online repositories such as *opengarages*. If no configuration at this level is done, the only responses to the parameters present in the OBD-II standard will be processed and sent to the cloud.

Finally, the system clock is configured to use the GPS receiver as a the prioritized time synchronization source. This is accomplished with the Network Time Protocol (NTP) protocol, through the *ntpd* daemon.

[5] *udev* is a userspace system that enables the operating system administrator to register userspace handlers for input/output events.

[6] A leap second is a one-second adjustment that is occasionally applied to the Coordinated Universal Time, to accommodate the difference between the International Atomic Time and the imprecise observed solar time, which varies due to irregularities and long-term slowdown in the earth's rotation.

## 4.4 Data Access

On the cloud, two endpoints for data access are provided. An MQTT broker is used for message subscription in real time, while an API can be used for historical data retrieval. Both were installed in *Arch Linux* servers. For TLS encryption, a public key certificate is generated with *certbot* [104], issued by *letsencrypt*[7] and tested with *openssl* [105].

*MQTT Broker*

The broker is instantiated with the development version of *mosquitto* [102][8]. A modification is done to its corresponding *systemd* [100] service file, to enable writing to a key log file the TLS session keys exchanged by the the different clients and the server. It's usage is explained in section 4.5.2.

As a complement to the broker, a Message Distribution Service is also developed, that subscribes to the local MQTT broker input queue and publishes relevant messages to other output queues, which are used by other clients to subscribe to the data.

*API*

The API was developed on top of the previously existing Plataforma Aberta para o desenvolvimento e experimentação de Soluções para MObilidade (PASMO) API [106]. It gets data from a *postgres* [107] database, where the SCoT platform backups various sensor data. It's developed with the *flask* framework [108], instantiating an API conforming to the REpresentational State Transfer (REST) architectural style. The following endpoints were added:

- **/vehicles**: Returns data from every vehicle during a time period specified;
- **/vehicles/stationID**: Returns data from a specific vehicle (stated by *stationID*) during a time period specified.

In addition to the endpoints created for the vehicular sensor data, more were created to accommodate stationary weather stations data, from another partner of the TRUST project:

- **/weather**: Returns the most recent weather information from every weather station installed;
- **/weather/stationID**: Returns weather data from a specific station (stated by *stationID*) during a time period specified;
- **/weather/stationID/measurement**: Returns a specific weather measurement (given by *measurement* from a specific station (stated by *stationID*) during a time period specified. A *groubpy* field can also be set to group the results by any length of time.

Order, limit and offset values can also be specified for every endpoint. The API is deployed using *ngnix*[9] [109] on a *Docker* container [110].

[7] *letsencrypt* is a non-profit certificate authority run by Internet Security Research Group (ISRG), that provides X.509 certificates for TLS encryption at no charge.
[8] *mosquitto* is an open source implementation of a server that applies the MQTT protocol.
[9] *ngnix* is a popular lightweight web application that is used for developing server-side applications.

## 4.5 Logging

Reliable logging mechanisms are required in order to test the strict CCAM timing requirements, to understand the applications and network behaviour, and their relationship with the connect cars behaviour.

Here, the MQTT broker was installed in a portuguese 5G MEC *Arch Linux* server. This server is directly connected to a spanish 5G MEC server, providing a low-latency channel for communications between the different servers. Another virtual network interface also exists for communications that come directly from different *gNodeB* radio towers. Figure 4.4 illustrates the different entities and relationships defined in the context of the project.

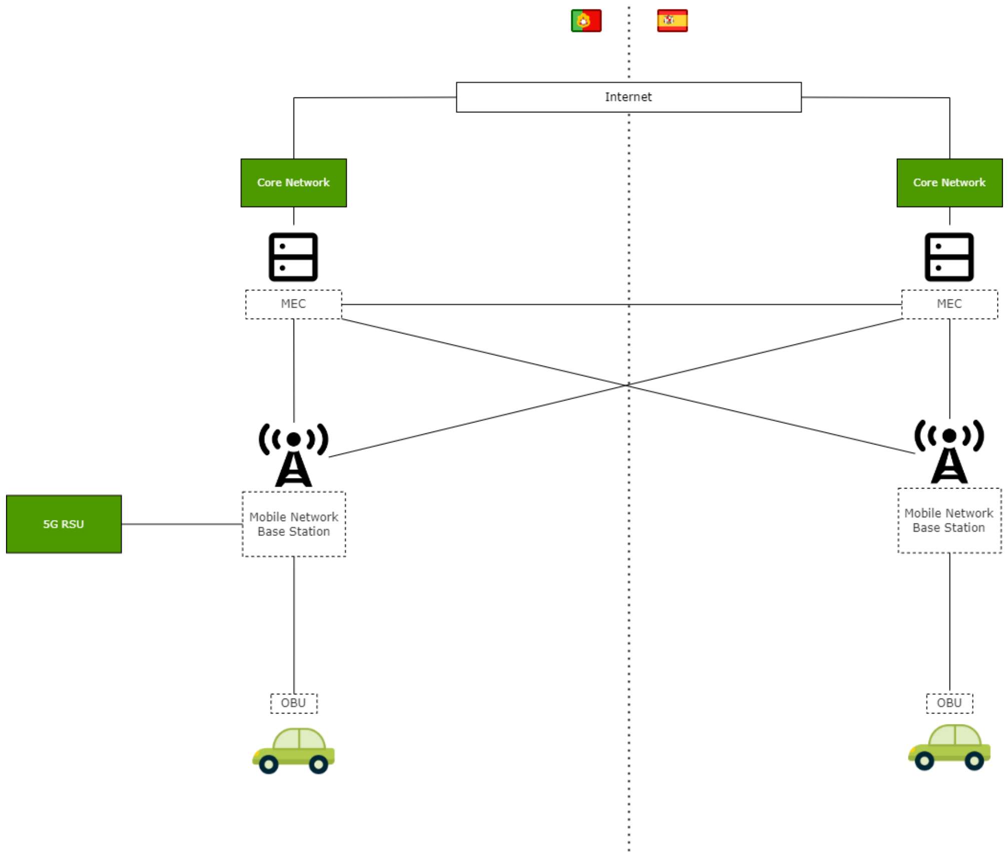


**Figure 4.4:** 5G-MOBIX deployed architecture in a cross-border region.

Due to roaming, according to the type of Subscriber Identity Module (SIM) card that each OBU has, they will either communicate with the MQTT broker in the portuguese MEC (with a portuguese SIM card), or to the Spanish one (with a spanish SIM card), regardless of the *gNodeB* radio tower they connect to.

Since connected cars can only be subscribed to one broker, and need to be aware of all other vehicles (despite of their type SIM card), the Message Distribution Service described in section 4.4 also publishes every message received in the input queue, to an inter-MEC queue, in addition to the output queue.

Logging at different levels is important to understand why problems might occur. For example, high delays or data-loss could caused by different reasons, such as a high overhead on the OBU CPU, or a congested network, or even poor Received Signal Strength Indication (RSSI) on the 5G module of the OBU. Therefore, logging is divided into different abstraction layers, which are applications, network and access.

The implementation described in this dissertation focuses on the logging part of the MEC server, which does not include access layer logging because only wired interfaces are used, defeating it's purpose.

### 4.5.1 Applications Layer

Accordingly to the project requirements, the following parameters are logged at the application level, by the MEC server for each MQTT message published in the broker.

- **Timestamp**: Absolute Unix timestamp in milliseconds;
- **Message**: Data contained in the MQTT message, serialized according to the ASN.1 standard [111];
- **Message Size**: Size in bytes of the data contained in the MQTT message;
- **Topic**: Topic to which the MQTT message was published;
- **Message Type**: C-ITS message type of the message, such as CAM or CPM;
- **Topic Direction**: Queue where the message was published (e.g.: input queue, output queue, inter-MEC queue).

This is achieved by the Message Distribution Service, where the logger is implemented. Logging every message that is published by other stations (such as OBUs or RSUs) to the input queue, and every message it republishes in other output queues. Figure 4.5 illustrates this mechanism.

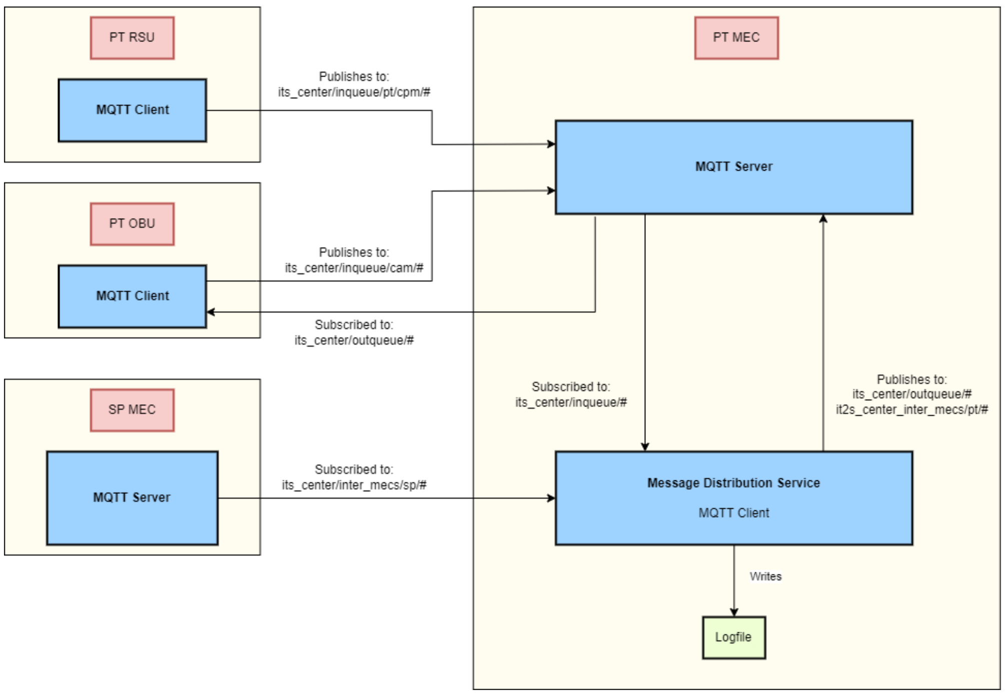


**Figure 4.5:** Application layer logging mechanism.

After every log is written to the logfile, they are be imported into a Structured Query Language (SQL) table, with a *python* script processing the file and inserting a column for each MQTT message.

### 4.5.2 Network Layer

Accordingly to the project requirements, the following parameters are logged at the network level, by the MEC server for each MQTT packet received or sent by the server.

- **Timestamp**: Absolute Unix timestamp in milliseconds;
- **Source Address**: Source IP address;
- **Destination Address**: Destination IP address;
- **Packet Type**: MQTT packet type, such as *CONNECT*, *SUBSCRIBE* or *PUBLISH*;
- **Message**: Data contained in the MQTT payload, serialized according to the ASN.1 standard;
- **Message Size**: Size in bytes of the data contained in the MQTT payload;
- **Message Type**: C-ITS message type of the message, such as CAM or CPM;
- **Topic**: Topic present in the MQTT payload.

This is achieved through *tcpdump*, a tool provided by the *wireshark* [112] command line interface. A capture is made both on the radio and the inter-MEC interfaces. Finishing the capture, these are merged with the *mergecap* tool.

*TLS Decryption*

Since all the communications with the server are encrypted via public key certificates, a way to decrypt the traffic from the captures must be implemented in order to extract the parameters specified in the previous section.

This is done using the TLS session keys already logged by the *mosquitto* server, as described in section 4.4.

In TLS, two communicating parties generate 4 session keys at the start of any communication session, during the TLS handshake. A session key is any encryption key used to symmetrically encrypt one communication session only. In other words, it's a temporary key that is only used once, during one stretch of time, for encrypting and decrypting data.

With the TLS sessions keys and the capture file, it is now possible to inject the keys into the capture file, filter the important fields out of each packet in the capture, and import these fields into a SQL database table. Figure 4.6 illustrates the steps to achieve this.

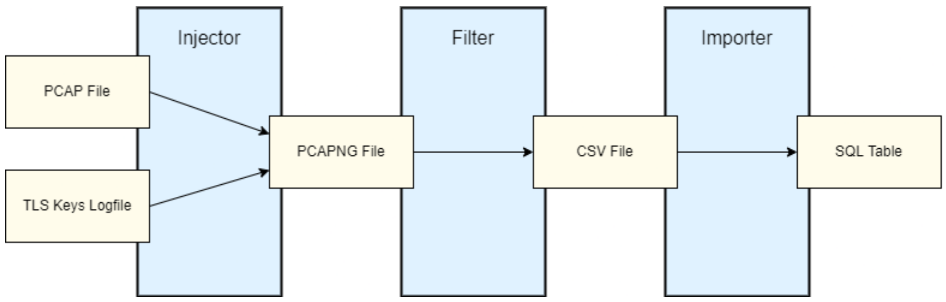


**Figure 4.6:** Network layer processing mechanism.

If multiple capture files exist (e.g: a capture for the radio and inter-mec interfaces of the 5G MEC server), these can be merged with *mergecap*. Then, the first step uses as input the capture file and a logfile containing the shared TLS keys, these files are given to *tshark*[10] to select the necessary keys, verify that the keys for every TLS session exist, and then with injecting them into the capture file, outputting a PCAP New Generation (PCAPNG) capture file. The next step takes the previous output, filters only the relevant packets ( MQTT connection , subscribe and publish packets), and outputs them into a file formatted as Comma Separated Values (CSV). Finally, the filtered packets are imported to a SQL database table.

All these steps are condensed into a single *python* script that automates these processes.

### 4.5.3 Processing

After acquiring the logfiles for the different elements and importing them to their respective SQL database tables, additional processing is required in order to extract valuable data, such as latency, throughput and data-loss ratios.

With this in mind, two types of scripts were created to correlate the data from different entities:

- **Matching**: This type of script matches each message/packet one-by-one, through their data. This allows the calculation of latency values per item and total amount of lost messages/packets;
- **Aggregations**: These are similar do the matching scripts, but with the addition of custom sliding time windows. This allows to count the number of messages/packets sent and received throughout multiple small time windows, outputting features such as average end-to-end latency, data-loss/packet-loss ratios, and data-rate/throughput values.

[10] *tshark* and *mergecap* are tools provided by the *wireshark* command line interface [112].

CHAPTER 5

# Tests and Results

*This chapter exposes the results from the tests performed to evaluate and validate the implementation of the system, as well as the tools used for testing. It starts with the results from the identification and extraction of the vehicular sensor data, then going over the validation of the communication mechanisms, followed by the usage of the data access channels and the validation of the logging procedures. Finally, an in-depth analysis of these results is discussed.*

## 5.1 Vehicular Sensor Data

To test the OBD-II parameter extraction, a development board by *OZEN Elektronic* was used. This board can simulate up to three different ECUs, generate DTCs, and control five parameters in real-time, using potentiometers. Figure 5.1 shows the board.

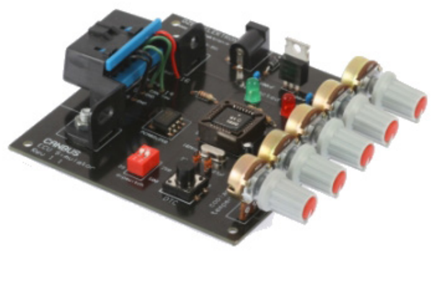


**Figure 5.1:** CAN bus OBD Simulator.

The controller used in this board is the *OE91C1610*, which is compatible with the ISO 15765-4 standard, supports multiple fixed OBD-II PIDs and a check engine light [113].

This was a vital component in the initial testing phase of this dissertation because it allowed to test the communications with the bus through the OBD-II port without needing a vehicle.

After developing and testing a basis for requesting and processing OBD-II data through the simulator, the next step was to make this requests in a real vehicle, log every CAN message that was detected in the bus, and finally correlating the data with the script developed at section 4.1.1. Figures 5.2 to 5.8 provide the comparison between the the evolution of different OBD-II parameters and the best candidate provided by the algorithm.

*Engine Speed*

As specified in the OBD-II standard, this value is given by $\frac{(256A+B)}{4}$, where $A$ is the first data byte, and $B$ the second. The algorithm identified the second and third bytes of the 0c9 CAN ID as the best candidates, with a pearson correlation factor $\rho = 0.97$ for the first byte, and $\rho = 0.86$ for the second.

