UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

IMPLEMENTATION AND EXPERIMENTAL EVALUATION OF COOPERATIVE AWARENESS BASIC SERVICE FOR V2X COMMUNICATIONS

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Charmae Franchesca Ugnaya Mendoza

In partial fulfilment

of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Advisors: Jordi Casademont-Serra, PhD

Daniel Camps-Mur, PhD

Barcelona, May 2019

Title of the thesis: Implementation and Experimental Evaluation of Cooperative Awareness Basic Service for V2X Communications

Author: Charmae Franchesca Ugnaya Mendoza

Advisor: Jordi Casademont-Serra, PhD and Daniel Camps-Mur, PhD

Abstract

A key aspect of Vehicle-to-Everything (V2X) communication is the concept of *cooperative awareness*, wherein the periodic exchange of status information allows vehicles to become aware of their surroundings for increased traffic safety and efficiency. This project aimed to implement the Cooperative Awareness (CA) basic service through the development of a low-cost, open-source On-board Unit (OBU)/Roadside Unit (RSU) that periodically broadcasts Cooperative Awareness Messages (CAM) using the 5.9 GHz band. Its proper operation and interoperability were verified by testing it with a commercial V2X device. This project also aimed to evaluate the effectiveness of the CA basic service through the development of an IEEE 802.11p-based V2X system simulator. The simulations were executed with varying vehicle traffic load (by changing the vehicle speed and the number of lanes) and CAM transmit frequency. The performance was then assessed by analyzing the Packet Reception Ratio (PRR), position error and Neighborhood Awareness Ratio (NAR) metrics. The presence of more vehicles in the slow speed and high lane count scenarios caused higher packet losses due to increased interference and collision probability, leading to low PRR and NAR values. Despite losing more CAMs, the slow speed scenarios had lower position errors since the displacement of vehicles was small. When the CAM transmit frequency was increased, the PRR decreased due to packet collisions. However, the position error was kept low as it benefited from the more frequent CAM transmissions and local database updates. Increasing the transmit frequency also increased the NAR, at least until a certain frequency threshold, beyond which the NAR started to worsen due to the dominant effect of interference in high message traffic situations.

Acknowledgements

I would like to express my sincere gratitude to my advisors, Prof. Jordi Casademont-Serra and Dr. Daniel Camps-Mur, for their guidance and for giving me the opportunity to learn a lot through this project. I would also like to thank Joaquim Oller, Joan Josep Aleixendri Cruelles, Miguel Catalan Cid and Julio Carlos Barrera Juez for all the help they provided. Special thanks to my colleague, Leandro Miguel Lopez, for his patience and cooperation. Finally, thank you to my family and friends back home for their continuous support and encouragement.

Revision History and Approval Record

Revision	Date	Purpose
0	30/03/2019	Document creation
1	14/05/2019	Document revision

Written by:		Reviewed and approved by:	
Date	15/05/2019	Date	15/05/2019
Name	Charmae Franchesca Ugnaya Mendoza	Name	Jordi Casademont-Serra, PhD
Position	Project Author	Position	Project Supervisor

Table of Contents

List of Figures

List of Tables

1. Introduction

Despite the continuous improvements in the traffic infrastructure and automobile technology, road accidents still remain one of the leading causes of deaths with an estimated total of 1.35 million each year [1]. While innovations ranging from simple seat belt to antilock braking systems (ABS) up to sophisticated vehicle sensors have contributed to automobile safety, these technologies remain isolated in individual vehicles. However, if vehicles were able to break this isolation and began communicating with each other, then they could alert each other and prevent potentially dangerous situations, such as unsafe overtaking and sudden stopping of vehicle ahead. Consequently, this would significantly reduce the number of traffic fatalities.

Vehicle-to-everything (V2X) communication is thus considered a breakthrough technology that would revolutionize road safety. Moreover, V2X aims to increase traffic efficiency (by optimizing the use of traffic infrastructure), to reduce environmental impact (by efficient driving), and to provide additional services (such as software provisioning and update, electronic commerce and media downloading), thereby improving the overall transport experience.

A key aspect of V2X communication is the periodic exchange of status information, including vehicle identifiers, location, and velocity. Such data are needed in the realization of several Cooperative Intelligent Transport Systems (C-ITS) applications. For instance, the collision avoidance warning application relies on the real-time position and speed information from surrounding vehicles in order to predict and prevent possible collisions. For this reason, ETSI introduced the Cooperative Awareness (CA) basic service in the ITS facilities layer as a common service that can be utilized by any application (in the upper layer of the V2X protocol stack). In particular, the CA basic service defines the Cooperative Awareness Message (CAM), which is periodically broadcasted by each vehicle to share status information.

1.1. Objectives and Scope

The main goals of this project were to implement the CA basic service and evaluate its effectiveness in enabling cooperative awareness among vehicles and traffic infrastructure. In order to achieve these goals, the project was divided into the following two subtasks:

1. Development of an On-board Unit (OBU)/Roadside Unit (RSU):

   Dedicated V2X platforms are expensive, in addition to their implementation of the V2X stack being proprietary. This makes it difficult for researchers to carry out field tests to evaluate the CA basic service. For this reason, one of the objectives of this project was to develop a low-cost OBU/RSU that implemented the ETSI C-ITS protocol stack (including the CA messaging capability) using open-source software, specifically the Vanetza library. The testbed was further modified to work in the 5.9 GHz frequency band, before being tested using a commercial V2X device to check interoperability.

2. Development of an IEEE 802.11p-based simulator:

   An existing system-level simulator from the V2X-Arch project [2] was used as the foundation in this project for the experimental evaluation of the CA basic service. The said simulator was then modified to enable the variation of important parameters (speed, number of lanes, CAM transmit frequency) in order to deduce their impact on system performance, and to perform more realistic simulations under different road topologies (highway, Manhattan grid). Moreover, different metrics (Packet Reception Ratio or PRR, position error, and Neighborhood Awareness Ratio or NAR) were designed to assess if the CA basic service indeed helped in making the vehicles aware of their surroundings.

   It is important to note that the experiments carried out did not focus on evaluating the physical layer and Medium Access Control (MAC) mechanisms, and as such, the corresponding parameters in the lower layers were kept constant. Moreover, while there are various V2X technologies (WLAN- and cellular-based) that could have been employed in the ITS access layer, only the IEEE 802.11p standard was used all throughout the project, and comparisons of the different V2X technologies were out of scope of the project.

1.2. Work Plan

As the project consisted of two separate tasks, it was necessary to work on them in parallel to ensure their successful and timely completion. The project was carried out from September 2018 to mid-May 2019, and each task was divided into three parts, namely the *study phase*, the *development phase*, and the *results and analysis phase.* The first month was dedicated to studying concepts that were necessary for the implementation of the project. This included both generic topics, such as C++ programming and the C-ITS protocol stack, and more specific ones, including understanding existing source codes to be utilized in the project.

The next months were allocated to working on the two tasks in parallel. In the case of the simulator development, a number of MAC and application layer functions, along with other road scenarios, were created and modified to implement the metrics for the CA basic service evaluation. On the other hand, the CAM receiver application was developed first, and then tested with the transmitter application using a Raspberry Pi. However, due to incompatibility issues (Section 3.4.1), it was necessary to switch to another hardware platform. For this reason, an additional period of time was spent studying the Linux wireless architecture in order to be able to continue with the OBU/RSU development using the new hardware.

Towards the end of the project, simulations had to be carried out multiple times. This was because it was often necessary to make adjustments to the code when unexpected results were observed. This phase was relatively time-consuming since a single batch of simulations took a few days to complete.

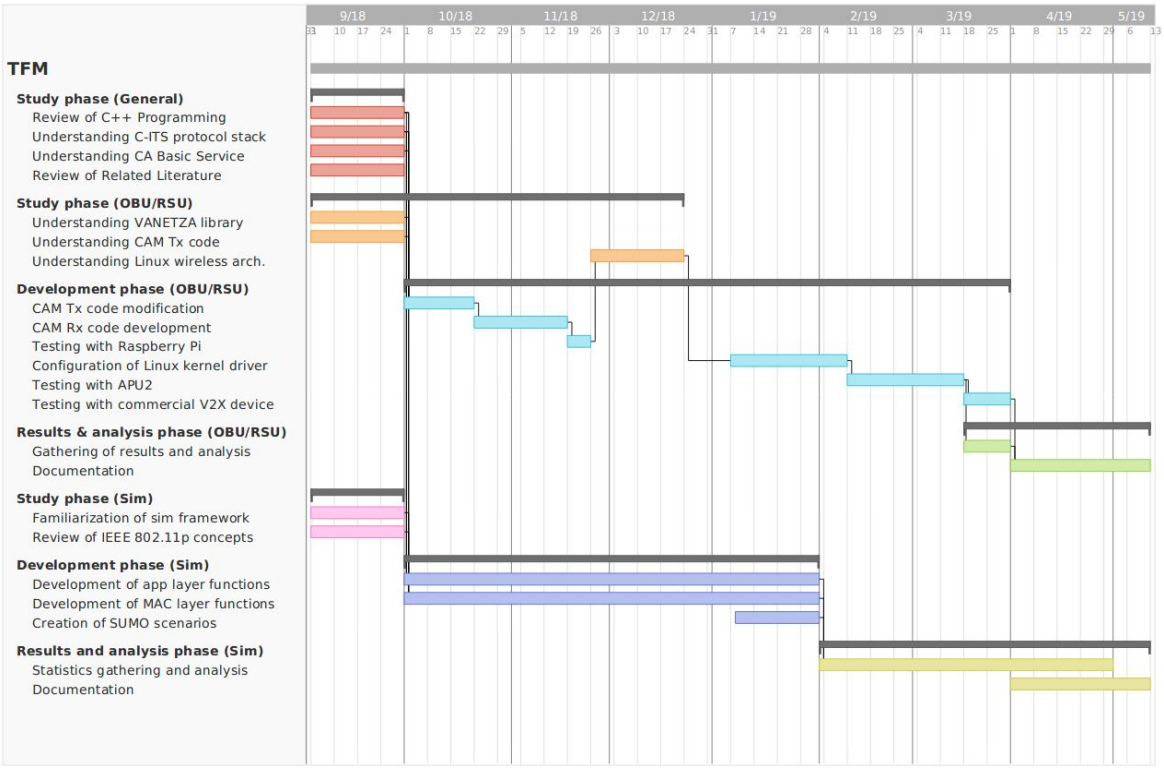


Figure 1. Project schedule

1.3. Outline

Chapter 2 introduces the C-ITS concepts necessary to understand and implement the project. Chapter 3 provides a detailed explanation of the methodology used to develop the open-source OBU/RSU. Chapter 4 discusses the simulator enhancements, and the scenarios and parameters employed in the simulations. Chapter 5 presents the results and analyzes them using specific performance metrics. Chapter 6 gives an estimate of the costs incurred while working on the project. Chapter 7 concludes the paper, and recommends future tasks and research direction. Finally, the appendix includes installation and configuration guides for setting up and replicating both the OBU/RSU and simulator environments.

