TÉCNICO
LISBOA

TagusVanet

A low-cost, integrated framework for development, validation and deployment of VANETs

João Manuel Ferreira Duarte Gomes

Dissertation submitted to obtain the Master Degree in

Telecommunications and Informatics Engineering

Supervisor: Prof. Teresa Maria Sá Ferreira Vazão Vasques

June 2014

Acknowledgments

First I would like to thank my supervisors, Professor Teresa Vazão, for her unwavering support and help during this work, and Professor Ricardo Chaves, for always having his door open and for his priceless contributions to this work.

Next, I want to thank my family for putting up with me throughout this work and supporting me during this stage of my life.

To the members of the CNM-TagusPark research group and the TagusPark's staff, whose input and help allowed this work to became what it is, thank you.

Finally, I thank my friends, for making me laugh and telling me ”No!” when I needed, you all know how you are. If you don't, you shouldn't be here anyway.

For all this and more,
Thank you,
João Gomes,
May 2014

Abstract

Vehicle Ad-Hoc NETworks are an emerging mobility paradigm focussed on improving road safety and traffic efficiency. While their potential benefits are immense, the vehicles' movement make communications very complicated, to address this new communication interfaces are being developed. However these interfaces are very expensive, which can become a barrier to the adoption of this kind of network. A number of applications can already be expected for this kind of network, but taking into account the possible factors that affect each application can make establishing a set of requirements very complicated. Also given the very different nature of the possible applications, each with its own set of requirements, developing and supporting each application can become a daunting task. Validating applications can also be very difficult, given the logistical overhead that field testing entails, not to mention the financial cost of having to use real vehicles for every test.

In this work, these issues are addressed. A low-cost platform, which uses commercial, off-the-shelf hardware is used. This platform supports a software architecture which is designed to address multiple application support and application validation. Applications are supported by using a policy-based approach which allows the platform to be configured, on a per-application base, in a simple way. Integrated with this approach, is a simple API which allows developers to focus on their application, not on how their application's requirements are met. To complete this solution, several tools are made available to allow for an easy validation of the applications, both in field and lab settings through emulation of field conditions. Testing is done to assure the emulator's behaviour is correct and two proof of concept applications show the capabilities of the proposed solution.

Keywords: TagusVanet, Low-cost Platform, Policy-based Approach, Integrated Framework, VANET Deployment, VANET Validation.

Resumo

As redes veiculares são um paradigma de mobilidade emergente cujo foco está em melhorar a segurança rodoviária e a eficiência de tráfego. Enquanto os benefícios têm um potencial imenso, o movimento dos veículos torna a comunicação entre si muito complicada, para mitigar este problema novas interfaces de comunicação estão a ser desenvolvidos. No entanto estas interfaces são dispendiosas, o que se pode transformar numa barreira à adopção deste tipo de redes.

Já são previsíveis, para este tipo de rede, um conjunto de aplicações, mas dada a quantidade de factores que afectam cada aplicação, uma análise de requisitos pode-se revelar complicada. Dado, ainda, a natureza distinta de cada aplicação possível, cada uma com os seus requisitos, desenvolver e suportar cada uma pode-se revelar uma tarefa quase impossível. Outra questão complicada é a validação do funcionamento de uma aplicação, que implica um esforço de planeamento logístico que evite gastos financeiros desnecessários, visto ser preciso um veículo dedicado para cada teste.

Neste trabalho, estes problemas são abordados. Uma plataforma de baixo custo, baseada em equipamento disponível ao consumidor usual, é usada. É nesta plataforma que uma arquitectura de *software*, projectada especificamente para estes problemas, é executada. Para suportar múltiplas aplicações, é usada uma solução baseada em políticas que permite a fácil configuração na plataforma dos requisitos de cada aplicação. Integrada nesta solução, está uma API simples que permite aos programadores concentrar-se na sua aplicação e não na satisfação dos requisitos da mesma. Para completar esta solução, são fornecidas várias ferramentas que permitem facilitar a validação da aplicação, tanto em testes de campo como em laboratório, através da emulação das condições de campo. O comportamento deste emulador é testado para garantir a sua correcção e duas aplicações são apresentadas que provam os conceitos base desta solução.

Palavras-chave: TagusVanet, Plataforma de baixo custo, Solução baseada em politicas, Sistema integrado, Implementação de redes veiculares, Validação de redes veiculares.

Contents

Acknowledgments **iii**

Abstract **v**

Resumo **vii**

List of Figures **xiv**

List of Tables **xv**

Acronyms **xvii**

1 Introduction **1**

1.1 Motivation 1

1.2 Problem statement 2

1.3 Proposed solution and thesis contribution 2

1.4 Outline 3

2 Related Work **5**

2.1 VANET overview 5

2.1.1 Components and interactions 5

2.1.2 Data dissemination modes 6

2.2 VANET Applications 7

2.2.1 Safety-oriented 7

2.2.2 Convenience-oriented 8

2.2.3 Commercial-oriented 9

2.2.4 Summary 9

2.3 Reference architecture 10

2.3.1 Communication Access for Land Mobile (CALM) 10

2.3.2 Wireless Access in Vehicular Environments (WAVE) 11

2.3.3 Access technologies 13

2.3.4 Summary and discussion 14

2.4 Platforms and testbeds 15

2.5 Policy-based Management 16
2.6 Propagation Models 17
2.6.1 802.11b Packet Error Rate Estimation 17
2.6.2 Attenuation Models 18
2.7 Summary 21

3 Architecture **22**
3.1 Requirements 22
3.2 TagusVanet Platform overview 23
3.3 Application Developer Interface module 24
3.3.1 Requirement analysis 24
3.3.2 Module architecture 24
3.3.3 Open issues 26
3.4 Vehicle Ad-Hoc NETwork (VANET) Emulator module 26
3.4.1 Requirement analysis 26
3.4.2 Module architecture 26
3.4.3 Open issues 28
3.5 Main system module 28
3.5.1 Requirement analysis 28
3.5.2 Module architecture 29
3.5.3 Open issues 30
3.6 Global SW architecture 31
3.7 Policy-based Approach 31
3.7.1 Application-oriented Policies 31
3.7.2 Network-oriented Policies 34
3.8 Summary 36

4 Implementation **38**
4.1 Hardware Architecture 38
4.1.1 TS-7500 39
4.1.2 Wireless Cards 40
4.1.3 Global Positioning System (GPS) Receiver 40
4.1.4 OnBoard Diagnostics II (OBD-II) interface 41
4.1.5 On-Board Unit (OBU) and Road Side Unit (RSU) Hardware Architecture 42
4.2 Software Architecture 43
4.2.1 Application Developer Interface 43
4.2.2 VANET Emulator Module 45
4.2.3 Main System Module 49
4.3 Summary 56

5 Evaluation **57**

5.1 Emulator Module 57

5.1.1 Test Description 57

5.1.2 Test Results 58

5.2 Platform evaluation 60

5.2.1 Test Description 61

5.2.2 Field Demonstrator 61

5.2.3 Legacy Application 62

5.3 Summary 64

6 Conclusions and Future Work **66**

A Wireshark Screen Captures **69**

Bibliography **75**

List of Figures

2.1 Basic VANET architecture. 5

2.2 Communication Access for Land Mobile (CALM), Operating in a multi-media environment. [22] 10

2.3 CALM Classic Architecture. [22] 11

2.4 WAVE communication stack indicating the standard that covers each set of layers. [23] 11

2.5 Basic Universal Mobile Telecommunications System (UMTS) architecture. 13

2.6 Simplified model of ground reflection interference. 19

2.7 Generic elements of the Single Knife Model. 21

3.1 Platform organisation 24

3.2 Platform organisation: Application Developer Interface module 25

3.3 Platform organisation: Emulator module 27

3.4 Platform organisation: Main system module 29

3.5 Platform architecture. 31

4.1 TS 7500 board. 39

4.2 SMC Wireless Card. 40

4.3 Holux GPS receiver. 41

4.4 ELM237 OBD-II interface. 41

4.5 OBU hardware solution. 42

4.6 RSU hardware solution. 42

4.7 Wireshark traffic analysis flow. 46

4.8 Sumo simulation flow. 47

4.9 Sumo network import example. 47

4.10 Sumo generated networks example. 48

4.11 Communication Manager / Message Dispatcher message handling. 51

4.12 Policy Manager's decision process. 52

4.13 RxUnicast policy's decision process. 52

4.14 RxBroadcast policy's decision process. 53

4.15 RxGeocast policy's decision process. 54

5.1 Wireshark screenshot showing the contents of a Basic Safety Message Verbose. 59

5.2 Wireshark screenshot showing the contents of a Probe Data Management Message. . . . 59
5.3 Propagation model behaviour. 60
5.4 Propagation model impact on the Control CHannel (CCH). 60
5.5 Impact of Service CHannel (SCH) traffic in CCH. 61
5.6 Proof of Concept applications' test scenario. 61
5.7 Hardware used for field demonstration (Upper left, OBD-II interface, upper right, Wireless cards, bottom, TS-7500 and GPS receiver). 62
5.8 Platform demonstration, RSU in cyan, OBU 1 in green and OBU 2 in pink. 63
5.9 Platform demonstration, RSU in cyan, OBU 1 in green and yellow and OBU 2 in pink. . . 64
5.10 Example of information retrieved from the platform. 65
5.11 Legacy application support, Proof of Concept. 65

A.1 Wireshark screenshot showing the contents of a Basic Safety Message Verbose. 69
A.2 Wireshark screenshot showing the contents of a Probe Data Management Message. . . . 70

List of Tables

2.1 Vehicle Ad-hoc NETwork (VANET) application requirements. 10
2.2 Access Technologies. 15
2.3 Summary of VANET platforms characteristics. 16

3.1 *Message Lifetime* and *Region Size* keywords. 32
3.2 *Region Format* and *Region Direction* keywords. 33

4.1 Implemented functionalities of the hardware platform. 39
4.2 Neighbour manager's record format. 55

List of Acronyms

ALCM A La Carte Message

AP Access Point

API Application Programming Interface

ASN.1 Abstract Syntax Notation 1

ASN1C Abstract Syntax Notation 1 (ASN.1) Compiler

BER Bit Error Rate

BSM Basic Safety Message

BSS Basic Service Set

C2C-CC Car 2 Car Communication Consortium

CA Certification Authority

CALM Communication Access for Land Mobile

CCH Control CHannel

CN Core Network

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

CVIS Co-operative Vehicle Infrastructure Systems

DCF Distributed Coordination Function

DER Distinguished Encoding Rules

DIFS Distributed Coordination Function (DCF) Inter-Frame Space

DSRC Dedicated Short Range Communications

EDCA Enhanced Distributed Channel Access

EFSM Empirical Free-Space Path Loss Model

ETSI European Telecommunications Standards Institute

FDD Frequency Division Duplex

FSM Free Space Path Loss Model

GNSS Global Navigation Satellite System

GPS Global Positioning System

HSDPA High-Speed Downlink Packet Access

HSUPA High-Speed Uplink Packet Access

I2I Infrastructure to Infrastructure

I2V Infrastructure to Vehicle

IBSS Independent Basic Service Set

IEEE Institute of Electrical and Electronics Engineers

IPv6 Internet Protocol Version 6

ISO International Standard Organization

LOS Line Of Sight

LTE Long Term Evolution

MAC Medium Access Control

MANET Mobile Ad-hoc NETwork

MIB Management Information Base

NMEA National Marine Electronics Association

OBD-II OnBoard Diagnostics II

OBU On-Board Unit

OFDM Orthogonal Frequency Division Multiplexing

PER Packet Error Rate

RSU Road Side Unit

SAE SAE International

SAFESPOT Co-operative Systems for Road Safety

SCH Service CHannel

SEVECOM Secure Vehicle Communications

SKM Single Knife Model

STA Station

SUMO Simulation of Urban MObility

TDD Time Division Duplex

TRGM	Two Ray Ground Model
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UTRAN	Universal Terrestrial Radio Access Network
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
VANET	Vehicle Ad-hoc NETwork
VSCC	Vehicle Safety Communications Consortium
WAVE	Wireless Access in Vehicular Environments
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WSMP	Wireless Access in Vehicular Environments (WAVE) Short Message Protocol
WSN	Wireless Sensor Network
WWAN	Wireless Wide Area Network

Chapter 1

Introduction

1.1 Motivation

The advances of wireless communications and embedded systems lead to the emergence of VANET, which is seen nowadays as a mean to increase road safety and travel efficiency. This new type of network introduces new challenges that have driven the attention of the research community and standardisation bodies, as surveyed in [1, 2]. Also new business opportunities have been identified by auto manufacturers, road operators, fleet transportation and software development companies.

The currently foreseeable VANET applications can be divided into three main categories: safety-oriented, convenience (or traffic management) and commercial [3]. Safety-oriented applications focus on assisting the driver in handling potentially dangerous situations, by actively monitoring the environment and listening to the messages being exchanged between nodes. Convenience applications enable more efficient traffic flow and increased vehicle throughput, by sharing information with the road infrastructure, centralised traffic control systems and even other vehicles. Finally, commercial applications focus on providing the driver with ways of improving satisfaction, entertainment and productivity. These new types of applications demand for new communications paradigms due to the specific requirements they have: safety-oriented applications need to guarantee the dissemination of time-critical information in a very short period of time; in traffic management applications, each node broadcasts information gathered from the vehicle sensors, to all the neighbour nodes, in a timely accurate way; and finally, commercial applications need to deal with the transport, routing and location problems that arise in VANET due to the nodes' mobility pattern and scenario characteristics. As all these problems are very challenging, a significant research activity is been driven to solve them. Therefore, a wide range of routing protocols [4], location services [5], efficient data dissemination strategies [6] and medium access control mechanisms [7] have been proposed.

A lot of standardisation is under going in the Institute of Electrical and Electronics Engineers (IEEE), International Standard Organization (ISO) and European Telecom Standardization Institute (ETSI). These efforts lead to the design of a new protocol stack specifically targeted to support safety-oriented applications, the Wireless Access for Vehicular Environment (WAVE). The IEEE also proposed a new 802.11-

based standard, which is able to cope with the specificities of VANET: the very short connection time that may arise in several mobility conditions; and the existence of multiple applications with different priorities that need to share the wireless medium. Nevertheless, they are is still in the early stage of development and several testbeds have been designed with existing IP protocol stack and available wireless technologies to validate the concept [8, 9, 10]. Besides the design cost, one of the main problems of building such demonstrators is the difficulty of assessing the physical conditions of a VANET operation, namely the signal propagation and nodes mobility. Hence, field tests are quite expensive and impossible to repeat in similar conditions and, due to this, there is an huge overhead associated with process of design and developing new applications and protocols.

1.2 Problem statement

In spite of interesting applications and good technological support that might be offered, VANETs will only succeed as a business when there is a huge number of costumers that support the network itself, providing the means to exchange data and leverage the technology and applications. At the current stage of development, it is expected that the technology will be offered on new vehicles, possibly only in the most expensive, which is not enough to create the necessary communication network. This might compromise the success of the technology, as long as costumers feel unable to take advantage of it and companies unable to get profit. Therefore, a fundamental question arises:

- *Is it possible to have a faster dissemination of VANET technology ?*

We believe it is possible, but, for this, two key aspects need to be guaranteed. First, we need to increase the number of vehicles that might use the technology right now. This means that, we need to have an easy way of introducing the technology in vehicles in circulation. Second, we need to increase the number of costumers that want to use the technology. For this, we need to have a very large number of non-expensive applications available so that each costumer is able to find the sort of application that fulfils his needs.

Given this point, the final question that this thesis aims to solve is the following one:

- *Is it possible to create a low-cost VANET operating system that offers a set of tools to create, test, and operate VANET's applications in most of the existing cars?*

1.3 Proposed solution and thesis contribution

This thesis addresses the problem previously stated by proposing a low-cost VANET platform that offers an integrated framework for development, validation and deployment of VANET applications in existing vehicles. To address the development issue, an abstraction layer is proposed that hides from the developer the inner functionalities of the VANET node that are needed to support its application. Additionally, information is provided using the available standards in order to promote interoperability. The validation is achieved through the creation of a VANET emulator, which can be used to recreate the field conditions in a controlled environment. A policy-based approach is selected to ease the deployment of the platform in large scale. Finally, standards are used to access to in vehicle information so that vehicles in circulation can benefit from the technology.

We consider that our main contribution is an embedded linux-based platform for VANETs, supported on standards. This contribution can be split into different components:

- **Application developer interface** – a library that is used by developers to interface the platform, which creates messages with standard format and setup the policies needed to guarantee that the application requirements are met.
- **A VANET node emulator** – a set of tools used to test applications, which reproduces, in the lab, the mobility pattern, the information gathered from the node and the propagation conditions and analyses the information gathered at the most widely used traffic probe, Wireshark.
- **A VANET "operating system"** – the basis of a software platform that supports the deployment of VANET in any linux-based embedded system, equipped with the interfaces needed to communicate with the in-board sensors, GPS device and other VANET nodes. Although the entire system may not be completed by the end of the thesis, it is expectable that it provides the relevant components, as well as the needed interfaces to include other modules.

1.4 Outline

This thesis is organised in 6 chapters.

- **Chapter 1** presents the motivation, problem statement, proposed solution and thesis contribution. It ends, with this section, that provides the outline of the document.
- **Chapter 2** describes the previous work in the field.
- **Chapter 3** describes the system requirements and the architecture of TagusVanet.
- **Chapter 4** describes the technologies chosen and the implementation of the TagusVanet platform.
- **Chapter 5** describes the performed tests and their results along with two Proof of Concept applications.
- **Chapter 6** summarises the work developed and future work.

Chapter 2

Related Work

2.1 VANET overview

This sections overviews the basic VANET concepts. It starts by presenting the components and interactions. After that, a brief overview of the data communication modes is performed. This description details more the **geocast** mode, which have been created to support VANETs, and the problems that it introduces, as well as the proposed solution.

2.1.1 Components and interactions

Being VANETs a new kind of network, there is a need to detail how different network components interact with each other. In figure 2.1 all the basic VANETcomponents and their interactions are illustrated.

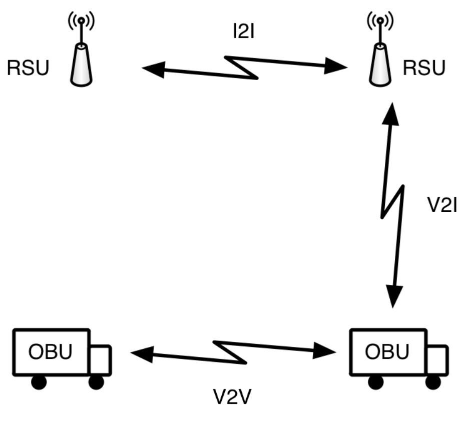


Figure 2.1: Basic VANET architecture.

As stated in the figure, a VANET comprises two major components: the On-Board Unit (OBU) and the Road Side Unit (RSU). The OBU is the VANET component located in the mobile node, the vehicle., and the RSU is located at fixed locations of the road infra-structure, such as traffic lights.

Vehicle to Vehicle (V2V) interactions are based on OBU communication [11], using single-hope or multi-hop communication, depending on the existence or not of intermediary retransmitting nodes. The communication range is usually small, due to the limited power of the OBUs antennas. There is a wide variety of possible applications, leading to different performance requirements. For same of them, high throughput may be needed, while for other latency and delivery are more relevant. As VANETs are expected to support safety critical applications, one need to guarantee that their requirements are met. Hence, these interactions may require that traffic is handled in a differentiate way, depending on the application.

V2V can be supported by different technologies, although most of them are based in Institute of Electrical and Electronics Engineers (IEEE) 802.11. Recently, IEEE proposed a new version of this standard, the IEEE 802.11p, specifically designed to cope with VANET requirements.

In Vehicle to Infrastructure (V2I) interactions, all communications are between OBUs and RSUs. As in V2V, communications can be single- or multi-hop, depending on the number of intermediary nodes. This type of interaction can have a larger range due to the type of antennas usually used at the RSUs. and a lower rate than V2I. A number of different technologies can support this interaction, among them are IEEE 802.11 and WiMax [11].

The Infrastructure to Infrastructure (I2I) is another type of interaction, mainly used for application support and access to the Internet. This type of interaction can be supported by any kind of technology, wired or wireless [12]. These interactions can have the highest data rate as there may be a need for multiple applications and multiple users to access a node in the infrastructure through a RSU.

2.1.2 Data dissemination modes

The data produced by applications in a VANET can be transmitted in many different ways, depending on the addressing scheme used [13]. Current networks use unicast, anycast, multicast and broadcast communication modes. In an unicast communication the source node addresses a single destination node. In anycast, the source node sends data to a group of nodes that share the same IP address, but this data is forwarded to the nearest node of this group. In case of multicast communication, data is delivered to a group of nodes using a single transmission. Finally, when using the broadcast communication mode, data is received by all the neighbour nodes of the source node. For all this type of communications, standard addressing mode is used, as destination nodes are known in advance and may be identified by their addresses.

However, these addressing modes are unable to fulfil the needs of all the emergent VANET applications, as a significant set of them requires the dissemination of information within a given region of interest. For this type of applications a new communication mode is needed: the geocast communication. In geocast communication, data is disseminate within a certain area and, for this, nodes need to be identified by their position, instead of by their IP addresses. A proof of concept of this new addressing mode is used by the Network on Wheels projects [14].

The use of geocast communication may cause a situation, usually known as Broadcast-Storm [15], where all nodes in the destination region rebroadcast the message at the same time. This leads to a state of frequent contention and increased packet collisions, causing an increase in backoff timers.

To address this issue, a probability based scheme is introduced in [15], where each node computes a rebroadcast probability using the following formula, on a per-packet basis, where D_{ij} represents the relative distance between node *i* and *j* and *R* the average transmission range.

$$p_{ij} = \frac{D_{ij}}{R} \tag{2.1}$$

The distance used in this formula is the distance between the current node and node that last broadcasted the message. This way it's ensured that the nodes closer to the Source are less likely to rebroadcast the message when neighbours farther away have already done it.

2.2 VANET Applications

Although VANET related research is still recent, an extensive list of applications can already be envisioned and surveyed in different studies [2], [1] and [16]. Several different classifications have been proposed by many different sources, eg. [17, 18], but the applications tend to be have similar purpose. We decided to use the classification that is based on the European Telecommunications Standards Institute (ETSI) standard [17], which identifies three major groups of applications: **Safety-oriented**, **Convenience-oriented** and **Commercial-oriented**. This section presents these categories in more detail and presents a few application examples.

2.2.1 Safety-oriented

Safety-oriented applications are intended to reduce accidents and by consequence, injuries and economic loss. In the type of application, vehicle cooperate among each other and with RSUs to provide relevant information and assistance to the driver in order to avoid a potential hazard. Safety applications may be used with different purposes, such as: emergency response, support to authorities, danger road features, abnormal road conditions, danger of collision, crash eminent or crash occurred. To support these purposes a wide range of applications is defined and the most important requirements analysed.

Next, we will detail two different type of safety-oriented applications: Emergency Vehicle Warning and Traffic Condition Warning.

Emergency Vehicle Warning – the Emergency Vehicle Warning application allows an emergency vehicle to announce its presence, as in many countries, dispatched emergency vehicles take precedence over other vehicles. This enables emergency vehicles to reduce response time and collision risk between with other vehicles.

In order to fulfil this goal, the emergency vehicle periodically disseminates safety messages, informing other vehicles about its position, speed and movement direction. These messages must be received by all the drivers whose vehicles may interfere with the circulation of the emergency vehicle. This means that, they must be transmitted using V2V within region-of-interested. Therefore, geocast communication is needed. If the drivers are not aware of the existence of an approaching emergency vehicle or are informed too late, the purpose of the application is compromised. Hence, a major concern here is to guarantee that the safety information must reach all the vehicles of the region in a short period of time. However, the message frequency must be carefully selected, as disseminating too often may lead to the occurrence of broadcast storm [15]. According to the authors of [19], the minimum message frequency is 10Hz and the critical time is 100 ms.