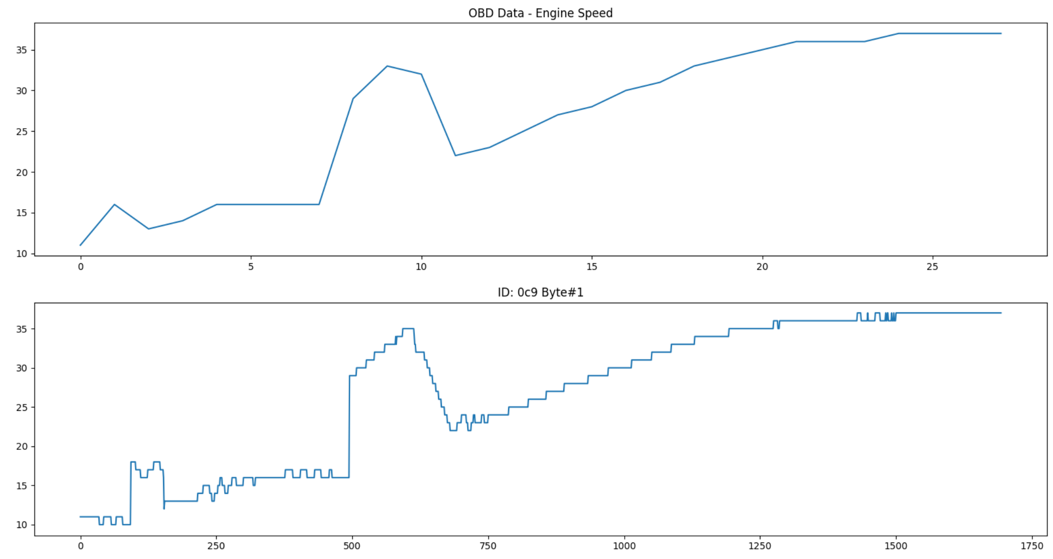


**Figure 5.2:** OBD data for engine speed (byte $A$) and CAN ID 0c9 (byte #1).

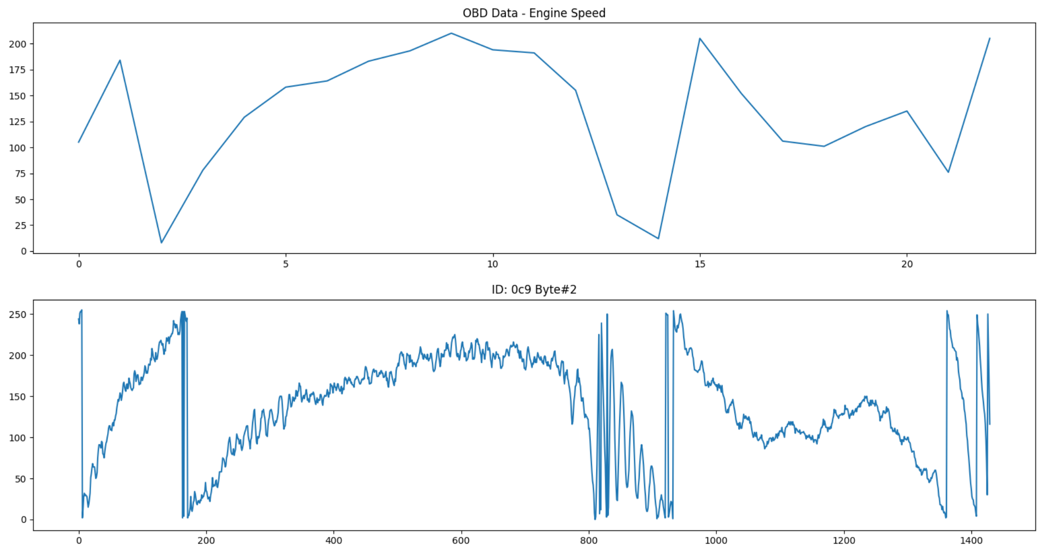


**Figure 5.3:** OBD data for engine speed (byte $B$) and CAN ID 0c9 (byte #2).

*Engine Load*

The engine load is given by $\frac{100}{255}A$, where $A$ is the first data byte. The algorithm identified the second byte of the 3d1 CAN ID as the best candidate, with a pearson correlation factor $\rho = 0.78$.

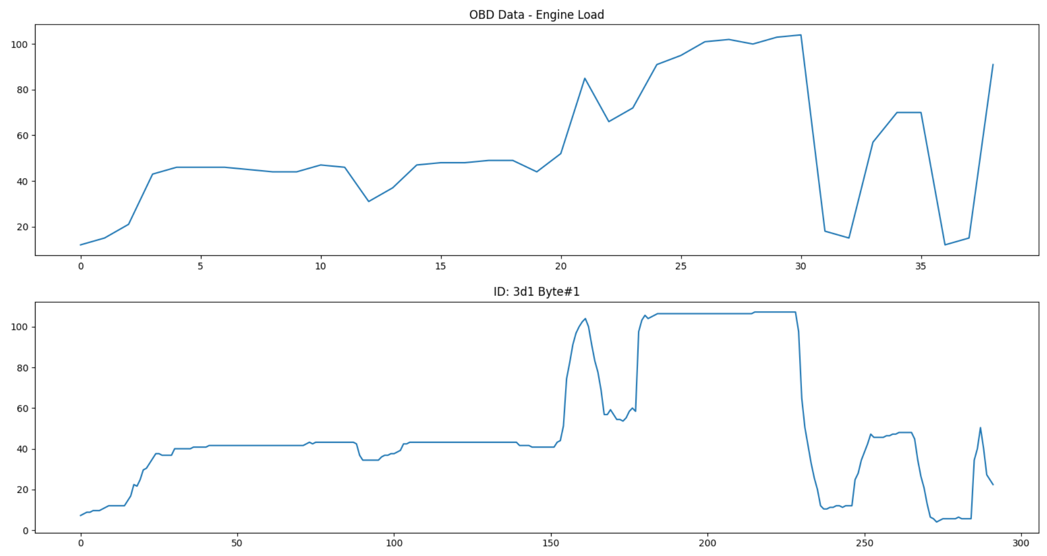


**Figure 5.4:** OBD data for engine load (byte $B$) and CAN ID 3d1 (byte #1).

*Vehicle Speed*

This parameter is given by $A$, the first data byte. Although only one data byte exists in the OBD-II response, the algorithm identified that the CAN messages containing the vehicle speed, used two bytes ($256A + B$). The first and seconds bytes of the 3e9 CAN ID formed the best candidate, with a pearson correlation factor $\rho = 0.95$.

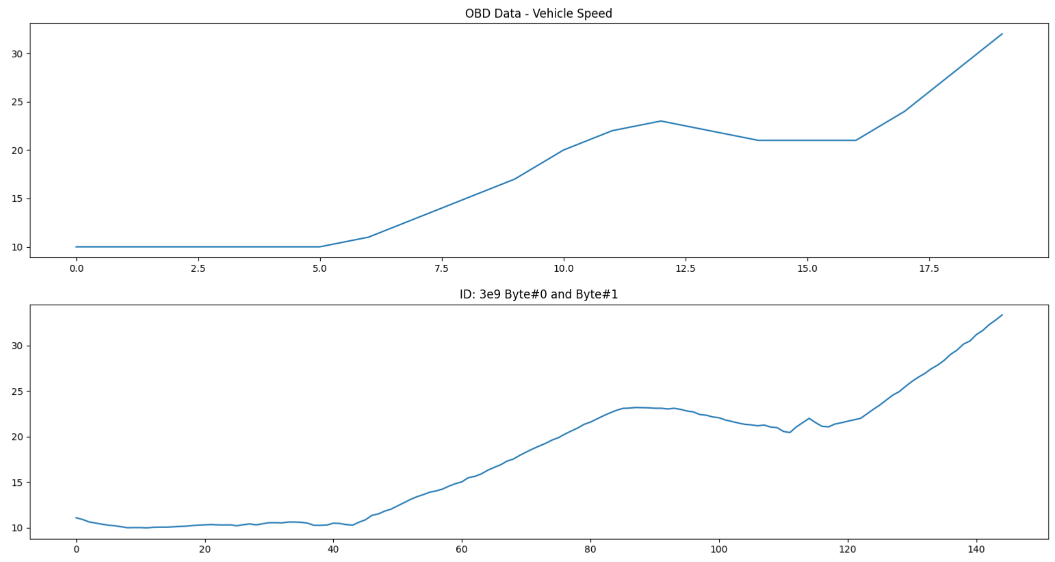


**Figure 5.5:** OBD data for vehicle speed (byte $A$) and CAN ID 3e9 (byte #0 and #1).

*Throttle Pedal Position*

This value is given by $\frac{100}{255}A$, where $A$ is the first data byte. The algorithm identified the fifth byte of the 0c9 CAN ID as the best candidate, with a pearson correlation factor $\rho =$ -0.89.

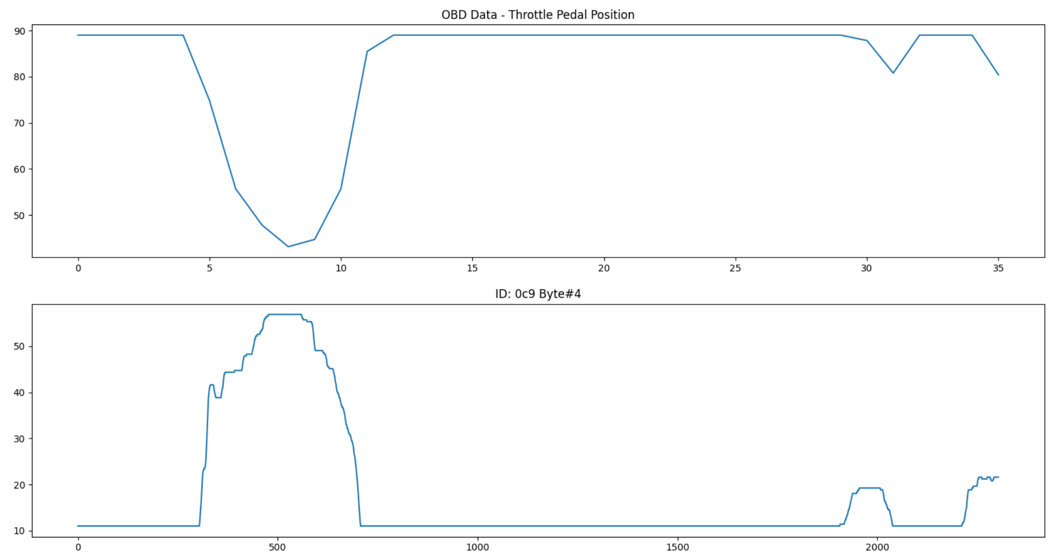


**Figure 5.6:** OBD data for engine load (byte $A$) and CAN ID 0c9 (byte #4).

*Wheels Speed*

Although this value is not specified in the OBD-II PID list, these can be guessed correlating the vehicle speed CAN data with the other CAN messages. The algorithm identified the 34a and 348 CAN IDs as the best candidates, both with a pearson correlation factor $\rho = 0.97$.

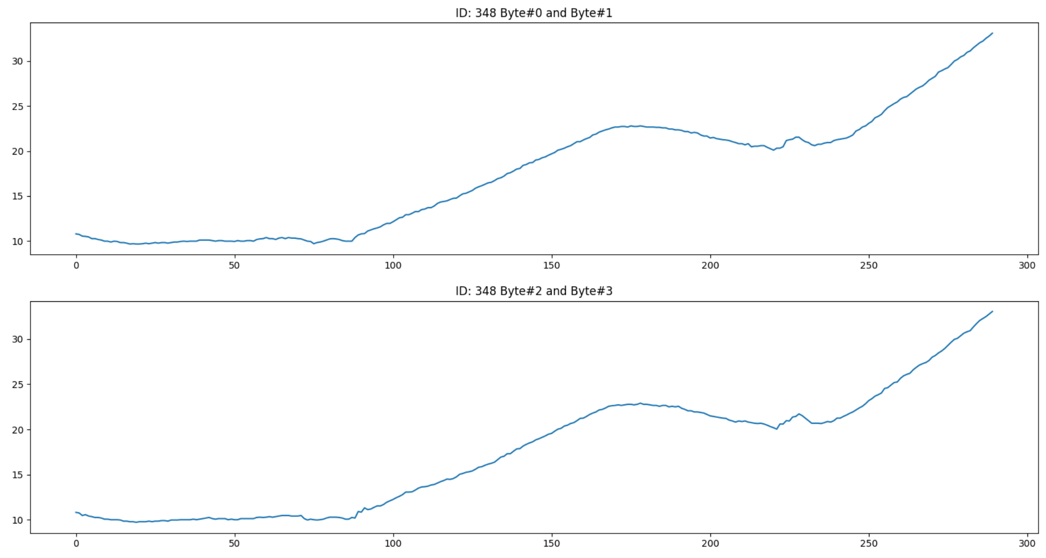


**Figure 5.7:** CAN ID 34a (byte #0 and #1) and CAN ID 34a (byte #2 and #3).

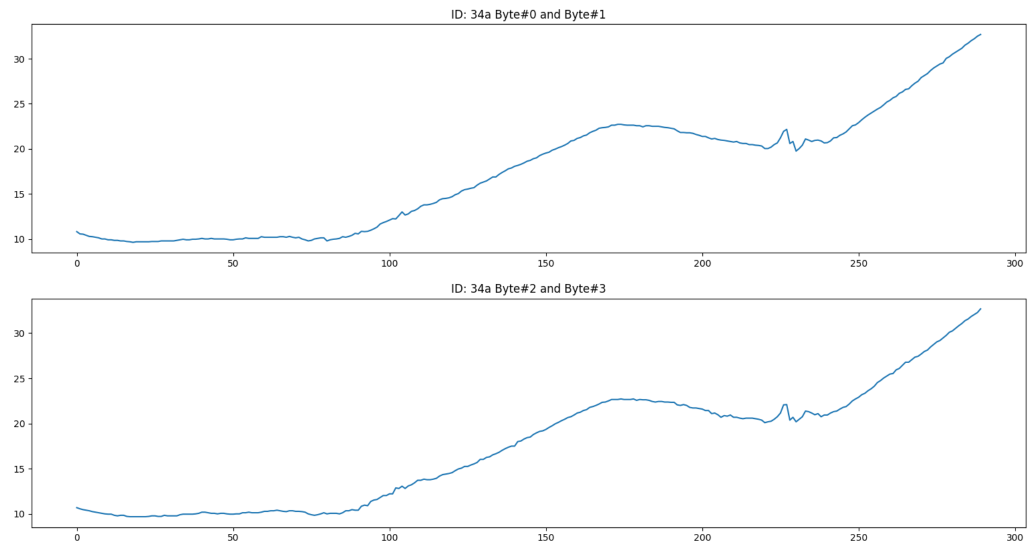


**Figure 5.8:** CAN ID 348 (byte #0 and #1) and CAN ID 348 (byte #2 and #3).

*Summary*

All the previous results were acquired using a Opel Adam from 2016, and different offset and scale values are used in comparison to their correspondant OBD-II data. A resume of the identified parameters and its features is presented in table 5.1.

| **Opel Adam (2016)** | **CAN ID (HEX)** | **Byte(s)** | **Unit** | **Offset** | **Scale** |
|---|---|---|---|---|---|
| Engine Speed | 0c9 | 1 | rpm | 0 | 1 |
| Engine Load | 3d1 | 1 | % | -20 | 4/5 |
| Vehicle Speed | 3e9 | 0-1 | km/h | 0 | 1/64 |
| Throttle Pedal | 0c9 | 4 | % | 100 | -1 |
| Wheels Speed | 34a | 0-1 | km/h | 0 | 1/32 |
| | 34a | 2-3 | km/h | 0 | 1/32 |
| | 348 | 0-1 | km/h | 0 | 1/32 |
| | 348 | 2-3 | km/h | 0 | 1/32 |

**Table 5.1:** Identified parameters of Opel Adam (2016) with its respective features.

In addition to the Opel Adam, tests concerning the same parameters were also made in a Peugeot 308 from 2017 and in a Seat Ibiza from 2011. The former provided similar results to the Opel, while in the latter, no CAN messages could be captured, in spite of being able to request OBD-II data and reading the responses without any problem.

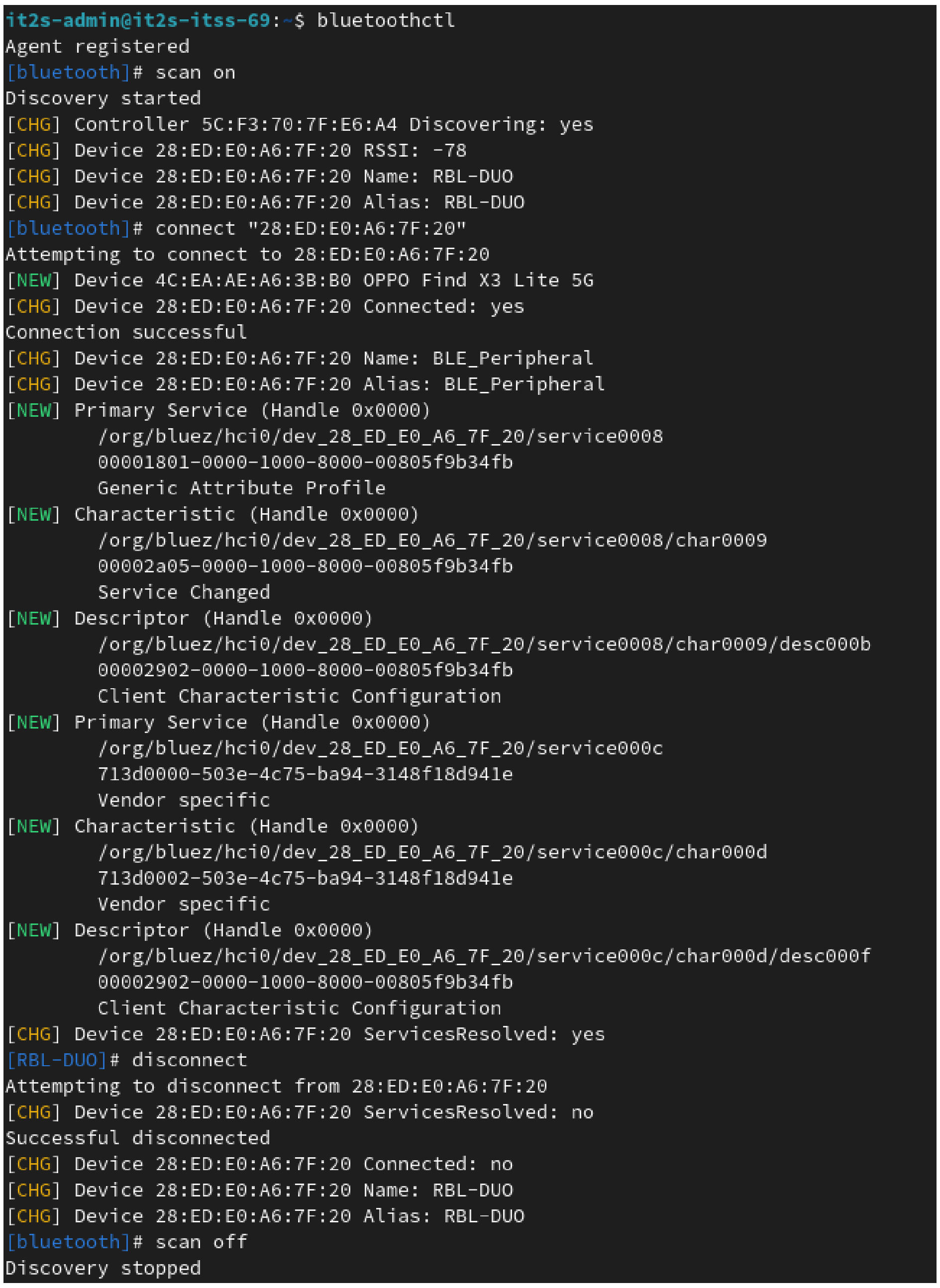
Furthermore, tests for other useful parameters such as headlight status, wipers activation and steering wheel angle were also made using the algorithms developed at [80]. This however proved to be quite inconsistent between the different cars. A reason for this may be that the control of these entities is done through the LIN bus, without the need to share messages through the CAN bus.