2. State of the Art

This chapter provides a brief description of C-ITS, while giving emphasis on the details of the ITS protocol stack, including the different layer functionalities and packet header structures. Moreover, it elaborates on some of the ITS facilities layer entities, specifically the CA basic service. Lastly, it presents a review of related work to further understand the motivation of carrying out this research study.

2.1. Cooperative Intelligent Transport Systems

C-ITS utilizes different wireless technologies to allow real-time communication and share useful information among road users and infrastructures. C-ITS aims to create a safer, greener and more convenient transportation environment for everyone.

As an enabler of C-ITS, V2X communication refers to the wireless exchange of information between a vehicle and another entity. There are currently two standardized V2X technologies being considered: IEEE 802.11p and Cellular V2X (C-V2X). A number of research studies have been conducted for the purpose of assessing their system performance to understand the advantages and disadvantages of each technology. While such comparative analysis contributes to the success of C-ITS, this project specifically employed the IEEE 802.11p protocol for evaluating the CA basic service in the ITS facilities layer.

2.2. Network Architecture

The C-ITS network architecture consists of different entities, or ITS stations (ITS-Ss), communicating with each other [3]. As shown in Figure 2, these are:

- Personal ITS-S - handheld devices of pedestrians
- Vehicle ITS-S - OBU mounted on vehicles
- Central ITS-S - traffic management centers
- Roadside ITS-S - RSU or fixed traffic infrastructures

The combination of any of these entities results to different communication modes. In particular, V2X is an umbrella term used to refer to the following.

- V2V: Vehicle-to-Vehicle
- V2I: Vehicle-to-Infrastructure
- V2P: Vehicle-to-Pedestrian
- V2G: Vehicle-to-Grid
- V2N: Vehicle-to-Network

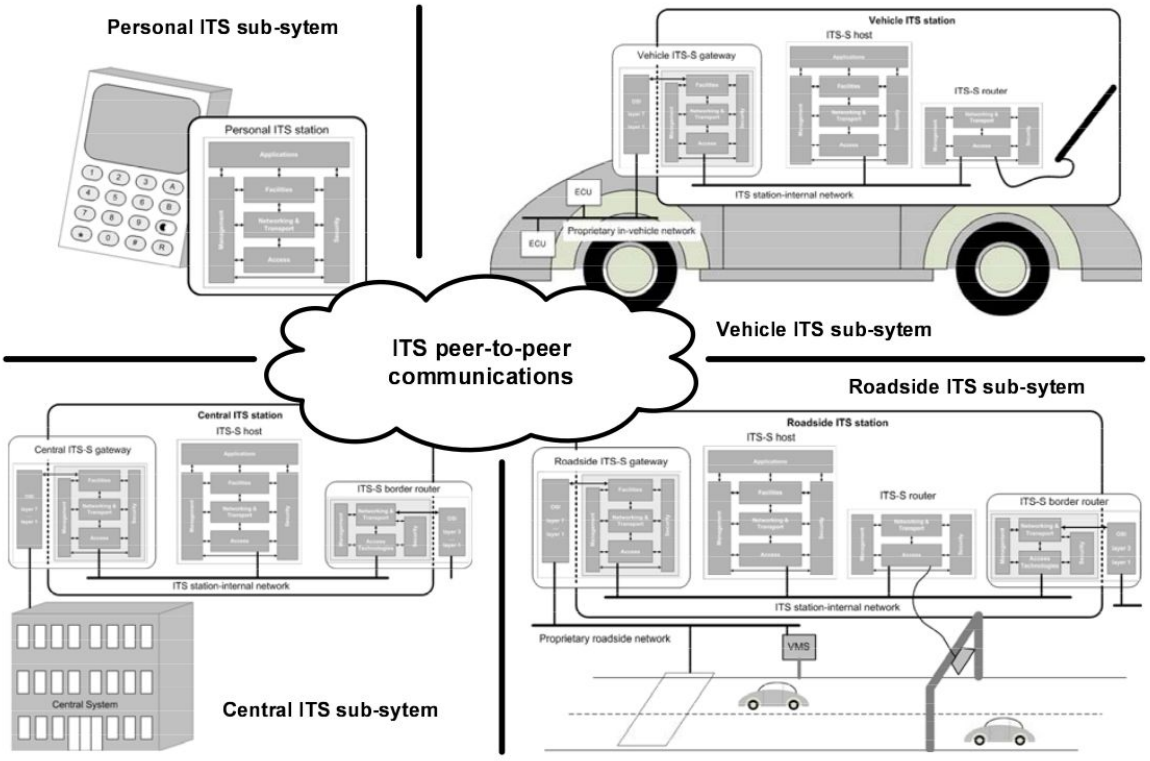


Figure 2. C-ITS Network Architecture [3]

2.3. ITS Station Reference Architecture

The ITS-S reference architecture defines the protocol stack implemented on each station. As illustrated in Figure 3, it comprises four horizontal layers along with two vertical entities [3]. It is analogous to the OSI model, except that it extends the model to include the ITS applications, depicted in the mapping of the two models in Table 1. The protocol layers are described in more detail in the following sections.

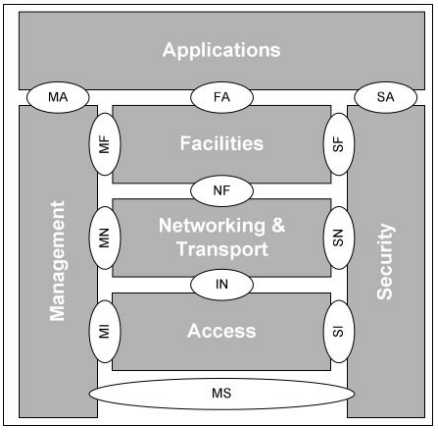


Figure 3. ITS-S reference architecture [3]

ITS-S Reference Architecture	OSI Model
Applications	-
Facilities	Application
	Presentation
	Session
Networking and Transport	Transport
	Network
Access	Data Link
	Physical

Table 1. Mapping between ITS-S reference architecture and OSI model

2.3.1. Applications

ITS applications are formed by complementary ITS-S applications (e.g., server-client scheme). A group of applications and use cases is known as the Basic Set of Applications (BSA). Furthermore, these use cases are categorized into the following three classes, which differ on reliability, latency and security requirements [4].

1. Active road safety

   The goal of this class is to improve traffic safety by preventing road casualties. Vehicles exchange status information (speed, position, etc.) periodically or in a event-triggered manner, creating cooperative awareness and possibly avoiding fatalities. This enables use cases such as collision risk warning and emergency vehicle warning.

2. Cooperative traffic efficiency

   The goal of this class is to improve to improve road traffic management, and increase the traffic efficiency in terms of travel times, fuel consumption and emissions, etc. This usually involves communication with the infrastructure. For instance, in the case of V2I communication, the roadside station sends specific information to the vehicles, enabling use cases like speed limit notification and optimal route recommendation.

3. Other applications

   These include applications providing other services such as those for infotainment. Some examples are point-of-interest notification and media downloading.

Applications class	Application	Use Cases
Active road safety	Driving assistance - Co-operative Awareness (CA)	Emergency vehicle warning Slow vehicle indication Intersection collision warning Motorcycle approaching indication
	Driving assistance - Road Hazard Warning (RHW)	Emergency electronic brake lights Wrong way driving warning Stationary vehicle - accident Stationary vehicle - vehicle problem Traffic condition warning Signal violation warning Roadwork warning Collision risk warning Decentralized floating car data - Hazardous location Decentralized floating car data - Precipitations Decentralized floating car data - Road adhesion Decentralized floating car data - Visibility Decentralized floating car data - Wind
Co-operative traffic efficiency	Speed Management (CSM)	Regulatory/contextual speed limits notification Traffic light optimal speed advisory
	Co-operative Navigation (CoNa)	Traffic information and recommended itinerary Enhanced route guidance and navigation Limited access warning and detour notification In-vehicle signage
Co-operative local services	Location Based Services (LBS)	Point of Interest notification Automatic access control and parking management ITS local electronic commerce Media downloading
Global internet services	Communities Services (ComS)	Insurance and financial services Fleet management Loading zone management
	ITS station Life Cycle Management (LCM)	Vehicle software/data provisioning and update Vehicle and RSU data calibration

Table 2. Basic set of applications [4]

2.3.2. Facilities

The ITS facilities layer maps to layers 5, 6 and 7 of the OSI reference model. As such, it exhibits the corresponding functionalities of those three layers combined with ITS-specific ones. Its main role is to provide service to the ITS applications in the upper layer, and thus, the facilities are also referred to as basic service. Some of the facilities are listed in Table 3, and can be grouped in two ways, according to: (1) type of support, and (2) scope of support provided to the ITS BSA [4].

Classification	Facility name	Short description
Common facilities for the application support facilities	Priority management	Message and use case priority assignment.
	Identities management	Manage the station identifier used by the applications and the V2X messages.
	HMI interface	Provide common interface to multiple HMIs.