Traffic Condition Warning – the Traffic Condition Warning is used to warn drivers about potential hazards that arise due to the traffic conditions in the vicinity of their destination, resulting in a decrease of rear-end collisions and traffic jams. This implies that the vehicles need to be able to detect traffic jams and warn oncoming entities of such situations. It is also necessary for interested vehicles to be able to receive and process such messages. The application is supported by periodic dissemination of messages by OBU or RSU or even by an authoritative traffic management entity. In order to provide the dissemination of the messages within a region of interest, a geocast communication must be used. According to the ETSI standard [17], the minimum frequency of the messages is 1Hz.

2.2.2 Convenience-oriented

Convenience-oriented applications also called “Traffic efficiency” applications, aim to improve traffic flow and management. This category can be divided into two sub-groups, Speed Management and Co-operative Navigation. Speed Management applications aim at reducing the number of unnecessary stop’s and provide a smoother driving experience. Applications focused on Co-operative Navigation improve traffic efficiency and vehicle throughput. Among these applications some of the more relevant are Electronic Toll Collect, for Speed management, and Platooning for Co-operative Navigation.

Electronic Toll Collect – the Electronic Toll Collect is intended to improve traffic fluidity at a toll collect network through the use of a RSU. It’s established a full duplex unicast communication between the RSU and the OBU, after an initial broadcast by the RSU, in a V2I communication paradigm. This application requires that both RSU and OBU are capable of processing the messages associated with the electronic toll service and that the RSU is capable of advertising its electronic toll payment capabilities [20]. The RSU must also be able to check if the OBU is valid using the I2I communication. In order to guarantee the billing process, there is a maximum latency time since the electronic toll starts at the OBU until the RSU checks the OBU and gathers the correspondent account information. This type of application require the use of different modes of communication: unicast may be used to support V2I and I2I communications, while broadcast is used to support Infrastructure to Vehicle (I2V) communication. The ETSI standard [17] defines that the maximum value of the latency time is of 200ms and, for that, the RSU advertisement rate must be 1Hz.

Platooning – the Platooning application allows vehicles in a highway or a specific lane to act safely as a platoon thus providing a safer journey. This may result in only the leading vehicle having a driver, which can be useful in transferring electrical shared vehicles between places at low speed. This is achieved through the use of unicast and multicast messages exchanged by the vehicles in the platoon. The use of these messages implies that all vehicles in the platoon must be able to process them and establishing unicast connections. In order to maintain the safety of everyone involved, vehicle positions must be accurate by less than 2m and messages must have a maximum latency of 100ms with a refresh rate of 2Hz [21].

2.2.3 Commercial-oriented

Commercial-oriented or infotainment applications are, as the name implies, intended for commercial use. In this category there are two kinds of services, Co-operative Local Services and Global Internet Services. Examples of these services are Local Electronic Commerce and Fleet Management, respectively.

Local Electronic Commerce – the Local Electronic Commerce is intended to allow a RSU present at a local business the capability of processing local payment in the case of a service reservation or purchase of goods. This requires that the RSU is able to broadcast its capabilities and both the RSU and the OBU are able to establish a full duplex unicast communication. This service is less restrictive in message latency, 500ms, but the same message rate is defined, 1Hz [19].

Fleet Management – the Fleet Management is aimed to provide economical and mobile efficiency to a fleet of professional vehicles. This is achieved through information exchanges between a local RSU and vehicles that compose the fleet, where the vehicles carry information pertaining to other vehicles in the fleet besides themselves. This implies that the RSU is able to advertise its capabilities and has an Internet connection. The vehicles in the fleet need to have the capability of receiving, aggregating and retransmitting the information relevant to managing the fleet [19]. The RSU must advertise its capabilities at a minimum rate of 1Hz and the maximum latency of the service is 500ms, as defined in [17].

2.2.4 Summary

In this section we described the type of VANET applications and presented a few examples

These applications were categorized according to their functionality into three major groups: Safety-oriented, Convenience-oriented and Commercial-oriented. Although there is a wide range of possible applications, we decided to focus our attention into a limited set which represents the different challenges that the applications introduce in the VANET platform design. For each application, the type of communication, message frequency and data dissemination mode was presented and is summarised in table 2.1. This table contains the most important requirements that were taken into account when our platform was designed.

Application Category	Application	Communication Type	Minimum Message Frequency	Data Dissemination
Safety	Emergency Vehicle Warning	V2V	10Hz	Geocast
Safety	Traffic Condition Warning	V2V V2I	1Hz	Geocast
Convenience	Electronic Toll Collect	V2I I2I	1Hz	Unicast Broadcast
Convenience	Platooning	V2V	2Hz	Multicast
Commercial	Local Electronic Commerce	V2I	1Hz	Unicast Broadcast
Commercial	Fleet Management	V2V V2I Internet	1Hz	Unicast Broadcast

Table 2.1: VANET application requirements.

2.3 Reference architecture

In the following section, a reference architecture for VANETs is presented. First, the CALM standard is detailed, followed by the WAVE standard. At the end of the section, the main access technologies are described.

2.3.1 CALM

The CALM architecture was introduced by the International Standard Organization (ISO) and is a standard mainly focused on providing a set of parameters for both medium and long range communication and air interface protocols, for a number of medium access technologies. Figure 2.2 represents the CALM standard operating in a multi-media scenario. Although CALM is aimed at I2I, figure 2.2 displays a number of V2I capable technologies [22].

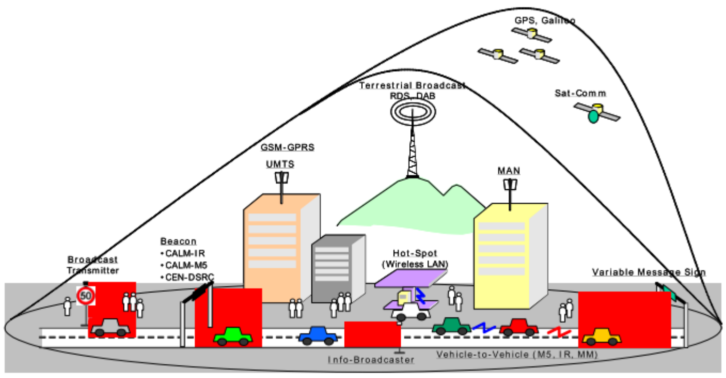


Figure 2.2: CALM, Operating in a multi-media environment. [22]

The central feature in CALM is multiple communication media support and enable networking between them [22]. This leads to a situation where if a medium becomes unavailable or another more suited medium becomes available, the change of medium is transparent to the user and as seamless as possible if the new medium fits in the users preferences [22].

As can be seen from Figure 2.3, CALM supports a wide variety of access technologies and tries to provide a easy way to support many different upper layer protocols. This is accomplished by using an unified network layer based on Internet Protocol Version 6 (IPv6). By using IPv6, CALM tries to make use of the support for Network Mobility and routing in ad hoc networks that this standard provides [12].

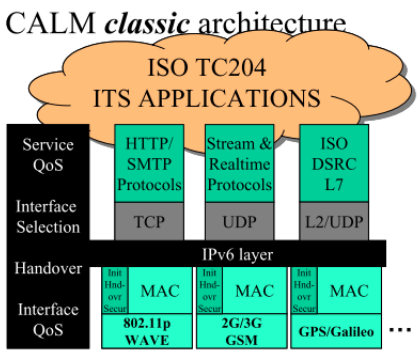


Figure 2.3: CALM Classic Architecture. [22]

2.3.2 WAVE

WAVE is a set of standards proposed by IEEE, composed by the IEEE 802.11p and IEEE 1609.1 through 1609.4 standards. These standards have the purpose of easing wireless access in vehicular environments, and their reference architecture is presented in Figure 2.4.

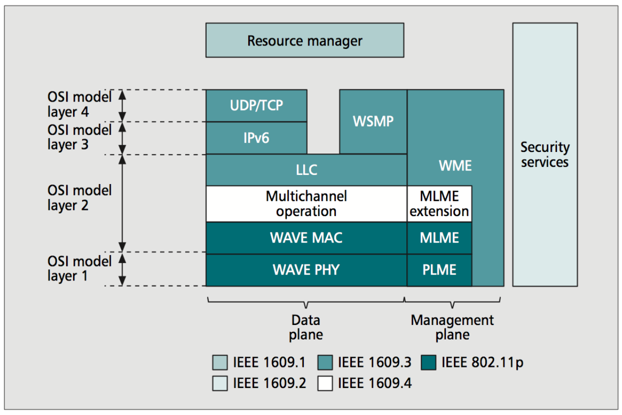


Figure 2.4: WAVE communication stack indicating the standard that covers each set of layers. [23]

Next a brief description of the standards, presented in Figure 2.4, is shown. Afterwards, each standard is described in more detail.

IEEE 802.11p Defines the PHY layer and Medium Access Control (MAC) sublayer, improved for vehicular environments.

IEEE 1609.1 Describes the resource manager, which is in charge of managing access to the system's communications resources.

IEEE 1609.2 Defines the security services associated with the network stack defined by the other 1609.x standards.

IEEE 1609.3 Specifies the LLC sublayer, Network and Transport layers.

IEEE 1609.4 Extends the 802.11p MAC sublayer to support multichannel operation and service priority management.

The IEEE 802.11p is based on the IEEE 802.11a physical layer design. The main differences are the channel width and the maximum allowed transmission power. The channel bandwidth is only 10MHz in 802.11p instead of the 20MHz in 802.11a. The modulation and allocated spectrum are the same, Orthogonal Frequency Division Multiplexing (OFDM) and 5GHz, respectively [24]. These changes allow the IEEE 802.11p to support speeds up to 250 km/h but limit the up and downlink rate to 27 Mbit/s [25].

One of the main components of WAVE is IEEE 1609.4, which coordinates multichannel operations. This standard separates channels by functionality and characteristics. One CCH and six SCHs are defined, the Control CHannel (CCH) has the highest maximum transmission power and is reserved for safety critical applications.

In addition to the multichannel operation, WAVE implements a prioritization scheme in the MAC-layer similar to the IEEE 802.11e Enhanced Distributed Channel Access (EDCA) function.

The network services, defined in IEEE 1609.3, standardize network and transport layer services, among which addressing and routing. It also defines WAVE Short Message Protocol (WSMP) thus providing an efficient alternative to IPv6. This protocol is specially tailored for safety applications, due to its short messages format. The Management Information Base (MIB) for the WAVE stack is also described in this standard [26].

In IEEE 1609.2 the security services for the WAVE stack are defined. The standard defines mechanism to provide confidentiality, authenticity and integrity based on cryptography [27]. This implies the definition of an authentication approach to allow secure one-way communication with low-latency for applications which require it. It also defines secure message exchange protocols and exchange triggering conditions [27]. While this standard defines the format of the certificates used while communicating, it doesn't define how this certificate management is done, leaving this management an open issue [28]. Although WAVE defines a lot of secure mechanisms for VANETs there is still some discussion as to how these solutions will be applied [29].

In order to be used with the WSMP, an application message set was defined, the SAE International (SAE) J2735 standard. In this standard a set of messages are defined, which are intended to support most VANET-oriented applications [30]. This message set is composed of a number of *Message Types*, being each *Message Type* composed by *Data Elements* or *Data Frames*:

1. *Data Elements* are the most basic type of information defined in this standard, for example, the vehicle's heading or latitude. They convey only a single type of information and in this standard

there are over 150 *Data Elements* defined.

2. *Data Frames* are composed of one or more *Data Elements* or *Data Frames*, for example a vehicle's 3D location is composed of its latitude, longitude and altitude above sea level. They are to relay more complex pieces of information and more than 70 defined in the standard.

Among the messages defined in this standard are a Basic Safety Message (BSM), which is used to carry vehicle-state information for safety-oriented application support, carrying information like the vehicle's coordinates, current speed and vehicle size. And a A La Carte Message (ALCM), a generic message with flexible content chosen by applications according to their needs.

In this standard, all the elements are defined in the formal language ASN.1 [31]. For the over-the-air translation, the Distinguished Encoding Rules (DER) encoding is used, in this format every data item is encoded in three fields, *Tag*, *Length* and *Value* [32].

2.3.3 Access technologies

In this section the main access technologies are detailed, as is their support for node mobility.

IEEE 802.11

This Wireless Local Area Network (WLAN) radio system is aimed at high data transfer rates for internet access in situations when vehicle speeds are moderate. Data rates can go up to 54Mb/s in a 20MHz wide channel, but a connection has to be setup prior to any communication can occur. In an infrastructure topology, communication range is determined by the Access Point (AP) coverage area.

Since in 802.11, MAC is based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), in which collisions occur randomly, there are some issues when the network load increases [33].

In order to provide some degree of traffic differentiation, the basic standard does not provide, the 802.11e amendment was introduced, it defines several traffic classes and prioritizes traffic by differentiation, thus allowing to some extent real-time traffic support [33].

Mobility support in the standard is limited as the highest speed supported in the 5GHz range, using amendment "a", is of 11 km/s [12].

This technology can be used either for V2V or V2I.

UMTS

The UMTS technology being a Wireless Wide Area Network (WWAN) is more complex than IEEE 802.11.

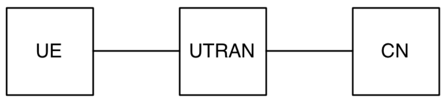


Figure 2.5: Basic UMTS architecture.

Figure 2.5 details a very simplified reference architecture [34]. The Universal Terrestrial Radio Access Network (UTRAN) is in charge of handling mobility, handover control, ciphering and deciphering the radio channel, etc. The User Equipment (UE) makes use of the UTRAN to establish communication with the Core Network (CN). Among the functionalities of the CN are gateway to other networks, inter-system handover and location management.

When the UMTS technology was first implemented the maximum downlink bandwidth was 384 kb/s and maximum uplink was 64kb/s. Now, with the use of High-Speed Downlink Packet Access (HSDPA) and High-Speed Uplink Packet Access (HSUPA), theses rates can be increased, in theory, to 7.2Mb/s and 5.8Mb/s respectively [35].

The maximum node velocity supported is 500km/h in rural environments and 120 km/h in suburban outdoor environments [36].

V2I is supported by this technology, as is I2I.

Worldwide Interoperability for Microwave Access (WiMAX)

The WiMAX technology is a Wireless Metropolitan Area Network (WMAN) radio system based on the IEEE 802.16 standard designed to be a wireless last mile alternative to wired technologies. But with the introduction of IEEE 802.16e amendment, this technology became a possible alternative to the fourth generation of wireless cellular communication systems [12].

WiMAX can operate in any frequency up to 66GHz, with various channel bandwidths available, typically 1.25, 5, 10 or 20 MHz [12]. This leads to data speeds of 70 MBit/s in up and down link [33].

The main functionalities of WiMAX are advanced antenna techniques, native QoS support at the MAC layer and mobility support.

With the introduction of the “802.16m” amendment mobility support was increased, this lead to the support of speeds up to 350 km/h and intercell handover [12].

By design this technology can be used for V2I and I2I.

Long Term Evolution (LTE)

The LTE is a recent technology, with the first standard released in 2009, designed as the evolution of UMTS, it is a fourth generation cellular communication system [12].

LTE operates with channel bandwidths of 1.4, 3, 5, 10, 15 and 20 MHz and supports data rates up to 316.6 Mbit/s in downlink and 86.4 Mbit/s in uplink.

Like WiMAX it uses advanced antenna techniques, like MIMO, but unlike WiMAX node velocity can be up to 500 km/h, well over the 100 km/h needed for a vehicular system [12].

Like UMTS, this technology can only support V2I and I2I.

2.3.4 Summary and discussion

As can be seen on Table 2.2 only IEEE 802.11 and IEEE 802.11p provide the possibility of V2V communications. The need by a number of applications, for example *Traffic Condition Warning*, to have

V2V communication excludes most of the technologies presented, as single communication technology. Although other technologies can provide higher throughputs, this capability makes these technologies uniquely suited for VANET environments.

Technology	IEEE 802.11a	UMTS	WiMAX	LTE	WAVE
Interactions	V2V V2I	V2I I2I	V2I I2I	V2I I2I	V2V V2I
Downlink (Mbit/s)	54	7.2	70	316.6	27
Uplink (Mbit/s)	54	5.8	70	86.4	27
Maximum Speed (km/h)	11	500	350	500	250

Table 2.2: Access Technologies.

When comparing IEEE 802.11a and IEEE 802.11p, it's possible to conclude that IEEE 802.11p has a lower data rate, 27 Mbit/s instead of 54Mbit/s IEEE 802.11a supports. However IEEE 802.11p's maximum supported speed is 250 km/h in opposition to the 11 km/h that IEEE 802.11a supports. This and the fact that WAVE defines a whole protocol stack intended only for VANET environments, makes WAVE a reasonable candidate to be used.

Given the VANET-oriented nature of the SAE J2735 standard and the fact the it was developed to be used in accordance with WSMP, it is also an interesting technology to be used in any VANET-related research.

2.4 Platforms and testbeds

VANET experimental solutions are essential for validation purposes and several testbeds have been proposed with the aim of assessing the technology and applications [37]. Technology focussed solutions experiment on the applicability of the currently available technology in the vehicular field and application focussed solutions analyse how current existing applications behave in vehicular environments. Next we will describe some relevant projects in the field in order to define the design goals of our system.

In the COM2REACT project [38] Vehicle to Vehicle (V2V) based communications are evaluated, with IEEE 802.11b based communications, with the conclusion that Line Of Sight (LOS) is one of the most important issues in vehicular communications. A multi-hop testbed created at the University of Murcia, Spain [37]. It comprises up to 4 vehicles and their communication is based on the Optimized Link State Routing (OLSR) protocol. The authors point out that while OLSR is a good reference point for VANET research, it fails to address all the issues related to vehicular communication. Vehicle to Infrastructure (V2I) communications are studied in FleetNet [39] and at the University of California, Davis [40] in different ways, but come to the same conclusion on the use of UDP instead of TCP due to TCP's poor performance in mobile wireless environments.

In C-Vet [10], V2V, V2I and Infrastructure to Infrastructure (I2I) communications are used, the VANET testing platform is supported by a mesh network of routers, working as a backbone. This solution focuses on testing application behaviour and how VANETs can support applications. The authors concluded that VANET-specific applications are mostly based in geographic routing and require the existence of a Location Service.

Another platform that studies application behaviour in VANET environments is called CarTorrent [41]. This work focusses on V2V information sharing and it addresses information retrieval strategies in highly mobile scenarios.

In Orbit [42] a large scale testing solution is presented. The solution incorporates up to 100 nodes and focuses on the study of packet delivery rates in dense VANET environments. The authors study the behaviour of some VANET applications but also state that the simulation of node mobility is an issue that remains to be addressed.

Finally, in the Network on Wheels project [43], a hybrid simulation and emulation solution is presented. This solution emulates the VANET network stack on top of the Linux network stack, and the emulated nodes run on a single machine. This solution does not enable field testing, meaning that it is only usable in a laboratory environment.

In Table 2.3 the characteristics of these solutions are summarised.

Platform	Focus	Field Testing	Laboratory testing
COM2REACT	Technology	X	-
University of Murcia	Technology	X	-
FleetNet	Technology	X	-
University of California, Davis	Technology	X	-
C-VeT	Application	X	-
CarTorrent	Application	X	-
Orbit	Technology / Application	-	X
Network on Wheels	Application	-	X

Table 2.3: Summary of VANET platforms characteristics.

From Table 2.3 it is possible to conclude that there are no platforms available that allow testing, both VANET oriented technology or applications, in both field and lab conditions. Therefore. our main goal is to design such platform.

2.5 Policy-based Management

One of the major challenges of the design and implementation of a large scale distributed system is to guarantee that node configuration is consistent for all nodes. Policy-based management is a management approach that has been proposed with the goal of providing scalability, easiness and robustness of configuration. These challenges are also shared by VANET, meaning that this might be an interesting approach to follow in our system design.

Policies are rules for governing the behaviour of a system. They are often used as a mean of implementing flexible and adaptable systems for management of internet services and distributed systems [44]. Since their appearance, it has been used in different scopes, namely QoS, security and mobility management, considering both wired and wireless networks [45].

Policy-Based Management is a paradigm that allows the definition of high-level configuration and operation policies. It was used by the Internet Engineering Task Force (IETF) [46] in order to enable a more efficient management of application requirements in DiffServ. A more dynamic solution is proposed, which tackles the problem of policy definition and enforcement in multiple targets after an initial configuration process [47]. It uses Ponder, a declarative language aimed at specifying management and security policies. And also defines an architecture for, in a flexible manner, defining and enforcing network policy rules.

Lupu and Sloman [48] focus on specifying implementable policies for the management of distributed systems, having in mind that the system cannot be offline during this process. They do this by formalising what a policy is and its notation. This notation can be used to define both organisation-level policies and implementable policies. In this way, by using the same notation, high level concepts can be related with low level ones in order to ease the implementation of the policies.

More recently, it is also proposed for Wireless Sensor Network (WSN) [49] and autonomic systems [50]. The authors claims for the advantages of using self-management capabilities.

2.6 Propagation Models

Our systems aims at proposing a set of tools to validate VANET applications through the use a new node emulator, specifically targeted to this scenario. One of the aspects that must be taken into account to built such emulator is the modelling of physical conditions where the network is operating. Hence, in this section the most important issues related to signal propagation and modelling are presented.

2.6.1 802.11b Packet Error Rate Estimation

The IEEE 802.11b standard supports multiple transfer rates, 1, 2 5.5 and 11 Mbps. To effectively estimate the Packet Error Rate (PER), it is necessary to take into account the different rates [51].

In order to simplify the calculations of the PER, it is assumed that the bits in the packet are independent [52]. Using the packet length, *L*, and a binomial test for every bit, the PER can be expressed using:

$$\text{PER} = 1 - (1 - \text{BER})^{L} \tag{2.2}$$

The Bit Error Rate (BER) calculation formula is dependent on the different transfer rates.

For the 1 Mbps data rate, the BER is expressed in:

$$\text{BER}_{\text{1Mbps}} = \frac{1}{2} e^{-\frac{E_b}{N_0}} \tag{2.3}$$

Where:

$$E_b = \frac{P_r}{R_b}$$

is the average energy per bit, calculated by dividing the received signal power by the binary transmission rate. For the 2Mbps, the BER formula is defined as:

$$\text{BER}_{\text{2Mbps}} = Q_1(a,b) - \frac{1}{2}I_0(ab)e^{-\frac{1}{2}(a^2+b^2)} \tag{2.4}$$

where $Q_1(a,b)$ is the Marcum Q-function and $I_0(ab)$ is the modified Bessel function of the first kind and zero order with parameters a and b defined as:

$$a = \sqrt{\frac{2E_b}{N_0}\left(1-\sqrt{\frac{1}{2}}\right)},\ b = \sqrt{\frac{2E_b}{N_0}\left(1+\sqrt{\frac{1}{2}}\right)}$$

Finally for the 5.5 and 11 data rates, the BER can be expressed by:

$$\text{BER}_{\text{5.5, 11Mbps}} = 1 - \frac{1}{\sqrt{2\pi}}\int_{-X}^{\infty}\left(\frac{1}{\sqrt{2\pi}}\int_{-(\nu+X)}^{\nu+X} e^{\frac{-y^2}{2}}\mathrm{d}y\right)^{\frac{N}{2}-1} e^{\frac{-\nu^2}{2}}\mathrm{d}\nu \tag{2.5}$$

Where N equals to 4 and 8 for 5.5 and 11 respectively and X takes the value:

$$X = \sqrt{\frac{2E_b}{N_0}}$$

2.6.2 Attenuation Models

In this section several models are detailed, each with its own characteristics. These attenuation models are used to estimate the received signal strength.