## 5.2 Dissemination

To test the different flows, isolated tests and assertions were made regarding each technology.

*BLE*

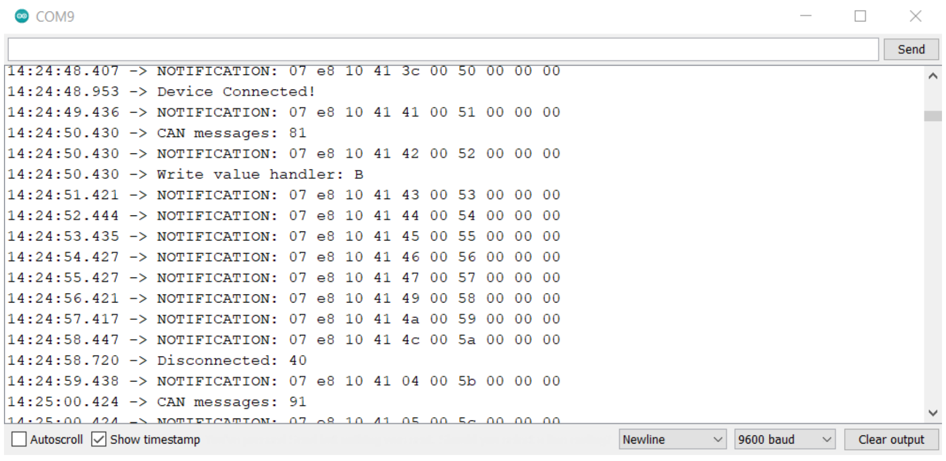
The connection from the central device (OBU) to the peripheral was tested using the *bluetoothctl* tool, provided by *bluez* [88]. A successful connection is represented in figure 5.9, also displaying a service discovery, validating the GATT profile specified in 4.2.1.



```
it2s-admin@it2s-itss-69:~$ bluetoothctl
Agent registered
[bluetooth]# scan on
Discovery started
[CHG] Controller 5C:F3:70:7F:E6:A4 Discovering: yes
[CHG] Device 28:ED:E0:A6:7F:20 RSSI: -78
[CHG] Device 28:ED:E0:A6:7F:20 Name: RBL-DUO
[CHG] Device 28:ED:E0:A6:7F:20 Alias: RBL-DUO
[bluetooth]# connect "28:ED:E0:A6:7F:20"
Attempting to connect to 28:ED:E0:A6:7F:20
[NEW] Device 4C:EA:AE:A6:3B:B0 OPPO Find X3 Lite 5G
[CHG] Device 28:ED:E0:A6:7F:20 Connected: yes
Connection successful
[CHG] Device 28:ED:E0:A6:7F:20 Name: BLE_Peripheral
[CHG] Device 28:ED:E0:A6:7F:20 Alias: BLE_Peripheral
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/hci0/dev_28_ED_E0_A6_7F_20/service0008
        00001801-0000-1000-8000-00805f9b34fb
        Generic Attribute Profile
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_28_ED_E0_A6_7F_20/service0008/char0009
        00002a05-0000-1000-8000-00805f9b34fb
        Service Changed
[NEW] Descriptor (Handle 0x0000)
        /org/bluez/hci0/dev_28_ED_E0_A6_7F_20/service0008/char0009/desc000b
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/hci0/dev_28_ED_E0_A6_7F_20/service000c
        713d0000-503e-4c75-ba94-3148f18d941e
        Vendor specific
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/hci0/dev_28_ED_E0_A6_7F_20/service000c/char000d
        713d0002-503e-4c75-ba94-3148f18d941e
        Vendor specific
[NEW] Descriptor (Handle 0x0000)
        /org/bluez/hci0/dev_28_ED_E0_A6_7F_20/service000c/char000d/desc000f
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
[CHG] Device 28:ED:E0:A6:7F:20 ServicesResolved: yes
[RBL-DUO]# disconnect
Attempting to disconnect from 28:ED:E0:A6:7F:20
[CHG] Device 28:ED:E0:A6:7F:20 ServicesResolved: no
Successful disconnected
[CHG] Device 28:ED:E0:A6:7F:20 Connected: no
[CHG] Device 28:ED:E0:A6:7F:20 Name: RBL-DUO
[CHG] Device 28:ED:E0:A6:7F:20 Alias: RBL-DUO
[bluetooth]# scan off
Discovery stopped
```

**Figure 5.9:** BLE connection output from the central side.

A callback function is also set on the peripheral device, to be called when a device connects. Figure 5.10 shows the output logs of the peripheral device when the central device performs a connection.

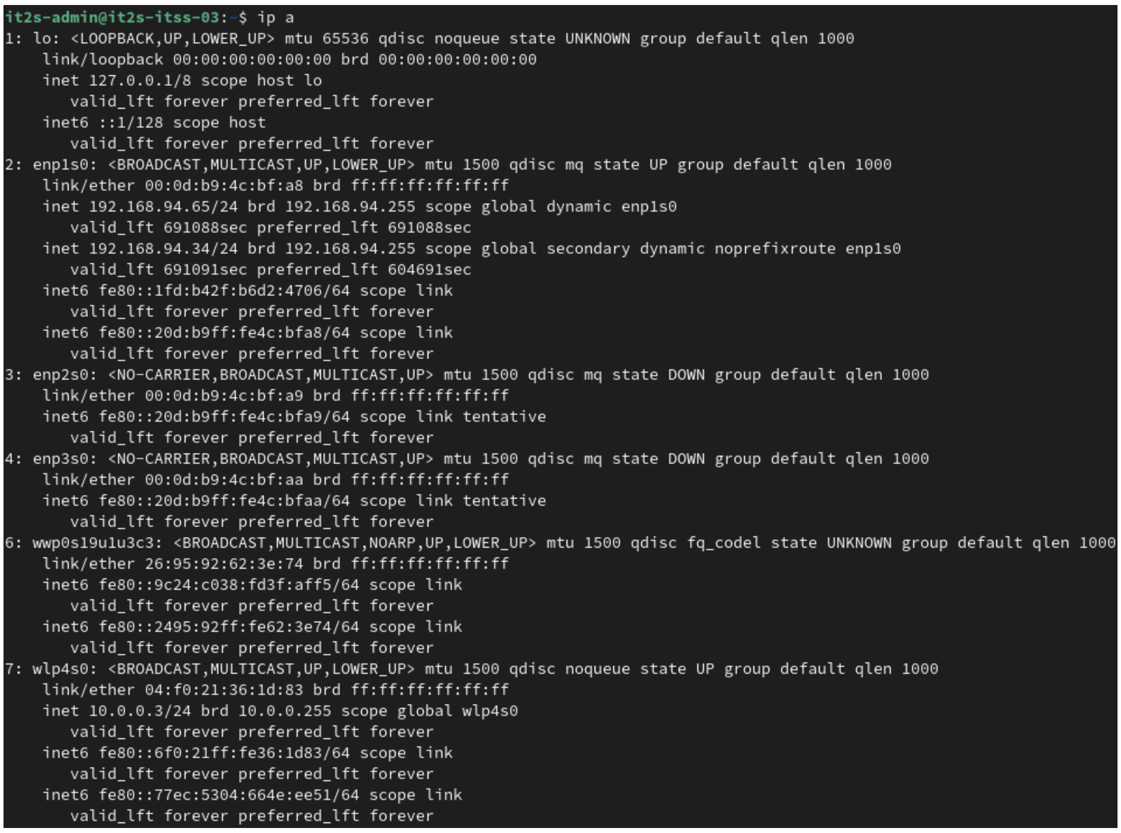


```
14:24:48.407 -> NOTIFICATION: 07 e8 10 41 3c 00 50 00 00 00
14:24:48.953 -> Device Connected!
14:24:49.436 -> NOTIFICATION: 07 e8 10 41 41 00 51 00 00 00
14:24:50.430 -> CAN messages: 81
14:24:50.430 -> NOTIFICATION: 07 e8 10 41 42 00 52 00 00 00
14:24:50.430 -> Write value handler: B
14:24:51.421 -> NOTIFICATION: 07 e8 10 41 43 00 53 00 00 00
14:24:52.444 -> NOTIFICATION: 07 e8 10 41 44 00 54 00 00 00
14:24:53.435 -> NOTIFICATION: 07 e8 10 41 45 00 55 00 00 00
14:24:54.427 -> NOTIFICATION: 07 e8 10 41 46 00 56 00 00 00
14:24:55.427 -> NOTIFICATION: 07 e8 10 41 47 00 57 00 00 00
14:24:56.421 -> NOTIFICATION: 07 e8 10 41 49 00 58 00 00 00
14:24:57.417 -> NOTIFICATION: 07 e8 10 41 4a 00 59 00 00 00
14:24:58.447 -> NOTIFICATION: 07 e8 10 41 4c 00 5a 00 00 00
14:24:58.720 -> Disconnected: 40
14:24:59.438 -> NOTIFICATION: 07 e8 10 41 04 00 5b 00 00 00
14:25:00.424 -> CAN messages: 91
14:25:00.424 -> NOTIFICATION: 07 e8 10 41 05 00 5c 00 00 00
```

**Figure 5.10:** BLE connection output from the peripheral side.

*LTE and ITS-G5*

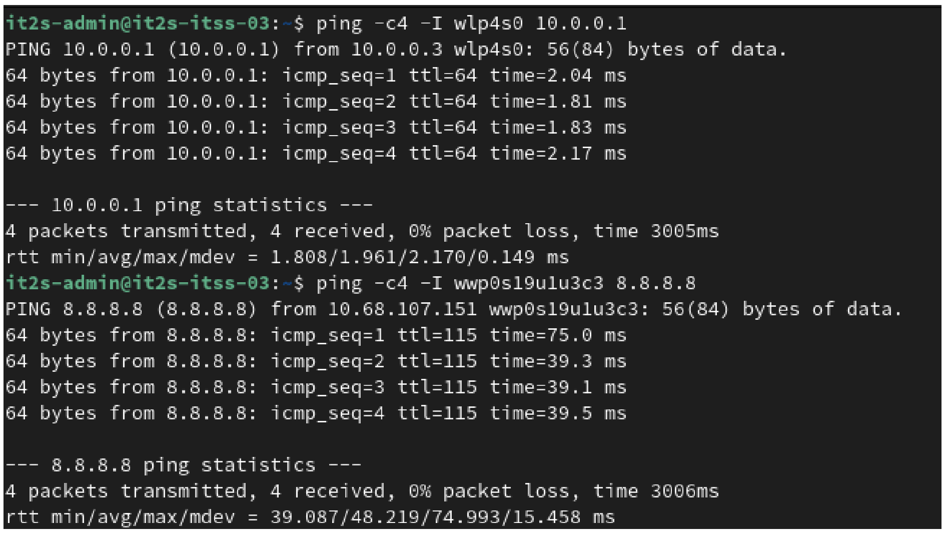
The first step is to assert that this interfaces are properly set up and with an assigned IP address. Figure 5.11 shows the interfaces description on the OBU. where *wwp0s19u1u3c3* is the LTE interface and *wlp4s0* the ITS-G5 interface.



```
it2s-admin@it2s-itss-03:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:0d:b9:4c:bf:a8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.94.65/24 brd 192.168.94.255 scope global dynamic enp1s0
       valid_lft 691088sec preferred_lft 691088sec
    inet 192.168.94.34/24 brd 192.168.94.255 scope global secondary dynamic noprefixroute enp1s0
       valid_lft 691091sec preferred_lft 604691sec
    inet6 fe80::1fd:b42f:b6d2:4706/64 scope link
       valid_lft forever preferred_lft forever
    inet6 fe80::20d:b9ff:fe4c:bfa8/64 scope link
       valid_lft forever preferred_lft forever
3: enp2s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether 00:0d:b9:4c:bf:a9 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20d:b9ff:fe4c:bfa9/64 scope link tentative
       valid_lft forever preferred_lft forever
4: enp3s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether 00:0d:b9:4c:bf:aa brd ff:ff:ff:ff:ff:ff
    inet6 fe80::20d:b9ff:fe4c:bfaa/64 scope link tentative
       valid_lft forever preferred_lft forever
6: wwp0s19u1u3c3: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 1000
    link/ether 26:95:92:62:3e:74 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::9c24:c038:fd3f:aff5/64 scope link
       valid_lft forever preferred_lft forever
    inet6 fe80::2495:92ff:fe62:3e74/64 scope link
       valid_lft forever preferred_lft forever
7: wlp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 04:f0:21:36:1d:83 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.3/24 brd 10.0.0.255 scope global wlp4s0
       valid_lft forever preferred_lft forever
    inet6 fe80::6f0:21ff:fe36:1d83/64 scope link
       valid_lft forever preferred_lft forever
    inet6 fe80::77ec:5304:664e:ee51/64 scope link
       valid_lft forever preferred_lft forever
```

**Figure 5.11:** Interfaces description on the OBU.

Following this validation, the actual exchange of packets is tested. *ping* commands are tested between the LTE interface and the internet, and between two ITS-G5 interfaces configured similarly. It's results are shown in figure 5.12.



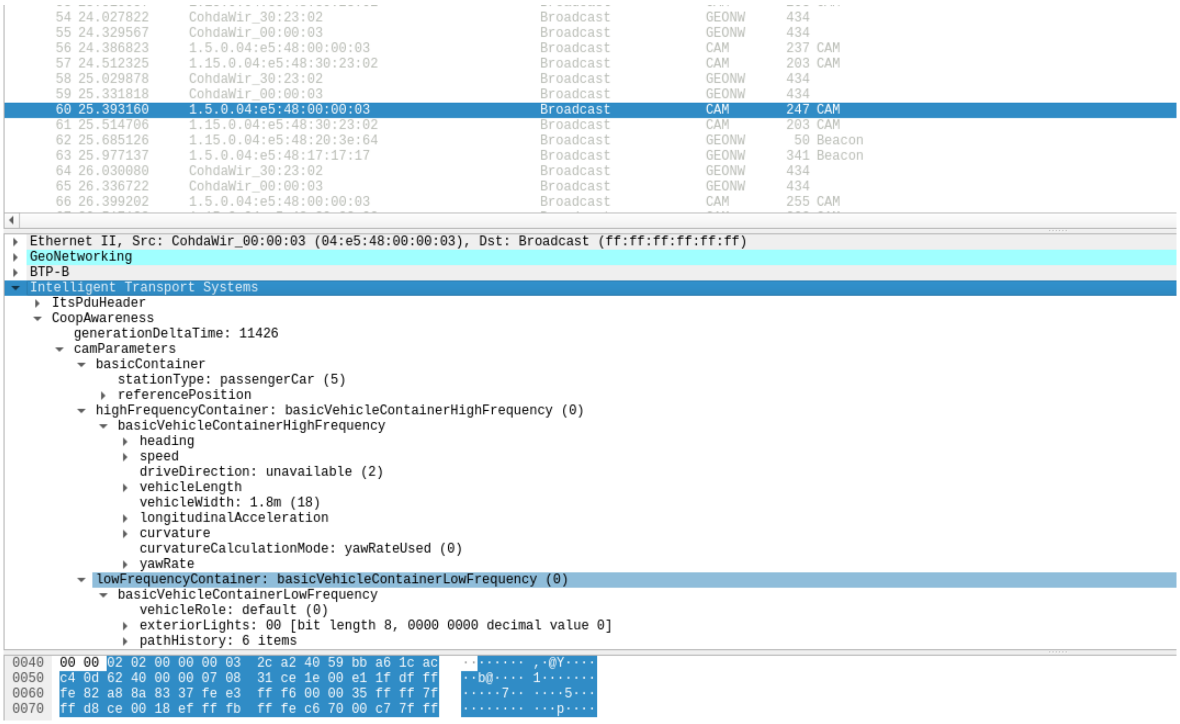
```
it2s-admin@it2s-itss-03:~$ ping -c4 -I wlp4s0 10.0.0.1
PING 10.0.0.1 (10.0.0.1) from 10.0.0.3 wlp4s0: 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=2.04 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1.81 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=1.83 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=2.17 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.808/1.961/2.170/0.149 ms
it2s-admin@it2s-itss-03:~$ ping -c4 -I wwp0s19u1u3c3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) from 10.68.107.151 wwp0s19u1u3c3: 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=75.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=39.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=39.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=39.5 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 39.087/48.219/74.993/15.458 ms
```