	CAM management	Provide management support for Cooperative Awareness Message.
	Security access management	Provide and manage the high layer security requirements and data to the security entity.
	Time management	Provide the time management and time synchronization service within the ITS station.
	Service management	Manage the supporting ITS service and applications within the ITS station.
Common facilities for the information support facilities	Station type/capability	Manage the ITS station type and capabilities information.
	Position management	Provide and manage the station position and movement information.
	Location referencing	Provide location referencing functionalities for the station positioning according to the application requirements.
	Data presentation	Provide presentation support for the V2X messages.
Common facilities for the communication support facilities	Communication management	Contribute from the high layer for the management and the selection of the optimal communication profiles to be used for the V2X message transmission.
	Addressing mode	Select the addressing mode for the V2X message transmission and provide the message dissemination requirements to the network and transport layer.
Domain facilities for the application support facilities	Mobile station dynamics	Manage the vehicle ITS station dynamics information from the in vehicle networks and vehicle electronic functions.
	Mobile station status monitoring	Monitor mobile station status from in vehicle network and vehicle electronic functions and provide information for applications.
	DENM management	Manage DENM and DENM protocol.
	Roadside ITS station state monitoring	Monitors the roadside ITS station status.
	Client ID management	Manage and define the service clients profile information.
	Web service	High layer protocols for the web service e.g. SOA application protocol support.
	Billing and payment	Provide service access to the billing and payment service.
	GIS support	Provide the interface to the GIS service.
	Discovery mechanism	Discover the users of a community service either by a service announcement (passive) or by a subscription (active).
	Station life cycle management	Provide the support for station software updating and data updating.
	Relevance check	Provide the relevance check for the received information from other ITS stations, according to the application requirements.
Domain facilities for the information support facilities	LDM	LDM database.
	Map data base	Provide interface to the map data base at the central ITS station.
	Service content database	Manage a database of the ITS service content.
	RSU registration	Manage the roadside ITS stations and their information that are under the

		control of a central ITS station.
	User repository	Management of the user information at a central ITS station providing an ITS service.
	Fleet Monitoring	Monitor the community service behaviour at the central ITS station relevant.
	Message queuing	Manage the V2X messages queuing based on the message priority and the client services/use case requirements.
Domain facilities for the communication support facilities	Session support	Support the communication session establishment and closure.

Table 3. List of ITS facilities [4]

Classification of facilities according to the type of support provided [4]:

1. Application support facilities - provide application support functionalities
2. Information support facilities - provide common data and database management functionalities
3. Communication support facilities - provide services for communication and session management

Classification of facilities according to the scope of support provided [4]:

1. Common facilities - provide basic core services and functions for all applications and for the operation of the ITS stations
2. Domain - provide specific services and functions for one or several applications

2.3.3. Networking and Transport

The Basic Transport Protocol (BTP) provides an end-to-end, unreliable and connectionless transport service. It is responsible for multiplexing the messages from the different processes at the ITS facilities layer, and at the other end, demultiplexing of messages received through the the GeoNetworking protocol. The way multiplexing/demultiplexing works is based on ports, which act as identifiers to distinguish different processes running on the ITS station. Moreover, BTP allows the facilities layer to access the services provided by the GeoNetworking protocol, as well as the exchange of protocol control information between those two entities [5]. A list of well-known BTP ports is given in Table 4.

There are two types of BTP headers, which is indicated in the Next Header (NH) field of the GeoNetworking Common header. BTP-A is for interactive packet transport, while BTP-B signals non-interactive. Moreover, they differ in packet structure, with BTP-A containing both the source and destination ports, and BTP-B specifying only the destination port with the addition of destination port information in case of well-known ports [5].

Well-known BTP port	ITS facilities layer entity
2001	CAM
2002	DENM
2003	MAP
2004	SPAT
2005	SAM

Table 4. BTP ports

Next Header (NH)	Encoding	Description
BTP-A	1	BTP-A header
BTP-B	2	BTP-B header

Table 5. Encoding of BTP header types in the Next Header field of the GeoNetworking Common Header

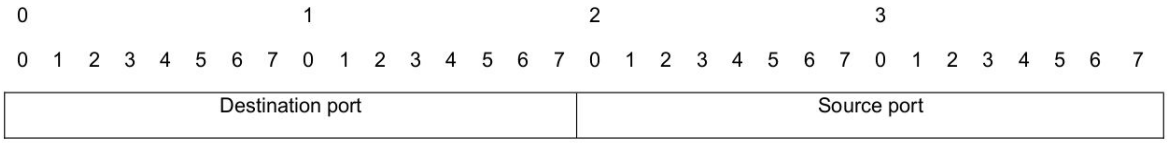


Figure 4. BTP-A header format [5]

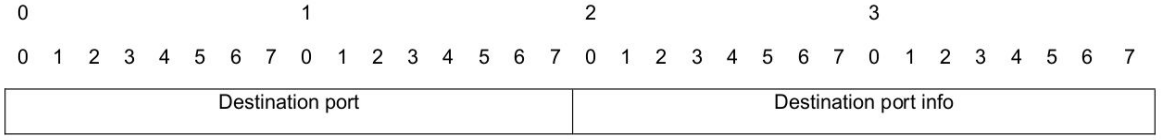


Figure 5. BTP-B header format [5]

The GeoNetworking protocol is a network-layer protocol that uses geographical positions and areas to route packets across the ITS ad hoc network. It enables infrastructure-less communication, and meets the vehicle networking requirements, such as support for high node mobility and continuously changing network topology [6].

The GeoNetworking protocol has the following main functions.

1. Geographical addressing

   A packet is sent to a destination node with a specific geographical position or to a number of destination nodes belonging to a geographical area.

2. Geographical forwarding

   Each node maintains a knowledge of the network topology. When a node receives a packet, it examines the destination field, and compares the indicated geographical address to its knowledge of the network topology to make forwarding decisions. This eliminates the need for complicated IP routing tables.

The GeoNetworking routing employs different packet forwarding schemes as depicted in Figure 6.

1. GeoUnicast

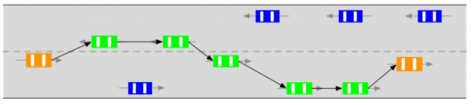
   The unicast packet is continuously forwarded by intermediate nodes (multi-hop) until it reaches its destination node.

2. GeoBroadcast

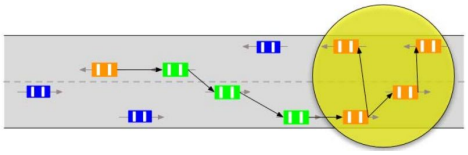
   The packet is continuously forwarded until it reaches its destination geographical area. The nodes inside the area re-broadcasts the packet, unlike in GeoAnycast, where a node inside the area receives the packet and does not resend it.

3. Topologically-scoped broadcast

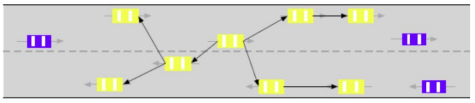
   The packet is continuously re-forwarded until the n-hop node.



(a) GeoUnicast



(b) GeoBroadcast



(c) Topologically-scoped broadcast

Figure 6. GeoNetworking routing schemes [6]

Figure 7. GeoNetworking header format [7]

As illustrated in Figure 7, the GeoNetworking header includes the mandatory Basic and Common headers, as well as an optional Extended header. To aid in understanding the

contents of a GeoNetworking packet, the following figures and tables examine the Basic and Common header formats and corresponding fields. The contents of the Extended header depends on the GeoNetworking packet header type specified in Table 8, and the header format for each type is detailed in [7].

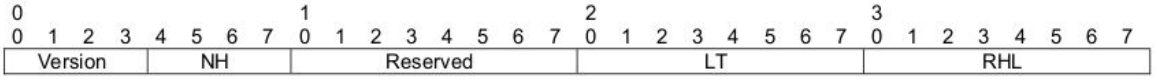


Figure 8. Basic header format [7]

Basic header field	Description
Version	version of the GeoNetworking protocol
Next Header (NH)	type of header following the Basic Header 0: ANY (unspecified) 1: Common header 2: Secured packet
Reserved	reserved, set to 0
Lifetime	maximum time a packet could be buffered before reaching destination
Remaining Hop Limit (RHL)	decremented by 1 (hop) every time the packet is forwarded by the GeoAdhoc router

Table 6. Basic header fields

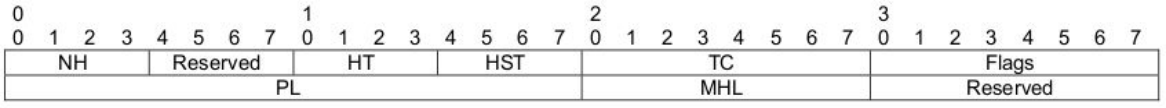


Figure 9. Common header format [7]

Common header fields	Description
Next header (NH)	type of header following the GeoNetworking headers 0: ANY (unspecified) 1: BTP-A 2: BTP-B 3: IPv6
Reserved	reserved, set to 0
Header Type (HT)	type of GeoNetworking header (refer to Table 8)
Header Subtype (HST)	sub-type of GeoNetworking header (refer to Table 8)
Traffic Class (TC)	represents requirements of facilities layer

Flags	indicates whether the ITS-S is mobile or stationary
Payload (PL)	size of packet following the GeoNetworking headers
Maximum Hop Limit (MHL)	maximum hop limit
Reserved	reserved, set to 0

Table 7. Common header field

Header Type (HT)	Header Sub-type (HST)	Encoding	Description
ANY		0	Unspecified
	UNSPECIFIED	0	Unspecified
BEACON		1	Beacon
	UNSPECIFIED	0	Unspecified
GEOUNICAST		2	GeoUnicast
	UNSPECIFIED	0	Unspecified
GEOANYCAST		3	Geographically-Scoped Anycast (GAC)
	GEOANYCAST_CIRCLE	0	Circular area
	GEOANYCAST_RECT	1	Rectangular area
	GEOANYCAST_ELIP	2	Ellipsoidal area
GEOBROADCAST		4	Geographically-Scoped Anycast (GAC)
	GEOANYCAST_CIRCLE	0	Circular area
	GEOANYCAST_RECT	1	Rectangular area
	GEOANYCAST_ELIP	2	Ellipsoidal area
TSB		5	Topologically-scoped broadcast (TSB)
	SINGLE_HOP	0	Single-hop broadcast (SHB)
	MULTI_HOP	1	Multi-hop TSB
LS		6	Location service (LS)
	LS_REQUEST	0	Location service request
	LS_REPLY	1	Location service reply

Table 8. Encoding of HT and HST fields [7]

The geographical area is defined by a geometric shape (circle, rectangle or ellipse), which is indicated in the header subtype of the Common header. The parameters and coordinates forming the boundaries of the geographical area are specified in the Extender header. Moreover, the mathematical functions representing the shapes are detailed in [8].