Free-Space Path Loss Model

The Free Space Path Loss Model (FSM) is the most simple, it represents ideal propagation conditions where the transmitter and receiver have a clear LOS and the antennas used are isotropic. This model is expressed using the following equation [53]:

$$L_{fsm} = 20log\left(\frac{4\pi d}{\lambda}\right) \qquad (dB) \tag{2.6}$$

Where:

L_{fsm} free-space loss (dB)

d distance between transmitter and receiver

λ signal wavelength

d and λ are expressed in the same unit.

By using the characteristics of the 802.11b standard, an operating frequency of 2.4 GHz, the Equation 2.6 can be be rewritten as:

$$L_{fsm} = 40.045 + 20log(d) \qquad (dB) \tag{2.7}$$

Empirical Free-Space Path Loss Model

In order to provide a better model of estimating free-space path loss, in [54] an Empirical Free-Space Path Loss Model (EFSM) is presented.

This model is oriented at VANET path loss estimation, the used antennas were omnidirectional with a central frequency of 5.890 GHz and the experiments were conducted without obstacles.

This model is based on an empirical adaptation of the FSM:

$$L_{efsm} = 10Log\left(16\pi^2 \frac{d^\alpha}{\lambda^\alpha}\right) \qquad (dB) \tag{2.8}$$

The experiments done in [54] found that the best value of α should be 2.2.

Using this information and the 802.11b standard frequency of 2.4 GHz, it is possible to rewrite the Equation 2.8 as:

$$L_{efsm} = 41.852 + 22Log(d) \qquad (dB) \tag{2.9}$$

Two Ray Ground Model

While the FSM and EFSM can be used to estimate path loss, they don't take into account that radio propagation usually suffers from one notable source of interference, ground reflection, as illustrated in Figure 2.6.

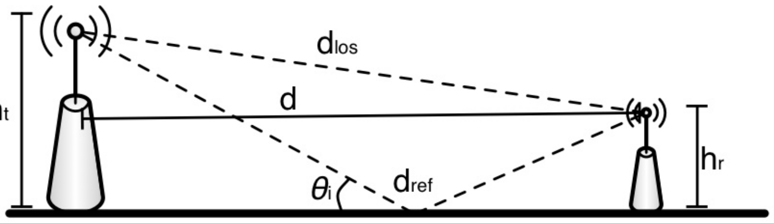


Figure 2.6: Simplified model of ground reflection interference.

This interference can be expressed by taking into account the phase difference of interfering rays φ and a reflection coefficient $\Gamma_\perp$ [55]. This leads to a *Two-Ray Interference* model, which is expressed in:

$$L_{tri} = 20log\left(\frac{4\pi d}{\lambda} \mid 1 + \Gamma_\perp e^{i\varphi} \mid^{-1}\right) \qquad (dB) \tag{2.10}$$

Where:

$$\begin{aligned}
\varphi &= 2\pi\frac{d_{los}-d_{ref}}{\lambda}\\
\Gamma_{\perp} &= \frac{\sin\theta_i-\sqrt{\epsilon_r-\cos^2\theta_i}}{\sin\theta_i+\sqrt{\epsilon_r-\cos^2\theta_i}}\\
d_{los} &= \sqrt{d^2+(h_t-h_r)^2}\\
d_{ref} &= \sqrt{d^2+(h_t+h_r)^2}\\
\sin\theta_i &= \frac{(h_t+h_r)}{d_{ref}}\\
\cos\theta_i &= \frac{d}{d_{ref}}
\end{aligned}$$

Since this calculation is far more complex than the free-space based models, in [55], this model is simplified for large distances and assuming there is perfect polarisation and reflection. This simplification is called Two Ray Ground Model (TRGM) and is expressed in:

$$L_{trg} = 20Log\left(\frac{d^2}{h_t h_r}\right) \qquad (dB) \tag{2.11}$$

For this model to be used in VANET simulations, network simulators usually use a combination of the FSM and TRGM [54]:

$$L_{fsm/trgm} = \begin{cases} L_{fsm} & \text{if } d \leq d_c \\ L_{trgm} & \text{if } d > d_c \end{cases} \qquad (dB) \tag{2.12}$$

This solution uses a cross-over distance, which is expressed by:

$$d_c = 4\pi\frac{h_t h_r}{\lambda}$$

Single Knife Model

In many cases the LOS between transmitter and receiver is obstructed, in the case of VANETs mainly by other vehicles [56]. The Single Knife Model (SKM) is the simplest obstacle model, which is used as a reference for other more complex models. This model assumes an ideal knife-edge obstacle with negligible thickness. In Figure 2.7 the generic elements of this model are represented.

Due to the operating characteristics of the WAVE stack, a radio frequency of 5.9GHz, the wavelength, in the order of 5cm, is small enough to meet the model's requirements [57].

The additional attenuation of this model can be expressed by:

$$A_{sk} = \begin{cases} 6.9+20log[\sqrt{(\nu-0.1)^2+1}+\nu-0.1], & \text{for } \nu > -0.7 \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

Where

$$\nu = \frac{\sqrt{2}H}{r_f}$$

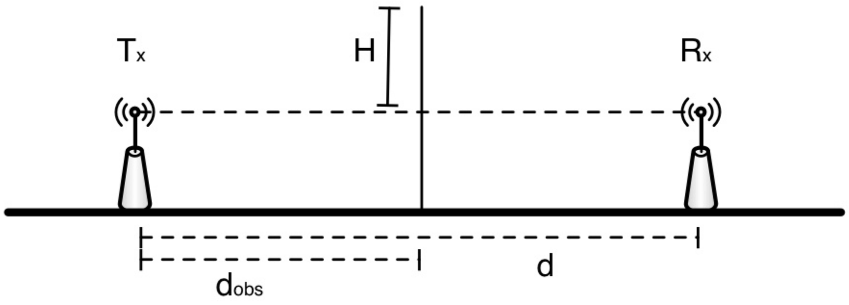


Figure 2.7: Generic elements of the Single Knife Model.

in which H is the difference between the height of the object and the direct line that connects the transmitter and the receiver, and r_f is the Fresnel ellipsoid radius, expressed by:

$$r_f = \sqrt{\frac{\lambda d_{obs}(d - d_{obs})}{d}}$$

Where d is the distance between transmitter and receiver, d_{obs} is the distance between the transmitter and the obstacle and λ is the signal wavelength.

2.7 Summary

VANETs have different needs and purposes from all other networks, this leads to the appearance of VANET-specific standards. These standards cover everything from application classification and message formats, to physical communication specifications. This standardisation effort is also based on a varied array of research work done in this field.

In this chapter, an overview of all these topics was given. First, an overview of the area is presented, comprising the components and interactions, as well as the data dissemination strategies. In section 2.2, the currently envisioned VANET applications were categorised and some examples given. Then, in Section 2.3, an overview of the current standardisation work was given, in the form of a reference architecture, several possible communication technologies were detailed and two all encompassing standards were explained, CALM and WAVE. Finally, an application-level standard was detailed in the form of the Dedicated Short Range Communications (DSRC) Message Set. In Section 2.4, some of the current research platforms and testbeds were described. These were separated into two categories, technology and application centred. Finally in the last two sections, the work done in two additional fields is detailed, policy based management and propagation models. While these fields have little relation with VANET standardisation and research in themselves, they are the basis for some of the work presented in this work.

Chapter 3

Architecture

This chapter describes the architecture of **TagusVanet**, an integrated platform for development, validation and deployment of VANETs. It comprises seven sections. In section 3.1, we start by defining the requirements that lead to design of our platform and, in 3.2, we present a brief overview of it. After that, each one of the main blocks is described in a specific section: the application developer interface is described in section 3.3; the VANET emulator, in section 3.4 and, finally, the VANET ”operating system” is described in section 3.5. In Section 3.6 the global architecture is shown.

Finally in Section 3.7 a policy-based architecture is defined, which is used to simplify the development of new VANET applications.

3.1 Requirements

We aim at creating a system that might be used to fasten the dissemination of VANETs. To achieve this, our goal is twofold: to ease the development of new VANET-oriented applications and to enable the use of the platform in existing vehicles. Hence, TagusVanet is a unique platform that integrates a set of tools to create, test, and deploy VANET applications, in existing vehicles. To accomplish this, the complete platform must satisfy the following set of requirements:

- **Cost-effectiveness** – in order to promote the fast dissemination of VANET technology, the platform must have a low cost, so that everyone can afford it without the need of spending too much money to buy it. Also, the process of application development, validation and test must be shortened and simplified. This means that, expensive field trials must be avoided as much as possible.

- **Extensibility** - Since VANET research is still a hot topic, the platform must be able to incorporate new breakthroughs in an easy and simple fashion. Hence, integration of new protocols and technologies must be easily achieved.

- **Flexibility** – the platform must flexible enough so that it can integrate into a unique system the creation, test and deployment of a VANET. Additionally, as there must be a significant number of applications with very different characteristics, the platform must be able to support each one of them.

3.2 TagusVanet Platform overview

Our aim is to create a low cost, extensive and flexible platform, which simplifies the creation of VANET-oriented applications and VANET-related research.

The cost–effectiveness is provided by concentrating into a unique platform the three main goals that have been identified - creation, validation and deployment of VANET applications - using off-the-shelf HW and open source SW.

The extensibility of the platform is achieved by the use of standards that promote inter-operability at different levels of the platform. Hence, one can easily develop new components or replace existing ones without breaking the entire system. Currently, we are addressing the two entry levels of our system. At the top-level, inter-operability among the applications of different vendors/developers is guaranteed by the use standards. At the bottom-level, standards are also used to gather data from the on-board sensors, as well as to access the GPS information. This way, the same interface to retrieve data from different vehicles is provided.

Concerning the last requirement, flexibility, it is provided in two ways: first, by organising the platform into different modules that cooperate among each other to fulfil the three goals of the platform: application development, validation and deployment; second, by the use of a policy-based approach. Instead of specifying policies related to specific applications, we specify a set of policies related to each application type. The policies strive to provide an application developer with a simple interface, which is used to define the application's needs and abstract some of the inner workings of a VANET. This enables the developer to focus more on the application's functionality, freeing it to create better applications. Network behaviour is modelled according to the application needs, by mapping higher layer policies into network layer ones.

According to this, we organised our platform into three different modules, as depicted in Figure 3.1:

- **Application Developer Interface module** – supports the development of new applications and offers an API that abstracts to the programmer the details of the standards implementation.

- **Emulator module** – supports the validation of the applications and deployment conditions by enabling the recreation of field tests in the laboratory.

- **Main system module** – supports the deployment of applications and the implementation of the VANET both in RSUs and OBUs.

The next sections provide additional insights into each one of them.

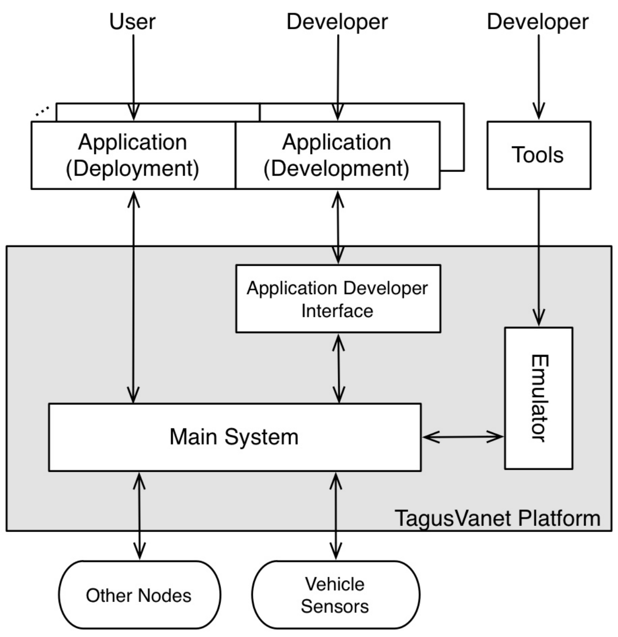


Figure 3.1: Platform organisation

3.3 Application Developer Interface module

3.3.1 Requirement analysis

Having in mind that the goal of the **Application Developer Interface** is to enable the creation of new VANET applications, the following set of requirements was established.

- **Easy development of applications** – VANETs enable the development of new types of applications, which have very different requirements and characteristics than those that exist in the current Internet applications. The platform must provide an easy interface to developers that hides the details of VANETs operation, allowing them to focus their efforts in application design.

- **Standard-compliant application design** – In order to promote inter-operability along different vendors, an extensible standardisation effort is been realised, comprising an entire set of standards that covers message formats for VANET applications. The platform must be compliant with these standards.

3.3.2 Module architecture

As stated before, the **App Developer Interface** module must be designed to be allow an easy development of applications and to be standard-compliant.

The use of a policy-based approach allows an easy development of applications, as their requirements can be defined in terms of Application-oriented policies that are converted into network-oriented policies in the platform. Hence, developers can focus their efforts in the application itself, leaving to the platform the task of enforcing that its requirements are met.

Standard-compliance is achieved by formatting messages received from the applications according to the standard format defined in the WAVE architecture: the SAE J2735 Message Set Dictionary [58]. The most important of these is the BSM, which conveys critical vehicle state information in support of V2V safety applications. However, other type of messages, needed to support other applications, are also defined.

The standard SAE J2735 defines the messages format, while leaving to other standards the specification of the minimum communication requirements. In our platform, this task is executed by the Main System module.

The **App Developer Interface** module is depicted in Figure 3.2. As stated in the figure, this module comprises a single component: the **Message Formatter**. The **Message Formatter** receives/sends messages to applications during development phase, using a very simple custom Application Programming Interface (API). These messages are converted into the standard SAE J2735 format so that they can be processed by the **Main System** module.

Before starting the validation or deployment of the application, the platform must be configured in order to met the application's requirements. This process is realised in the **Main System** module, by the **Application Policy Parser**.

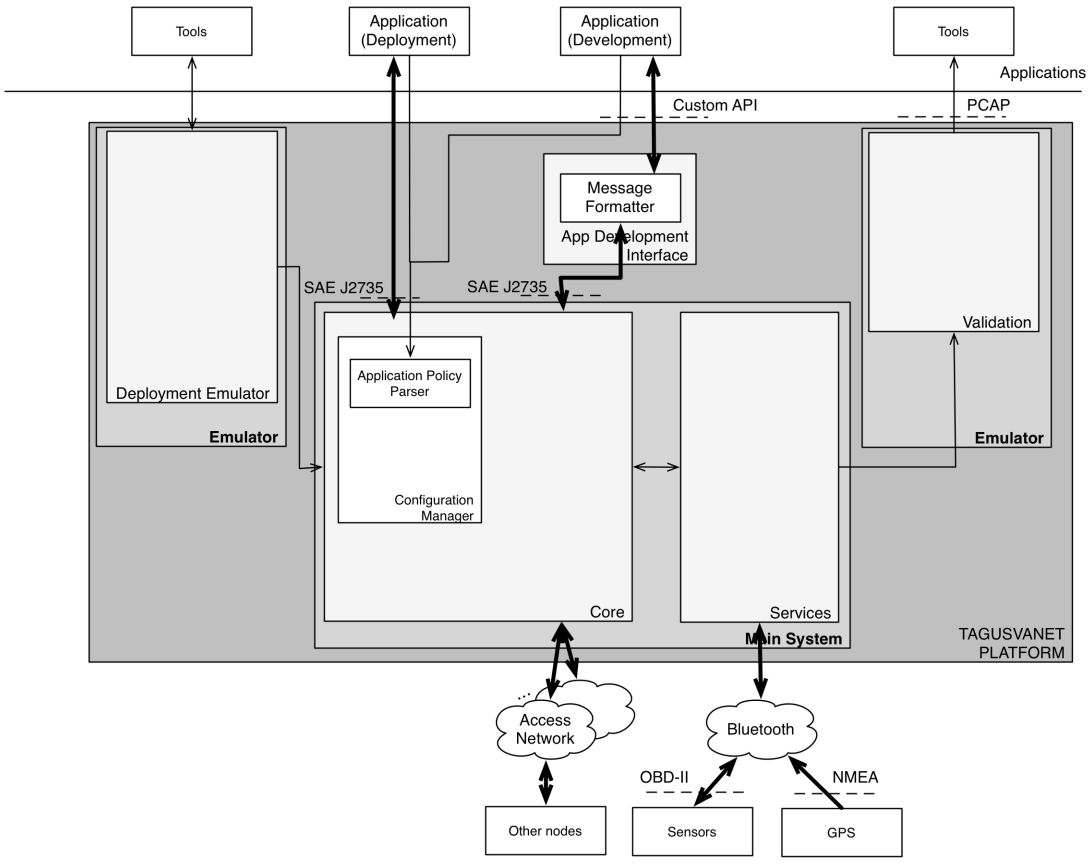


Figure 3.2: Platform organisation: Application Developer Interface module

3.3.3 Open issues

In the current version our platform offers a limited set of facilities for application developers, mainly related with standard conversion. This means that, our main focus is safety or traffic management applications, that are the ones that are supported in the standard.

While the platform was designed with Unicast-based communication in mind, this is an issue that remains to be addressed. This type of communication is the main form of communication for commercial applications, but it is still possible to support most of the currently foreseeable VANET applications.

3.4 VANET Emulator module

3.4.1 Requirement analysis

In order to successfully support VANET emulation, after careful analysis, the following requirements were established.

- **Easy validation of applications** – validating a VANET application might be an expensive and time-consuming process due to the difficulty of implementing a proof-of-concept testbed. The platform must provide a set of tools that allows the developer to validate their applications without realising expensive tests, requiring complex field trials.
- **Use of realistic conditions** – an important part of validating applications is the capability of recreating the same conditions multiple times. Therefore the platform must be able of emulating node mobility patterns, either real or simulated, in a simple way. In addition to node mobility, the platform must also be able to emulate sensor information and supply that information to the applications. Finally, in order to fully emulate multiple field trial conditions, the platform must provide multiple propagation models which are used to establish if a message is received or not.
- **Use of well-known tools** – to ease the learning curve and reach more developers, the platform must be able compatible with well-known tools and respect commonly used network analysis procedures. For this end, the platform must be compatible with Wireshark and allow the developers to analyse the network communication between the nodes.

3.4.2 Module architecture

The **Emulator** module was designed to allow an easy validation of applications, using realistic test conditions and well-known tools.

The aim of the **Emulator** module is to create a set of tools that allow the developer to emulate a real VANET, without going out of the laboratory.

These tools should allow an easy validation of the application in the deployment environment. These two aspects are spliced in the module into two different components: the **Validation** and the **Deployment**.

Concerning the **Validation**, we can consider that the validation process is twofold: first, we need to guarantee that the messages are adequately built; second, we need to verify whether the application's requirements are met or not. The validation of the message format may be easily performed by capturing and visualising the generated message with a network probe. To verify the requirements fulfilment, the entire message flow must be monitored. This means that, every node must be able to store in a log file all the information that passes through it (received or generated by the node itself). Logging is a service provided by the **Main system** module and used by the **Emulator** module. An integration with Wireshark tool facilitates the debugging.

Concerning the **Deployment**, we can think that the application's deployment is validated if we can reproduce in the laboratory real working conditions of the VANET. These conditions imply that each node in the lab has a behaviour similar to one in a real environment. Hence, real data gathered from vehicles (vehicle position) and adequate propagation models are provided. The platform can also be supplied with simulated data from vehicles. This data is provided by Simulation of Urban MObility (SUMO), which can be seeded with real data from road networks.

All these components are part of the **Emulator** module, whose architecture is represented in Figure 3.3.

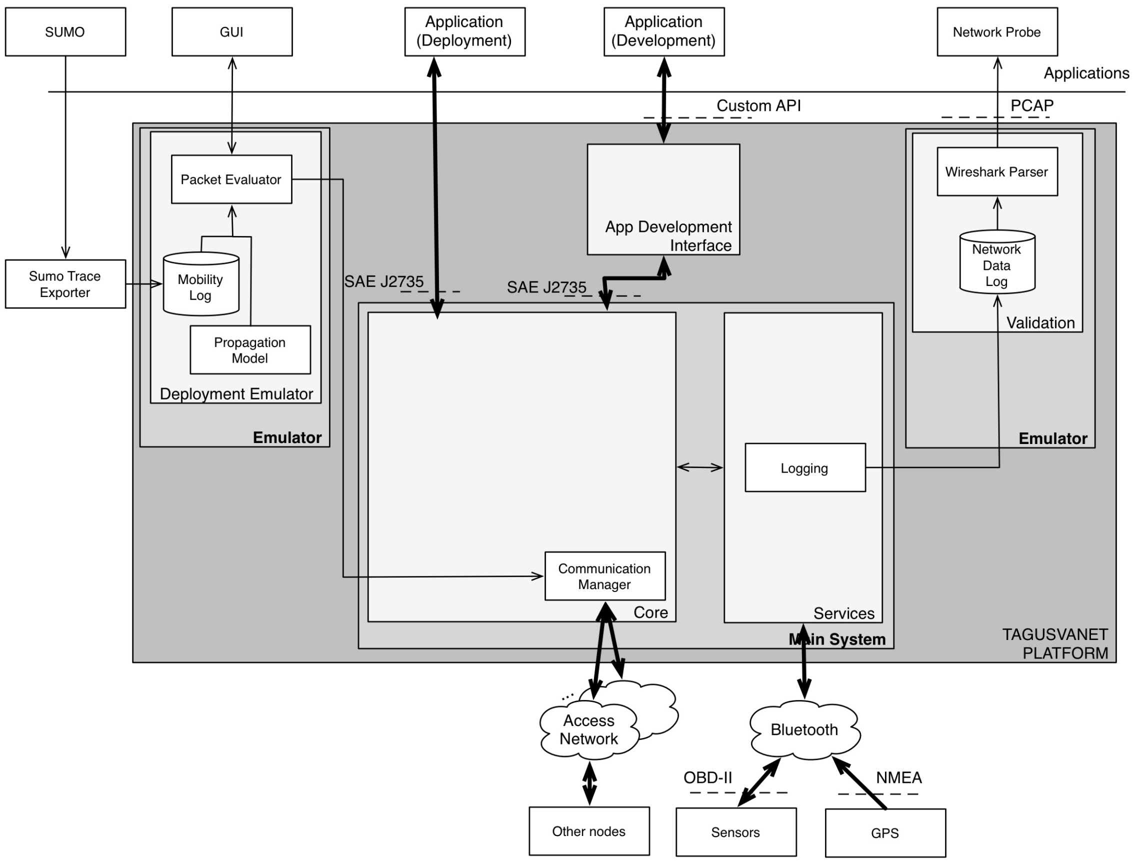


Figure 3.3: Platform organisation: Emulator module

3.4.3 Open issues

One downside to emulating VANET behaviour is that complex scenarios with a very dense concentration of nodes can be very expensive. One possible solution is to allow nodes to emulate more than one node, this issue is not addressed in the current version of the platform.

Another issue is that in order for the propagation estimation to be realistic, the models need to take into account the shadowing caused by other vehicles and buildings. Currently only vehicle shadowing is supported, which is useful when emulating highway and rural scenarios. One solution for urban scenario emulation is the use of more complex propagation models which take into account the presence of buildings.

3.5 Main system module

3.5.1 Requirement analysis

As stated in Section 3.2, the **Main System** is in charge of supporting application deployment and VANET implementation in both RSUs and OBUs. To achieve this the following requirements were established:

- **Configurability** – each application has its own requirements. Therefore, the platform must provide a set of configuration mechanisms in order to supply a way of configuring the node according to the application's requirements.
- **Support Product Differentiation** – applications can be developed by many sources. This may lead to applications with very different characteristics, this implies that the platform must support applications from multiple vendors.
- **Support Multiple Addressing Schemes** – VANETs are by nature a cooperative environment but not all applications are designed to be cooperative. The platform must support multiple addressing schemes, in this case, Broadcast, Geocast and Unicast.
- **Geocast Support** – not all Geocast-based applications have the same addressing requirements. With this in mind, the platform must be able to cope with different formats of the destination region.
- **Support multiple communication technologies** – in order to assure constant connectivity, even in areas where some communication technologies may not have coverage, the platform must posses multiple access technologies.
- **Integration in existing vehicles** – vehicles of different models or from different manufacturers offer different types of sensing information and might use different type of GPS devices. To be usable by the majority of vehicles and GPS devices available, the platform must support standard interfaces to interact with vehicles on-board unit.
- **Support Application Data Exporting** – to enable the recreation of real field test conditions in a controlled lab environment, the platform must be able to export the data used in field tests to be used later as needed.