**Figure 5.12:** Results of *ping* commands through LTE and ITS-G5 interfaces.

A trace file was also captured using *wireshark*, containing the exchange of C-ITS messages between different stations, with data, where simulated data such as vehile speed and headlights status is inserted to the CAM containers. A part of this capture is shown in figure 5.13.



```
54 24.027822   CohdaWir_30:23:02              Broadcast    GEONW   434
55 24.329567   CohdaWir_00:00:03              Broadcast    GEONW   434
56 24.386823   1.5.0.04:e5:48:00:00:03        Broadcast    CAM     237 CAM
57 24.512325   1.15.0.04:e5:48:30:23:02       Broadcast    CAM     203 CAM
58 25.029878   CohdaWir_30:23:02              Broadcast    GEONW   434
59 25.331818   CohdaWir_00:00:03              Broadcast    GEONW   434
60 25.393160   1.5.0.04:e5:48:00:00:03        Broadcast    CAM     247 CAM
61 25.514706   1.15.0.04:e5:48:30:23:02       Broadcast    CAM     203 CAM
62 25.685126   1.15.0.04:e5:48:20:3e:64       Broadcast    GEONW    50 Beacon
63 25.977137   1.5.0.04:e5:48:17:17:17        Broadcast    GEONW   341 Beacon
64 26.030080   CohdaWir_30:23:02              Broadcast    GEONW   434
65 26.336722   CohdaWir_00:00:03              Broadcast    GEONW   434
66 26.399202   1.5.0.04:e5:48:00:00:03        Broadcast    CAM     255 CAM
```

```
▸ Ethernet II, Src: CohdaWir_00:00:03 (04:e5:48:00:00:03), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ GeoNetworking
▸ BTP-B
▾ Intelligent Transport Systems
  ▸ ItsPduHeader
  ▾ CoopAwareness
      generationDeltaTime: 11426
    ▾ camParameters
      ▾ basicContainer
          stationType: passengerCar (5)
        ▸ referencePosition
      ▾ highFrequencyContainer: basicVehicleContainerHighFrequency (0)
        ▾ basicVehicleContainerHighFrequency
          ▸ heading
          ▸ speed
            driveDirection: unavailable (2)
          ▸ vehicleLength
            vehicleWidth: 1.8m (18)
          ▸ longitudinalAcceleration
          ▸ curvature
            curvatureCalculationMode: yawRateUsed (0)
          ▸ yawRate
      ▾ lowFrequencyContainer: basicVehicleContainerLowFrequency (0)
        ▾ basicVehicleContainerLowFrequency
            vehicleRole: default (0)
          ▸ exteriorLights: 00 [bit length 8, 0000 0000 decimal value 0]
          ▸ pathHistory: 6 items
```

```
0040  00 00 02 02 00 00 00 03  2c a2 40 59 bb a6 1c ac   ········ ,·@Y····
0050  c4 0d 62 40 00 00 07 08  31 ce 1e 00 e1 1f df ff   ··b@···· 1·······
0060  fe 82 a8 8a 83 37 fe e3  ff f6 00 00 35 ff ff 7f   ·····7·· ····5···
0070  ff d8 ce 00 18 ef ff fb  ff fe c6 70 00 c7 7f ff   ········ ···p····
```

**Figure 5.13:** Capture on ITS-G5 interface showing exchanged packets between stations.

## 5.3 Data Access

To validate the data endpoints correct behaviour and the full end-to-end flow, several trips were made through the city of Aveiro where roadside ITS-S stations were deployed by IT, as well from Aveiro to Oliveira de Azeméis. The data collected in this trips consist of only parameters accessible through the OBD-II standard. The OBU was powered with the vehicle own battery, through a power inverter. Its specifications are presented in table 5.2.

| | |
|---|---|
| Model | Bapdas Power Inverter DC 12V to AC 220 V |
| Input DC Voltage | 12 Volts |
| Output AC Voltage | 220 Volts |
| Output Power | 500 Watts |
| Peak Efficiency | > 85 % |
| Dimensions | 150 x 95 x 55 mm |
| Weight | 540 g |

**Table 5.2:** Power inverter specifications.

Picture 5.14 shows a picture taken in Praia da Barra, where a RSU is deployed (in the background, on the leftmost orange pillar), with a Seat Ibiza used for the tests, equipped with the GPS receiver and ITS-G5 antennas on top of it. Picture 5.15 displays the CAN controller attached to the OBD-II port, and 5.16 shows the OBU on top of the right seat.

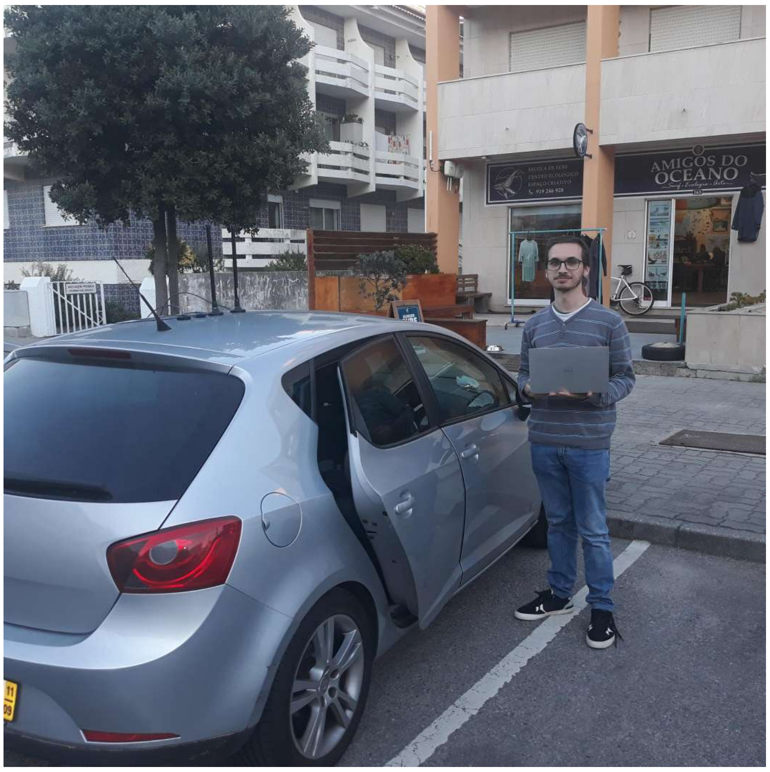


**Figure 5.14:** Vehicle equipped with GPS receiver and ITS-G5 antennas. RSU on the background.

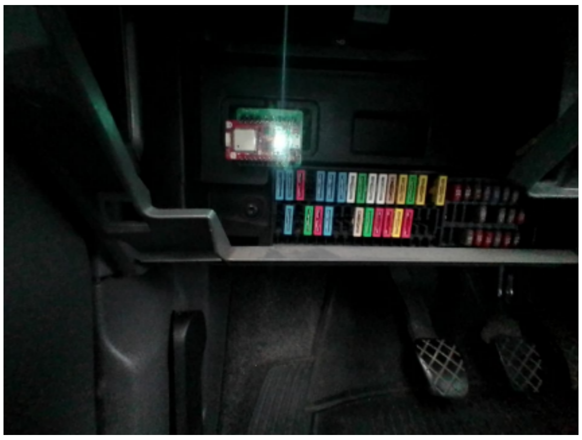


**Figure 5.15:** Installed CAN controller.

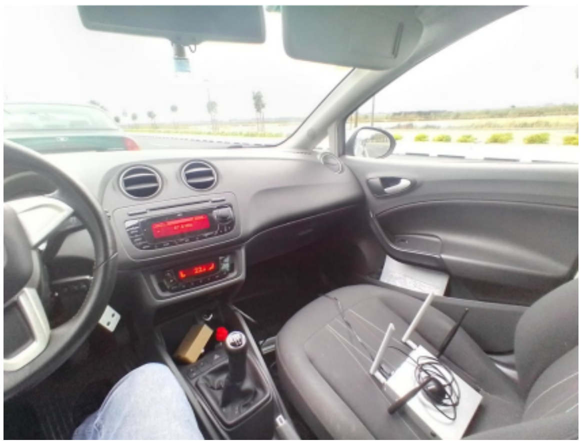
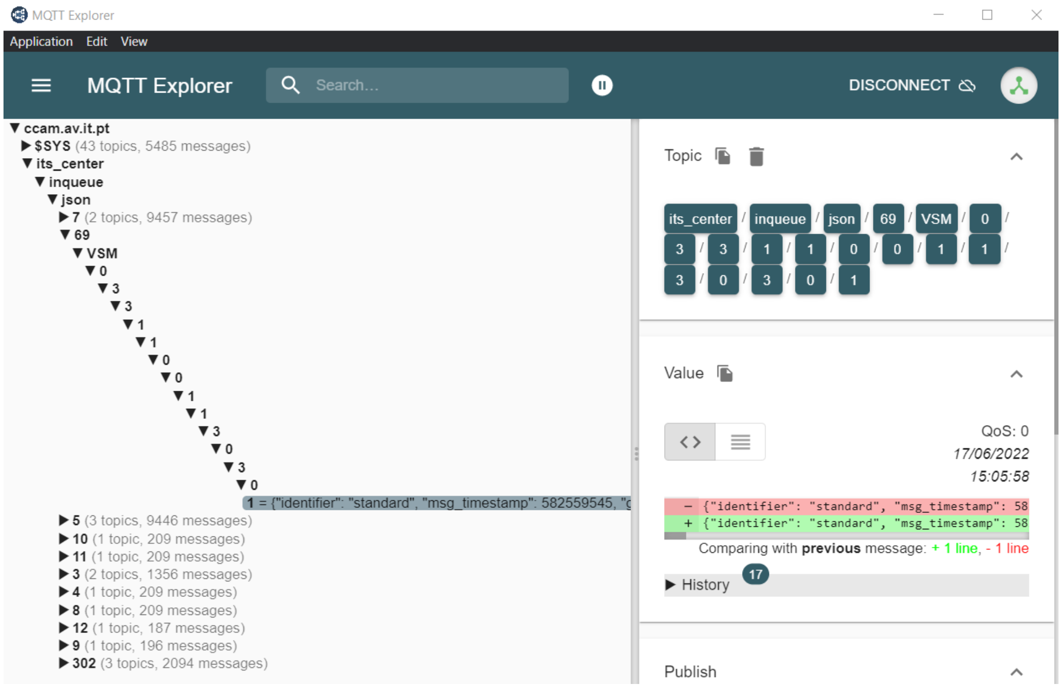


**Figure 5.16:** OBU inside the vehicle.

*MQTT Broker*

This endpoint was tested with the *MQTT Explorer* tool [114], which is comprehensive MQTT client that connects to a broker and outputs the different messages it gets to a graphic interface. Figure 5.17 shows the output of this tool, when various ITS-S stations were sending C-ITS messages, while another was publishing vehicular sensor data (seen as VSM), all according to the topic structure described in section 3.2.



```
▼ ccam.av.it.pt
  ▶ $SYS (43 topics, 5485 messages)
  ▼ its_center
    ▼ inqueue
      ▼ json
        ▶ 7 (2 topics, 9457 messages)
        ▼ 69
          ▼ VSM
            ▼ 0
              ▼ 3
                ▼ 3
                  ▼ 1
                    ▼ 1
                      ▼ 0
                        ▼ 0
                          ▼ 1
                            ▼ 1
                              ▼ 3
                                ▼ 0
                                  ▼ 3
                                    ▼ 0
                                      1 = {"identifier": "standard", "msg_timestamp": 582559545, "g
        ▶ 5 (3 topics, 9446 messages)
        ▶ 10 (1 topic, 209 messages)
        ▶ 11 (1 topic, 209 messages)
        ▶ 3 (2 topics, 1356 messages)
        ▶ 4 (1 topic, 209 messages)
        ▶ 8 (1 topic, 209 messages)
        ▶ 12 (1 topic, 187 messages)
        ▶ 9 (1 topic, 196 messages)
        ▶ 302 (3 topics, 2094 messages)
```

```
 − {"identifier": "standard", "msg_timestamp": 58
 + {"identifier": "standard", "msg_timestamp": 58
Comparing with previous message: + 1 line, - 1 line
```

**Figure 5.17:** MQTT explorer during dissemination of the vehicular sensor data to the broker.

### *API*

The data collected during trips from Aveiro to Oliveira de Azeméis was used to validate the REST API. The evolution of several parameters gathered are represented from figure 5.18 to figure 5.28 (taken from the API).