2.3.4. Access

The ITS access layer maps to the data link and physical (PHY) layers of the OSI reference model. The data link layer consists of the MAC and the Logical Link Control (LLC) sublayers. The ITS access layer technology is termed ITS-G5, which is based on the IEEE 802.11-2012 Wireless Local Area Network (WLAN) standard. In particular, IEEE 802.11p corresponds to the PHY and MAC layers and is an enhancement of IEEE 802.11 (specifically, IEEE 802.11a) to meet the requirements of ITS applications.

IEEE 802.11p employs an almost identical physical layer as that of IEEE 802.11a, including the use of Orthogonal Frequency Division Multiplexing (OFDM) (total of 52 subcarriers, of which 48 are for data and 4 for pilot carriers). However, some differences are needed to be introduced for it to be able to handle the high node mobility and steadily changing vehicular environments. For one, IEEE 802.11p utilizes the 10 MHz frequency channel bandwidth, as opposed to the 20 MHz of IEEE 802.11a, to make the signal more robust to fading and other propagation effects. Table 9 lists down the resulting data rates for IEEE 802.11p using different modulation and coding schemes (MCSs), with the 3, 6 and 9 Mbps required for all ITS-S. The duration of one OFDM symbol is 8μs, with the number of data bits per symbol depending on the MCS used [9].

Transfer rate (Mbit/s)	Modulation scheme	Coding rate	Data bits per OFDM symbol	Coded bits per OFDM symbol
3	BPSK	1/2	24	48
4.5	BPSK	3/4	36	48
6	QPSK	1/2	48	96
9	QPSK	3/4	72	96
12	16-QAM	1/2	96	192
18	16-QAM	3/4	144	192
24	64-QAM	2/3	192	288
27	64-QAM	3/4	216	288

Table 9. MCS and data rates for IEEE 802.11p [9]

Figure 10 illustrates the physical packet structure for IEEE 802.11p, and the fields are briefly described in Table 10. The preamble and signal fields are transmitted using BPSK, while the MCS varies for the data part.

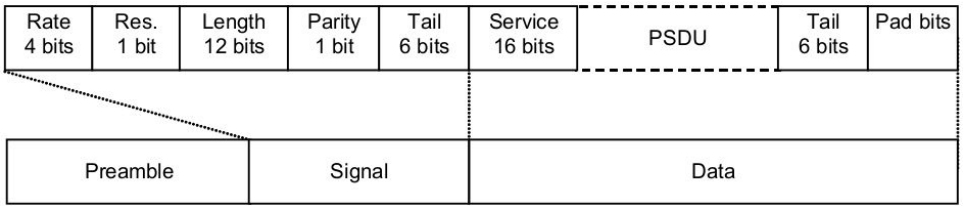


Figure 10. IEEE 802.11p PHY packet structure [9]

Field	Subfield	Description	Duration (µs)
Preamble	N/A	Synchronizing receiver. Consists of a short and a long training sequence.	32
Signal	Rate	Specifies the transfer rate at which the data field in the PPDU will be transmitted.	8
	Reserved	For future use.	
	Length	The length of the packet.	
	Parity	Parity bit.	
	Tail	Used for facilitating decoding and calculation of rate and length subfields.	
Data	Service	Used for synchronizing the descrambler at receiver.	Depending on selected transfer rate and packet length.
	PSDU	The data from the MAC layer including header and trailer, i.e. MPDU.	
	Tail	Used for putting the convolutional encoder to zero state.	
	Pad bits	Bits added to reach a multiple of coded bits per OFDM symbol (i.e. 48, 96, 192, 288)	

Table 10. IEEE 802.11p PHY packet fields [9]

2.3.4.1. ITS-G5 Frequency and Channel Allocation

ITS-G5 frequencies are allocated depending on their purpose of use, which also differ on performance requirements. To enable various ITS applications, one control channel (CCH) and seven service channels (SSH) are allocated [9].

	Channel type	Center Frequency (MHz)	Frequency range (MHz)	IEEE channel number	Channel spacing (MHz)	Default data rate (Mbit/s)	Tx power limit (dBm EIRP)	Usage
ITS-G5A	G5-CCH	5900	5895-5905	180	10	6	33	ITS road safety related applications
	G5-SCH 2	5890	5885-5895	178	10	12	23	
	G5-SCH 1	5880	5875-5885	176	10	6	33	
ITS-G5B	G5-SCH 3	5870	5865-5875	174	10	6	23	ITS non-safety

	G5-SCH 4	5860	5855-5865	172	10	6	0	applications
ITS-G5C	G5-SCH 7	Refer to ETSI EN 301 893	5470-5725	94-145	several	Dependent on channel spacing	30 (DFS master) 23 (DFS slave)	RLAN (BRAN, WLAN)
ITS-G5D	G5-SCH 5	5910	5905-5915	182	10	6	0	Future ITS applications
	G5-SCH 6	5920	5915-5925	184	10	6	0	

Table 11. ITS-G5 channels

2.3.4.2. Enhanced Distributed Coordination Access

IEEE 802.11p uses a MAC algorithm known as the Enhanced Distributed Coordination Access (EDCA). It works like the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) algorithm but allows the prioritization of data traffic. It defines separate queues corresponding to different access categories (ACs). In the order of lowest to highest priority, these are: AC_BK (Background), AC_BE (Best effort), AC_VI (Video) and AC_VO (Voice) [9][10].

AC	TC ID	CW (min)	CW (max)	AIFS	Intended Use
AC_VO	0	3	7	58 μs	High priority DENM
AC_VI	1	7	15	71 μs	DENM
AC_BE	2	15	1023	110 μs	CAM
AC_BK	3	15	1023	149 μs	Multihop DENM, other data traffic

Table 12. ITS-G5 Traffic classes

2.3.4.3. Decentralized Congestion Control

In an ITS ad hoc network, the network topology constantly varies, and in particular, the number of vehicles within the communication range is unpredictable. In the case of high density scenarios, the communicating vehicles may require a number of resources beyond the capacity of the channel. As such, the Decentralized Congestion Control (DCC) mechanism is necessary to avoid channel congestion and allow a fairer access to the limited resources. The way DCC works is that the vehicle adapts its transmission parameters according to the measured channel load. Moreover, the applications running in the upper layers must be aware of the channel load situation to be able to prioritize among different possible transmissions [11]. Thus, DCC operates across several layers as shown in Figure 11.

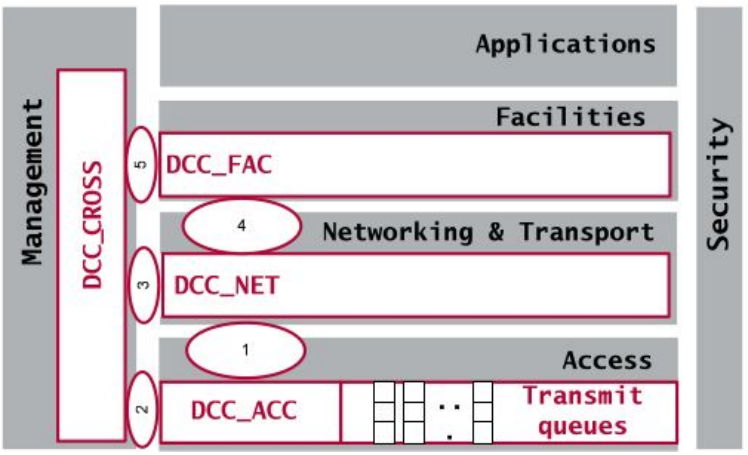


Figure 11. DCC architecture [11]

The different DCC access techniques used to control the channel load is described below.

- Transmit Power Control (TPC) - adjusts the output power, such that it is lowered to reduce the resulting interference in high load scenarios.
- Transmit Rate Control (TRC) - adjusts the time between consecutive packets, such that it is increased in high density scenarios.
- Transmit Data rate Control (TDC) - adjusts the transfer rate, such that it is lowered at high load scenarios.

2.3.4.4. Outside the Context of a BSS

One of the ITS requirements, particularly in safety-related applications, is minimizing latency/delay in the vehicular environment. To satisfy this criterion, a new operation mode called Outside the Context of a BSS (OCB) is introduced in IEEE 802.11p. This is activated by configuring the Management Information Base (MIB) parameter `dot11OCBActivated` to `true`. In this mode, communication outside a Basic Service Set (BSS) is possible, which eliminates the need for the vehicle to undergo the MAC authentication and association phases. Moreover, since it does not try to join a BSS, frequency channel search is also unnecessary, as a predefined channel must be set by default [9]. Both of these features contribute to the reduction of overhead and latencies in the network.

2.3.5. Management and Security

The Management entity is used to configure the ITS-S and exchange information among the different horizontal layers of the ITS-S reference architecture. The Security entity provides services geared towards secure and private communications [12].

These two vertical protocol entities were out of scope of the project.

2.4. Basic Services

As previously mentioned, the main goal of the facilities layer is to provide services to the application layer. This section focuses on two facilities, which play an important role on the realization of different ITS applications, particularly those that improve traffic safety and efficiency.

2.4.1. Cooperative Awareness Basic Service

The Cooperative Awareness (CA) basic service is a mandatory facility for all ITS-Ss, and is responsible for generating, processing and managing the Cooperative Awareness Message (CAM).

CAM is periodically sent by an ITS-S to all ITS-Ss within its communication range using single hop communications to create cooperative awareness. For instance, knowing how close the surrounding vehicles are enables applications such as collision avoidance to prevent possible road casualties. In particular, this message includes status information (speed, position, time, etc.), as well as attribute information (vehicle type, dimensions, etc.). The frequency of CAM generation, or the time interval between consecutive CAMs, must be at least 100 ms and not exceeding 1000 ms. Moreover, the CAM generation time, or elapsed time from the instant at which the CAM generation is triggered to that when the CAM reaches the networking and transport layer, must be less than 50 ms. The CAM format is given by the Abstract Syntax Notation One (ASN.1) unaligned packed encoding rules (PER) [13].