3.5.2 Module architecture

Considering the previously stated requirements the **Main System** must be designed to be configurable. It must also support product differentiation and multiple addressing schemes, more specifically geocast. To ease application deployment it has to support multiple communication technologies and integration with currently existing vehicles. Finally, since the platform is designed to support application development, the **Main System** has to be able to export application data.

To achieve these goals the **Main System** is divided into three components, the *Configuration Manager*, which is in charge of configuring the system's behaviour, the *Services*, in charge of providing support for the *Core* and applications as required and the *Core*, which implements the network policies and handles data from and to the applications.

A detailed view of the **Main System** module is represented in figure 3.4.

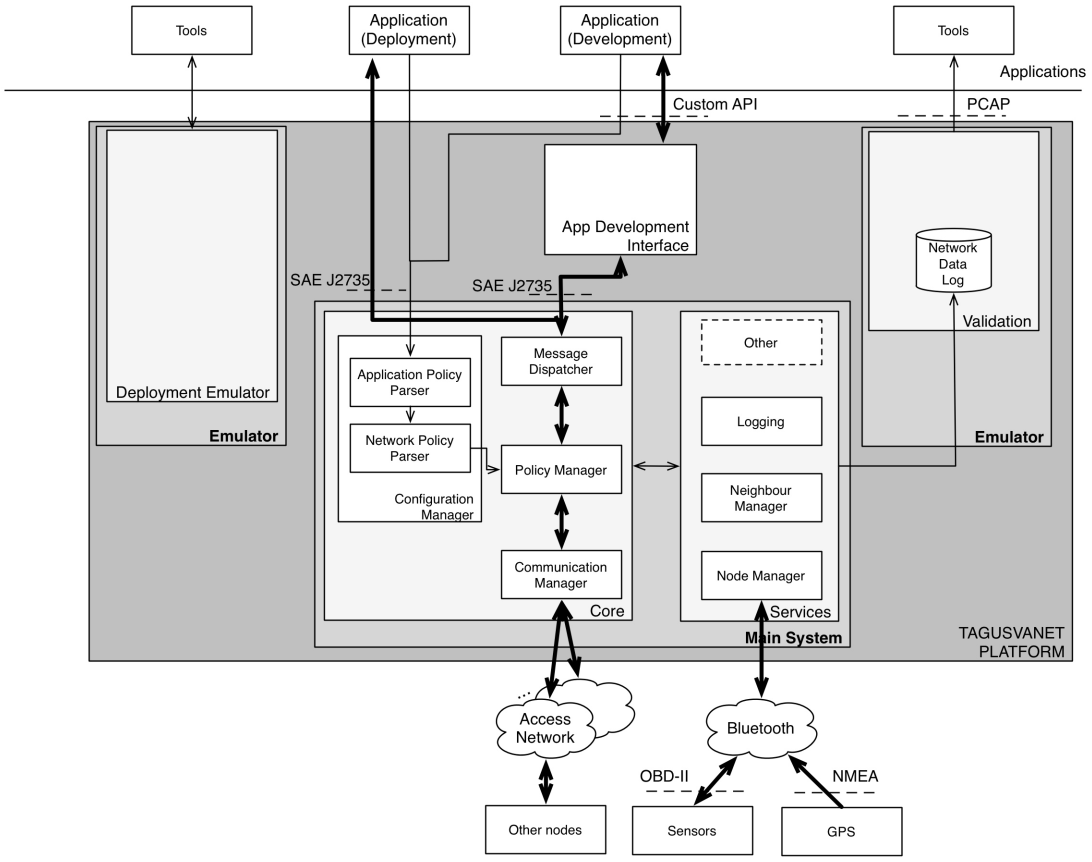


Figure 3.4: Platform organisation: Main system module

Before an application can send and receive messages it needs to register itself, by informing the platform of its needs, for this end the **Configuration Manager** is used.

The **Configuration Manager** is in charge of parsing the applications needs, in form of an application policy, and converting them to network-level policies which are used by the *Core*. To achieve this, the *Configuration Manager* is composed of two elements, the **Application Policy Parser** and the **Network Policy Parser**.

The *Application Policy Parser* is charged with receiving from an application its needs in the form of an application-policy and parsing them to a network-policy. The *Network Policy Parser* then takes the network-policy and configures in the **Policy Manager** that application's needs.

All the services supported by the platform are grouped into the **Services** component. The current version, includes the basic services needed to support the majority of applications, namely: the **Logging**, **Neighbour Manager** and **Node Manager**.

To allow the developers a simple way of debugging the application, a *Logging* service is available. The developers can use this service to understand the state of the application whenever they require it.

As most VANET-oriented applications are cooperative, the applications can request the platform to be notified when a new neighbour appears or a neighbour leaves. In the platform, the **Neighbour Manager** is used to keep track of new neighbours and leaving neighbours, and notifying the applications that have requested this information.

In order to ease application development, the *Node Manager* handles the connection with the GPS receiver, through the National Marine Electronics Association (NMEA) 0183 protocol [59], making the location and related information available to the applications through a custom API. This custom API is independent of the GPS receiver, this implies that the application code is independent of the installed receiver. The *Node Manager* is also in charge of interfacing with the in-car sensors, using the OBD-II standard, to retrieve information and make it available to the applications.

The **Core** component is responsible for the coordination of the communication process. For this, three different core services are needed: the **Message Dispatcher**, **Policy Manager** and the **Communication Manager**.

The **Message Dispatcher** is in charge of information flow. It receives data from the lower layers and sends it to the **App Interface** block, or it selects and sends messages from the applications or from the services.

The **Policy Manager** is in charge of enforcing the *Network-oriented Policies* needed to support the applications and defined according to the associated *Application-Policy*.

To enable the platform to support multiple access networks, the **Communication Manager** is used. It allows the platform to be used in multiple contexts, from production systems, equipped with IEEE 802.11p radios, to testbeds, which can be based in any access network available.

3.5.3 Open issues

While the platform was developed with Unicast traffic support in mind, the main focus of this work is to support the development of new applications and the emulation of VANET behaviour. Because of this, most of the expended effort was focused on supporting new VANET-specific applications, as such Unicast traffic support was not implemented.

Another issue is the lack of support for the full WAVE architecture. The equipment's cost and the infancy of the standard make the full support of the standard impossible. It is possible to recreate some of a WAVE receiver behaviour, but this work did not focus on these solutions.

3.6 Global SW architecture

The complete software architecture of the platform, comprising all the modules and their inter-relations, is depicted in Figure 3.5.

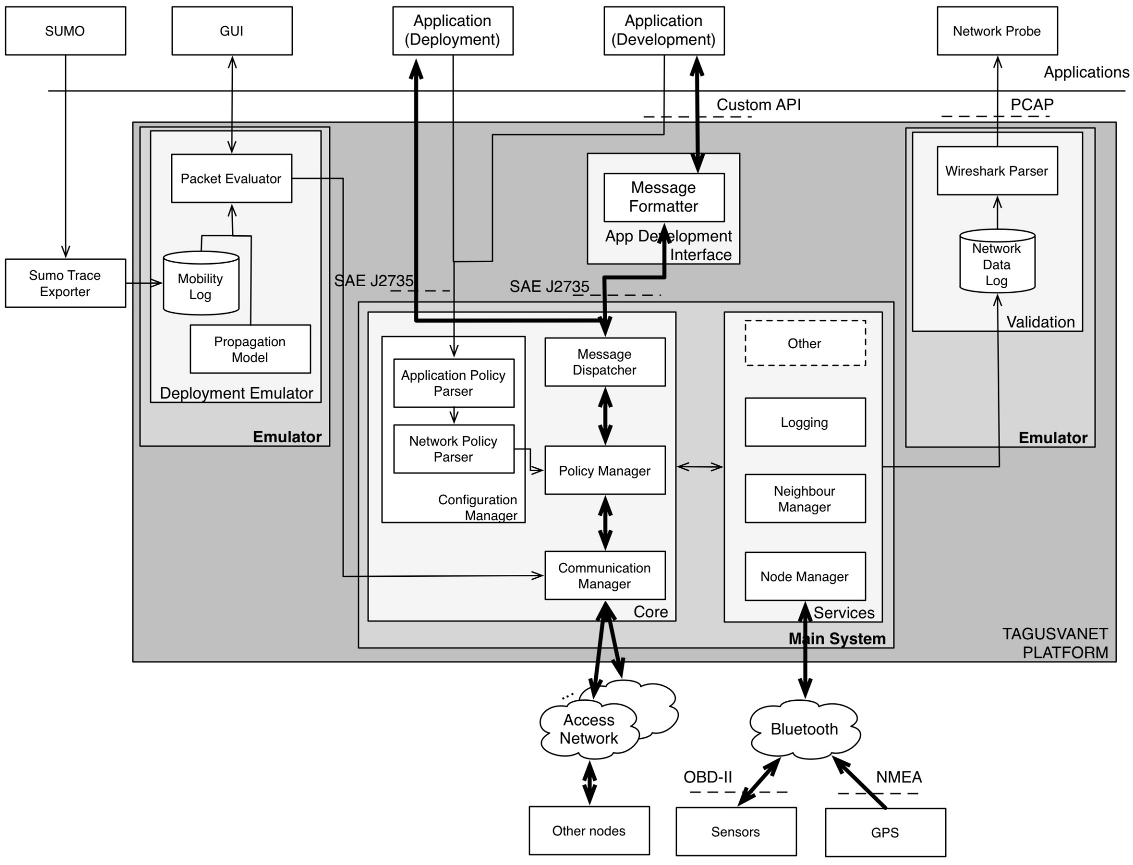


Figure 3.5: Platform architecture.

3.7 Policy-based Approach

In order to simplify the development of new VANET applications, several types of policies were defined. While some policies like the Application-oriented Policies define what an application developer has to worry about when developing a new application. Others, like the Network-oriented Policies define how each message is handled. The next sections will detailed them.

3.7.1 Application-oriented Policies

As stated in Section 2.2, VANET-oriented applications have multiple requirements. For instance, a given type of application may need to use broadcast to disseminate information during a short period of time, while, other type, may need to send data using geocast communication, for a single event. These requirements must be introduced in the platform in order to create the appropriate processing rules, at the network level.

Next, we will present the policies that were defined at the application-level.

Policy-Definition

Regardless of the purpose of the application, all VANET-oriented applications have some of the same requirements. These requirements can be expressed in the following policy:

App Name | App ID | Communication Type | <Optional>

The **App Name** field is defined by the developer and is used to provide the developer with a simple, user friendly way of identifying the application in case there is a need to update the application's requirements.

The **App ID** is used by the policy enforcer when handling messages from each application. Because it is used for each application this field has to be chosen in a way that, while it may not be user friendly, the enforcer is able to use it efficiently. A possible solution is to attribute a different integer value to each application.

The **Communication Type** field is set by the developer according to the application's addressing requirements. This field's value can be one of the following: *Unicast*, *Geocast* or *Broadcast*. These values come from the possible addressing strategies explained in Section 2.1.2.

The **Optional** field is, as the name implies, optional and can be used to specify useful information that does not fit anywhere else. This field can be repeated as necessary, the different information is separated by colons, in order to improve readability. So far this field can be one of the following: *Message Lifetime*, *Region Format*, *Region Size*, *Region Direction*. All of these fields, need to have values. For this end, the optional fields have the following format: *Field = Value*.

In order to further ease the developer's work, the most common numerical values of these fields have been converted to a small set of keywords. These keywords can be used by the developer to ease the development but these fields can also be defined using a custom numerical value if the developer sees fit. In the following tables this capability is represented by the *Custom* keyword.

In the *Message Lifetime* and *Region Size* fields, the conversion between keywords and value is shown in Table 3.1.

Keyword	Value
Short	100 ms
Medium	2 min
Long	10 min
Custom	*Input* in ms/s/min/h

Keyword	Value
Small	200 m
Medium	500 m
Wide	2 km
Custom	*Input* in m/km

Table 3.1: *Message Lifetime* and *Region Size* keywords.

The *Region Format* and *Region Direction* fields have no need for numeric values, in this case the keywords possible are shown in Table 3.2.

For the *Region Format* keywords, the destination region specification in the IEEE 1609.2 standard [27] was used and a new keyword was defined. The Sector keyword is defined as a circular sector, with

Format	Direction
Circular	Back
Rectangular	Front
Polygonal	Left
Sector	Right

Table 3.2: *Region Format* and *Region Direction* keywords.

a 45° angle.

The *Region Direction* field is to be used with the Sector keyword. This field defines the orientation of the circular sector in relation to the current heading of the source node.

Policy-Examples

This formal policy can be applied to the applications referenced in Section 2.2.

For example, in the case of an *Emergency Vehicle Warning (EVW)* application, a formal policy will take the following form:

```
EVW | 0 | Geocast | Message Lifetime = Short, Region Format = Sector, Region Size = Small, Region
Direction = Front
```

What this policy states is that the application with the **App Name** EVW uses *Geocast*-based communication and its **App ID** is 0. This policy also defines the following geographical characteristics: the region of interest encompasses all the nodes within a *200 m* long circular *Sector* in *Front* of the source node and that its messages have a lifetime of *100ms*.

For another application, say in the case of an *Accident Warning (AW)*, the policy is similar, but the *Region Format* and *Region Size* fields are different and there is no need for a *Region Direction* field:

```
AW | 1 | Geocast | Message Lifetime = Short, Region Format = Circle, Region Size = Medium
```

This policy, unlike the first, specifies an application, called *AW*, that is also *Geocast*-based, whose region of interest includes all the nodes in a *500m Circle* around the source node and whose messages have a lifetime of *100ms*.

Since not all VANET-oriented applications have the same requirements, other applications' requirements may be associated with simpler policies. For example, a *Remote Diagnostics (RD)* application has the need to address a single node, and no specific lifetime requirements, in this case the policy would take the following form:

```
RD | 2 | Unicast |
```

On the other hand, for a *Fleet Management (FM)* application, as the one described in Section 2.2, is based in *Broadcast* communication. In spite of not needing to address a specific node in the network, this application still has very specific message lifetime needs. In this case these needs are expressed using a numerical value for the message lifetime.

```
FM | 3 | Broadcast | Message Lifetime = 24h
```

3.7.2 Network-oriented Policies

Policy-Definition

As noted in the previous sections, each application has its own requirements, in terms of addressing and message lifetime. The applications also expect that every message is delivered only once and as such they don't have be prepared for duplicate messages.

These requirements can be summarised in some policies. These policies can be applied to each received message, with the objective of meeting the application's requirements and assuring the application developer of how each message is handled.

More formally all policies have the following format:

```
Id | Target | Trigger | Rules | Actions | Constraints
```

The **Id** is used to refer to the policy. The **Target** is who the message is intended to, the target can be a single node, the *Destination Node*, a *Set of Nodes*, the Region of Interest, or *all the nodes* in the source's neighbourhood.

The **Trigger** is the event that prompted the execution of this policy. The event can be either a new message is *Received*, an *Internal Action* or a *Timeout*, both generated by another policy.

The **Rules** are applied in a chain, where each rule has two possible outcomes, true or false. When a rule returns true, the next rule is evaluated, otherwise the according action is taken. In the last rule, two actions are taken, one for a positive outcome and one for the negative outcome.

An **Action** specifies what is to be done in case a **Rule** fails. In the case of the last **Rule**, two **actions** are specified, one for each possible output. The actions taken can be as simple as *Discard*, *Send to Network* or *Deliver to Application*, which don't need additional processing, or more complex actions, which are described by other policies, for example *Forward* or *Update Broadcaster*.

The **Constraints** are applied to each **Action** the policy has, these can be of any type or format that apply to the action they pertain. For example, the constraints of the *Forward* action are in the form of a probability, with possible values between 0 and 1.

Policy-types

This formal policy can be used to specify how each message is treated according to its application's needs. The following policies show how the application's needs are addressed at a network level.

The more simple case is when a message is broadcasted, a Policy for handling Broadcast messages can be:

RxBroadcast | All Nodes | Received Message | { isValid?; notDuplicate?; isSource? } | { Discard; Discard; Deliver to Application; Forward() } | { ;;; where $p = 1$ }

This Policy, named *RxBroadcast*, is targeted at *All Nodes* in the current nodes neighbourhood and is only triggered when a Broadcast message is received. Next the message is passed through a chain of rules, where some verifications are applied. The first check is if the message is *Valid*, then if it is not a *Duplicate*. If the message fails one of these checks, it is discarded. Next the current node is check if it is the *Source*, if this check fails the message is *Delivered to the Application*. Otherwise the message is *Forwarded* to all the nodes in the sources neighbourhood, with the constraint that the probability defined in Equation 2.1 is equal to one, meaning that this message is always transmitted.

In the case of Unicast, this Policy looks like this:

RxUnicast | Destination Node | Received Message | { isValid?; notDuplicate?; isNextHop?; isDestination? } | { Discard; Discard; Discard; Forward(); Deliver to Application } | { ;;; where $p = 1$; }

What this means is that the Policy, which is named *RxUnicast*, has a target, that is the *Destination Node*, which is called when a new unicast message is *Received*. This message goes through a chain of rules to ensure that the application's requirements are met. These rules verify is the message is *Valid*, *Duplicate* or if the current node is the *Destination Node*. If the message is *Invalid* or *Duplicate*, the message is discarded. Next the current node checks if it is the message's *Next Hop*, the message is discarded if it isn't for this node. The choice of *Next Node* can be done using one of the many Unicast-oriented protocols available. Next if the current node is the *Destination Node*, the message is delivered to the application it belongs. Otherwise the message is *Forwarded* to the next hop, as in broadcasting messages, these messages have the constraint that the forwarding probability is equal to one.

In Geocast, like in broadcast and unicast, every message has to be checked if it is still relevant. However when a message isn't relevant, in the case of Geocast, some actions need to be taken in order to prevent the broadcast storm problem described in Section 2.1.2. The Geocast-oriented Policy is the following:

RxGeocast | Node Set | Received Message | { isValid?; inRegion?; notDuplicate?; } | { Discard; Discard; UpdateBroadcaster(); Forward() + Deliver to Application } | { ;;; where $p = p_{ij}$ }

So the Policy named *RxGeocast* is targeted at all the *Nodes in Region of Interest* of this message. The message is then checked for its *Validity* and *Discarded* if it is *Invalid*. Next the current node checks if it is *In* the Region of Interest, if this message is irrelevant to the current node, the message is *Discarded*. Then if the message is duplicate, the *latest broadcaster* information is updated to so that the Broadcast prevention mechanism is used. Otherwise, the message is *Forwarded* to the neighbouring nodes, using the Broadcast prevention mechanism, and *Delivered* to the application it belongs.

In order to fully implement the Broadcast-Storm prevention mechanism, three more policies have to be defined. These specify the *Forward*, *UpdateBroadcaster* and *PreventDieOut* policies, that support such mechanism.

The *Forward* policy is the following:

Forward | Node Set | Internal Action | isProbOverThreshold? | { PreventDieOut(); Send To Network } | { Where $timeout = t$; }

The *Forward* policy is targeted at a *Node set* and triggered by an *Internal Action*. Next the message forwarding probability is checked *if it's above* 50%, if it is, the message is *Sent to the Network*, otherwise, the PreventDieOut policy is called.

The PreventDieOut policy is used, as its name implies, to prevent the message dying out while preventing Broadcast-Storms, as specified by Tonguz and Wisitpongphan [15].

PreventDieOut | Node Set | Timeout | Rebroadcast? | { Discard; Forward() } | { ; Where $p = 1$}

Since the *PreventDieOut* policy is only used when in Geocast, the policy is targeted at the *Set of Nodes* in the region of interest and triggered by a *Timeout*. The message is then checked if it's suitable for *Rebroadcasting*. The message is *Discarded* if it's not and *Forwarded* with a probability of one, if it is.

For a message to be validated for Rebroadcasting, it is necessary to keep track of messages rebroadcasted by other nodes. The policy that handles this behaviour is expressed in by the following:

UpdateBroadcaster | Node Set | Internal Action | isCloser? | { Discard; Discard } | { ; }

The *UpdateBroadcaster* policy, as the *PreventDieOut* policy, is targeted at the *Set of Nodes* in the region of interest and, like the *Forward* policy, triggered by an *Internal Action*. This policy checks if the message broadcaster *is Closer* to the current node than the last broadcast of the same message, in either case the message is *Discarded*.

3.8 Summary

In this chapter the proposed solution's architecture was detailed, this architecture was designed to cope with a set of requirements, which were defined in Section 3.1 as:

- **Cost-effectiveness** – in order to promote the fast dissemination of VANET technology, the platform must have a low cost, so that everyone can afford it without the need of spending too much money to buy it. Also, the process of application development, validation and test must be shortened and simplified. This means that, expensive field trials must be avoided as much as possible.
- **Extensibility** - Since VANET research is still a hot topic, the platform must be able to incorporate new breakthroughs in an easy and simple fashion. Hence, integration of new protocols and technologies must be easily achieved.

- **Flexibility** – the platform must flexible enough so that it can integrate into a unique system the creation, test and deployment of a VANET. Additionally, as there must be a significant number of applications with very different characteristics, the platform must be able to support each one of them.

These requirements shaped the design choices made and the resulting architecture is split into three main modules, the *Application Developer Interface*, the *Emulator Module* and the *Main System Module*. This solution is the basis of the remaining work.

In Chapter 4, the implementation details and decisions made in order to create the architecture proposed in this work are detailed.

Chapter 4

Implementation

This chapter addresses the main decisions taken in order to make the solution presented in Chapter 3 a reality.

This chapter is split into two main sections. In Section 4.1 the hardware architecture which supports the proposed solution is detailed, first all the components are detailed and after the possible node hardware configurations are described.

In Section 4.2 the implementation of the software architecture is described. This section is divided into three subsections, one for each module of the proposed architecture. Each of these subsections, details the implementation choices made in order to create the software modules defined in Chapter 3.

4.1 Hardware Architecture

The Software architecture defined in Section 3.6 has several requirements in terms of hardware. These requirements are: the existence of several communication interfaces available, including an IEEE 802.11p and the use of a GPS receiver which provides position information and clock synchronisation. For the vehicles, an interface for the on board sensors through an OBD-II interface was also necessary and, for RSUs, an external network interface.

Table 4.1 summarises the characteristics of the developed platform and compare them to the initial requirements established in Section 3.6.

This hardware platform is centred around a computation board, which is in charge of implementing the software platform described in Section 3.6. Given that there was no IEEE 802.11p wireless card available, the hardware platform uses two WLAN cards. To provide the location information, a GPS receiver was used, the communication between the centre board and the receiver is done using a Bluetooth piconet. The communication between the centre board and the OBD-II receiver which is used to interact with the on-board sensors, is also done using the Bluetooth piconet. In the piconet, the computation board is the master and the sensors the slaves.

Functionality	Implemented
Communication Technology	
802.11p	-
WLAN	X
UMTS	-
WiMax	-
LTE	-
Location	
GPS	X
In-Vehicle Sensing	
OBD-II	X
CAN	-
External Network Connection	
Ethernet	X

Table 4.1: Implemented functionalities of the hardware platform.

4.1.1 TS-7500

The developed hardware platform central computation board used was a Technologic Systems (TS) 7500 board. These boards are embedded computers with an ARM processor from Cavium Networks, capable of booting Linux OS. Their small size and low energy requirements makes them an attractive to VANET-oriented development. These boards run a Linux Kernel version 2.6.24.4, provided by the board's vendor.