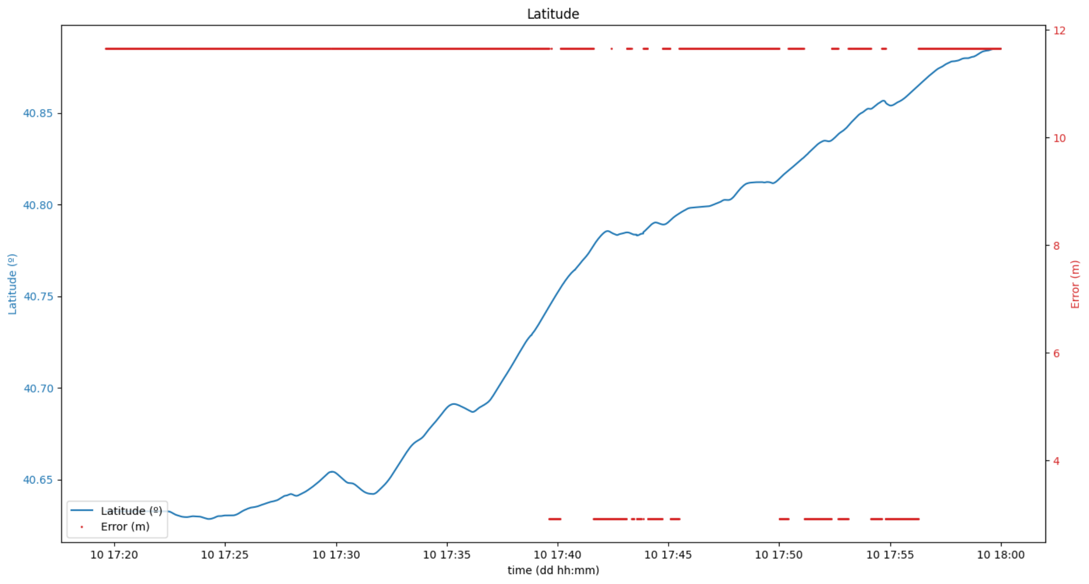


**Figure 5.18:** Latitude and respective error values.

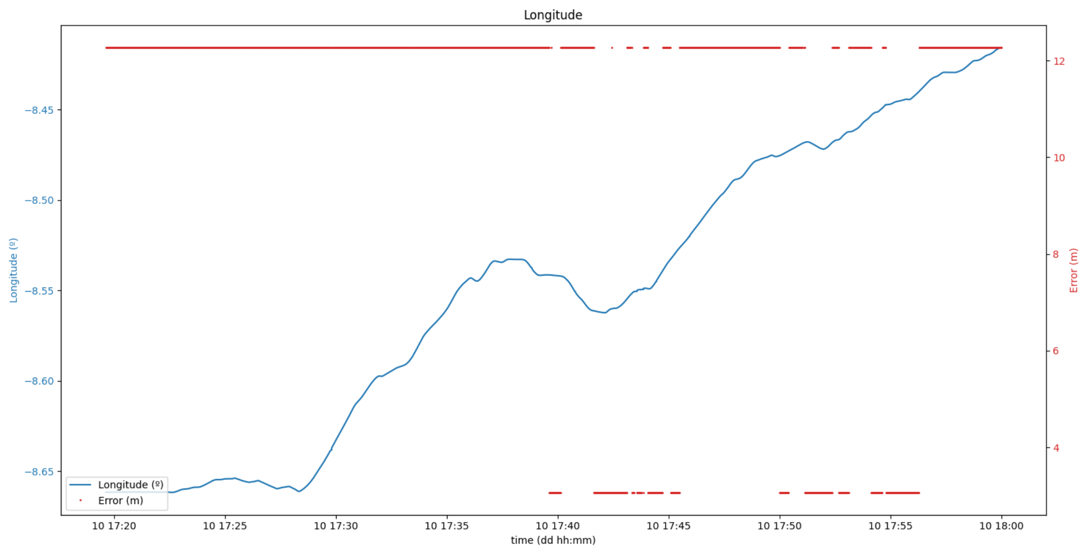


**Figure 5.19:** Longitude and respective error values.

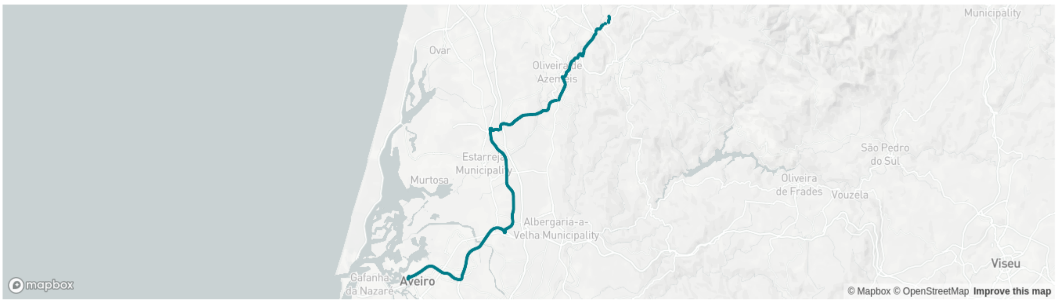


**Figure 5.20:** Representation of the trip in a map (values taken from the API).

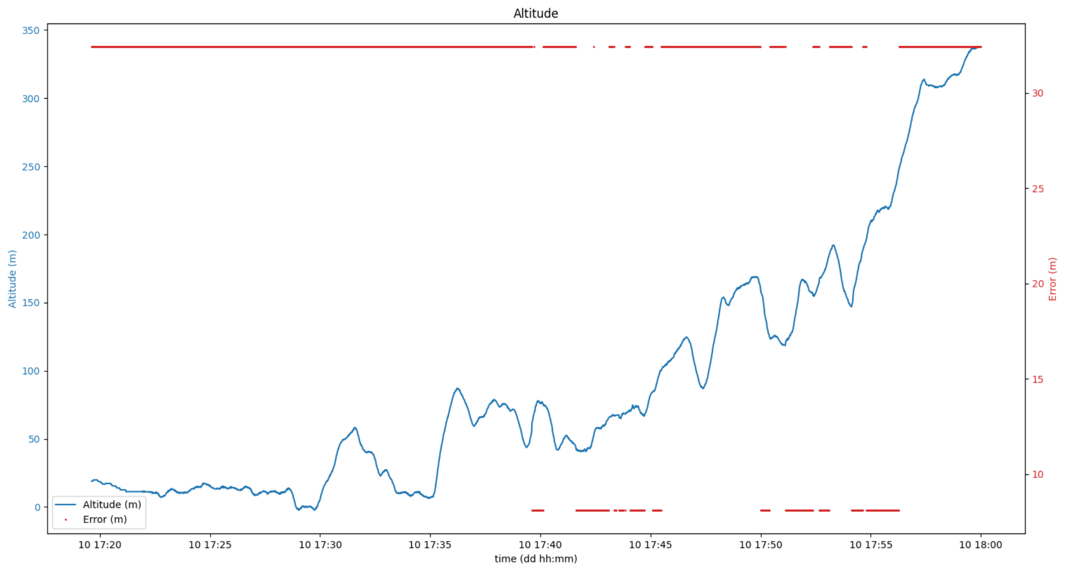


**Figure 5.21:** Altitude and respective error values.

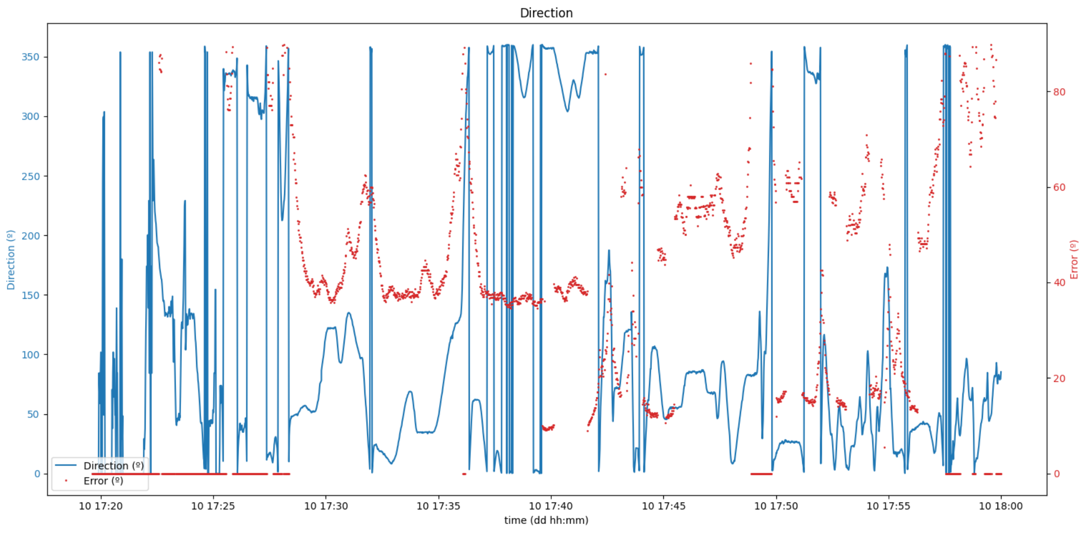


**Figure 5.22:** Direction and respective error values.

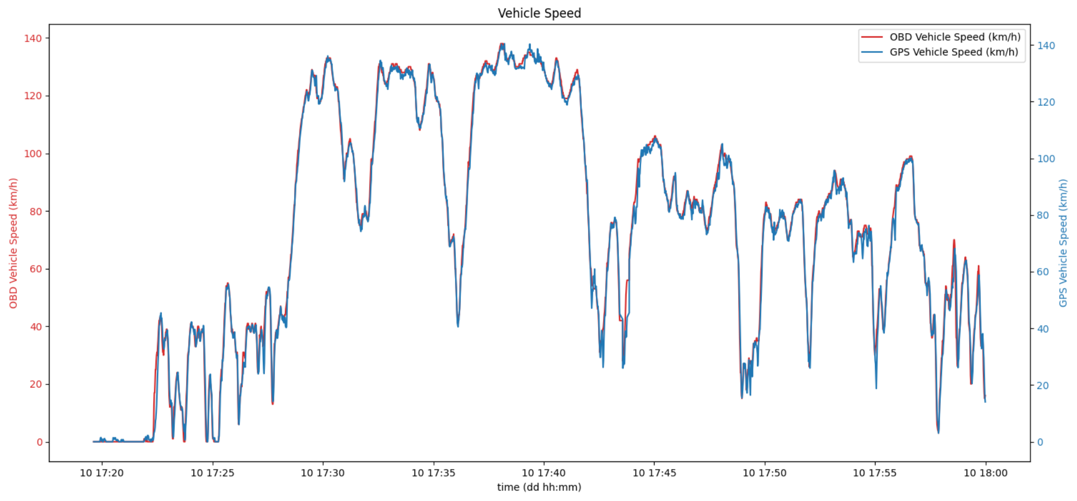


**Figure 5.23:** Vehicle speed, according to OBD (red) and GPS (red) data.

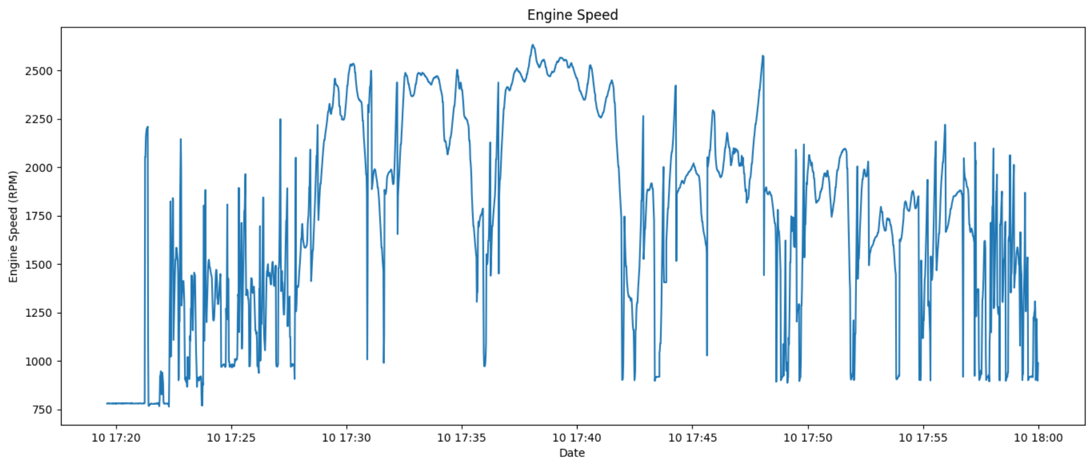


**Figure 5.24:** Engine speed values.

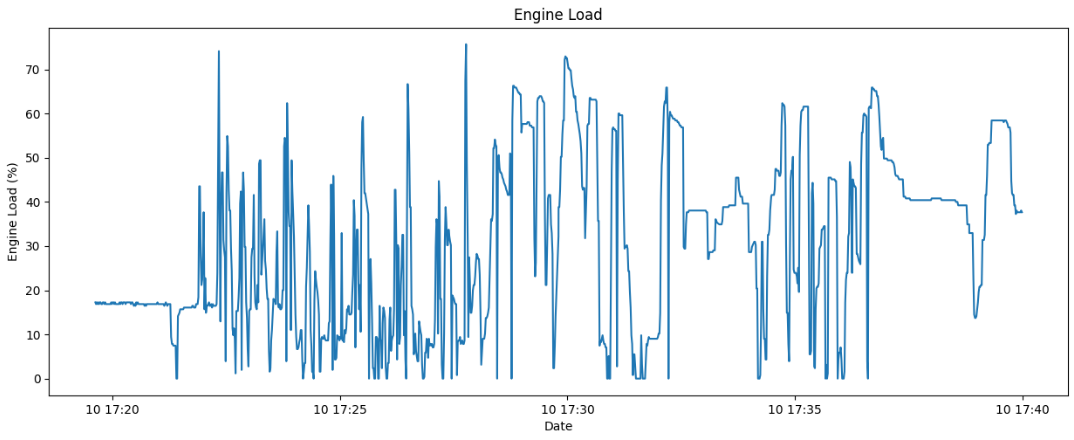


**Figure 5.25:** Engine load values.

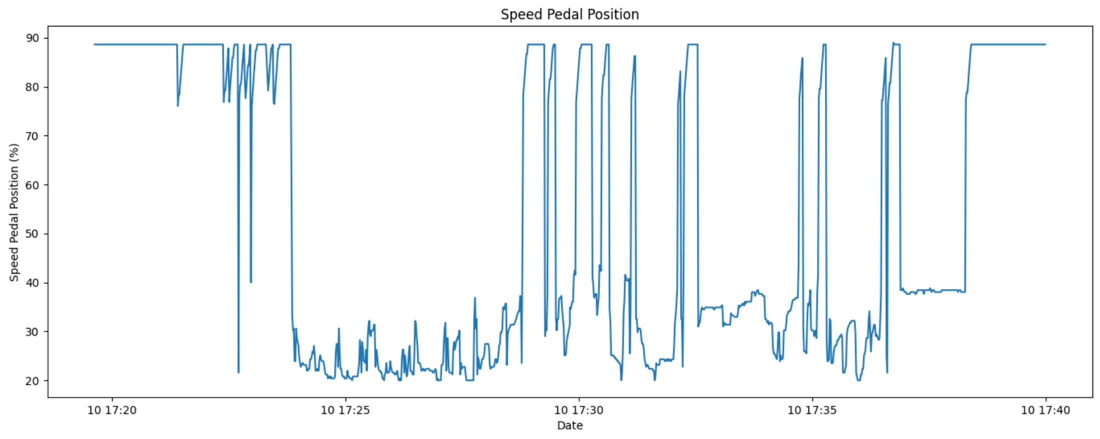


**Figure 5.26:** Throttle pedal position values.

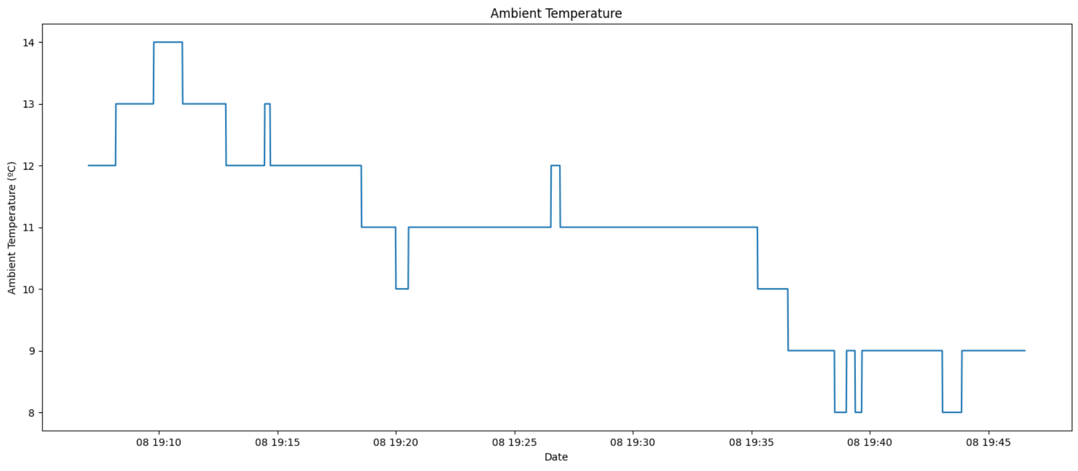


**Figure 5.27:** Ambient temperature values.

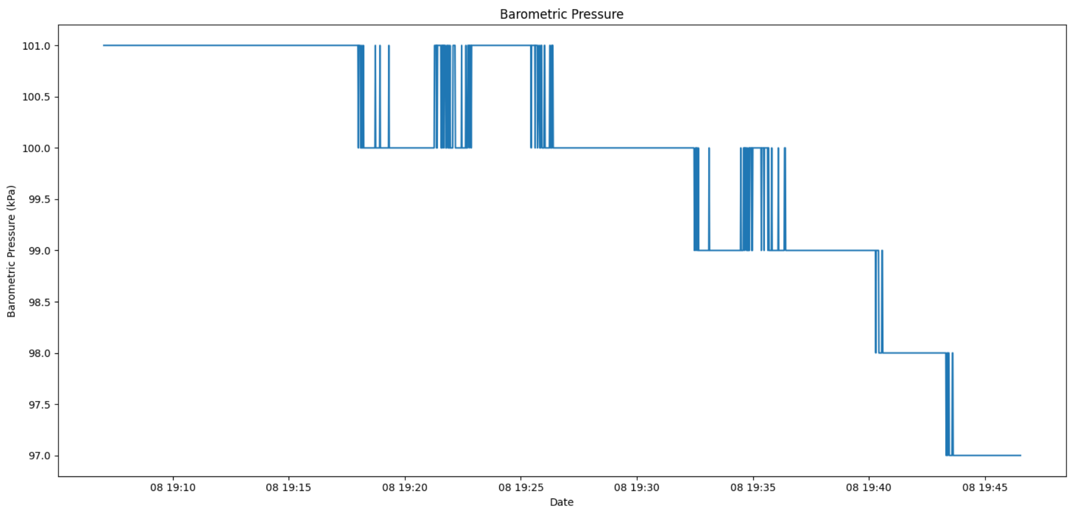


**Figure 5.28:** Barometric pressure values.

## 5.4 Logging

The logging mechanisms were validated in two different scenarios:

- **Single PLMN**: In this scenario, both the RSU and OBUs are equipped with PT SIM cards while under the full coverage of a PT network *gNodeB*. Every device is connected to the same network and to the PT MEC MQTT broker;
- **Inter PLMN**: In this scenario, OBUs are equipped with a spanish SIM card while the connected vehicles depart from Portugal and travel to Spain. The OBU is always connected to the ES MQTT broker, while performing handover from the visited network (PT) to the home network (ES).

*Single PLMN Scenario*

The flows generated by CAM and CPM messages are illustrated in figure 5.29, where the numbers represent the order. These tests were performed on the A28 highway, in Porto.

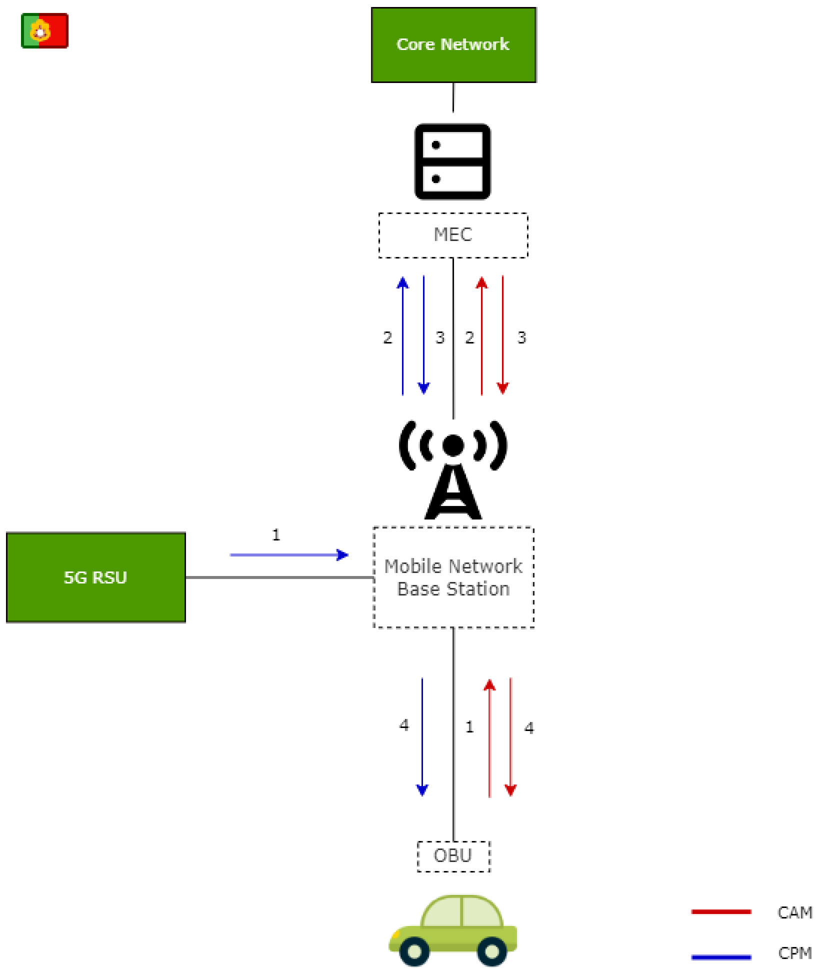


**Figure 5.29:** CPM and CAM messages flow in the single PLMN scenario.

After processing the logfiles generated by every element present in the scenario, analysis and debugging can be facilitated with the use of the scripts described in section 4.5.

Figures 5.30 and 5.31 illustrate message-by-message metrics from different abstraction layers in respect to the uplink flow direction (from the OBU/RSU to the MEC server), for both CAM and CPM messages, where the Packet Delivery Ratio (PDR) was equal to 100%.

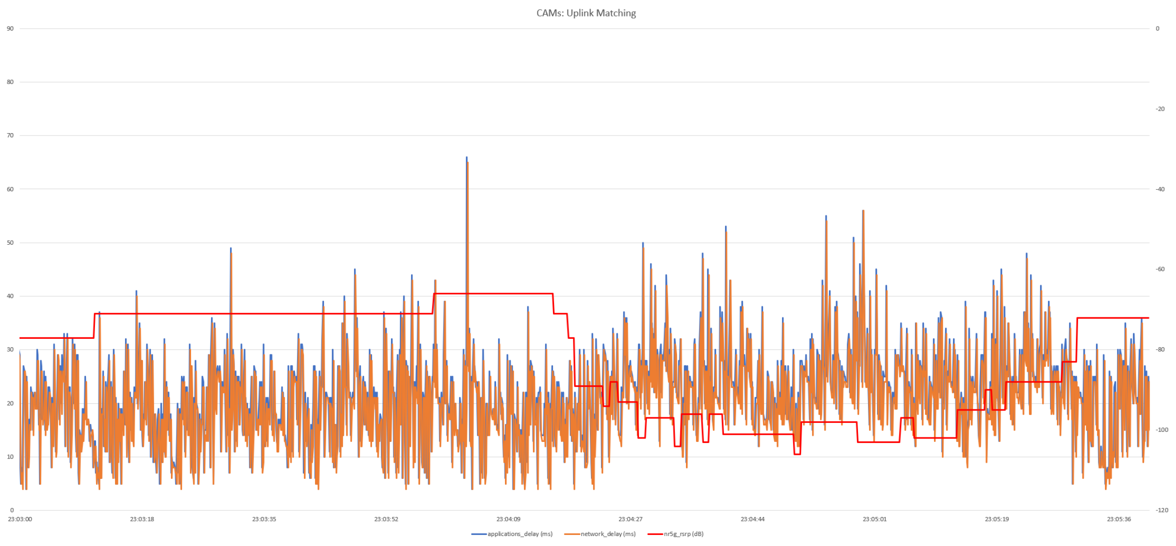


**Figure 5.30:** Single PLMN: Uplink, message-by-message, applications and network latency (blue and orange, respectively) and 5G RSRP (red) for CAM messages.