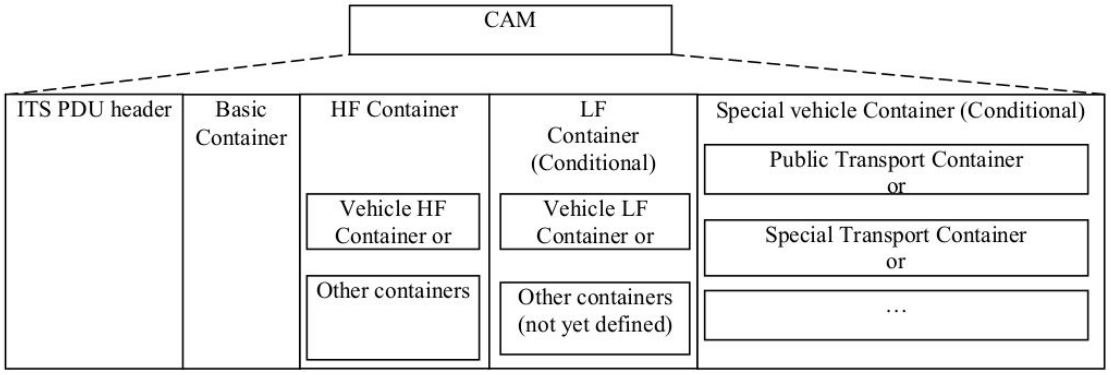


Figure 12. CAM general structure [13]

The structure of the CAM is shown in Figure 12. Depending on the type of ITS-S, some parts may be omitted. The containers are briefly described below, while more details are available in [13].

- ITS PDU header - specifies the protocol version, message type, ITS-S ID
- Basic container - includes the station type, geographical position during CAM generation
- High-frequency (HF) container - indicates dynamic or fast-changing vehicle information

- Low-frequency (LF) container - specifies static or slow-changing vehicle information
- Special vehicle container - includes additional information depending on the vehicle role indicated in the LF container

With reference to [14], a summary of the possible values for the mandatory fields is presented in Table 13 and Table 14.

ITS PDU header field	Description
`protocolVersion`	version of ITS message
`messageID`	type of message 1: Decentralized Environmental Notification Message (DENM) 2: CAM 3: Point of Interest (POI) 4: Signal Phase and Timing (SPAT) 5: MAP 6: In-vehicle Information(IVI) 7: Electric vehicle recharging spot reservation (EV_RSR) 8: Tyre Information System (TIS), Tyre Pressure Gauge (TPG) 9: Traffic Light Signal Request Message 10: Traffic Light Signal Request Status Message 11: Electrical Vehicle Charging Spot Notification 12: Services Announcement Extended Message 13: Radio Technical Commission for Maritime Services (RTCM) Message
`statitionID`	identifier of originating ITS-S

Table 13. ITS PDU header

Basic container field	Description
`StationType`	type of ITS-S 0: unknown 1: pedestrian 2: cyclist 3: moped 4: motorcycles 5: passenger car 6: bus 7: light truck 8: heavy truck 9: trailer 10: special vehicle 11: tram 15: RSU
`ReferencePosition`	specifies the geographical position of the ITS-S, including the: - latitude - longitude - position confidence ellipse (accuracy of geographical position) - altitude

Table 14. CAM basic container

2.4.2. Decentralized Environmental Notification Basic Service

The Decentralized Environmental Notification (DEN) basic service is the facility in charge of generating, processing and managing the Decentralized Environmental Notification Message (DENM).

Unlike CAM, DENM is an event-triggered message that is disseminated to warn about (detected) hazardous events. It is transmitted to all the users within the affected area using multi-hop communications. For instance, DENM could be transmitted to notify about road accidents or that an emergency vehicle is approaching. Having been properly informed, the users can then make the appropriate maneuver and act accordingly, such as giving way to an approaching ambulance. The DENM format is also defined by ASN.1 unaligned PER [15].

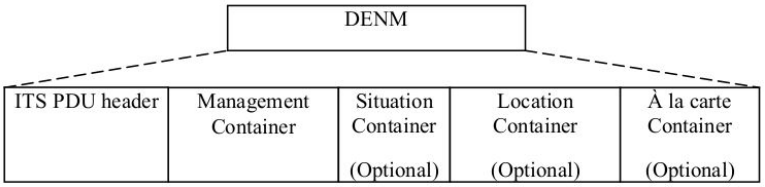


Figure 13. DENM general structure [15]

The message structure of the DENM is depicted in Figure 13. Same with CAM, some parts may not be included depending on the type of transmitting ITS-S. A brief description of the containers is presented below, while in-depth information can be found in [15].

- ITS PDU header - specifies the protocol version, message type, station ID
- Management container - includes information related to managing the DENM
- Situation container - describes the detected event
- Location container - indicates the location of the event
- À la carte container - specifies additional useful information not included in the other containers

Although the DEN basic service is also an important ITS facilities layer entity for the realization of C-ITS, it was out of scope of the project.

2.5. Related Work

As IEEE 802.11p in C-ITS is a relatively mature technology, there are already a number of scientific work pertaining to its system performance evaluation. This section narrows down these references to those that are more relevant to the goals of the project.

The paper of [16] presented one of the initial implementations of IEEE 802.11p on a Linux system. While it provided a description on the modifications made in the Linux kernel, it did not specify how the system was verified to check its operation using the IEEE 802.11p frequencies. The paper of [17] started by conducting a survey on different wireless cards to determine their suitability, followed by an explanation of the changes made on the Linux kernel. The project then upgraded a commercial device to use the applicable

wireless cards previously studied. However, the tests performed were more focused on evaluating whether the performance of the wireless cards met the requirements specified in the standards, such as in terms of throughput, switching channel timing and power transmission. In [18], city-scale field tests were carried out using low-cost, open-source prototypes, also built by modifying the Linux kernel. However, while the tests involved actual exchange of CAMs, the prototypes were not tested against a commercial OBU or RSU to confirm its interoperability. Another implementation was presented in [19], where the system was tested using a commercial V2X platform. In this case, the test packets used employed the US WAVE protocol stack, as opposed to the CAMs of the EU ITS-G5 stack. While the testbed in [20] allowed switching to the 760 Mhz (Japan) when the 5.9 GHz (US and UK) communication fails, the testbed was only tested using IP packets.

The study of [21] investigated the practical limits of cooperative awareness in vehicular communication by carrying out small-scale field tests, before performing large-scale simulations using the Geometry-based Efficient propagation Model for V2V communication (GEMV^2) and SUMO. Central to their analysis was a metric called the Neighboring Awareness Ratio (NAR), defined as the ratio of the number of vehicles from which a CAM was received over the total number of vehicles within a given range. The paper suggested a direct correlation between the Packet Delivery Ratio (PDR) and NAR, which both decreased as the distance between communicating vehicles increased. The maximum communication range and cooperative awareness were very much affected by the link quality and propagation conditions. Moreover, it was concluded that after a certain threshold value, increasing the CAM transmission rate did not anymore significantly improve the NAR, but only increased the channel load. On the other hand, increasing the transmit power while lowering the transmit rate, was a more effective way to increase the NAR, but this also increased the interference of far away vehicles. Thus, it was important to find the balance between awareness and interference.

In [22], the CAM and DENM messaging services were evaluated by utilizing commercial IEEE 802.11p transceivers in a testbed deployed in a real driving environment. From its analysis of the Received Signal Strength Indicator (RSSI) measurements, it verified that the signal quality was heavily affected by the line-of-sight conditions and distance between the transmitter and receiver, as well as their relative altitude. The signal quality directly impacted the PDR, such that high signal quality translated to high PDR. The experiment also showed that the PDR was lower for faster vehicle speeds (although this is not the case when later analyzing the simulation results of this project). Moreover, the Connection Time and Connection Distance metrics had been defined, which referred to the time and distance elapsed between the first and last correctly received CAMs. Although these metrics are capable of providing deeper insights on the performance of the CA basic service, the paper did not perform extensive study to elaborate on them. In addition, the test only involved two entities (one vehicle/OBU and one RSU), with some of the results being highly dependent on a single environment topology.

As depicted in the above discussion of related work, further study still needs to be done in order to evaluate the performance of the CA basic service. In particular, this project was focused on implementing the CA basic service and understanding its effectiveness in providing awareness among the ITS entities of a cooperative transport system.

3. On-board/Roadside Unit Development

Field tests to capture real-world measurements for the purpose of evaluating the CA basic service usually require the use of special V2X platforms. In addition to these devices being really costly, they are also proprietary systems, and as such, access to their implementation is very limited. Thus, one of the goals of this project was to develop a low-cost OBU/RSU that implemented the ETSI C-ITS protocol stack using open-source software and commercial off-the-shelf hardware. Ultimately, the developed OBU/RSU was expected to work with any dedicated V2X hardware using the 5.9 GHz channel, such that it would be able to transmit/receive CAMs to/from a commercial device. The project focused on the delivery of CAMs by implementing the CA basic service that is a mandatory ITS facilities layer entity for all ITS-Ss.

This chapter details the steps carried out in developing the OBU/RSU, including the creation of the CAM application, and the selection and modification of a suitable hardware platform that supports IEEE 802.11p.

3.1. Vanetza Library

The Vanetza library is an open-source implementation of the ETSI C-ITS protocol suite [23]. It includes the following protocols and feature, which were utilized in the OBU/RSU development.

- GeoNetworking
- Basic Transport Protocol (BTP)
- Support for ASN.1 messages

As discussed in Section 2.3.3, the GeoNetworking protocol is a network layer protocol that uses geographical information in routing packets over the ITS ad hoc network, while BTP is responsible for the end-to-end unreliable delivery of packets. In developing the source code for the project, we took advantage of Vanetza's built-in functions that implement the services provided by these protocols. Moreover, this library contains ASN.1 modules for easy CAM encoding and decoding.

Vanetza also includes a demo application called `socktap`, which runs Vanetza on top of Linux raw packet sockets allowing it to work without dedicated V2X hardware [23]. It has several variants; however, for this project, `socktap-cam` was used. This variant periodically sends CAMs, and as such, it could be readily used to function as the CAM transmitter application. On the other hand, the source code for the CAM receiver application needed to be developed from scratch.

3.2. CAM Receiver Source Code

The central idea when creating the CAM receiver application was being able to correctly receive and parse all the incoming CAMs. Some of the built-in functions needed were actually private functions in the Vanetza library. In order to have access on a number of specific variables, these functions were copied as is to the `rcv_parse.cpp`.