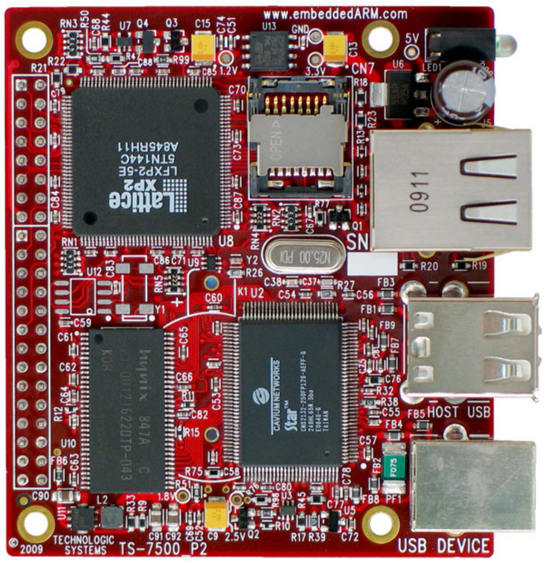


Figure 4.1: TS 7500 board.

Another advantage of these boards is the OS they use. Linux has a large developer base and therefore has a large number of development tools when compared to other embedded OS's, like TinyOS. Developing is also easier, since it is possible to write programs in well known programming languages, such as C.

The board is capable of booting a minimal Linux Kernel is less than three seconds. The filesystem is located within a SD card. The board has 64MB of RAM available as well as two USB ports and an Ethernet port, with Power-over-Ethernet capabilities. It also has another USB port used to power up the board, which means it can be powered by only five volts.

The processor in compatible with an ARM9, meaning it only supports a subset of the ARM9 Instruction Set Architecture. Finally, the board has thirty-three Digital Input/Output pins, which can be used to communicate with external devices, such as the CAN bus, present in most vehicles.

4.1.2 Wireless Cards

As stated previously two wireless cards were used to simulate the IEEE 802.11p wireless interface.

Figure 4.2: SMC Wireless Card.

These two wireless cards are used to simulate the two channels present in IEEE 802.11p, the CCH and Service CHannel (SCH). Each card is in its own channel to prevent them from interfering and these channels are non-overlapping. In spite of the use of these two channels to emulate a real 802.11p interface, it is important to note that there are still important differences between the new standard and the available technology. The most important one is the association procedure that was removed from 802.11p to mitigate the short time available for data communication in a VANET.

The used cards were SMC EZ Connect^{TM}N 150Mbps Wireless Adapters. These cards were chosen for the Ralink RT2870 chipset they possess, as the TS-7500 wireless driver is able of using all of the possible capabilities these cards. From enabling the wireless Ad-Hoc mode to forcing the cards to use only the IEEE 802.11b, 802.11g or 802.11n standard or any combination of the previous.

4.1.3 GPS Receiver

Since the developed platform requires that a node is able to access its current location, each node is equipped with a GPS receiver.

The used receiver is the Holux M-1200E Bluetooth Global Positioning System (GPS) Logger, which is equipped with BLuetooth and USB interfaces and a built-in rechargeable battery. The GPS data can be retrieved by Bluetooth or USB through serial port emulation.

This receiver can search up to 66 satellite-search channels in parallel and has a sensitivity up to -165 dBm. It has an altitude limit of 18km, speed of 515 m/s and 4G of acceleration.

The receiver has a cold start, where the cached satellite data is invalid, of 36 seconds and a warm start, where the information about the satellites in view is invalid of under 33 seconds and a hot start of under one second.

Figure 4.3: Holux GPS receiver.

The receiver output is encoded in NMEA 0183 format [59], with support for GGA, GSA, GSV and RMC sentences, the used *Datum* is WGS84. The receiver has a operating temperature range from -10 °C to +60 °C.

This receiver was chosen because of its cost, operational capabilities and available Bluetooth interface. The receiver's output is retrieved by a Bluetooth emulated serial port, using the Bluetooth piconet used to connect all the on-board sensors.

4.1.4 OBD-II interface

To interact with the on-board sensors and retrieve information about the vehicle status, an OBD-II Bluetooth interface was used. This interface uses a Serial Port over Bluetooth to receive commands and respond with the requested data.

This interface is based on the ELM237[1] OBD-II to RS232 interpreter. The interpreter supports all the mandatory OBD-II protocols [60].

Figure 4.4: ELM237 OBD-II interface.

[1] http://www.elmelectronics.com/

The receiver has an operating temperature range of -40 °C to +85 °C. The interface is powered directly by the vehicle through the OBD-II socket.

This receiver was used because of its price and large user base, meaning that a lot of documentation is available.

4.1.5 OBU and RSU Hardware Architecture

VANETs can be composed of two different types of nodes, OBUs and RSUs. Each type of node has its own hardware requirements.

An OBU needs to have access to the vehicle's sensors, using the OBD-II interface and possibly a Human Machine Interface hardware, which is used to convey the information coming from the applications to the driver. Figure 4.5 depicts the hardware needs of an OBU.

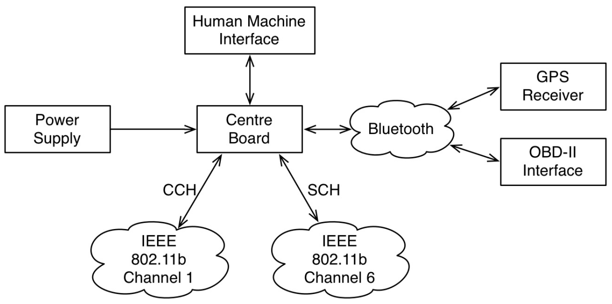


Figure 4.5: OBU hardware solution.

In the case of a RSU, there is no need for a Human Machine Interface or an OBD-II interface. There is, however, a need to provide support for the applications running on OBUs. For this, the RSUs are equipped with an wired network interface, which is used to provide access to an external network. This configuration is shown in Figure 4.6.

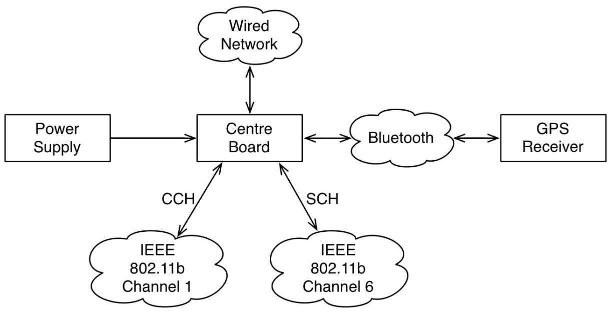


Figure 4.6: RSU hardware solution.

4.2 Software Architecture

4.2.1 Application Developer Interface

In Section 3.3 it is established that the Application Developer Interface is required to enable the easy development of applications and that applications have to be standard-compliant.

To achieve these goals a **Custom API** was created, which abstracts the creation of standard-compliant messages. This *Custom API* is integrated in a library, the SAE J2735 Library, which also contains the **Message Formatter**. The *Message Formatter* is used to address the requirement of applications using standard-compliant messages, by supplying a number of ASN.1 encoding and decoding facilities. This library can be imported by the developer into his application, in the same way another library is imported.

Custom API

The Custom API is used by the developer to interact with the **TagusVanet** platform. It supplies the developer with facilities to retrieve information from the vehicle's sensors and to abstract the creation of standard-compliant messages. It also supplies a way for the developer to inform the platform of the application's needs and enables the developer to request to be notified when a neighbour appears or disappears.

The API is composed of the following functions:

```
int register_application(char* filename);
uint8_t get_app_id(char* app_name);
int register_message_callback(uint8_t app_id, void (*message_hdlr)(char* msg, int len));
void send_message(uint8_t app_id, char* msg, int len, char* destination);
int register_new_neighbor_callback(uint8_t app_id, void (*new_neighbour_hdlr)(char* neighbour_id));
int register_remove_neighbor_callback(uint8_t app_id, void (*remove_neighbour_hdlr)(char* neighbour_id));
void get_current_location(Location* location);
void get_current_time(GPSTime* gps_time);
uint16_t get_current_rpm(void);
uint16_t get_current_speed(void);
```

The *register_application* function is used by the developer to inform the platform of its application's needs, the developer supplies a filename from where the requirements are read. The requirements are specified in JSON using the following format:

```
{
  "app_name":"<App_Name>",
  "app_id":<App_ID>,
  "properties":{
    "communication_type":"<Unicast, Geocast, Broadcast>",
    "region_of_interest":{
      "format":"<Circle, Rectangle, Polygon, Sector>",
      "direction":"<Front, Back, Left, Right>",
      "size":"<Small, Medium, Large, Custom>"
    },
    "event_lifetime":"<Short, Medium, Long, Custom>"
  }
}
```

This format, for specifying the application's requirements, is the representation of the Application-level policies specified in Section 3.7.1. After the application has informed the platform of its needs, it can call the *get_app_id* function to convert from the user friendly application name to the application id, which the platform uses to identify the application.

For now, applications wishing to receive messages from the platform need to supply a callback for when a message arrives. This is accomplished through the *register_message_callback*. On the other hand, when an application wishes to send a message, the *send_message* function is supplied. This function receives the message to be sent and the message length. For Unicast communication this function also receives the destination node's identification.

Given the cooperative nature of VANETs, the API also allows the applications to be notified when a new neighbour appears or when a neighbour leaves. An application can request to be notified when a neighbour appears by using the *register_new_neighbour_callback*, the registered callback is called with the new neighbour's id. The same applies to the *register_remove_neighbour_callback*.

Finally, the Custom API, provides a number of utility functions which allow the application to access the information from the vehicle's sensors. The *get_current_location* and *get_current_time* functions supply the information gathered from the GPS receiver. The *Location* structure has Latitude, Longitude and Altitude fields and the *GPSTime* structure provides all the time related information from year to second. To access the OBD-II interface information, the *get_current_rpm* and *get_current_speed* functions are supplied.

SAE J2735 Library and ASN.1 Compiler

To ease the creation of new applications, a statically linked library was created, for the developer to use when building an application.

This library gathers all the code needed to support the *Custom API* described previously and also the code that supports the *Message Formatter*, which is in charge of encoding the SAE International (SAE) J2735 compliant messages.

Since no translation of the SAE J2735 standard was available in C, the open source tool ASN.1 Compiler (ASN1C)[2] was used. The ASN1C creates the code that supports the standard data structures

[2]http://lionet.info/asn1c/compiler.html

and also provides encoding and decoding functionality.

The ASN1C tool supports a wide range of ASN.1 syntaxes, namely Basic Encoding Rules (BER), Packet Encoding Rules (PER) and XML Encoding Rules (XER), and also generates code for constraint validation. It also support a wide number of platforms, meaning the library can be used in multiple platforms for testing purposes.

Open Issues

One issue that is not addressed is the use of standardised APIs to allow the developer to interact with the platform. While not all the required functionality in the *Custom API* can be expressed by standard APIs, like POSIX, one of the most important aspects can, message sending and delivering, these can be standardised through the Sockets API.

4.2.2 VANET Emulator Module

As stated in Section 3.4, one of the requirements of the **VANET Emulator module** is the use of well-known tools and realistic test conditions.

To be possible for the developers to use Wireshark for traffic analysis a new dissector was created. The Wireshark tool and the dissector are detailed in the following sections.

To provide simulated node mobility patters, the SUMO simulator is used and a tool, for converting the simulator's output traces to the NMEA standard, was created. These are explained in more detail after.

Finally, the most relevant details of the propagation models implementation are also presented.

Validation component

Wireshark Wireshark[3] is the world's most popular network analyser [61], it is an open source community-driven tool and is available in most common Operating Systems.

The analysed traffic can come from two sources, live traffic capture or previously captured traffic retrieved from files. Packet capture in Wireshark is done according to the OS it is running on, in *NIX hosts, traffic capture is done using the industry standard *libpcap* [61]. In the case of Windows systems, traffic is captured using the windows port of *libpcap*, WinPCAP.

After the packets are captured, they are processed by the core engine and the correct dissector is chosen, according to the protocol used. Finally the dissected packet is displayed to the user through a Graphical User Interface (GUI). In Figure 4.7 this process is depicted.

In Wireshark, the core engine is used to ensure that dissectors are portable and that the packets are processed in the same way on every host. This means that the core engine supplies a number of facilities that can be used to access information in a packet in a portable way. The core engine is also in charge of calling dissectors as needed. For example, after an IPv4 Header dissector has finished parsing a packet, the core engine choses the appropriate dissector to parse the IPv4 packet payload.

[3]http://www.wireshark.org/

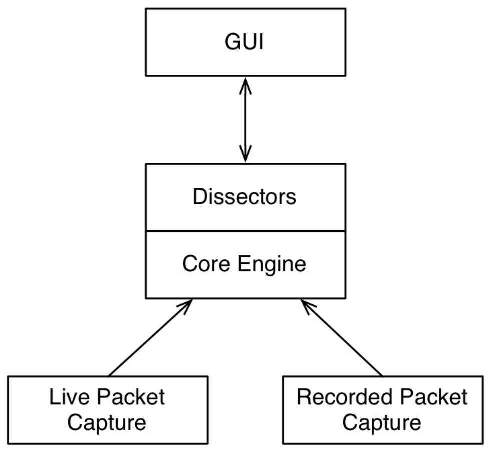


Figure 4.7: Wireshark traffic analysis flow.

Dissectors are used to parse the information in a packet, they use the facilities provided by the core engine to access information and supply the meaning of that information to the GUI, which then supplies it to the user. Dissectors can also enable the analysis of multiple packets in a stream, for example, a TCP dissector can be able to reconstruct the complete data stream, in order to help the user understand what is happening.

In addition to these capabilities, Wireshark also has a large user-base and extensive documentation available.

Wireshark Dissector Since the information crossing the platform can be saved in PCAP format, a simple and easy to used solution has to be available to the developer, to address this issue a Wireshark dissector was created.

This dissector allows the analysis of the captured information, easing the debugging of the applications being developed and the research being done with the platform. This dissector uses the ASN.1 definition of the SAE J2735 Message Set, to analyse each received packet and present the information in it to the developer.

Deployment component

SUMO The SUMO simulator [62] can be used to simulate vehicle mobility.

The simulator uses a set of configuration files and support tools to create the road network and the vehicles trips. The road network can be created in three possible ways, as shown in Figure 4.8, the network can be manually designed, imported from other network definitions or by generating a new network.

In a manually designed networks the user creates a set of network description files and feed's them to the **Netconvert** tool. These files, written in XML, describe the network nodes and how they are connected, the *Node* and *Link* files in Figure 4.8 respectively. Optionally the user can specify different types of links, for example rural road or highway, using the *Link Type* file and different lane connections in special intersections through the *Lane Connection* file, as in Figure 4.8.

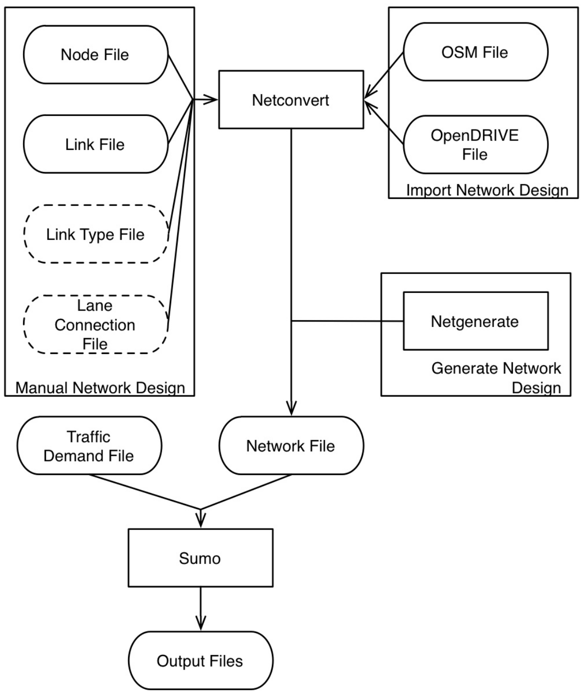


Figure 4.8: Sumo simulation flow.

Network descriptions can also be imported from other formats, using the **Netconvert** tool. Two of them are the OpenDRIVE[4], which is an open standard for the logical description of road networks, and the Open Street Map[5], where it is possible to select a portion of a map to export. In Figure 4.9, this process, using the Open Street Map export tool, is exemplified. This process still requires, sometimes, some manual editing of the exported file, but it has the added value of representing real road scenarios.

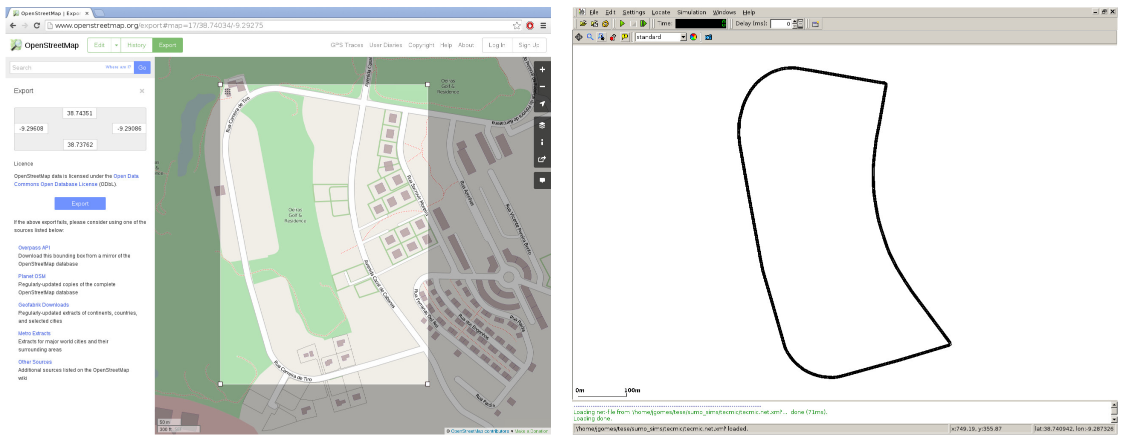


Figure 4.9: Sumo network import example.

The final way of creating a network, is through generation. This is accomplished using the **Netgenerate** tool. The **Netgenerate** can create abstract networks, these can be one of three types, grid spider or random networks. An example of each of these networks in shown in Figure 4.10.

After the *Network* file is created, the user creates a *Traffic Demand* file, where the routes vehicles

[4]http://www.opendrive.org/
[5]http://www.openstreetmap.org/

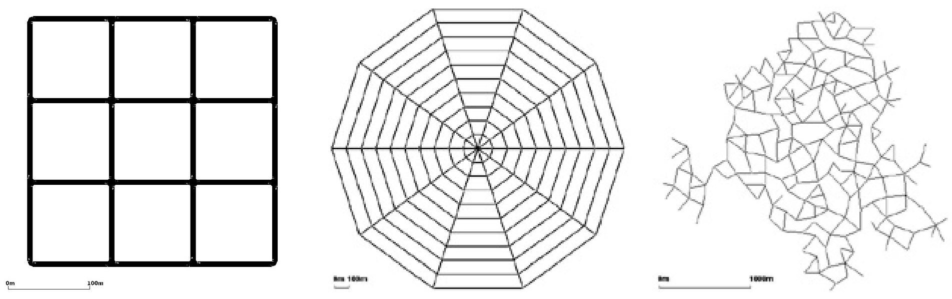


Figure 4.10: Sumo generated networks example.

are to take are described. Then the user supplies the **Sumo** simulator with these two files.

After the simulation is done, the **Sumo** simulator can save the output in a number of formats. The output formats can range from microscopic information, information regarding each vehicle status at every step of the simulation, to macroscopic information, information regarding lane-wise traffic and queue information throughout the simulation.

SUMO trace exporter While SUMO can be used to simulate traffic it doesn't have the capability of exporting such information in NMEA 0183 format, which can then be fed to the platform.

To overcome this issue a new tool was created, called the **SUMO Trace Exporter**. This tool reads the data from the SUMO trace file, splits the information according to VehicleID and then creates a GPS trace file for each vehicle. The GPS trace file is formatted in NMEA 0183 format so that the data can be fed to the platform, using this standard.

The SUMO simulator exports data in what it calls a *gpsdat* format, where the vehicleID, time, the node's position and current speed are detailed, as in this example:

```
34  2014-03-04 02:43:17.840688  -9.290986  38.738414  0  0.0  0.000000
```

From the shown *gpsdat* information exported by SUMO, this tool created the following NMEA sentences:

```
$GPGGA,024318.840,3844.3056,N,00917.4598,W,2,9,1.43,0,M,50.7,M,0000,0000*5a
$GPRMC,024318.840,A,3844.3056,N,00917.4598,W,11.090800,0.00,1434,,,D*4e
```

Propagation models To enable the use of realistic test conditions, the propagation models described in Section 2.6, were used. All of the attenuation models are implemented and at the disposal of the developer. From the packet error models, only the simplest 1Mpbs error model was used, Equation 2.3, this is because it has the simplest calculations, in order to not overload the system, and it represents the more conservative communication solution, where the communication between nodes is slower but more error resistant. In order to convert the calculated PER to an ACCEPT/DROP action, a random value is generated per packet and is compared with the calculated PER. The outcome of this comparison is then reported to the *Main System Module*, to allow the appropriate action to be taken.

Open Issues

One aspect of this module that can be improved are the propagation models, so far they only take into account the shadowing effect caused by other vehicles. This is useful when emulating environments which don't have many buildings, for example a highway scenario, but falls short in urban scenarios, rich in buildings that affect signal propagation.

4.2.3 Main System Module

The Main System Module, is the basis of the platform and given its importance, a number of requirements have to be addressed. In Section 3.5, it was established that these requirements are the following:

- Configurability
- Support product differentiation
- Support multiple addressing schemes
- Geocast support
- Support multiple communication technologies
- Integration in existing vehicles
- Support application data exporting

In the following sections how these requirements were addressed is detailed. Some requirements were not fully addressed, these will be detailed in the final section, *Open Issues*.

The Main System Module is divided into two submodules, the **Core Module**, which is the core of the platform, and the **Services**, which provides support the *Core Module* and the *Applications*.

SAE J2735 Message Examples

In order to be standard compliant, the platform uses the SAE J2735 standard for its messages format.

As specified in the standard, in the CCH, the heartbeat message, the BSM, is used. Since the CCH is reserved for critical information only the platform can send messages in this channel.

The BSM's definition in ASN.1, as per defined in the SAE J2735 message standard, is the following:

```
BasicSafetyMessage ::=  SEQUENCE {
  -- Part I
  msgID       DSRCmsgID,                -- 1 byte
  -- Sent as a single octet blob
  blob1       BSMblob
  -- The blob consists of the following 38 packed bytes:
  --
  -- msgCnt      MsgCount,             -x- 1 byte
  -- id          TemporaryID,          -x- 4 bytes
  -- secMark     DSecond,              -x- 2 bytes
  -- pos      PositionLocal3D,
    -- lat       Latitude,             -x- 4 bytes
    -- long      Longitude,            -x- 4 bytes
    -- elev      Elevation,            -x- 2 bytes
    -- accuracy  PositionalAccuracy,   -x- 4 bytes
  -- motion   Motion,
    -- speed     TransmissionAndSpeed, -x- 2 bytes
    -- heading   Heading,              -x- 2 byte
    -- angle     SteeringWheelAngle    -x- 1 bytes
    -- accelSet  AccelerationSet4Way,  -x- 7 bytes
  -- control  Control,
  -- brakes      BrakeSystemStatus,    -x- 2 bytes
  -- basic    VehicleBasic,
  -- size        VehicleSize,          -x- 3 bytes
}
```

For the applications to transmit their data, the ALCM is used. As stated previously, this is a generic message to be used by applications as they see fit. The following ASN.1 definition is the one used in the platform and created by the *Custom API*. It contains all the needed information for each node to handle the message according to the application's requirements. The payload field is filled with the information the application desires.

```
AlaCarteMessage ::=  SEQUENCE {
  msgID               DSRCmsgID,            -- the message type (defined in the standard)
  appID               Count,                -- the application id
  CHOICE {
    hopCount          Count,                -- the message's hop count for unicast
    lifetime          TerminationTime       -- the message's lifetime for broadcast and geocast
  },
  msgCount            MsgCount,             -- used to check for duplicates
  senderID            TemporaryID,          -- the sender's temporary id
  destinationID       TemporaryID OPTONAL,  -- the destination's temporary id for unicast
  CHOICE {
    nextHopID         TemporaryID,          -- the next hop's temporary id for unicast
    lastBroadcasterID TemporaryID           -- the last broadcaster's temporary id for geocast
  } OPTIONAL,
  destination         ValidRegion,          -- the destination region
  payload             PayloadData           -- the message's payload
}
```

The use of this standard allows the platform to support applications from multiple vendors, given that the ALCMs have the fields specified previously.