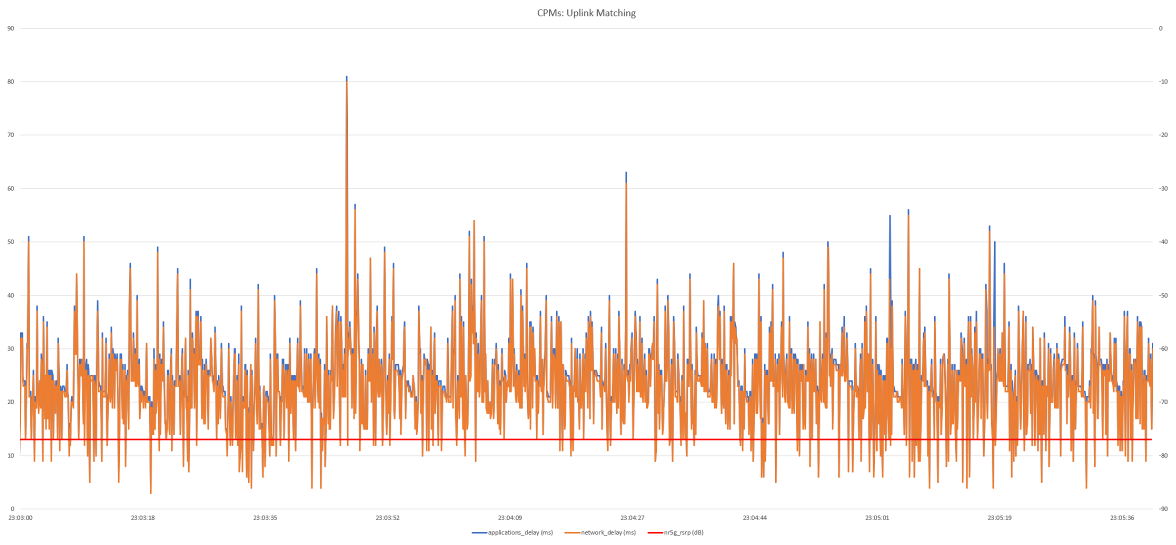


**Figure 5.31:** Single PLMN: Uplink, message-by-message, applications and network latency (blue and orange, respectively) and 5G RSRP (red) for CPM messages.

Here, the latencies from both layers are very similar, with an average difference of 2 ms, and 12 ms at most. These are almost ideal results, showing that the overhead of the applications that implement this data flow is minimal on the MEC/OBU platform CPU.

Figures 5.32 and 5.33 illustrate message-by-message metrics from different abstraction layers in respect to the downlink flow direction (from the broker in the MEC server to the OBU), for both CAM and CPM messages, where the PDR was equal to 100%.

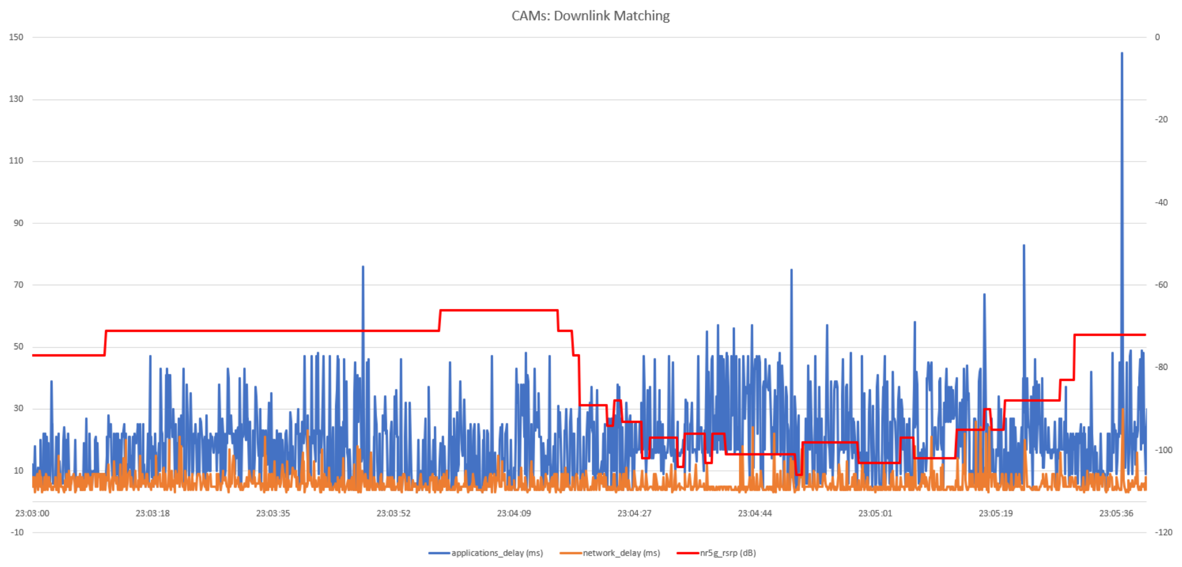


**Figure 5.32:** Single PLMN: Downlink, message-by-message, applications and network latency (blue and orange, respectively) and 5G RSRP (red) for CAM messages.

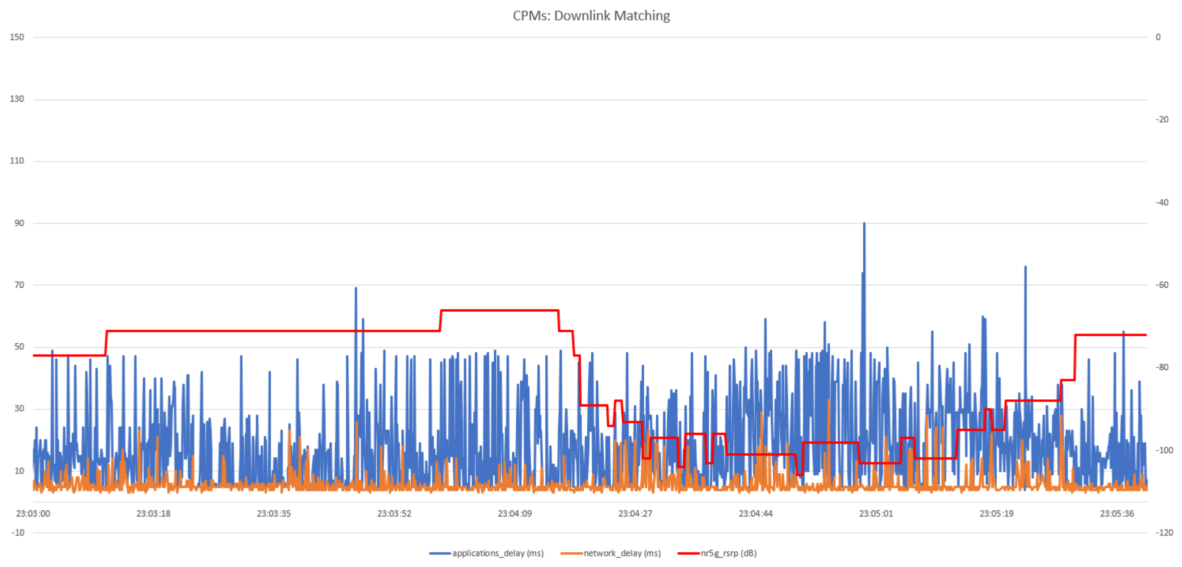


**Figure 5.33:** Single PLMN: Downlink, message-by-message, applications and network latency (blue and orange, respectively) and 5G RSRP (red) for CPM messages.

In this case, the latencies from both layers are very distinct, with an average difference of 28 ms, and 114 ms at most. This shows that the overhead of the applications that implement this data flow is very significant on the MEC/OBU platform CPU, and optimization techniques should be implemented.

Aggregating the previous downlink message-by-message approach by 1000 millisecond time windows, it's verified that on average 30 CAM and 10 CPM messages are forwarded from the broker to the OBU at every time aggregation, averaging a downlink throughput of 165 Bytes/second for CAMs and 84 Bytes/second for CPMs. Figures 5.34 and 5.35 illustrate the aggregated metrics, including the vehicle speed data.

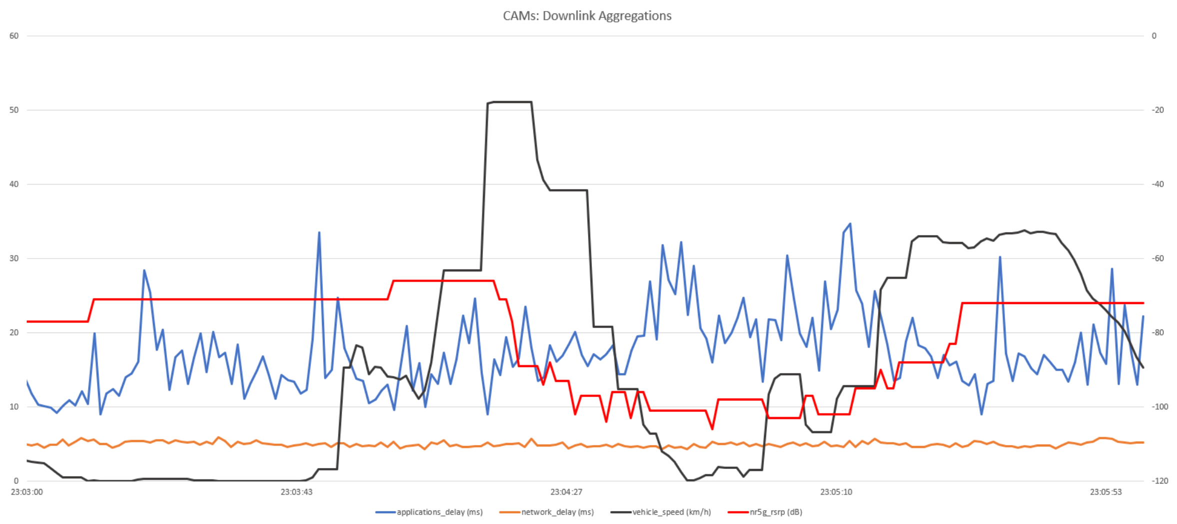


**Figure 5.34:** Single PLMN: Downlink, aggregated by 1000 miliseconds, applications and network latency (blue and orange, respectively), 5G RSRP (red), and vehicle speed (black) for CAM messages.

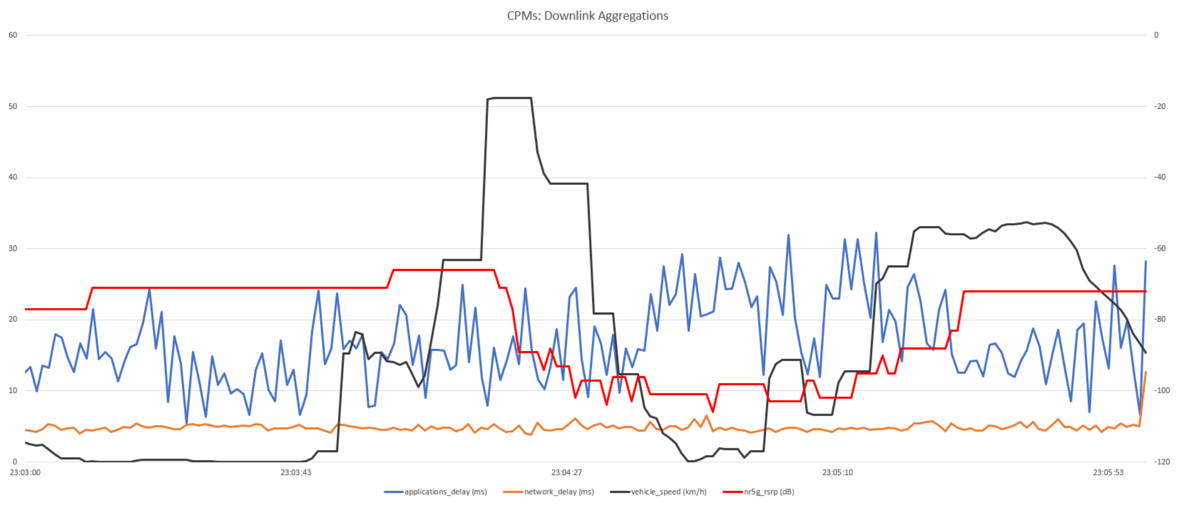


**Figure 5.35:** Single PLMN: Downlink, aggregated by 1000 miliseconds, applications and network latency (blue and orange, respectively), 5G RSRP (red), and vehicle speed (black) for CPM messages.

*Inter PLMN Scenario*

Different flows for each type of message are now defined in the presence of a handover event. These are illustrated in figures 5.36 (before the handover) and 5.37 (after the handover), where the numbers represented the order. These tests were performed at the border between Portugal and Spain, on the A3 highway.