The CAM receiver started by defining a buffer in `router_context.cpp`. This buffer continuously received the CAMs that were periodically transmitted by the `socktap-cam` application.

The contents of the buffer were then processed layer-by-layer in `rcv_parse.cpp`. The parsing of the received message started from the physical layer, where 0 byte was assigned. Moving on to the link layer, the 14 bytes represented the source (6 bytes), destination (6 bytes) and ether type (2 bytes), which was set to `0x8947` for ITS GeoNetworking. In the GeoNetworking layer, only the Basic and Common headers were processed. The optional extended header was skipped, as the project always assumed single-hop broadcast and disabled security attributes for simplification. The payload size following the GeoNetworking header was indicated in the Common header. As BTP had a fixed 4 bytes for its header, the remaining bytes corresponded to the CAM itself, which was handled in the upper layer. Vanetza included built-in containers that help to easily decode the CAM. This parsing process was repeated for each of the CAM received.

Note that, depending on the Vanetza version in use, it may be necessary to make some adjustments to properly represent certain field values included in the CAM. In this project, modifications were made in the data representation of the speed and geographical information (latitude, longitude).

3.3. Setting up the CAM Transmitter and Receiver Environment

This section provides a brief description on how to set up the CAM transmitter and receiver applications that were key components of the OBU/RSU project. The commands corresponding to the following procedure are listed on Appendix A.1. As mentioned in Section 3.1, both applications used Vanetza to implement specific protocols and features. For this reason, the first step was to obtain Vanetza from [23], and subsequently, install and compile the library along with the software dependencies.

Following this, the applications must be compiled. Since both of them build upon raw packet sockets, it was necessary to run them with special privileges [23]. Moreover, the GPS functionality was a prerequisite. In this project, a GPS signal was emulated instead of using a GPS receiver. In this case, a recorded GPS file was played in the background while running the application, as discussed in Appendix A.2.

With no error occuring, both applications were executed. The initial testing was done using only the loopback interface (i.e., no hardware platform needed). In this setup, it was necessary to start the applications in separate command windows of the local PC.

3.4. Implementation using Raspberry Pi

Raspberry Pi is a low-cost, credit card sized computer, which was originally intended to promote education by making the platform accessible to everyone who wants to learn how to program. Same with a standard computer, several peripherals could be connected, including a monitor, keyboard, mouse, etc. The recent models offer more advanced functionalities, such as support for bluetooth, Power-over-Ethernet (PoE) and

dual-band WiFi (2.4GHz/5GHz) [24]. Moreover, the Raspberry Pi boasts a number of advantages with its low price, small size, customizability and availability in the market. Taking these into consideration, the Raspberry Pi 3 Model B+ was the selected hardware platform to implement the OBU/RSU during the initial stages of the project development. At the time of working on the project, this was the latest model of Raspberry Pi, with the specifications available in [24]. The Emlid Raspbian was used as the operating system [25].

Figure 14. Raspberry Pi 3 Model B+ [24]

3.4.1. OBU/RSU Testing using Raspberry Pi

The first attempt to test the OBU/RSU project using an actual hardware platform was using the Raspberry Pi. In this case, two devices were employed, one continuously sent CAMs, while the other received and displayed the message contents in the command line. The first thing to do was to download the files to the Raspberry Pis, and then install and compile all the necessary programs as explained in Section 3.3. Before being able to proceed with the testing, an ad hoc network must be configured to enable wireless communication between the two devices. The process of setting up an ad hoc network is detailed in Appendix A.3. Afterwards, the CAM transmitter was executed on one Raspberry Pi, and the receiver on the other device. At this point, it was confirmed that both applications were working properly using the Raspberry Pis.

The next step was to configure the devices to operate using the IEEE 802.11p frequencies. This was crucial for the OBU/RSU project to be able to work with commercial OBU/RSUs, which utilize the 5.9 GHz frequency band. By default, the Raspberry Pi 3 Model B+ supports 5 GHz WLAN, and as such, modifications had to be made to tune it to the desired ITS frequencies.

One good reference for the implementation of IEEE 802.11p on Linux is [17]. It presented a detailed guide on how to modify the ATH9K driver to work with IEEE 802.11p. ATH9K is a Linux kernel driver for Atheros PCI/PCI Express (PCIe) wireless cards, with the compatible chipsets/devices listed in [26]. Unfortunately, after doing some research, it was concluded that the current Raspberry Pi models are unable to operate using IEEE 802.11p as they do not support PCI interfaces. Thus, it was necessary to implement the OBU/RSU project using a different hardware.

3.5. Implementation using APU2

The APU2 platform developed by PC Engines is a single-board computer for networking. Although it is not as small as the Raspberry Pi, it offers similar advantages such as cheap price and ease of purchase. More importantly, APU2 is a suitable device that meets the hardware requirements discussed in the previous section. It has two mini PCIe slots, along with the several features listed in [27].

Figure 15. APU2 platform [27]

The wireless module used is the WLE200NX [28]. This mini PCIe module also supports both 2.4 GHz and 5 GHz WLAN, similar to the Raspberry Pi 3 Model B+. A key difference is that it uses the Qualcomm Atheros AR9280 chipset, which is compatible to the ATH9K kernel driver. As such, it was then possible to modify the driver to support IEEE 802.11p.

Figure 16. WLE200NX wireless module [28]

3.5.1. Linux Wireless Architecture

As mentioned in Section 2.3.4, one of the MAC layer modifications specific to IEEE 802.11p is the introduction of the OCB mode. In the latest Linux kernel versions, this mode could be enabled using the kernel configuration menu. However, further modifications have to be made in the ATH9K kernel driver to fully implement IEEE 802.11p on a Linux system [17].

Before diving into the detailed procedure of how to modify the driver, an overview of the Linux wireless architecture, depicted in Figure 17, will be briefly discussed. Applications and processes run in the user space on top of the kernel space, while the hardware device drivers can be found at lowest level of the architecture.

Figure 17. Linux wireless architecture [16]

- `mac80211`

  `mac80211` is a framework for developing Soft MAC drivers. Wireless cards can be classified as either a Full MAC (also known as Hard MAC) or Soft MAC. The difference between the two is that the former manages the IEEE 802.11 MAC Sublayer Management Entity (MLME) in the hardware, while the latter, implements it in the software. The MLME is a management entity where the PHY MAC state machines reside. Soft MAC is more commonly used nowadays, as it enables more precise control of the hardware, including the implementation of the IEEE 802.11 frame management in software [29].

- `cfg80211`

  `cfg80211` is a configuration API for IEEE 802.11 Linux-based devices. It acts as a link between the user space and the drivers. Full MAC drivers target `cfg80211`, as `mac80211` is only for Soft MAC devices. Moreover, it provides support for regulatory compliance through the use of `wireless-regdb` and `CRDA` [29].

- `nl80211`

  `nl80211` is used to configure `cfg80211`. Using the Netlink socket, it enables the communication between the user space and the kernel. It is responsible for the

user space part of the configuration management of wireless devices, while `cfg80211` is for the kernel space [29].

- `iw`

  `iw` is a Command-Line Interface (CLI) utility based on `nl80211` and is used for configuring wireless devices [29].

- `wireless-regdb`

  The `wireless-regdb` is a regulatory database used by `CRDA`. Each country has its own regulations on frequency allocation and acceptable transmission power levels. These are specified in the `wireless-regdb` to help ensure regulatory compliance. The database is conveniently placed in the user space, so that changes could be performed without upgrading the kernel. Moreover, together with the database, an RSA signature is embedded in the generated binary file (`regulatory.bin`) to ensure the authenticity of the file [29].

- `CRDA`

  The `Central Regulatory Domain Agent` (`CRDA`) in the user space uploads the wireless regulatory domain into the kernel. The kernel triggers the CRDA once it has detected changes in the regulatory domain [29].

3.5.2. ATH9K Driver Modifications

This section describes how to modify the ATH9K Linux kernel driver to support IEEE 802.11p. The complete list of commands is available in Appendix A.4.

The basic requirement to start this process was to download the Linux kernel from [30]. The kernel version used was 4.20.7, which was the latest stable kernel at the time of developing the project. The functionalities supported by the kernel depend on its version. For instance, the OCB mode has been implemented from version 3.19. As the project used a much later version, this operation mode was already available in the kernel configuration menu.

After obtaining the Linux kernel source file and installing the software dependencies, the following ATH9K driver source codes had to be modified.

- `drivers/net/wireless/ath/ath9k/ani.c`
- `drivers/net/wireless/ath/ath9k/common-init.c`
- `drivers/net/wireless/ath/ath9k/hw.h`
- `drivers/net/wireless/ath/ath9k/main.c`
- `drivers/net/wireless/ath/regd.c`

Details about the changes are specified in Appendix A.5. In order to support IEEE 802.11p, the modifications were related to enabling the OCB mode and ITS-G5 frequencies, as well as updating the number of channels. As mentioned above, it is possible that these changes have already been incorporated in the later kernel versions. Thus, it is advisable to check before modifying them.

The next step was to configure the kernel modules to be included. To ensure the validity of the kernel configuration file, an option was to copy the configuration file of the currently

running kernel. After obtaining this, executing the `make menuconfig` command launched the kernel configuration menu, where the kernel modules could be enabled/disabled accordingly. Several options were shown, including those related to the components of the Linux wireless architecture discussed in the previous section. The configuration used in the project is presented in Appendix A.6, where the settings in the paths listed below were examined. In particular, `Verbose OCB debugging` must be enabled.

- `Networking support > Wireless`
- `Device Drivers > Network device support > Wireless LAN`
- `Networking support > Wireless > Select mac80211 debugging features`

Following this, compile the kernel for the changes to take effect, then install the kernel modules and the kernel. It was also necessary to enable the kernel for boot and restart the system, as explained in Appendix A.4.

3.5.3. Verifying `iw`

As discussed in Section 3.5.1, `iw` is used to manage WLAN in Linux, much like the `ifconfig` for wired networks. The procedure for setting up `iw` is detailed in Appendix A.7. Note that it is important to check if the running `iw` version supports the OCB mode. This was done by entering the following command, with the expected output also specified below.

```
$ /sbin/iw | grep -i ocb

        dev <devname> ocb leave

        dev <devname> ocb join <freq in MHz> <5MHZ|10MHZ> [fixed-freq]
```

Figure 18. Testing the `iw` program

3.5.4. `wireless-regdb` Modifications

`wireless-regdb` is a regulatory database that helps ensure compliance with the regulations enforced by each country. The procedure for setting up the `wireless-regdb` is explained in Appendix A.8.