This standard also allows the messages to be exported in a structured way for later use. These messages can be exported in simple text or PCAP format, compatible with the Wireshark dissector, described in Section 4.2.2.

Core components

Policy parsing and enforcing In the *Core Module*, the policies defined in Section 3.7 are implemented. This module has an inner module, the **Configuration Manager**, which is responsible for parsing the application's requirements, supplied by the application using the *Custom API* described earlier.

The *Configuration Manager* receives the Application Policy from the application and first checks if it is correctly formatted. After the policy's format is verified, it is parsed and the requirements specified are used to create the corresponding network policy's configuration. While in Section 3.5, the *Configuration Manager* is split into two different submodules, in the platform's implementation, these were combined. Using this mechanism, the application is able to configure the platform on how its messages are to be handled. This information is then used by the **Policy Manager**.

In Section 3.5, there is an entity called the **Message Dispatcher**, which in the current implementation, its main functionality is integrated into the other components of the *Core Module*. The **Communication Manager**'s main goal is interfacing with the access network and abstracting it from the rest of the platform. When a new message arrives from the network, the *Communication Manager* identifies its type and handles it accordingly, as shown in Figure 4.11. In this case, message reception, the *Message Dispatcher*'s functionality is inserted into the *Communication Manager*.

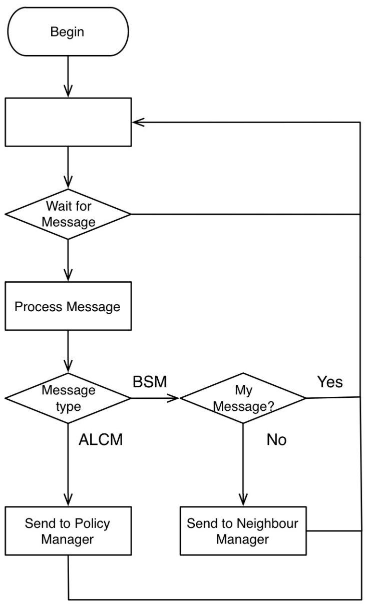


Figure 4.11: Communication Manager / Message Dispatcher message handling.

In this case, the BSMs from the current node are discarded, because their information is available from the node manager, otherwise that message is given to the Neighbour Manager.

Next, when the message is an ALCM, it is given to the **Policy Manager**, which is in charge of enforcing the network policies defined in Section 3.7.2. The *Policy Manager* then checks if the application, whom the messages belong, is registered in the platform, if there is no record of that application, the message is discarded. After the message's application has been identified, the corresponding network policy is applied according to the communication type defined by the application policy. This behaviour is shown in Figure 4.12.

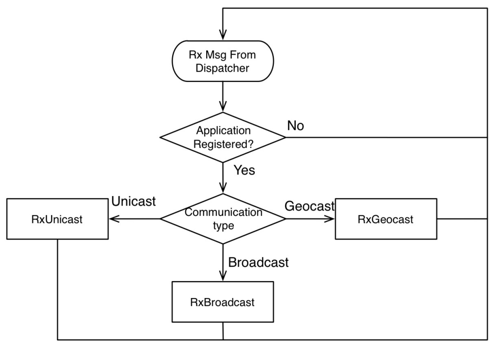


Figure 4.12: Policy Manager's decision process.

As described in Section 3.7.2, each of these policies has the following behaviour.

The **RxUnicast** policy first checks if the message's hop count is still within the limit defined by the sender, if the hop count is invalid, the message is discarded and the policy ends, returning to the *Policy Manager*. Then the message is verified if it is a duplicate and if it is, the message is discarded and execution returns to the *Policy Manager*. Next the message's next hop is verified, if the current node isn't the next hop, the message is discarded. Finally, the policy checks is the current node is the destination node, if this is true, the message is delivered to the application, otherwise, the next hop is chosen according to the current chosen unicast routing protocol and the message is sent to the network. Figure 4.13 summarises this decision process.

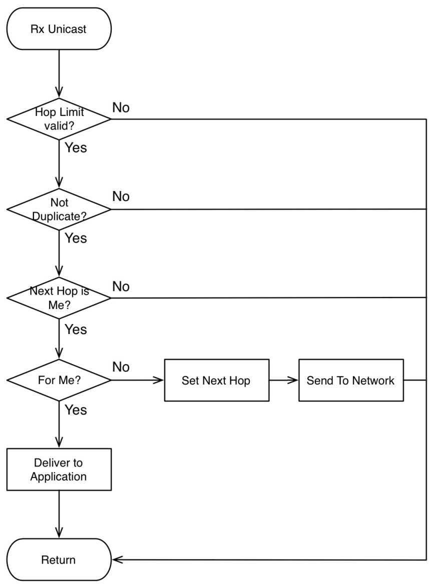


Figure 4.13: RxUnicast policy's decision process.

If the message belongs to an application which is broadcast based, the first thing the **RxBroadcast** policy checks is if the message is still valid according to the message lifetime defined by the application. Next if the message is a duplicate, it is discarded and the current execution of the policy ends. After, if the message is from the current node it is sent to the network, otherwise the message is delivered to the corresponding application. In Figure 4.14 this process is summarised.

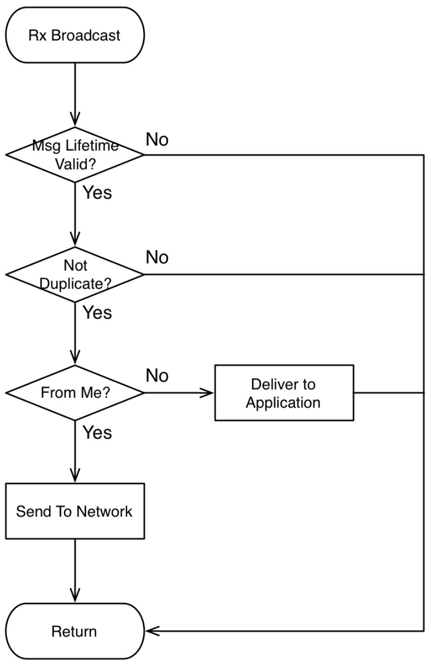


Figure 4.14: RxBroadcast policy's decision process.

The more complicated case of a geocast based message is depicted in Figure 4.15. When a new message of this type arrives, the message's lifetime is checked, according to the application's requirements, if the message is invalid it is discarded. Next, the policy checks if the current node is in the message's destination region. If the current node is in the destination region, the policy assess if the message is a duplicate, if this is true, the latest broadcaster information is updated, then the message is discarded. Otherwise the message is delivered to the application it belongs and then forwarded to the current neighbours, according to the broadcast prevention mechanism described in Section 3.7.2

Service components

Sensor communication The *Node Manager* is in charge of communicating with the on-board sensors and assessing the vehicle's capabilities. The communication is done using a Bluetooth piconet and an emulated serial port. The *Node Manager* is build in a way that it makes use of the file emulation provided by the Linux system and treats both sensors as files. This advantage, and the use of standardised interfaces to communicate with the sensors, however doesn't make the *Node Manager* independent from the installed sensors.

The main dependency is the OBD-II interface, while it is standardised, not all vehicles have the same sensors available. More recent vehicles have more information available than older vehicles.

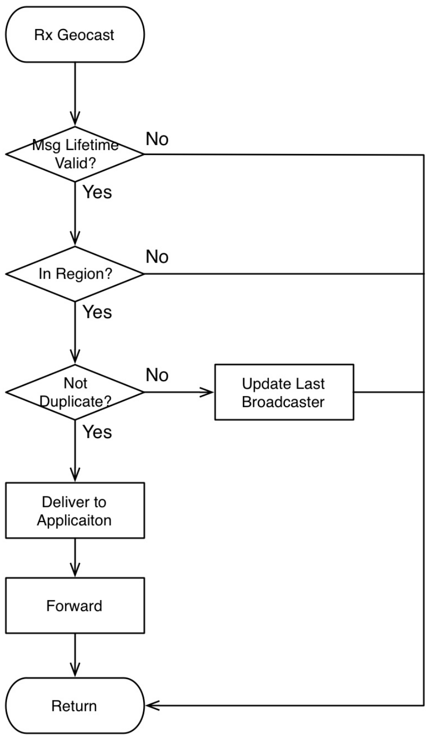


Figure 4.15: RxGeocast policy's decision process.

This is because the standard defines the behaviour and message format the interface must have, if the manufacturer choses to abide by it.

In the European Union, in an effort to decrease vehicle pollution and to allow the diagnosis, service and repair of a vehicle with little hindrance throughout its territory, an European Directive was approved in 1998 [63]. This requires that by 1 of January of 2002 and 1 of January of 2005, all petrol and diesel vehicles, respectively, come equipped with this standard. But there are still many differences in the supported information by each manufacturer, in spite of this, most support the retrieval of information related to the vehicle's speed and engine RPM. Given this, only these two pieces of information are available in the platform.

Implemented services As stated in Section 3.5, the *Main System Module* has a **Services** submodule. This submodule supports the *Core* and the applications, this was achieved by implementing three modules, the **Node Manager**, **Neighbour Manager** and the **Logging**. The *Node Manager* and the *Neighbour Manager* are essential for the platform's function and as such, their existence is mandatory. The *Logging* service is optional and exists to supply the application developer with the log information they need for debugging and to prepare data for exporting.

The *Node Manager* interacts with the on-board sensors, this service's implementation is dependant on the sensors used and is in charge of abstracting the sensors from the rest of the platform. This information is stored in the format that is supplied to the applications. The location and time related information is stored using the following structures:

```
typedef struct _location {
  double Latitude;
  double Longitude;
  double Elevation;
} Location;

typedef struct _gps_time {
  int Year;
  int Month;
  int Day;
  int Hours;
  int Minutes;
  int Seconds;
} GPSTime;
```

The vehicle's engine information is stored in the following format:

```
uint16_t obd_speed;
uint16_t obd_rpm;
```

This information can be retrieved by applications using the Custom API defined previously.

The *Neighbour Manager* is the receiver of the BSM sent in the CCH. Using this information, the *Neighbour Manager* keeps a record of the node's current neighbours and this record is soft state based. The record is composed of the received BSM, which can be used to retrieve the neighbours information, and a timer, as shown in Table 4.2.

Received BSM	Neighbour timer

Table 4.2: Neighbour manager's record format.

This timer is used for the soft state and is reset every time a new message is received. This means that after a predefined number of seconds without contact with a specific neighbour, its record is deleted. The time threshold used for the soft state timer, for now, is set when the platform starts and cannot be changed while the platform is running. The used time threshold in the development of the platform was 10 seconds.

The *Neighbour Manager* is also in charge of notifying, the applications that request it, of the appearance and disappearance of a neighbour. In order to allow the application to distinguish between neighbours and make choices depending on which neighbours it has, the neighbour's temporary id is also supplied when they are notified.

Open Issues

While a lot of effort was put into developing this platform, some issues are still to be addressed.

As shown in this section, the platform was developed and is prepared for Unicast based communication. However, some issues still have to be addressed. Several Unicast routing protocols exist for VANET communication and this issue can be addressed by creating a dedicated *Unicast Routing* service. Service which would chose the appropriate next hop, according to a specific protocol, possibly chosen by the application, among many available. Also in order to support this service, another is needed, a *Location* service, which can itself implement one of the many location solutions available for VANETs. Like in the *Unicast Routing* service, the *Location* service's choice of solution can possibly be chosen by the application. These two services working together would allow the platform to support Unicast-based communication, along with the others currently supported.

Another issue, is the need for applications that supply the platform directly with SAE J2735 messages, have these messages formatted as defined previously. While that format allows the platform to enforce the defined policies, there is no requirement for ALCMs to be formatted as defined. This raises the problem of how the platform should handle messages that aren't in the expected format. This issue remains to be addressed with a proper solution.

As stated previously, in Section 3.5.3, so far the platform does not support the full WAVE architecture. Nor does it support multiple communication technologies. For now the platform only supports IEEE 802.11b based communication, this solution was chosen given its ability to support V2V and V2I communications and low cost. A WAVE based solution was discarded given the price of each network interface. The other communication technologies, described in Section 2.3.3, were not used because they fail to support all the needed communication types, more specifically V2V.

4.3 Summary

In this chapter the implementation choices were described. The chapter begins with a detailed description of the hardware architecture, first the components by themselves and next the possible hardware configurations.

Then the implemented software architecture is described, this work tries to address as much functionalities as possible. When some feature is not supported, one possible solution is presented.

At the end of this chapter, a working implementation of the proposed solution is available. This solution meets the requirements specified in the previous chapter, its cost-effective, using only commercial off-the-shelf hardware, extensible and flexible, thanks to a modular architecture that allows the applications to choose what features to use according to their needs.

In the next chapter this solution is tested and some proof of concept applications are detailed.

Chapter 5

Evaluation

In order to evaluate the developed platform, several tests were performed. The tests were conducted in both real and emulated scenarios, in order to access the usability of the features included in the platform. All the tests were performed using the hardware architecture described in Section 4.1 and the software platform detailed in Section 4.2. The following sections detail these tests. In Section 5.1, the emulator-specific tests are detailed and in Section 5.2 the platform-wide tests are explained.

5.1 Emulator Module

Given the nature and purpose of the emulator module, several tests were performed in order to assess its behaviour.

5.1.1 Test Description

The tests performed on the emulator module were separated into tests regarding the created Wireshark dissector, the propagation models and the behaviour of the platform using the modules.

Wireshark tests To test the Wireshark dissector a mock packet capture was created and then supplied to the dissector. This packet capture featured a number of different messages, including the BSM used in the CCH, with different options to test all the aspects of the dissector.

Propagation models tests In order to test how each model behaves, each model was tested by itself. For each the physical characteristics of the 802.11b physical layer were used, in this case a frequency of 2.4GHz and a channel bandwidth of 20MHz. The test consisted on varying the value of the distance between sender and receiver. In the case of the SKM, it was assumed that the obstructing vehicle was positioned between the sender and the receiver and had the same height as the communicating parties. To reduce the effect of the random value chosen in for each packet, for each value of the distance, a million random numbers were generated, each representing a new packet, and each was tested to ascertain if the corresponding packet would be accepted or dropped.

After the models had been tested independently, the models were tested in conjunction with the platform. To understand how the models and the platform behaved in laboratory conditions, three tests were performed. In all three tests, the nodes were emulated to be stationary and within LOS.

In the first test, the impact of the propagation models in the CCH was studied, for this the only traffic generated were the beacons, used by the platform. The test started with only two nodes communicating and after five minutes a third node was added, after another five minutes a fourth node was added. In this test, two propagation models were used, FSM and SKM, the latter configured as in the previous test. These models were chosen for their representation of both the best and worst possible communication conditions. The FSM represents the best, most ideal, propagation conditions and the SKM represents a situation were communication is semi-obstructed.

For the second test, a service announcement application, which uses broadcast communication, was emulated, the chosen model was FSM. This application sent a new message through the SCH every second, in an attempt to mimic the requirements, established in [17], for such applications. In this test, as in the previous, there were only two nodes in the beginning and after five minutes another node joined the network, as did another node after five minutes.

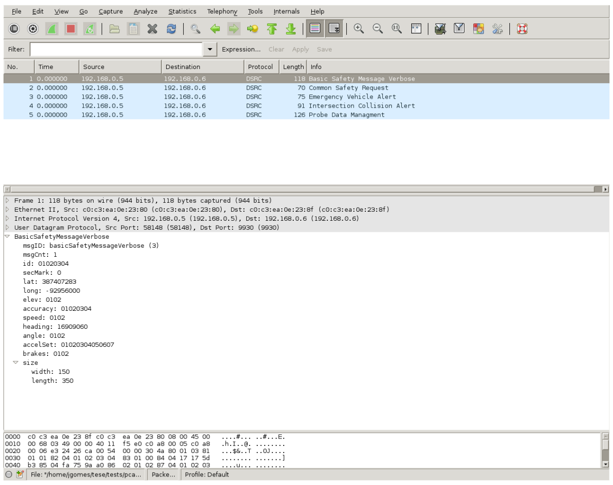
The third test, emulated an accident warning application, which used geocast communication, and FSM was also the used model. This accident warning application generated a message every 100ms, as specified in [17]. As with the previous two tests, there were two nodes in the beginning and two other nodes joined the network after five minutes each.

5.1.2 Test Results

Wireshark tests results As stated previously in order to test the created Wireshark dissector, a mock packet capture was created, in Figures 5.1 and 5.2 a couple of screenshots of the dissector in action are presented, these images are available in Appendix A with more resolution. Figure 5.1 shows the dissection of a Verbose Basic Safety Message, which is defined by the standard as a testing only message, and Figure 5.2 shows a Probe Data Management Message, which is composed of mandatory and optional fields.

Propagation model test results After each model was tested individually in the previously detailed setting, for each value of distance, the Packet Delivery Ratio was computed. This computation consisted of summing all the "would-be" delivered packets, for a distance value, and dividing that value for the number of test packets for that value. The resulting PDR graphs are depicted in Figure 5.3.

From these results it is possible to conclude that, as stated in [54], at the distances nodes in a VANET are able to communicate, the difference between FSM and TRGM is zero. From these results, it is also concluded, that the results for FSM are consistent with the findings in [52], where the authors found that the PDR experiences a sudden and steep drop at a distance of 140m. Another result shown in this graph, is the similarity between the EFSM and SKM, this can result from the specialised nature of EFSM to the vehicular environment, given that all the attenuation models were configured to emulate an operating frequency of 2.4GHz, instead of the 5.9GHz the model was designed to use.



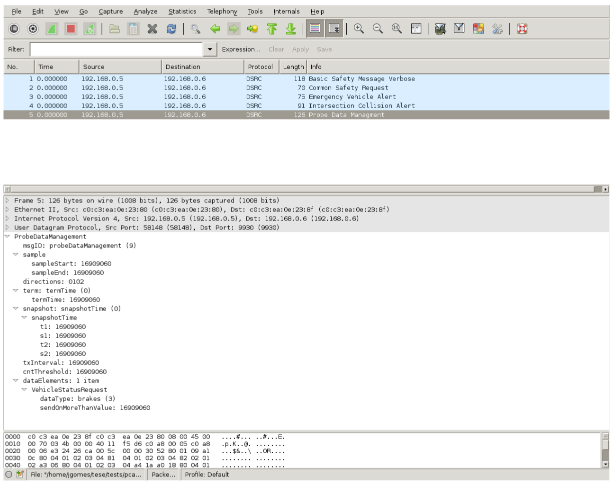
```
No.  Time      Source        Destination   Protocol Length Info
  1 0.000000  192.168.0.5   192.168.0.6   DSRC        118 Basic Safety Message Verbose
  2 0.000000  192.168.0.5   192.168.0.6   DSRC         70 Common Safety Request
  3 0.000000  192.168.0.5   192.168.0.6   DSRC         75 Emergency Vehicle Alert
  4 0.000000  192.168.0.5   192.168.0.6   DSRC         91 Intersection Collision Alert
  5 0.000000  192.168.0.5   192.168.0.6   DSRC        126 Probe Data Managment
```

```
▷ Frame 1: 118 bytes on wire (944 bits), 118 bytes captured (944 bits)
▷ Ethernet II, Src: c0:c3:ea:0e:23:80 (c0:c3:ea:0e:23:80), Dst: c0:c3:ea:0e:23:8f (c0:c3:ea:0e:23:8f)
▷ Internet Protocol Version 4, Src: 192.168.0.5 (192.168.0.5), Dst: 192.168.0.6 (192.168.0.6)
▷ User Datagram Protocol, Src Port: 58148 (58148), Dst Port: 9930 (9930)
▽ BasicSafetyMessageVerbose
    msgID: basicSafetyMessageVerbose (3)
    msgCnt: 1
    id: 01020304
    secMark: 0
    lat: 387407283
    long: -92956000
    elev: 0102
    accuracy: 01020304
    speed: 0102
    heading: 16909060
    angle: 0102
    accelSet: 01020304050607
    brakes: 0102
  ▽ size
      width: 150
      length: 350
```

```
0000  c0 c3 ea 0e 23 8f c0 c3  ea 0e 23 80 08 00 45 00   ....#... ..#...E.
0010  00 68 03 49 00 00 40 11  f5 e0 c0 a8 00 05 c0 a8   .h.I..@. ........
0020  00 06 e3 24 26 ca 00 54  00 00 30 4a 80 01 03 81   ...$&..T ..0J....
0030  01 01 82 04 01 02 03 04  83 01 00 84 04 17 17 5d   ........ .......]
0040  b3 85 04 fa 75 9a a0 86  02 01 02 87 04 01 02 03   ....u... ........
```

File: "/home/jgomes/tese/tests/pca... | Packe... | Profile: Default

Figure 5.1: Wireshark screenshot showing the contents of a Basic Safety Message Verbose.



```
No.  Time      Source        Destination   Protocol Length Info
  1 0.000000  192.168.0.5   192.168.0.6   DSRC        118 Basic Safety Message Verbose
  2 0.000000  192.168.0.5   192.168.0.6   DSRC         70 Common Safety Request
  3 0.000000  192.168.0.5   192.168.0.6   DSRC         75 Emergency Vehicle Alert
  4 0.000000  192.168.0.5   192.168.0.6   DSRC         91 Intersection Collision Alert
  5 0.000000  192.168.0.5   192.168.0.6   DSRC        126 Probe Data Managment
```

```
▷ Frame 5: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
▷ Ethernet II, Src: c0:c3:ea:0e:23:80 (c0:c3:ea:0e:23:80), Dst: c0:c3:ea:0e:23:8f (c0:c3:ea:0e:23:8f)
▷ Internet Protocol Version 4, Src: 192.168.0.5 (192.168.0.5), Dst: 192.168.0.6 (192.168.0.6)
▷ User Datagram Protocol, Src Port: 58148 (58148), Dst Port: 9930 (9930)
▽ ProbeDataManagement
    msgID: probeDataManagement (9)
  ▽ sample
      sampleStart: 16909060
      sampleEnd: 16909060
    directions: 0102
  ▽ term: termTime (0)
      termTime: 16909060
  ▽ snapshot: snapshotTime (0)
    ▽ snapshotTime
        t1: 16909060
        s1: 16909060
        t2: 16909060
        s2: 16909060
    txInterval: 16909060
    cntThreshold: 16909060
  ▽ dataElements: 1 item
    ▽ VehicleStatusRequest
        dataType: brakes (3)
        sendOnMoreThanValue: 16909060
```

```
0000  c0 c3 ea 0e 23 8f c0 c3  ea 0e 23 80 08 00 45 00   ....#... ..#...E.
0010  00 70 03 4b 00 00 40 11  f5 d6 c0 a8 00 05 c0 a8   .p.K..@. ........
0020  00 06 e3 24 26 ca 00 5c  00 00 30 52 80 01 09 a1   ...$&..\ ..0R....
0030  0c 80 04 01 02 03 04 81  04 01 02 03 04 82 02 01   ........ ........
0040  02 a3 06 80 04 01 02 03  04 a4 1a a0 18 80 04 01   ........ ........
```

File: "/home/jgomes/tese/tests/pca... | Packe... | Profile: Default

Figure 5.2: Wireshark screenshot showing the contents of a Probe Data Management Message.