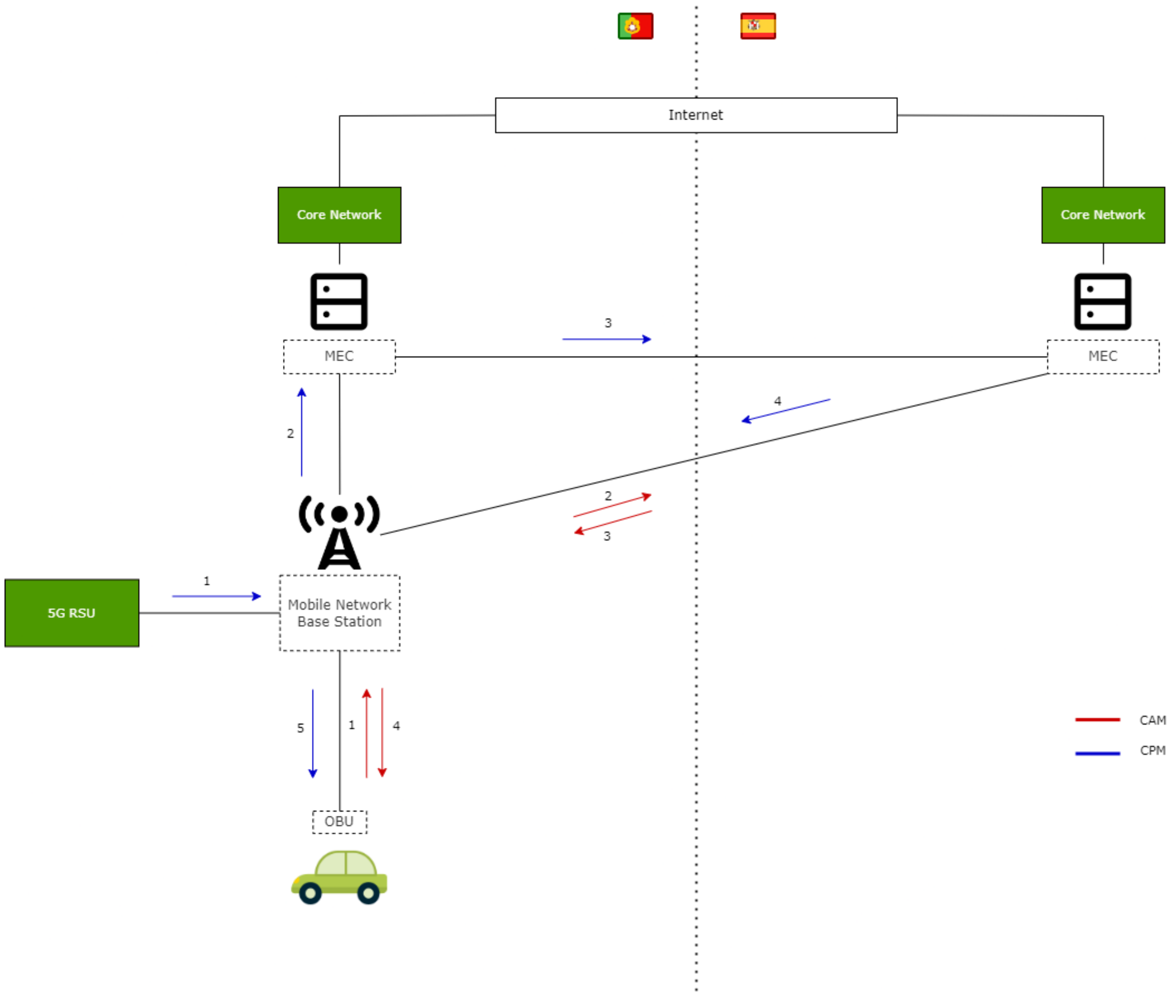


**Figure 5.36:** CPM and CAM messages flow in the inter PLMN scenario, before then handover event.

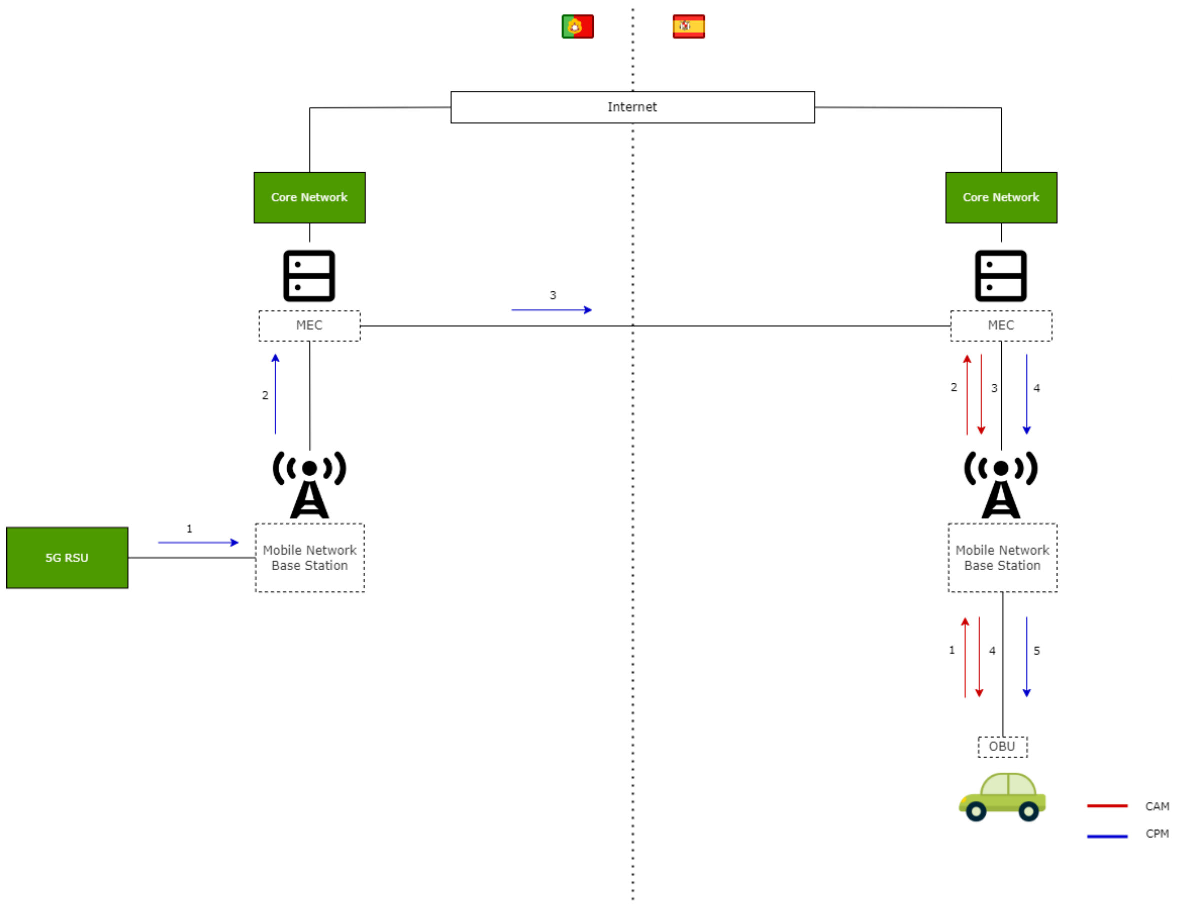


**Figure 5.37:** CPM and CAM messages flow in the inter PLMN scenario, after then handover event.

These tests were performed earlier in development, where only applications and access level logging were available, therefore no network layer latencies are provided in the results. Figures 5.38 and 5.39 illustrate the end-to-end delay application level latency and signals strength for both CAM and CPM messages, where the PDR was equal to 99.9%. The handover event is identified using the access layer logs, searching for a change in the Mobile Country Code (MCC) and Mobile Network Code (MNC) values.

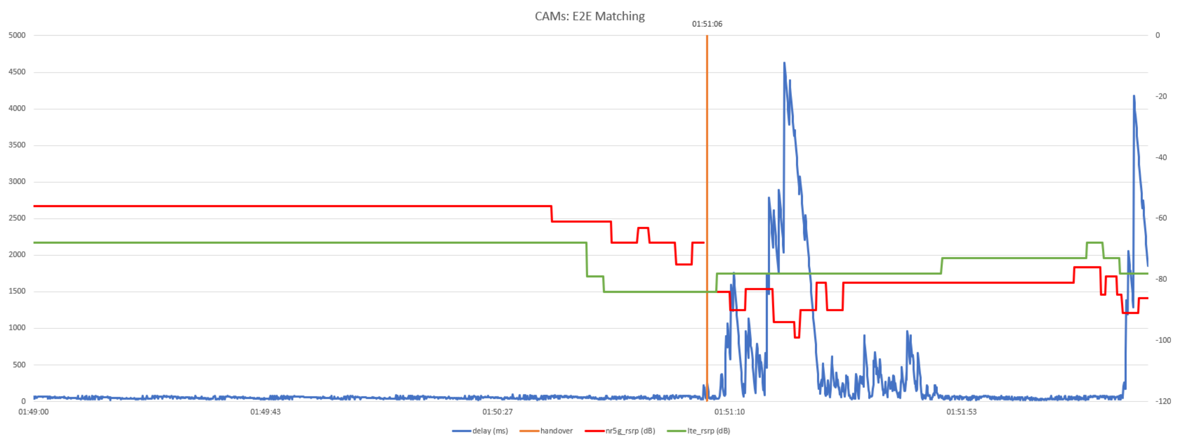


**Figure 5.38:** Inter PLMN: End-to-end, message-by-message, applications level latency (blue), 5G RSRP (red), and LTE RSRP (green) for CAM messages.

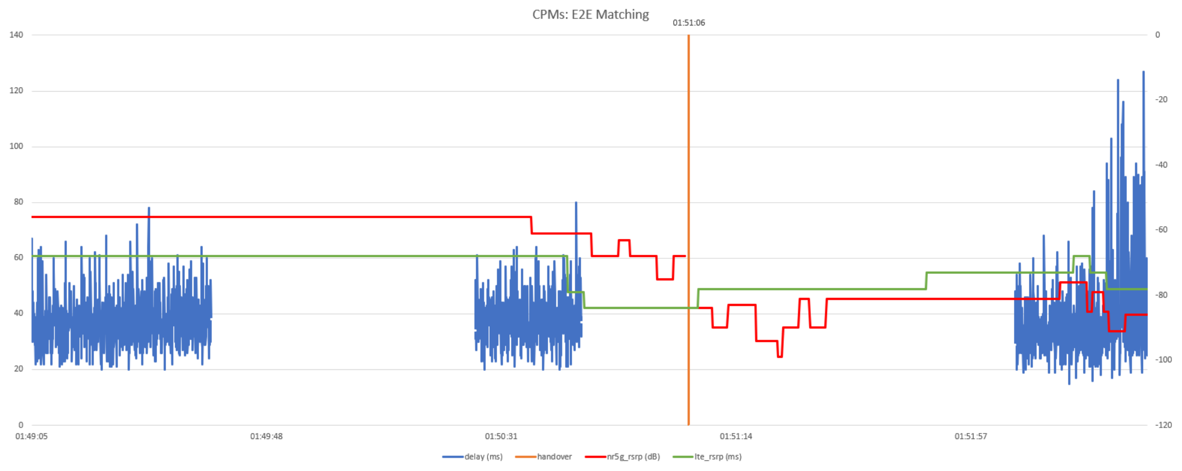


**Figure 5.39:** Inter PLMN: End-to-end, message-by-message, applications level latency (blue), 5G RSRP (red), and LTE RSRP (green) for CPM messages.

Aggregating the previous results by 1000 millisecond time windows, it's verified that on average 10 CAMs and 10 CPM messages are forwarded in an end-to-end flow direction, every time window. This results in a throughput of 74 Bytes/second for CPM and 52 Bytes/second for CAMs. Figures 5.40 and 5.41 illustrate the aggregated metrics, including the vehicle speed data.

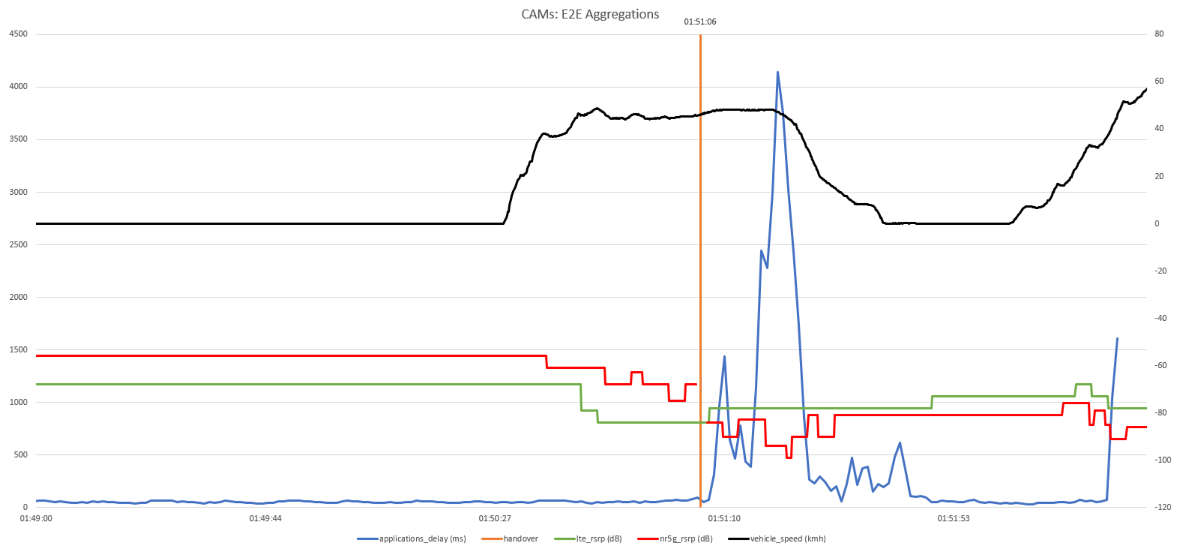


**Figure 5.40:** Inter PLMN: End-to-end, aggregated by 1000 miliseconds, applications latency values (blue) values, 5G RSRP (red), LTE RSRP (green) and vehicle speed (black) for CAM messages.

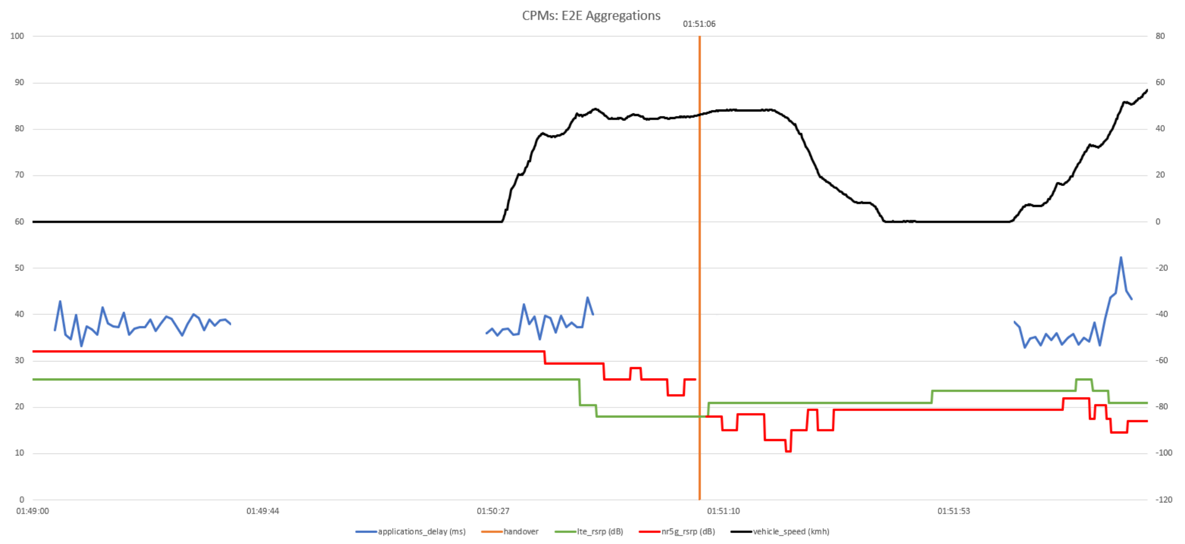


**Figure 5.41:** Inter PLMN: End-to-end, aggregated by 1000 miliseconds, applications latency values (blue) values, 5G RSRP (red), LTE RSRP (green) and vehicle speed (black) for CPM messages.

## 5.5 Results Discussion

*CAN Data Extraction*

The developed algorithm for CAN IDs identification proved to be mostly successful, providing satisfying results that seem to correctly match the data from different sources, with the exception of the engine load parameter that had the lowest pearson correlation factor, with slight deviations being apparent when comparing both graphs. Identification with algorithms previously developed also demonstrated to be very inconsistent and time demanding, with different vehicles providing very different results. Furthermore, a 100% guarantee that these values actually correspond to the correlated parameters is impossible to affirm, because of the intrinsic nature of reverse engineering. This issue could be mitigated through the addition of more external sensors to add redundancy to the data, but ultimately, to work with a ground truth for these parameters a CAN DBC would be needed, supplied directly by manufacturers, which they currently are very reticent to provide because of the lack of authentication or encryption in the bus, to not encourage car hacking. Another realistic solution would be the natural evolution of the CAN protocol, that could provide authentication mechanisms and standardized CAN IDs, or through the OBD protocol implementing more PIDs.

Without a ground truth list of CAN IDs, identification through reverse-engineering proves to be unreliable, time consuming and unusable to achieve absolute certainty. Nonetheless, the usefulness of this data in the context of road security is undeniable, and manufacturers/standardization organizations should be encouraged by proof-of-concept works like this, in order to evolve the protocols and standards used in vehicular networks, to provide easy and safe ways to access relevant parameters that can be utilized to form a better road environment.

*Vehicular Sensor Data*

Even just with data collected through the standardized OBD-II PIDs, the value of extracting sensor data from a connected car is clear.

Meteorological data such as temperature and barometric pressure could be used to improve the current RWM based on stationary RWS by using the inherent mobility of the vehicles, enabling the identification of areas where ice is more probable to be form (through temperature values), as well as zones where forest fires could be more prone to occur (many researches have been done correlating differences in atmospheric pressure to the flammability of different materials [115]).

This data can also be valuable to other drivers and connected cars through the broker or ITS-G5, in order to detect anomalies in the traffic such as congestions, dangerous drivers, mismatches between the vehicles speed and their wheels speed (that could mean hydroplaning) or inadequate driving that causes high engine load values, directly influencing fuel consumption. This enables the notification in near real-time about important and dangerous events, consequently improving the road environment.

Furthermore, this data could also be used by authorities, to identify zones more sensible to traffic congestion, accidents and where drivers don't respect the speed limits, indicating the need of reinforcing the security mechanisms in these zones. Additionally, statistical data about road population is very valuable to marketing companies and can be used as selling points to different services.

Finally, the framework developed allows to easily scale this and add more parameters through external sensors, adding the possibility of collecting other valuable data such as luminosity, humidity, noise levels and air quality.

*Logging*

The logging mechanism proves to be vital in the context of C-ITS development, enabling the debugging of the different stations through the interpretation of data from the different abstraction layers.

It allows to study and analyse the different cellular technologies in the context of C-ITS and CCAM, to verify timing restrictions and to test the behaviour of the network in different conditions, with and without the influence of events such as handovers or traffic congestion.

The results themselves demonstrate the 5G technology capabilities, that provide low network delays that satisfy the CCAM timing requirements, at least in a single PLMN scenario. More testing is necessary in the multi PLMN scenario that includes network level logging, to understand if it the high delays are related to the handover event or to the applications overhead.

In spite of the tests being based on use cases not directly related to vehicular sensor data dissemination, they were modeled using the same standardized messages used in a C-ITS environment that can also benefit from the sensor data to include in their body. Therefore, the logging methods can be directly imported in order to fit other related use cases.

CHAPTER 6

# Conclusion and Future Work

## 6.1 Conclusion

This work studies the identification and extraction of raw CAN and OBD-II messages transmitted within the vehicle communication networks. A solution to enrich the current RWM and the ITS environment was designed, implemented and tested, consisting of a device to be connected to the OBD-II port of the vehicles, an OBU and various cloud services.

This work was developed due to the incentive of providing a more secure road environment, the primary objective being the implementation of a reliable vehicular sensor extraction framework, capable of disseminating valuable data to relevant stakeholders, with appropriate methods to assert its requirements. Several steps were defined to achieve this objective, starting from the study of the concepts involved with this dissertation, a review of the state-of-the-art, the design of the system architecture, and finally its implementation and testing.

The results demonstrate the limitations of the reverse-engineering methods to acquire parameters from the CAN bus without ground truth bases. Nonetheless, the value and importance of vehicular sensor data is made clear in the result analysis that specify how the ITS environment benefits from the sensor data collection and dissemination to other vehicles and the cloud.

It is concluded that a well founded and useful proof-of-work is provided, that extracts and disseminates vehicular sensor data, while also including logging methods capable of asserting the restrictions set up by the standards, bringing a contribute to the area [116].

## 6.2 Future Work

During the development of this work, improvements to the overall architecture of the system and implementation were identified, to improve future increments on this work. These include acquiring a CAN DBC for a vehicle in order to work with a ground truth set of parameters, adding more external sensors to the OBU in order to add redundancy and to enrich the data collected, and most important, the development of a mobile application for drivers, that condensates the relevant sensor data and notifies users about important events.

# Bibliography

[1] European Automobile Manufacturers Association, “Vehicles in use Europe 2022,” ACEA, European Union, 2022, p. 21. [Online]. Available: https://www.acea.auto/files/ACEA-report-vehicles-in-use-europe-2022.pdf.

[2] ——, “The Automobile Industry Pocket Guide,” ACEA, 2020, p. 88. [Online]. Available: https://www.acea.be/uploads/publications/ACEA_Pocket_Guide_2020-2021.pdf.

[3] Eurostat, “Transport statistics at regional level,” European Union, 2020, p. 17. [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Transport_statistics_at_regional_level#Road_transport_and_accidents.

[4] W. H. Organization, “Global status report on road safety,” World Health Organization, 2018, p. 403. [Online]. Available: https://apps.who.int/iris/bitstream/handle/10665/276462/9789241565684-eng.pdf.

[5] A. Christodoulou, P. Christidis, and J. R. Centre, “Measuring congestion in European cities,” 2020, p. 31. [Online]. Available: https://op.europa.eu/en/publication-detail/-/publication/e590d815-9b08-11ea-9d2d-01aa75ed71a1/language-en/format-PDF/source-178542437.