One of the challenges encountered when implementing IEEE 802.11p on Linux was the modification of `db.txt` to add the ITS-G5 channels and transmission power values. With reference to [17], the initial goal was to add a new country `AA` (in `db.txt`), and under which, the said channels and power values were to be specified. Ideally, issuing the `iw reg set AA` command changes the wireless regulatory domain in use to `AA`. Subsequently, the corresponding changes in `db.txt` will be reflected upon executing the `iw reg get` and `iw list` commands. However, after several trials, this method did not work. Instead, the ITS-G5 channels could not be enabled when trying to add them by creating a new country `AA`. A possible cause was that the Atheros wireless card used in

the project was limited to the pre-defined countries only, and as such, it did not allow the use of non-existent countries like `AA`.

A workaround was to add the channels and power values under an existing country, such as `country ES`, depicted in Appendix A.8. Moreover, it was necessary to set `CTRY_SPAIN` in `regd.c` as indicated in Appendix A.5. After doing these modifications, make sure to restart the system for the changes to take effect. At this point, the ITS-G5 channels and power levels were then displayed when issuing both the `iw reg get` and `iw list` commands.

```
apu@loki:~$ iw reg get
global
country ES: DFS-ETSI
        (2400 - 2483 @ 40), (N/A, 20), (N/A)
        (5150 - 5250 @ 80), (N/A, 20), (N/A), NO-OUTDOOR
        (5250 - 5350 @ 80), (N/A, 20), (N/A), NO-OUTDOOR
        (5470 - 5725 @ 80), (N/A, 26), (0 ms), DFS
        (5840 - 5935 @ 10), (N/A, 30), (N/A)
        (57240 - 65880 @ 2160), (N/A, 40), (N/A), NO-OUTDOOR

phy#1
country ES: DFS-ETSI
        (2400 - 2483 @ 40), (N/A, 20), (N/A)
        (5150 - 5250 @ 80), (N/A, 20), (N/A), NO-OUTDOOR
        (5250 - 5350 @ 80), (N/A, 20), (N/A), NO-OUTDOOR
        (5470 - 5725 @ 80), (N/A, 26), (0 ms), DFS
        (5840 - 5935 @ 10), (N/A, 30), (N/A)
        (57240 - 65880 @ 2160), (N/A, 40), (N/A), NO-OUTDOOR

phy#0
country ES: DFS-ETSI
        (2400 - 2483 @ 40), (N/A, 20), (N/A)
        (5150 - 5250 @ 80), (N/A, 20), (N/A), NO-OUTDOOR
        (5250 - 5350 @ 80), (N/A, 20), (N/A), NO-OUTDOOR
        (5470 - 5725 @ 80), (N/A, 26), (0 ms), DFS
        (5840 - 5935 @ 10), (N/A, 30), (N/A)
        (57240 - 65880 @ 2160), (N/A, 40), (N/A), NO-OUTDOOR
```

Figure 19. `iw reg get` output

```
* 5850 MHz [170] (30.0 dBm)
* 5855 MHz [171] (30.0 dBm)
* 5860 MHz [172] (30.0 dBm)
* 5865 MHz [173] (30.0 dBm)
* 5870 MHz [174] (30.0 dBm)
* 5875 MHz [175] (30.0 dBm)
* 5880 MHz [176] (30.0 dBm)
* 5885 MHz [177] (30.0 dBm)
* 5890 MHz [178] (30.0 dBm)
* 5895 MHz [179] (30.0 dBm)
* 5900 MHz [180] (30.0 dBm)
* 5905 MHz [181] (30.0 dBm)
* 5910 MHz [182] (30.0 dBm)
* 5915 MHz [183] (30.0 dBm)
* 5920 MHz [184] (30.0 dBm)
* 5925 MHz [185] (30.0 dBm)
```

Figure 20. `iw list` output

3.5.5. Verifying CRDA

The procedure for setting up the CRDA is presented in Appendix A.9. It involved copying the public keys installed by the wireless-regdb, and generating the regulatory.bin file. To verify that everything was configured correctly, the following command was issued, which specifically checked the CRDA and regulatory.bin. The expected output is shown below.

```
$ sudo /sbin/regdbdump /lib/crda/regulatory.bin | grep -i ocb
        country 00: invalid
        (5850.000 - 5925.000 @ 20.000), (20.00), NO-CCK, OCB-ONLY
```

Figure 21. Testing CRDA and regulatory.bin

3.5.6. OCB Interface and IEEE 802.11p Channel Configuration

After enabling the ITS-G5 channels, the next step was to create an OCB interface by configuring a wireless interface to OCB mode. This OCB interface was then used in attempting to join an IEEE 802.11p channel. The detailed procedure is explained in Appendix A.10.

To verify whether the whole process was successful, the iw dev and iwconfig commands were used. In the following figures, it was confirmed that the OCB interface was indeed able to join an ITS-G5 channel. In particular, this channel was the Control Channel (CCH), which had a center frequency of 5900 MHz, channel number of 180 and channel spacing of 10 MHz. At this point, we had successfully implemented IEEE 802.11p on a Linux system. It was then possible to test applications implemented in the upper layers, such as transmitting and sending CAMs using the ITS-G5 frequencies. Note that following this procedure, it was necessary to set up the Vanetza library and CAM applications in the APU2 device as specified in Section 3.3.

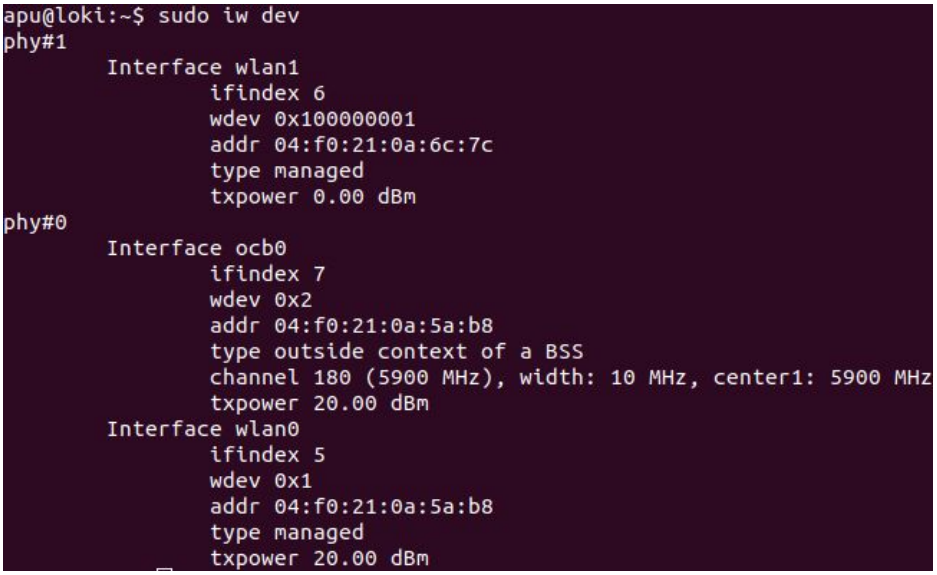


Figure 22. `iw dev` output

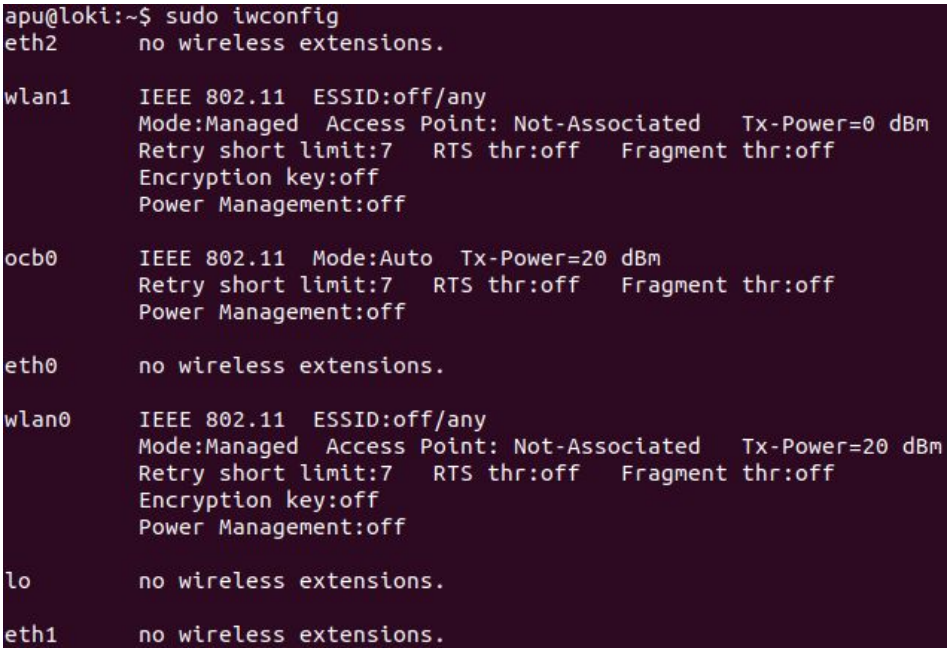


Figure 23. `iwconfig` output

4. IEEE 802.11p-based Simulator Enhancement

This chapter provides an overview of the existing IEEE 802.11p-based simulator, including details about extending its functionality to enable the experimental evaluation of the CA basic service. It also describes the scenarios and parameter configurations utilized in the simulations.

4.1. Simulation Framework Overview

The IEEE 802.11p-based simulator is composed of several simulation frameworks of different functionality. This section provides an overview of the simulator architecture illustrated in Figure 24.