After each model was tested, all models were integrated into the platform, then a number of tests were performed. In these tests, the maximum number of nodes is limited to four (4) due to hardware constraints. As described earlier, in the first test, the impact of the propagation models is assessed.

In Figure 5.4, the impact of two models in the CCH delivery success rate is depicted. The two tested models were FSM and SKM, because these represent the most favourable and unfavourable conditions.

As expected, the more conservative model has a greater impact on the delivery success rate. Another result, this test shows is that when the number of nodes is increased, the success rate is decreased, meaning that the number of packet collisions increases and this may lead to critical information being lost.

In Figure 5.5, the outcome of the tests designed to measure the impact of SCH traffic in the CCH is shown. In both tests the chosen model was FSM given, as stated previously, its similar behaviour with real field conditions.

Due to an issue with the logging, only the information regarding the CCH is available. When this problem was detected, it was impossible to repeat the tests because the used equipment started to show signs of extensive use and stopped working, in some cases correctly, in other completely.

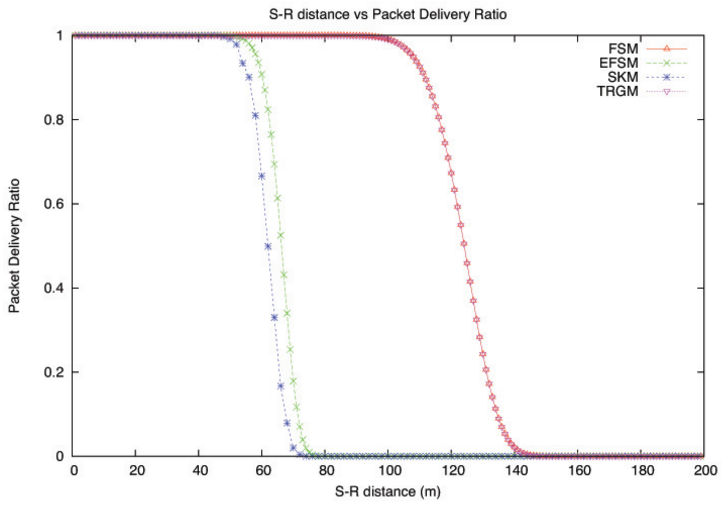


Figure 5.3: Propagation model behaviour.

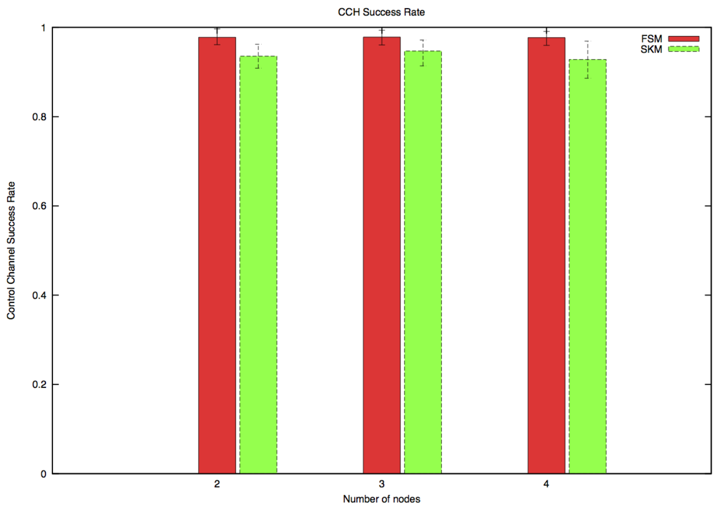


Figure 5.4: Propagation model impact on the CCH.

But, regardless of this issue, it is possible to conclude that the presence of traffic in the SCH has virtually no impact on the CCH, as is the purpose of using one channel for each type of information.

5.2 Platform evaluation

Aside from the model related tests done in the previous section, two proof of concept applications were developed. The first application was used in a field demonstration of the platform's features and abilities, the second was developed to address the requirements posed by extending a legacy application.

This section is organised as follows, first the field demonstrator is explained and its results shown. Next the second application is detailed and its results shown.

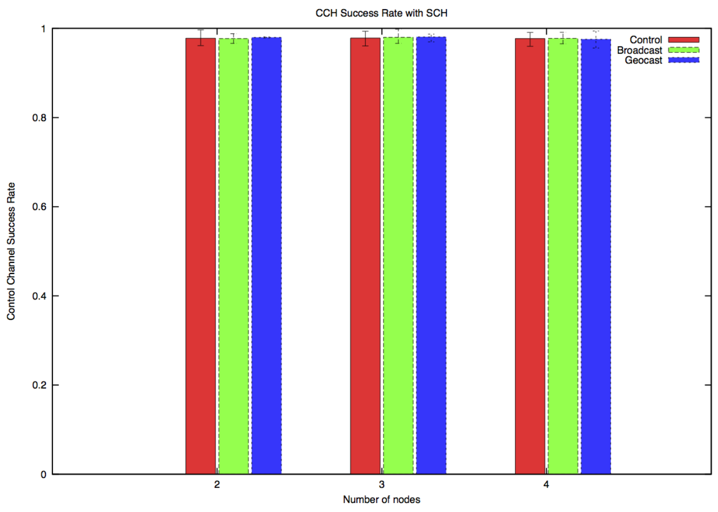


Figure 5.5: Impact of SCH traffic in CCH.

5.2.1 Test Description

For both applications, two OBUs and one RSU were used. The application's RSU component, supplied a central server with the information it retrieved from the platform using a wired connection, organised as shown in Figure 5.6.

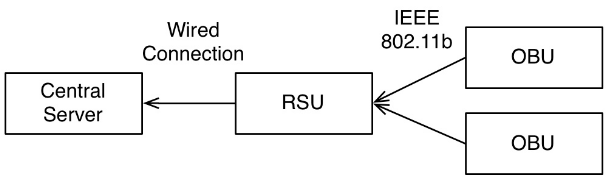


Figure 5.6: Proof of Concept applications' test scenario.

5.2.2 Field Demonstrator

The first use case aims to validate the use of geocast communication mode to disseminate data gather from sensors, using real vehicles that communicate with a RSU located near the window of an office, at the second floor of IST main building. It is placed in the location identified in Figure 5.8. One of the OBUs was a parked vehicle, which is identified by the pink pin in the same figure. The green line shows the mobility pattern of the mobile OBU, this mobility pattern was established to demonstrate the capabilities of the platform both in direct LOS, and direct communication, and with the building in the middle, through the use of multi-hop communication. At this point, it is important to point that the background image in the following figures doesn't represent the current layout of both the building and the roads, but there is no image available that reflects the current layout. At present, the building extends from its current end in the image to the road at the middle of the image. There is also a new road, starting in the bottom of the screen and passing near the end of the football field. While it is possible to deduce this by simply looking at the images, it is important to acknowledge this fact.

The cars are equipped with our platform and the hardware setup is displayed is Figure 5.7.

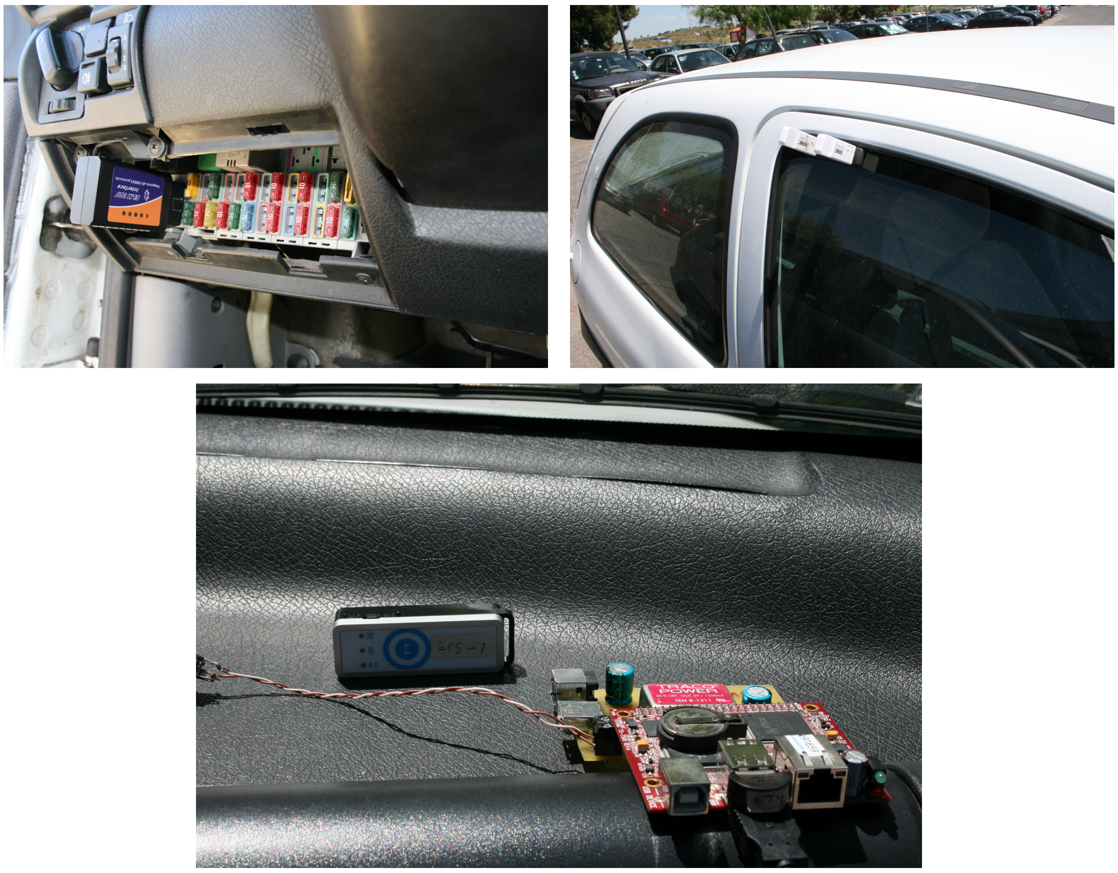


Figure 5.7: Hardware used for field demonstration (Upper left, OBD-II interface, upper right, Wireless cards, bottom, TS-7500 and GPS receiver).

To implement the application the following Application-oriented policy was created.

```
PoC1 | 0 | Geocast | Region Format = Circle, Region Size = Medium, Message Lifetime = Medium
```

In Figure 5.9 shows using a yellow pin, every single message received in the RSU, these messages are from OBU 1. The overlay of the mobile node's mobility pattern and the received messages helps understand where multi-hop and single-hop communication was achieved. Each of these yellow pins contains the information the PoC1 application wished to transmit using the platform.

In Figure 5.10, the information transmitted is displayed. This figure shows that the application transmitted the node's position, the time at which the information was generated, the current speed and RPM. This information was then processed at the central server and this image was generated.

5.2.3 Legacy Application

The second Proof of Concept application developed was used to support a legacy application. The emulated legacy application was a Fleet Management application and the platform was used as a means of communication between nodes when other communication technologies were not available.

The created application, works as an interface between the legacy application and the platform. According to the legacy application's requirements, the created application had the following requirements:

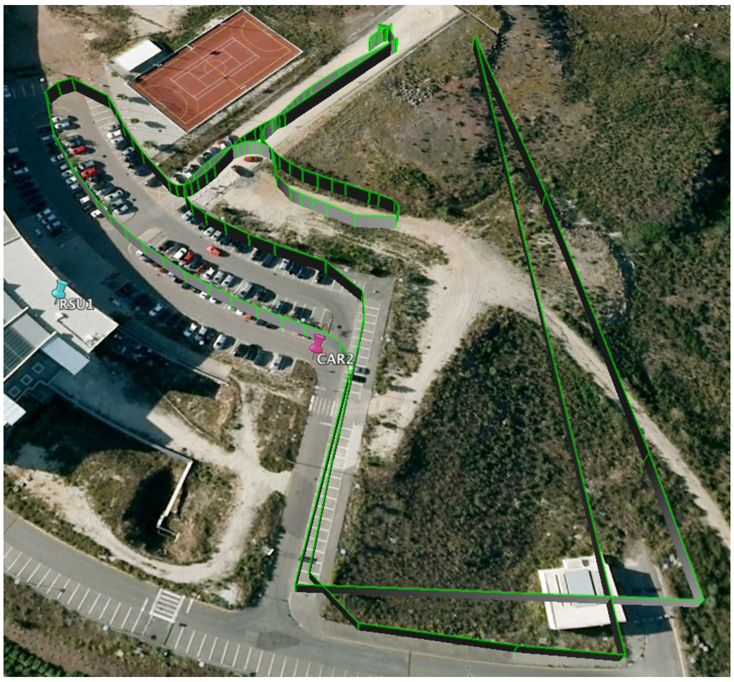


Figure 5.8: Platform demonstration, RSU in cyan, OBU 1 in green and OBU 2 in pink.

PoC2	1	Broadcast	Message Lifetime = 24h

The purpose of the PoC2 application is to epidemically transmit messages, in the hope that some will reach the central control server. To do this the application receives messages from the legacy application and stores them locally and when a new neighbour appears, transmits the messages in waiting. The transmitted messages are both from the current node and from other nodes that may have crossed paths with this node.

This application was tested using the platform's capabilities for emulating field conditions. The node mobility was emulated using the tools described in Section 4.2.2. In Figure 5.11 the RSU's view of the world is depicted. With yellow pins, the starting positions of the OBUs, ”CAR1” and ”CAR2” are shown. The RSU's position is shown using the white arrow. The first OBU, ”CAR1”, follows the path shown in blue, which takes it past ”CAR2” and then to the RSU. The green OBU, ”CAR2”, follows the road in front and then, when it reaches ”CAR1”'s position, turns back and stops were it started.

What this scenario tries to illustrate is the capability of the developed application to transport information from ”CAR2”, which is never in sight of the RSU, to the intended destination, in this case the RSU. When ”CAR1” reaches the neighbourhood of the RSU it transmits all the information it has. This scenario also shows how the platform is able to emulate field conditions indoor. This test was done in lab conditions, with no field testing done, and the application perceives its environment as it would be expected in field conditions.

The creation of this application and its results are evidence not only of the platform's capability of supporting legacy applications but also of the platform's intended purpose in supporting and easing the development of new applications.

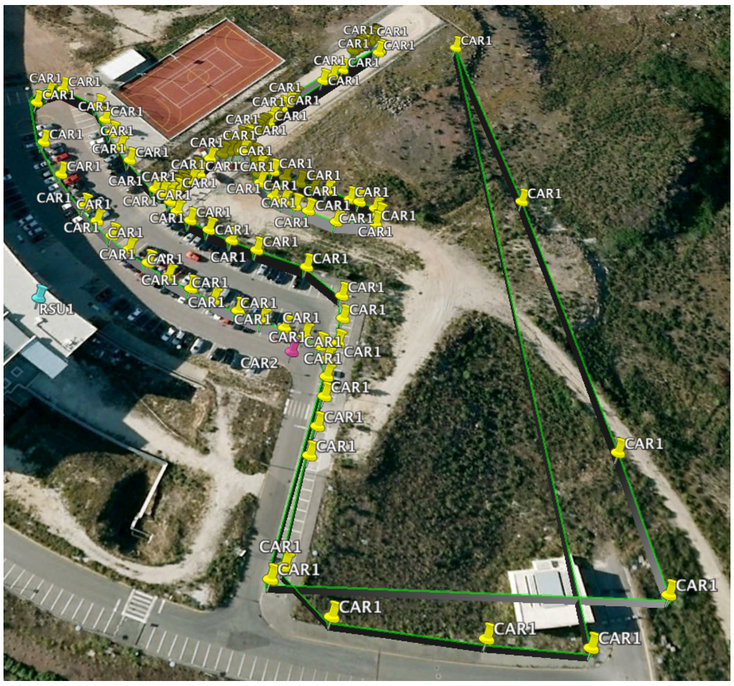


Figure 5.9: Platform demonstration, RSU in cyan, OBU 1 in green and yellow and OBU 2 in pink.

5.3 Summary

In this chapter the tests done to assure the platform's behaviour were detailed, two proof of concept applications are also described.

These tests focussed on the *Emulator Module* and the propagation models it encompasses. The results confirmed that the platform's emulated behaviour is correct and consistent with the observations made in field trials. However, because of hardware issues, some results could not be obtained and more extensive testing, which would allow to show with even more confidence, the platform's correctness could not be done.

This chapter also details two proof of concept applications, one use to demonstrate the field capabilities of the designed platform and one which show how the platform supports both legacy applications and application development.

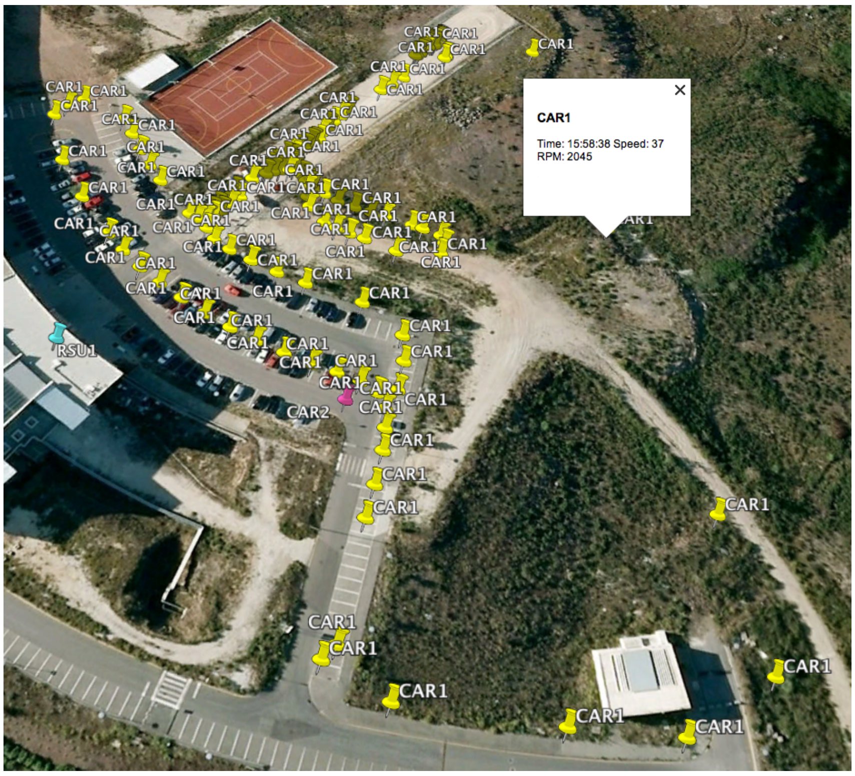


Figure 5.10: Example of information retrieved from the platform.

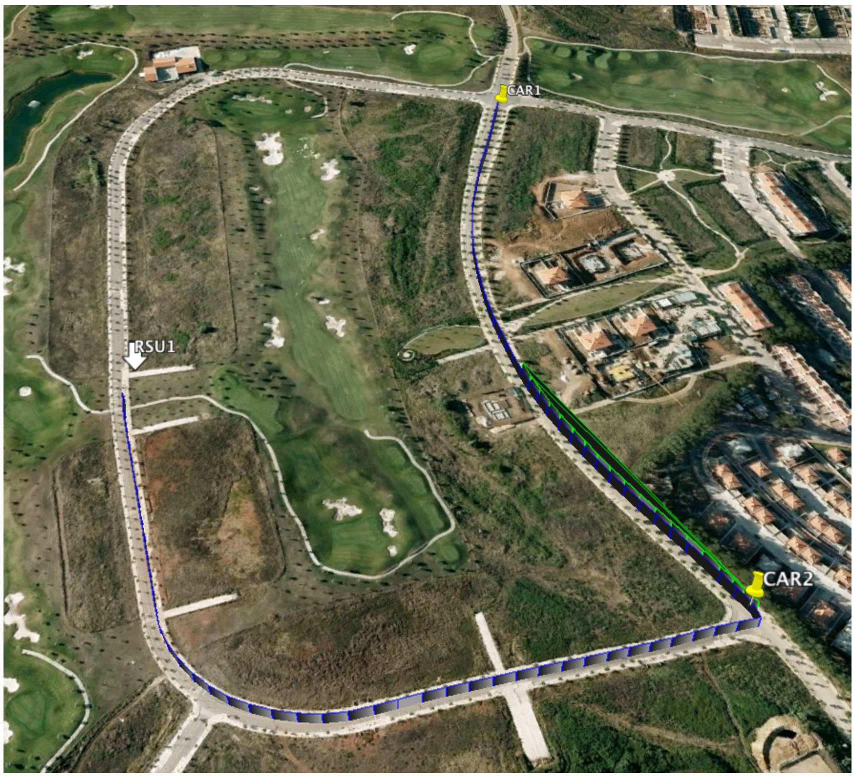


Figure 5.11: Legacy application support, Proof of Concept.

Chapter 6

Conclusions and Future Work

This thesis began with two questions.

- *Is it possible to have a faster dissemination of VANET technology?*
- *Is it possible to create a low-cost VANET operating system that offers a set of tools to create, test, and operate VANET's applications in most of the existing cars?*

To address these questions, this work proposed a low-cost VANET platform, which is focussed on providing support for the development, validation and deployment of VANET applications in existing vehicles.

This is achieved by having a solution based on commercial, off-the-shelf hardware, thus reducing the cost of the overall platform. This solution also defines a software architecture, which is split into three main components, the **Application Developer Interface**, **Emulator** and **Main System**. In order to ease the developer's work in defining and configuring the platform to meet his application's requirements a policy-based solution for this problem is also defined.

The *Application Developer Interface* defines a Custom API, which allows the developers to inform the platform of their applications' needs. The Custom API can also be used by developers to access the vehicle's sensor information, send and receive messages and be notified when a neighbour appears or leaves the neighbourhood.

The *Emulator* supplies a number of tools to support the development of applications. These tools are a Wireshark dissector and a pcap exporter, to allow the analysis of the application's network traffic, a trace exporter for the SUMO software, which allows the use of generated mobility models to be fed to the platform thus emulating node movement. And to complement these tools, a set of propagation models are included in the platform, these models allow the developer to test applications' behaviour in production, while still working in a lab environment.

Finally, the *Main System* is the core the platform, it implements the policy enforcement mechanism and is the platform's contact with the other nodes. It receives messages from applications and other nodes and dispatches them accordingly. It also keep's track of the current neighbours, their appearance and disappearance, notifying the applications, who wish it, of these events. It is in charge of communicating with the onboard sensors and abstracting these from the application and the rest of the platform.

New functionality can be added to the platform's core in the form of a service, these services can be used by either the applications or the platform itself, according to the purpose of the service.

After the platform was defined and developed, a series of tests were devised to test its features, the main focus was the *Emulator*. The tests showed that the used propagation models were correctly emulating the field conditions and that the impact on the CCH of having application traffic in the SCH in negligible. However, the amount of testing done was limited by hardware malfunctions and more testing is still required.

A couple of Proof of Concept applications were also developed, one was used to demonstrate the platform's functionalities in the field, with great results. The second showed how the platform can be used to support and extend legacy applications, this application also allowed the field emulation to be shown, speeding up the development of the application.