[6] A. Kaźmierczak, A. E. Envirnment, Mitigation, E. T. C. f. A. P. Change, and Climate, “Unequal exposure and unequal impacts: social vulnerability to air pollution, noise and extreme temperatures in Europe,” 2019, p. 97. DOI: 10.2800/324183. [Online]. Available: https://www.eea.europa.eu/publications/unequal-exposure-and-unequal-impacts.

[7] *Obd-ii background*, Accessed: January 2020. [Online]. Available: http://www.obdii.com/background.html.

[8] E. Comission, “A European strategy on Cooperative Intelligent Transport Systems, a milestone towards cooperative, connected and automated mobility,” 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%5C%3A52016DC0766.

[9] *5g-mobix*, https://www.5g-mobix.com/, Accessed: 2022-06-14.

[10] R. B. GmbH, “Can specification, version 2.0,” 1991.

[11] *Canbus: The central networking system of vehicles*, Accessed: December 2019, Jun. 2019. [Online]. Available: https://premioinc.com/blogs/blog/can-bus-the-central-networking-system-of-vehicles.

[12] *Controller area network (can bus) protocol*, Accessed: May 2022, 2022. [Online]. Available: https://www.kvaser.com/can-protocol-tutorial/.

[13] *Can bus explained - a simple intro [2022]*, Accessed: May 2022, 2022. [Online]. Available: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial.

[14] “Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling,” International Organization for Standardization, Standard, Dec. 2015.

[15] “Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit,” International Organization for Standardization, Standard, Dec. 2016.

[16] “Road vehicles — Controller area network (CAN) — Part 3: Low-speed, fault-tolerant, medium-dependent interface,” International Organization for Standardization, Standard, Jun. 2016.

[17] *Understanding microchip's can module bit timing*, Accessed: May 2022, 2001. [Online]. Available: https://ww1.microchip.com/downloads/en/Appnotes/00754.pdf.

[18] *Controller area network physical layer requirements*, Accessed: May 2022, Jan. 2008. [Online]. Available: https://www.ti.com/lit/an/slla270/slla270.pdf?ts=1653330451251.

[19] *Can physical layer and termination guide*, Accessed: May 2022, 2020. [Online]. Available: https://www.ni.com/pt-pt/innovations/white-papers/09/can-physical-layer-and-termination-guide.html.

[20] *Introduction to the controller area network (can)*, Accessed: May 2022, 2016. [Online]. Available: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf.

[21] I. Jennions, "Evaluation of can bus security challenges," *Sensors (Basel)*, Apr. 20, 2020.

[22] *Can fd explained - a simple intro (2020)*, Accessed: November 2020. [Online]. Available: https://www.csselectronics.com/screen/page/can-fd-flexible-data-rate-intro/language/en.

[23] *Can fd - the basic idea*, Accessed: November 2020. [Online]. Available: https://www.can-cia.org/can-knowledge/can/can-fd/.

[24] *Controller area network (can) and flexible data-rate (can fd)*, Accessed: November 2020. [Online]. Available: https://www.kvaser.com/about-can/can-fd/.

[25] *Can xl: Next step in can evolution*, Accessed: November 2020. [Online]. Available: https://www.bosch-semiconductors.com/news/t-newsdetailpage-4.html.

[26] *Can xl: Bridging the bitrate gap between can fd and ethernet*, Accessed: November 2020. [Online]. Available: https://www.kvaser.com/can-xl-bridging-the-bitrate-gap-between-can-fd-and-ethernet.

[27] *Introduction to the lin bus*, Accessed: May 2022. [Online]. Available: https://www.kvaser.com/about-can/can-standards/linbus/.

[28] *Lin bus explained - a simple intro [2022]*, Accessed: May 2022, 2022. [Online]. Available: https://www.csselectronics.com/pages/lin-bus-protocol-intro-basics.

[29] *Local interconnect network*, Accessed: May 2022, 2014. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/local-interconnect-network.

[30] *What is obd? understanding on-board diagnostics | noregon*, Accessed: May 2022. [Online]. Available: https://www.noregon.com/what-is-obd/.

[31] ". Miller", *Obd1 vs obd2: What nobody told you before*, Accessed: May 2022, Nov. 2021. [Online]. Available: https://www.obdadvisor.com/difference-obd1-obd2-scanners/.

[32] "Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics — Part 3: Diagnostic connector and related electrical circuits: Specification and use," International Organization for Standardization, Standard, Apr. 2016.

[33] "Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics — Part 5: Emissions-related diagnostic services," International Organization for Standardization, Standard, Aug. 2015.

[34] *Obd2 explained - a simple intro (2020)*, Accessed: October 2020, 2020. [Online]. Available: https://www.csselectronics.com/screen/page/simple-intro-obd2-explained/.

[35] Z. L. Haohuang Wen Qi Alfred Chen, "Plug-n-pwned: Comprehensive vulnerability analysis of obd-ii dongles as a new over-the-air attack surface in automotive iot," Oct. 4, 2021.

[36] *Bluetooth vs. bluetooth low energy: What's the difference?* Accessed: May 2022, 2021. [Online]. Available: https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy.

[37] *Bluetooth vs. bluetooth low energy: What's the difference?* Accessed: May 2022, Mar. 2019. [Online]. Available: https://www.centare.com/blog/what_is_bluetooth_low_energy/.

[38] "Bluetooth Core Specification Version 5.3," SIG Bluetooth, Tech. Rep., Jul. 2022, Accessed: July 2020.

[39] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[40] *Introduction to bluetooth low energy (ble) | argenox*, Accessed: May 2022. [Online]. Available: https://www.argenox.com/library/bluetooth-low-energy/introduction-to-bluetooth-low-energy-v4-0/.

[41] *Bluetooth® low energy (ble) physical layer*, Accessed: May 2022. [Online]. Available: https://microchipdeveloper.com/wireless:ble-phy-layer.

[42] *Ble link layer roles and states*, Accessed: May 2022, 2021. [Online]. Available: https://microchipdeveloper.com/wireless:ble-link-layer-roles-states.

[43] *Bluetooth® low energy packet types*, Accessed: May 2022, 2021. [Online]. Available: https://microchipdeveloper.com/wireless:ble-link-layer-packet-types.

[44] *Ble protocol stack — host controller interface (hci)*, Accessed: May 2022, Sep. 2019. [Online]. Available: https://medium.com/@pcng/ble-protocol-stack-host-controller-interface-hci-44dd5697bd8.

[45] *Logical link control and adaptation layer protocol (l2cap)*, Accessed: May 2022. [Online]. Available: https://software-dl.ti.com/lprf/simplelink_cc2640r2_latest/docs/blestack/ble_user_guide/html/ble-stack-3.x/l2cap.html.

[46] *What is ble attribute protocol (att)?* Accessed: May 2022. [Online]. Available: https://www.carrentalgateway.com/glossary/attribute-protocol/.

[47] *Gap | introduction to bluetooth low energy*, Accessed: May 2022. [Online]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap.

[48] *Gatt | introduction to bluetooth low energy*, Accessed: May 2022. [Online]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

[49] M. Cäsar, T. Pawelke, J. Steffan, and G. Terhorst, "A survey on bluetooth low energy security and privacy," *Computer Networks*, Mar. 14, 2022.

[50] W. S. Jeon, M. H. Dwijaksara, and D. G. Jeong, "Performance analysis of neighbor discovery process in bluetooth low-energy networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 2, pp. 1865–1871, 2017.

[51] W. Bronzi, R. Frank, G. Castignani, and T. Engel, "Bluetooth low energy performance and robustness analysis for inter-vehicular communications," *Ad Hoc Networks*, vol. 37, pp. 76–86, 2016.

[52] J. Wannstrom, *High Speed Packet data Access.* [Online]. Available: https://www.3gpp.org/technologies/keywords-acronyms/99-hspa (visited on 04/05/2021).

[53] 3GPP, "Security architecture and procedures for 5G System," 3rd Generation Partnership Project (3GPP), TS 33.501, 2020. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169.

[54] ——, "Requirements for Evolved UTRA (E-UTRA) and Evolved UTRAN (E-UTRAN)," 3rd Generation Partnership Project (3GPP), TR 25.913, 2006. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/25913.htm.

[55] ETSI, *4G - Long Term Evolution | LTE Standards.* [Online]. Available: https://www.etsi.org/technologies/mobile/4G (visited on 04/05/2021).

[56] C. Cox, *An Introduction to LTE: LTE, LTE-Advanced, SAE, VoLTE and 4G Mobile Communications.* Chennai: John Wiley & Sons Ltd, 2014, p. 488. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?bknumber=8039851.

[57] M. Nohrborg, *LTE Overview.* [Online]. Available: https://www.3gpp.org/technologies/keywords-acronyms/98-lte (visited on 04/05/2021).

[58] A. Ghosh, J. Zhang, J. G. Andrews, and R. Muhamed, *Fundamentals of LTE*, 1st. USA: Prentice Hall Press, 2010, ISBN: 0137033117. [Online]. Available: https://dl.acm.org/doi/book/10.5555/1941366.

[59] 3GPP, *ITU-R Confers IMT-Advanced (4G) Status to 3GPP LTE*, 2010. [Online]. Available: https://www.3gpp.org/news-events/1319-ITU-R-Confers-IMT-Advanced-4G-Status-to-3GPP-LTE (visited on 04/05/2021).

[60] ——, "Release description; Release 15," 3rd Generation Partnership Project (3GPP), TR 21.915, 2018. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/21915.htm.

[61] Accessed: May 2022. [Online]. Available: https://www.3gpp.org/release-16.

[62] Accessed: May 2022. [Online]. Available: https://www.3gpp.org/release-17.

[63] Accessed: May 2022. [Online]. Available: https://www.3gpp.org/release18.

[64] *Mqtt version 5.0*, Mar. 7, 2019.

[65] M. Pawar, *Mqtt protocol overview - everything you need to know*, Accessed: May 2022. [Online]. Available: https://www.velotio.com/engineering-blog/mqtt-protocol-overview.

[66] ETSI, "EN 302 665 - V1.1.1 - Intelligent Transport Systems (ITS); Communications Architecture," Tech. Rep., 2010, p. 44.

[67] ——, "EN 302 636-3 - V1.2.1 - Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 3: Network Architecture," Tech. Rep., 2014.

[68] ——, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions," Tech. Rep., 2009.

[69] ——, "EN 302 637-2 - V1.4.1 - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service," Tech. Rep., 2019.

[70] ——, "EN 302 637-3 - V1.2.1 - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service," Tech. Rep., 2014.

[71] ——, "TR 103 562 - V2.1.1 - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Analysis of the Collective Perception Service (CPS); Release 2," Tech. Rep., 2019.

[72] ——, "EN 302 663 - V1.3.1 - Intelligent Transport Systems (ITS); ITS-G5 Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band," Tech. Rep., 2019.

[73] D. Jiang and L. Delgrossi, "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments," in *VTC Spring 2008 - IEEE Vehicular Technology Conference*, 2008, pp. 2036–2040. DOI: 10.1109/VETECS.2008.458.

[74] IEEE, "IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: High Speed Physical Layer in the 5 GHz band," *IEEE Std 802.11a-1999*, pp. 1–102, 1999. DOI: 10.1109/IEEESTD.1999.90606.

[75] ——, "IEEE Standard for Information technology–Local and metropolitan area networks–Specific requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 8: Medium Access Control (MAC) Quality of Service," *IEEE Std 802.11e-2005 (Amendment to IEEE Std 802.11, 1999 Edition (Reaff 2003)*, pp. 1–212, 2005. DOI: 10.1109/IEEESTD.2005.97890.

[76] J. Almeida, J. Ferreira, and A. Oliveira, "Development of an ITS-G5 station, from the physical to the MAC layer," in *Intelligent Transportation Systems From Good Practices to Standards*, 2016, ISBN: 978-1-4987-2186-8. [Online]. Available: http://hdl.handle.net/10773/21887.

[77] H. M. Song and H. K. Kim, "Discovering can specification using on-board diagnostics," *IEEE Design & Test*, 2020.

[78] D. Frassinelli, S. Park, and S. Nürnberger, "I Know Where You Parked Last Summer : Automated Reverse Engineering and Privacy Analysis of Modern Cars," *IEEE Xplore*, 2020. DOI: 10.1109/SP40000.2020.00081. [Online]. Available: https://ieeexplore.ieee.org/document/9152789.

[79] A. Buscemi, I. Turcanu, G. Castignani, R. Crunelle, and T. Engel, “CANMatch: A Fully Automated Tool for CAN Bus Reverse Engineering Based on Frame Matching,” *IEEE Xplore*, pp. 12 358–12 373, 2021. [Online]. Available: `https://ieeexplore.ieee.org/document/9599500`.

[80] R. Correia, “Cooperative in-vehicle sensing of adverse road-weather conditions,” M.S. thesis, Universidade de Aveiro, 2021.

[81] J. Guerrero-Ibáñez, S. Zeadally, and J. Contreras-Castillo, “Sensor Technologies for Intelligent Transportation Systems,” *sensors*, 2018. DOI: `10.3390/s18041212`. [Online]. Available: `https://www.mdpi.com/1424-8220/18/4/1212`.

[82] T. Bogaerts, S. Watelet, C. Thoen, *et al.*, “Enhancement of road weather services using vehicle sensor data,” *IEEE Xplore*, 2022. DOI: `10.1109/CCNC49033.2022.9700658`. [Online]. Available: `https://ieeexplore.ieee.org/document/9700658`.

[83] T. Bogaerts, S. Watelet, N. De Bruyne, *et al.*, “Leveraging Artificial Intelligence and Fleet Sensor Data towards a Higher Resolution Road Weather Model,” *sensors*, 2022. DOI: `10.3390/s22072732`. [Online]. Available: `https://www.mdpi.com/1424-8220/22/7/2732`.

[84] G. Velez, E. Bonetto, D. Brevi, *et al.*, “5g features and standards for vehicle data exploitation,” 2022. [Online]. Available: `https://www.researchgate.net/publication/359936950_5G_Features_and_Standards_for_Vehicle_Data_Exploitation`.

[85] ETSI, “ETSI TS 133 501 V15.4.0 - 5G; Security architecture and procedures for 5G System (3GPP TS 33.501 version 15.4.0 Release 15),” 2019.

[86] A. L. Marques, “Device magement in the scot platform,” M.S. thesis, Universidade de Aveiro, 2016.

[87] labapart, *Gattlib*, `https://github.com/labapart/gattlib`, 2022.

[88] bluez, *Bluez*, `https://github.com/bluez/bluez`, 2022.

[89] GNOME, *Glib*, `https://gitlab.gnome.org/GNOME/glib`, 2022.

[90] *Mobile Broadband Interface Model v1.0.* [Online]. Available: `https://www.usb.org/document-library/mobile-broadband-interface-model-v10-errata-1-and-adopters-agreement` (visited on 09/28/2021).

[91] *MBIM-NETWORK man page.* [Online]. Available: `https://www.freedesktop.org/software/libmbim/man/latest/mbim-network.1.html` (visited on 06/11/2022).

[92] *MBIMCLI man page.* [Online]. Available: `https://www.freedesktop.org/software/libmbim/man/latest/mbimcli.1.html` (visited on 06/11/2022).

[93] M. D. Correia, “Roadside infrastructure to support cooperative intelligent transportation systems,” M.S. thesis, Universidade de Aveiro, 2021.

[94] E. Vieira, J. Almeida, J. Ferreira, T. Dias, J. Ribeiro, and L. Moura, “Roadside and Cloud Architecture for C-ITS Services,” *The Intitution of Engineering and Technology*, 2015.

[95] archlinux, *Linux*, `https://github.com/archlinux/linux/tree/v5.13.13-arch1`, 2022.

[96] *SVN man page.* [Online]. Available: `https://linux.die.net/man/1/svn` (visited on 06/11/2022).

[97] *makepkg.* [Online]. Available: `https://archlinux.org/pacman/makepkg.8.html` (visited on 06/11/2022).

[98] *mkinitcpio.* [Online]. Available: `https://man.archlinux.org/man/mkinitcpio.8` (visited on 06/11/2022).

[99] *GNU GRUB - GNU Project.* [Online]. Available: `https://www.gnu.org/software/grub/` (visited on 06/11/2022).

[100] systemd, *Systemd*, `https://github.com/systemd/systemd`, 2022.

[101] gpsd, *Gpsd*, `https://gitlab.com/gpsd/gpsd`, 2022.

[102] eclipse, *Mosquitto*, `https://github.com/eclipse/mosquitto`, 2022.

[103] curl, *Curl*, `https://github.com/curl/curl`, 2022.

[104] certbot, *Certbot*, `https://github.com/certbot/certbot`, 2022.

[105] openssl, *Openssl*, `https://github.com/openssl/openssl`, 2022.

[106] *Pasmo api*, `https://pasmo.es.av.it.pt/docs/`, 2022.

[107] postgres, *Postgres*, `https://github.com/postgres/postgres`, 2022.

[108] pallets, *Flask*, `https://github.com/pallets/flask`, 2022.

[109] nginx, *Nginx*, `https://github.com/nginx/nginx`, 2022.

[110] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[111] "X.680 : Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation," International Telecommunications Union, Standard, Feb. 2021.

[112] wireshark, *Wireshark*, `https://github.com/wireshark/wireshark`, 2022.

[113] *Canbus obd ecu simulator mobydic1610*, Accessed: June 2022. [Online]. Available: `https://www.ozenelektronik.com/canbus-obd-ecu-simulator-p.html`.

[114] thomasnordquist, *Mqtt-explorer*, `https://github.com/thomasnordquist/MQTT-Explorer`, 2022.

[115] C. Kuang, L. Yuanzhou, S. Zhu, and J. Li, "Influence of different low air pressure on combustion characteristics of ethanol pool fires," *ScienceDirect*, 2013.

[116] S. Hosseini, M. Jooriah, D. Rocha, *et al.*, "CCAM Service Continuity in a Cross-Border MEC Federation Scenario," *Frontiers*, 2022.