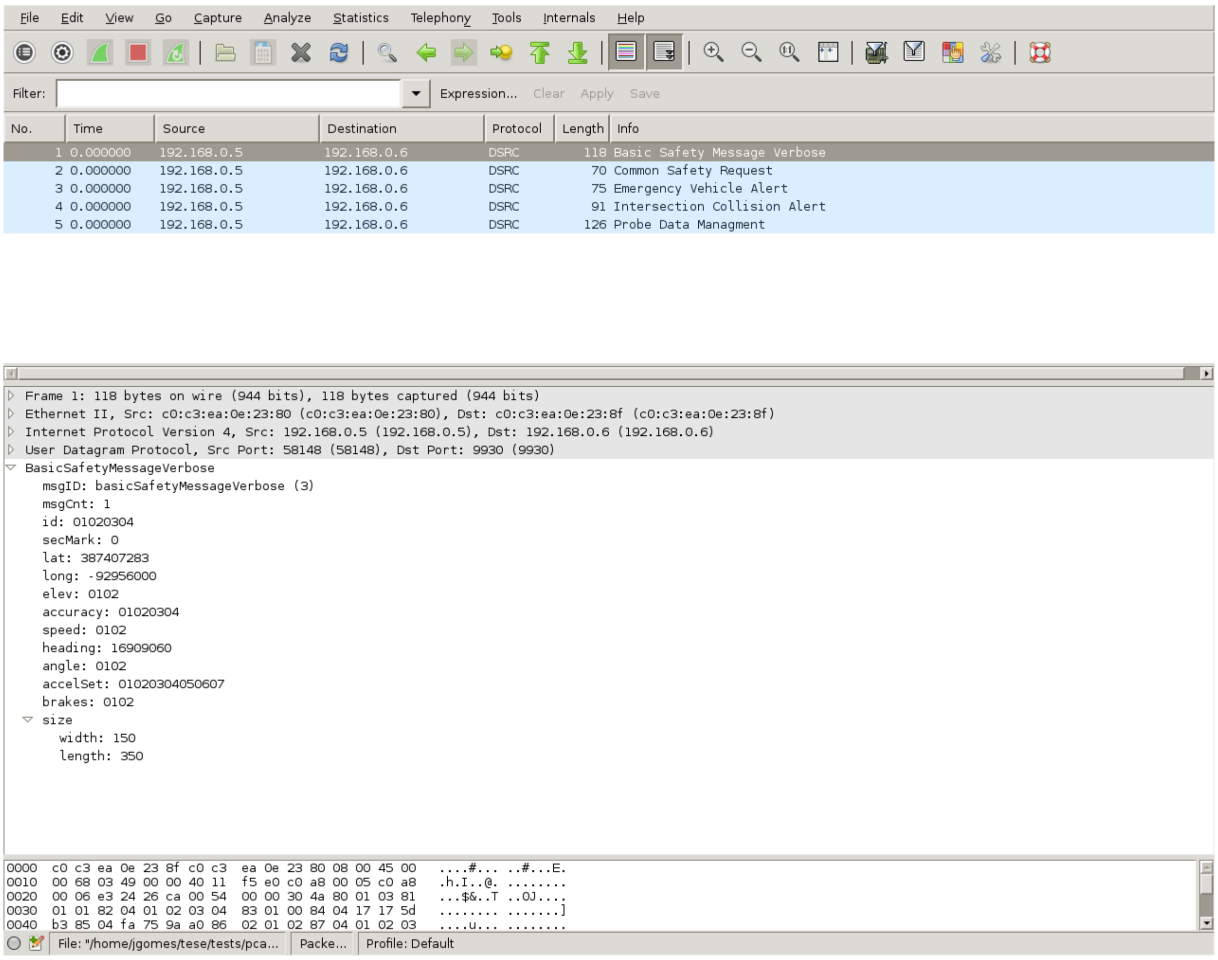
For future work remains the need for further testing to address the issues that could not be because of the hardware issues. The present implementation doesn't support Unicast applications correctly, while the platform was designed with these applications in mind and has some support for these applications, critical support services, like a Location service, support for Unicast routing protocols remain to be addressed. These two aspects of Unicast communication have been heavily addressed by the research community and the platform should support as many of these solutions as possible.

During the development of this work there was no VANET-specific communication technology, namely IEEE 802.11p, so the platform had to be developed using the available IEEE 802.11b network interfaces. In the future the platform should incorporate this new technology as its base communication technology, while that doesn't happen, the current interfaces should be adapted to reflect as much as possible the behaviour of the IEEE 802.11p interfaces. One of the main features that should be supported is the possibility of communication without association, which is supported in IEEE 802.11p but not in 802.11b.

With these improvements to the current work, what is already a very useful solution for the identified problems, can become an essential piece in VANET operation and research, allowing this technology to completely fulfil its purpose.

Appendix A

Wireshark Screen Captures

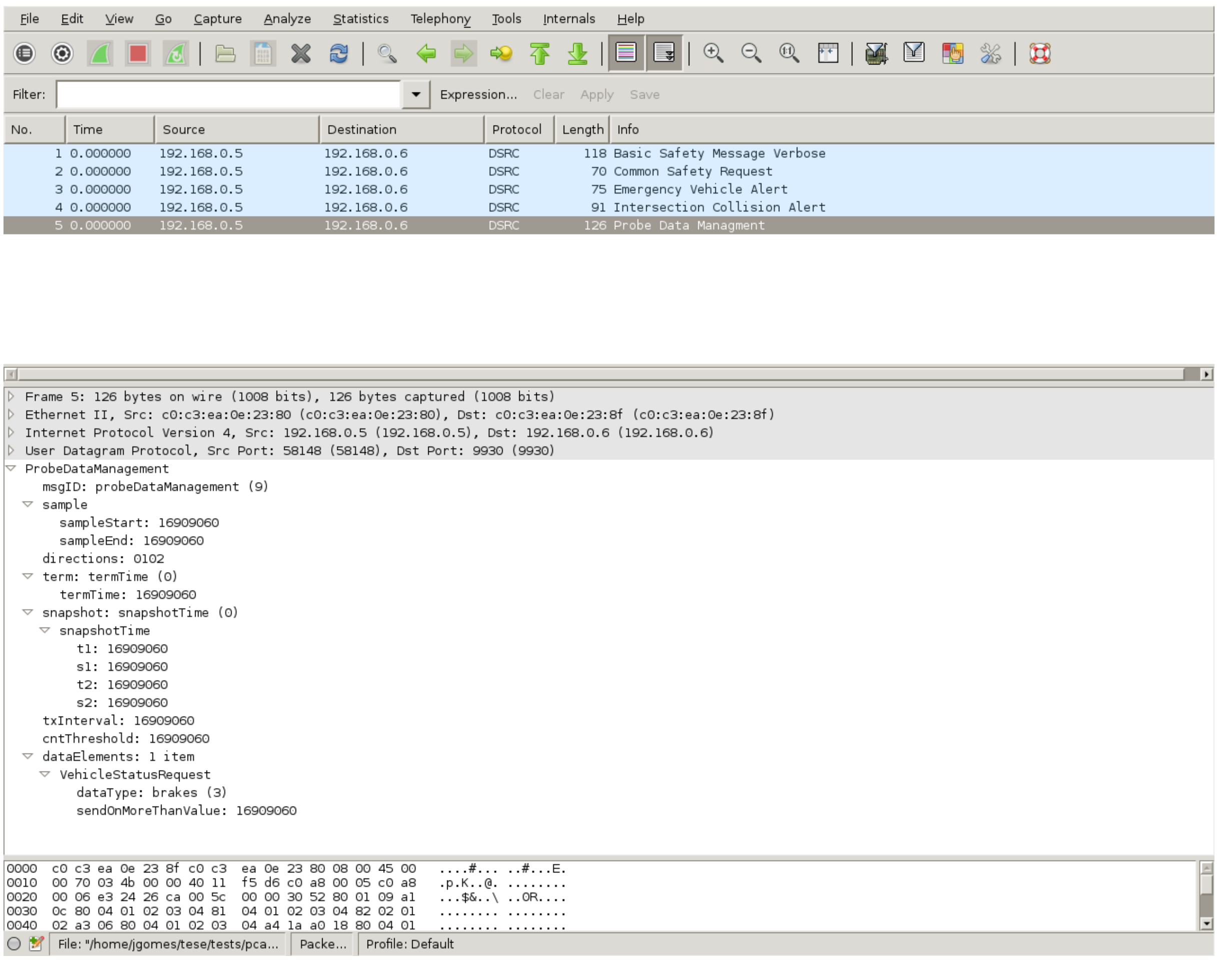

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time      Source       Destination  Protocol Length Info
  1 0.000000   192.168.0.5  192.168.0.6  DSRC        118 Basic Safety Message Verbose
  2 0.000000   192.168.0.5  192.168.0.6  DSRC         70 Common Safety Request
  3 0.000000   192.168.0.5  192.168.0.6  DSRC         75 Emergency Vehicle Alert
  4 0.000000   192.168.0.5  192.168.0.6  DSRC         91 Intersection Collision Alert
  5 0.000000   192.168.0.5  192.168.0.6  DSRC        126 Probe Data Managment
```

```
▷ Frame 1: 118 bytes on wire (944 bits), 118 bytes captured (944 bits)
▷ Ethernet II, Src: c0:c3:ea:0e:23:80 (c0:c3:ea:0e:23:80), Dst: c0:c3:ea:0e:23:8f (c0:c3:ea:0e:23:8f)
▷ Internet Protocol Version 4, Src: 192.168.0.5 (192.168.0.5), Dst: 192.168.0.6 (192.168.0.6)
▷ User Datagram Protocol, Src Port: 58148 (58148), Dst Port: 9930 (9930)
▽ BasicSafetyMessageVerbose
    msgID: basicSafetyMessageVerbose (3)
    msgCnt: 1
    id: 01020304
    secMark: 0
    lat: 387407283
    long: -92956000
    elev: 0102
    accuracy: 01020304
    speed: 0102
    heading: 16909060
    angle: 0102
    accelSet: 01020304050607
    brakes: 0102
  ▽ size
      width: 150
      length: 350
```

```
0000  c0 c3 ea 0e 23 8f c0 c3  ea 0e 23 80 08 00 45 00   ....#... ..#...E.
0010  00 68 03 49 00 00 40 11  f5 e0 c0 a8 00 05 c0 a8   .h.I..@. ........
0020  00 06 e3 24 26 ca 00 54  00 00 30 4a 80 01 03 81   ...$&..T ..0J....
0030  01 01 82 04 01 02 03 04  83 01 00 84 04 17 17 5d   ........ .......]
0040  b3 85 04 fa 75 9a a0 86  02 01 02 87 04 01 02 03   ....u... ........
```

Figure A.1: Wireshark screenshot showing the contents of a Basic Safety Message Verbose.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time      Source       Destination  Protocol  Length Info
   1 0.000000  192.168.0.5  192.168.0.6  DSRC         118 Basic Safety Message Verbose
   2 0.000000  192.168.0.5  192.168.0.6  DSRC          70 Common Safety Request
   3 0.000000  192.168.0.5  192.168.0.6  DSRC          75 Emergency Vehicle Alert
   4 0.000000  192.168.0.5  192.168.0.6  DSRC          91 Intersection Collision Alert
   5 0.000000  192.168.0.5  192.168.0.6  DSRC         126 Probe Data Managment
```

```
Frame 5: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
Ethernet II, Src: c0:c3:ea:0e:23:80 (c0:c3:ea:0e:23:80), Dst: c0:c3:ea:0e:23:8f (c0:c3:ea:0e:23:8f)
Internet Protocol Version 4, Src: 192.168.0.5 (192.168.0.5), Dst: 192.168.0.6 (192.168.0.6)
User Datagram Protocol, Src Port: 58148 (58148), Dst Port: 9930 (9930)
ProbeDataManagement
  msgID: probeDataManagement (9)
  sample
    sampleStart: 16909060
    sampleEnd: 16909060
  directions: 0102
  term: termTime (0)
    termTime: 16909060
  snapshot: snapshotTime (0)
    snapshotTime
      t1: 16909060
      s1: 16909060
      t2: 16909060
      s2: 16909060
  txInterval: 16909060
  cntThreshold: 16909060
  dataElements: 1 item
    VehicleStatusRequest
      dataType: brakes (3)
      sendOnMoreThanValue: 16909060
```

```
0000  c0 c3 ea 0e 23 8f c0 c3  ea 0e 23 80 08 00 45 00   ....#... ..#...E.
0010  00 70 03 4b 00 00 40 11  f5 d6 c0 a8 00 05 c0 a8   .p.K..@. ........
0020  00 06 e3 24 26 ca 00 5c  00 00 30 52 80 01 09 a1   ...$&..\ ..0R....
0030  0c 80 04 01 02 03 04 81  04 01 02 03 04 82 02 01   ........ ........
0040  02 a3 06 80 04 01 02 03  04 a4 1a a0 18 80 04 01   ........ ........
```

Figure A.2: Wireshark screenshot showing the contents of a Probe Data Management Message.

Bibliography

[1] Hartenstein, H., Laberteaux, K.P.: A tutorial survey on vehicular ad hoc networks. Comm. Mag. **46**(6) (jun 2008) 164–171

[2] Toor, Y., Mühlethaler, P., Laouiti, A., de La Fortelle, A.: Vehicle ad hoc networks: Applications and related technical issues. IEEE Communications Surveys and Tutorials (2008) 74–88

[3] Bai, F., Krishnan, H., Sadekar, V., Holland, G., ElBatt, T.: Towards characterizing and classifying communication-based automotive applications from a wireless networking perspective. (2006)

[4] Hong, X., Xu, K., Gerla, M.: Scalable routing protocols for mobile ad hoc networks. Network, IEEE **16**(4) (Jul 2002) 11–21

[5] Friedman, R., Kliot, G.: Location services in wireless ad hoc and hybrid networks: A survey. Techanical Report, Technion Computer Science (2006)

[6] Chen, W., Guha, R.K., Kwon, T.J., Lee, J., Hsu, Y.Y.: A survey and challenges in routing and data dissemination in vehicular ad hoc networks. Wireless Communications and Mobile Computing **11**(7) (2011) 787–795

[7] Menouar, H., Filali, F., Lenardi, M.: A survey and qualitative analysis of mac protocols for vehicular ad hoc networks. Wireless Communications, IEEE **13**(5) (2006) 30–35

[8] Lee, K.C., Lee, S.H., Cheung, R., Lee, U., Gerla, M.: First experience with cartorrent in a real vehicular ad hoc network testbed. In: 2007 Mobile Networking for Vehicular Environments, IEEE (2007) 109–114

[9] Pinart, C., Sanz, P., Lequerica, I., García, D., Barona, I., Sánchez-Aparisi, D.: Drive: A reconfigurable testbed for advanced vehicular services and communications. In: Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities. TridentCom '08, ICST, Brussels, Belgium, Belgium, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2008) 16:1–16:8

[10] Cesana, M., Fratta, L., Gerla, M., Giordano, E., Pau, G.: C-vet the ucla campus vehicular testbed: Integration of vanet and mesh networks. In: Wireless Conference (EW), 2010 European. (April 2010) 689–695

[11] Sichitiu, M., Kihl, M.: Inter-vehicle communication systems: a survey. Communications Surveys & Tutorials, IEEE (2008) 88–105

[12] Gosse, K., Bateman, D., Janneteau, C., Kamoun, M., Kellil, M., Roux, P., Olivereau, A., Patillon, J.N., Petrescu, A., Yang, S.: Standardization of vehicle-to-infrastructure communication. In: Vehicular Networking: Automotive Applications and Beyond. John Wiley Sons, Ltd (2010) 171–201

[13] Krishnan, H., Bai, F., Holland, G.: Commercial and public use applications. In: Vehicular Networking: Automotive Applications and Beyond. John Wiley & Sons, Ltd (2010) 1–28

[14] Festag, A., Noecker, G., Strassberger, M., Lübke, A., Bochow, B., Torrent-Moreno, M., Schnaufer, S., Eigner, R., Catrinescu, C., Kunisch, J.: NoW - Network on Wheels: Project Objectives, Technology and Achievements. Proceedings of 5th International Workshop in Intelligent Transportation (March) (2008) 211–216

[15] Tonguz, O., Wisitpongphan, N.: On the broadcast storm problem in ad hoc wireless networks. (2006)

[16] Willke, T.L., Tientrakool, P., Maxemchuk, N.F.: A survey of inter-vehicle communication protocols and their applications. Communications Surveys & Tutorials, IEEE **11**(2) (2009) 3–20

[17] ETSI, T.: 102 638-Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions. ETSI, Sophia Antipolis Cedex, France **1** (2009)

[18] Toor, Y., Muhlethaler, P., Laouiti, A.: Vehicle ad hoc networks: applications and related technical issues. Communications Surveys Tutorials, IEEE **10**(3) (2008) 74 –88

[19] Emmelmann, M., Bochow, B., Kellum, C.C., eds.: Vehicular Networking: Automotive Applications and Beyond. John Wiley & Sons, Ltd (2010)

[20] Antolino Rivas, D., Barceló-Ordinas, J.M., Guerrero Zapata, M., Morillo-Pozo, J.D.: Security on VANETs: Privacy, misbehaving nodes, false information and secure data aggregation. Journal of Network and Computer Applications **34**(6) (November 2011) 1942–1955

[21] Isaac, J., Zeadally, S., Camara, J.: Security attacks and solutions for vehicular ad hoc networks. Communications, IET **4**(7) (30 2010) 894 –903

[22] Williams, B.: The CALM Handbook. the official ISO TC204 CALM handbook: (2008)

[23] Uzcátegui, R., Acosta-Marum, G.: WAVE: A Tutorial. Communications Magazine, IEEE **47**(5) (May 2009) 126 –133

[24] Eichler, S.: Performance Evaluation of the IEEE 802.11p WAVE Communication Standard. Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th (2007) 2199–2203

[25] Bouchez, B., de Coen, L.: Communication systems for railway applications. In: Vehicular Networking: Automotive Applications and Beyond. John Wiley Sons, Ltd (2010) 83–104

[26] Schaffnit, T.: Automotive standardization of vehicle networks. In: Vehicular Networking. John Wiley Sons, Ltd (2010) 149–169

[27] IEEE: IEEE Trial-Use Standard for Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages. IEEE Std 1609.2-2006 (2006) 1–105

[28] Kenney, J.: Dedicated Short-Range Communications (DSRC) Standards in the United States. Proceedings of the IEEE **99**(7) (july 2011) 1162 –1182

[29] Moalla, R., Lonc, B., Labiod, H., Simoni, N.: How to secure ITS applications? In: Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean. (june 2012) 113 –118

[30] International, S.: DSRC Implementation Guide. (Feb 2010) 1 –210

[31] ITU-T: Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation. Recommendation X.680 (Nov. 2008) 1 – 194

[32] ITU-T: Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). Recommendation X.690 (Nov. 2008) 1 – 38

[33] Bossom, R., Brignolo, R., Ernst, T., Evensen, K., Frötscher, A., Höfs, W., Jääskeläinen, J., Jeftic, Z., Kompfner, P., Kosch, T., Kulp, I., Kung, A., Mokaddem, A.K., Schalk, A., Uhlemann, E., Wewetzer, C.: D31 European ITS Communication Architecture. Technical report (2009)

[34] Schiller, J.: Mobile Communications. 2nd edn. Pearson Education Limited, USA (2003)

[35] Wewetzer, C., Caliskan, M., Meier, K., Luebke, A.: Experimental Evaluation of UMTS and Wireless LAN for Inter-Vehicle Communication. 2007 7th International Conference on ITS Telecommunications (June 2007) 1–6

[36] ETSI: UMTS 30.01. Technical report, ETSI, Valbonne - FRANCE (1998)

[37] Santa, J., Tsukada, M., Ernst, T., Gomez-Skarmeta, A.: Experimental analysis of multi-hop routing in vehicular ad-hoc networks. In: Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops, 2009. TridentCom 2009. 5th International Conference on. (April 2009) 1–8

[38] González, V., Santos, A.L., Pinart, C., Milagro, F.: Experimental demonstration of the viability of ieee 802.11b based inter-vehicle communications. In: Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities. TridentCom '08, ICST, Brussels, Belgium, Belgium, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2008) 1:1–1:7

[39] Festag, A., Fußler, H., Hartenstein, H., Sarma, A., Schmitz, R.: FLEETNET: Bringing Car-to-Car communications into the real world. Computer **4**(L15) (2004) 16

[40] Hui, F., Mohapatra, P.: Experimental characterization of multi-hop communications in vehicular ad hoc network. In: Proceedings of the 2nd ACM international workshop on Vehicular ad hoc networks, ACM (2005) 85–86

[41] Lee, K., Lee, S.H., Cheung, R., Lee, U., Gerla, M.: First experience with cartorrent in a real vehicular ad hoc network testbed. In: 2007 Mobile Networking for Vehicular Environments. (May 2007) 109–114

[42] Ramachandran, K., Gruteser, M., Onishi, R., Hikita, T.: Experimental analysis of broadcast reliability in dense vehicular networks. In: Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th. (Sept 2007) 2091–2095

[43] R. Costa, S. Sargento, R.A.W.: Development of a Hybrid Simulation and Emulation Testbed For VANETs. (June 2009)

[44] Damianou, N.C., Bandara, A.K., Sloman, M.S., Lupu, E.C.: A survey of policy specification approaches. Technical report (2002)

[45] Pujolle, G., Chaouchi, H.: Qos, security, and mobility management for fixed and wireless networks under policy- based techniques. In: Communication Systems: The State of the Art (IFIP World Computer Congress). (2002) 167–180

[46] Moore, B., Ellesson, E.: Policy Core Information Model – Version 1 Specification. RFC 3060 (February 2001)

[47] Lymberopoulos, L., Lupu, E., Sloman, M.: An adaptive policy based management framework for differentiated services networks. In: Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on. (2002) 147–158

[48] Lupu, E., Sloman, M.: Conflicts in policy-based distributed systems management. Software Engineering, IEEE Transactions on **25**(6) (Nov 1999) 852–869

[49] Bourdenas, T., Sloman, M.: Starfish: policy driven self-management in wireless sensor networks. In: SEAMS. (2010) 75–83

[50] Sloman, M., Lupu, E.C.: Engineering policy-based ubiquitous systems. Comput. J. **53**(7) (2010) 1113–1127

[51] Mahasukhon, P., Hempel, M., Sharif, H., Zhou, T., Ci, S., Chen, H.H.: BER Analysis of 802.11b Networks Under Mobility. In: Communications, 2007. ICC '07. IEEE International Conference on. (June 2007) 4722–4727

[52] Pham, D., Şekercioğlu, Y., Egan, G.K.: Performance of IEEE 802.11b Wireless Links: An Experimental Study. In: TENCON 2005 2005 IEEE Region 10. (Nov 2005) 1–6

[53] ITU-R: P.525 : Calculation of Free-Space Attenuation. Recommendation P.525-2 (Aug. 1994) 1 – 3

[54] Sommer, C., Dressler, F.: "Using the Right Two-Ray Model? A Measurement based Evaluation of PHY Models in VANETs". 17th ACM MobiCom, Poster Session, Las Vegas, NV, (September 2011)

[55] Rappaport, T.: Wireless Communications: Principles and Practice. Prentice Hall communications engineering and emerging technologies series. Dorling Kindersley (2009)

[56] Boban, M., Vinhoza, T., Ferreira, M., Barros, J., Tonguz, O.: Impact of vehicles as obstacles in vehicular ad hoc networks. Selected Areas in Communications, IEEE Journal on **29**(1) (January 2011) 15–28

[57] ITU-R: P.526 : Propagation by diffraction. Recommendation P.526-13 (Nov. 2013) 1 – 43

[58] SAE International: Dedicated Short Range Communications (DSRC) Message Set Dictionary. (November 2009)

[59] NMEA: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp Visited in: Jan 2014.

[60] SAE International: E/E Diagnostic Test Modes. (February 2012)

[61] Chappell, L.: Wireshark Network Analysis The Official Wireshark Certified Network Analyst Study Guide. Second edn. Protocol Analysis Institute (July 2012)

[62] Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker, L.: Recent development and applications of SUMO - Simulation of Urban MObility. International Journal On Advances in Systems and Measurements **5**(3&4) (December 2012) 128–138

[63] Union, E.: http://eur-lex.europa.eu/lexuriserv/lexuriserv.do?uri=celex:31998l0069:en:html Visited in: May 2014.