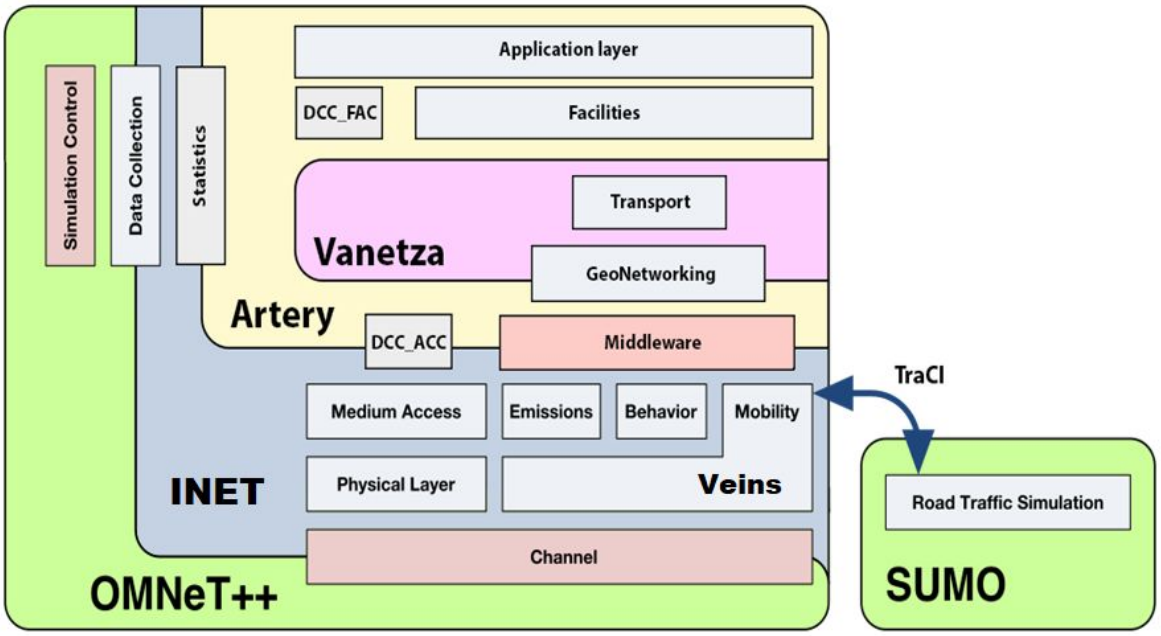


Figure 24. Simulation Framework Overview [V2X-Arch]

The Objective Modular Network Testbed in C++ (OMNeT++) is an extensible and modular simulation library and framework [31]. It works by assembling individual components/modules (written in C++) to larger components and models using NED, which is a network description language for creating network topologies. This modularity makes it easy for the models to be reused and incorporated to different applications. Moreover, although OMNeT++ is mainly used for building network simulators, it is also considered as a network simulation platform by its growing number of users. Model frameworks are often used in conjunction with OMNeT++ to implement more specific functionalities.

The INET simulation framework is an open-source library containing various models to simulate communication networks, and is particularly written for the OMNeT++ environment [31]. Some of its features include models for the Internet stack (IPv4, IPv6,

TCP, UDP) and wired/wireless interfaces (Ethernet, IEEE 802.11), and support for physical environment modelling (propagation model, presence of obstacles). Moreover, INET could be used as a base for creating other simulation framework such as Veins.

The Veins simulation framework is an open-source library consisting of numerous models specific to vehicular networking [32]. For instance, it has an IEEE 802.11p model, which includes multi-channel operation, QoS channel access and noise/interference effects. Veins simulation requires parallel execution of two simulators, namely OMNeT++ (for network simulation) and SUMO (for road traffic simulation). The interaction between these simulators is made possible using a TCP socket and a standardized protocol known as the Traffic Control Interface (TraCI). As such, the movement of vehicles in SUMO is represented as the movement of nodes in OMNeT++.

Simulation of Urban MObility (SUMO) is an open-source road traffic simulator [33]. It allows the creation of different road topologies for simulation, such as freeway and Manhattan grid scenarios, as well as the experimentation of various mobility models. Moreover, it is microscopic, as vehicles are individually modelled (including vehicle color, shape, maximum speed, route), and move independently through the network. By default, the simulations are deterministic, with the option of adding randomness to the simulation.

Artery was originally developed as an extension of Veins, although, it could now be used independently [34]. Artery corresponds to the application and facilities layers, which enable the generation of CAMs and DENMs. Moreover, Artery's middleware provides common facilities to the multiple ITS-G5 services running on individual vehicles.

Lastly, Vanetza is an open-source implementation of the ETSI C-ITS protocol suite. In particular, it implements the GeoNetworking (for routing) and BTP (for transport) protocols, and supports ASN.1 messages (CAM, DENM) [23].

4.2. IEEE 802.11p Simulator Functions

This section provides a brief description of the functions that were created and modified to extend the functionality of the existing simulator. These functions were designed and programmed in order to be able to acquire the statistics, which were defined according to the goals of the project.

4.2.1. `GlobalMapper`

The `GlobalMapper` function (in `artery/src/artery/application`) maintains a global database that keeps track of all the nodes present in the scenario. Each entry in the database represents a single node, and has the following information:

- Node name
- Module name of node
- `TxTime` - current simulation time during CAM generation
- `TxPos` - position of transmitting node during CAM generation
- `TxVel` - speed of transmitting node during CAM generation

The node itself uploads these information to the database whenever it generates and periodically sends a new CAM. This ensures that these information are constantly being

updated/refreshed, so that the `GlobalMapper` is always aware of the current status of all nodes at any given time.

4.2.2. `CaService`

The `CaService` function (in `artery/src/artery/application`) is responsible for checking the CAM trigger conditions, as well as generating and transmitting the CAMs. It includes a step-by-step process of creating these messages using the CAM containers discussed in Section 2.4.1. Moreover, this is where the node updates the `GlobalMapper` of its latest status information (position, speed, etc.) every CAM generation instance. The function also implements an empirical way to know the average speed of each node, by dividing the total time spent in the scenario over the total distance travelled.

4.2.3. `Rx`

The `Rx` function (in `artery/extern/inet/src/inet/linklayer/ieee80211/mac`) handles the reception of CAMs from other nodes. It counts the following statistics in the link layer:

- Total number of received packets
- Total number of correctly received packets
- Total number of erroneous packets received

As these are taken in the link layer, they do not include the packets that were filtered out based on power-related threshold values in the physical layer, namely the Receiver Sensitivity, Energy Detection and SNIR threshold. Moreover, the error model used is the `Ieee80211NistErrorModel`, which is a readily available model for IEEE 802.11 network interfaces. It works by using the SNIR value in the computation of the BER.

Each (receiving) node maintains a local database called `lastKnownPos`. It stores the following information for all of the surrounding (transmitting) nodes.

- `startEntry` - current simulation time when database was updated
- `lastPos` - last known position of the transmitting node
- `TxRxDist` - distance from the transmitting node

Each database entry represents a single (transmitting) node. The corresponding entry is updated whenever a CAM is correctly received from a surrounding node. In effect, the receiving node is able to keep track of all of its neighboring vehicles, with their perceived position/distance constantly being updated in its local memory. As such, it can detect when a neighboring vehicle is already too close compared to a defined safe distance. This information is especially useful in applications such as collision avoidance warning.

The database is periodically checked every 100ms to ensure the validity of its entries. Moreover, an expiry time of 2s (or 20 CAMs for a CAM transmit frequency of 100ms) is configured. That is, when an entry is not updated after 2s in the database, it is automatically deleted.

4.2.4. `SystemMonitor`

The `SystemMonitor` (in `artery/src/artery/application`) is a new function mainly created for statistics collection, and providing better data representation and visualization of results. During the initial stages of the project development, the results were plotted using solely the OMNeT++ analysis file, which placed a constraint on the way graphs were created (i.e., limited to the built-in features of OMNeT++). To solve this problem, the `SystemMonitor` function was developed to allow customizing the plots by being able to modify more properties. For instance, it allows changing the number of bins to adjust the granularity of the histograms. Moreover, it is capable of exporting the raw data into an Excel/csv file, giving more freedom on how results would be post-processed. In this way, it helps in analyzing and understanding the results better by using more effective visualizations.

4.3. SUMO Scenario

This section details the the creation of different road topologies and mobility scenarios using the SUMO simulation framework.

4.3.1. SUMO Files

Each scenario is created using three SUMO files [35], which are located in `v2x-arch_11p/artery/scenarios/artery/my_roads`.

4.3.1.1. Network File (`*.net.xml`)

The network file contains the description of the physical topology of the scenario. This may include the roads, intersections, traffic logics and even roundabouts. Using the SUMO naming convention, the roads or streets are referred to as edges, and the intersections as junctions or nodes. That is, two edges are connected by junctions. Figure 25 shows the contents of the network file.

```
<?xml version="1.0" encoding="UTF-8"?>

<net version="0.27" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/net_file.xsd">

    <location netOffset="0.00,0.00" convBoundary="0.00,0.00,1000.00,0.00" origBoundary="2.852656,41.882297,2.898048,41.899504"
projParameter="+proj=utm +zone=31 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"/>

    <edge id="E0" from="J0" to="J1" priority="1">
        <lane id="E0_0" index="0" allow="passenger" speed="33.33" length="1000.00" shape="0.00,-11.55 1000.00,-11.55"/>
        <lane id="E0_1" index="1" allow="passenger" speed="33.33" length="1000.00" shape="0.00,-8.25 1000.00,-8.25"/>
        <lane id="E0_2" index="2" allow="passenger" speed="33.33" length="1000.00" shape="0.00,-4.95 1000.00,-4.95"/>
        <lane id="E0_3" index="3" allow="passenger" speed="33.33" length="1000.00" shape="0.00,-1.65 1000.00,-1.65"/>
    </edge>

    <junction id="J0" type="unregulated" x="0.00" y="0.00" incLanes="" intLanes="" shape="0.00,-0.05 0.00,-13.15"/>
    <junction id="J1" type="unregulated" x="1000.00" y="0.00" incLanes="E0_0 E0_1 E0_2 E0_3" intLanes="" shape="1000.00,-13.15 1000.00,-0.05"/>

</net>
```

Figure 25. SUMO network file

The `location` field specifies details about the network projection in case the original network was not using Cartesian coordinates, and therefore, needed to be transformed.

The `edge` field describes the created lanes including the allowed type of vehicles (or pedestrian), speed limit, lane length and geometry. The `junction` field defines the lanes that the junctions connect.

The network file may be created using a tool called NETEDIT. It is a graphical editor for creating and modifying networks, as illustrated in Figure 26.

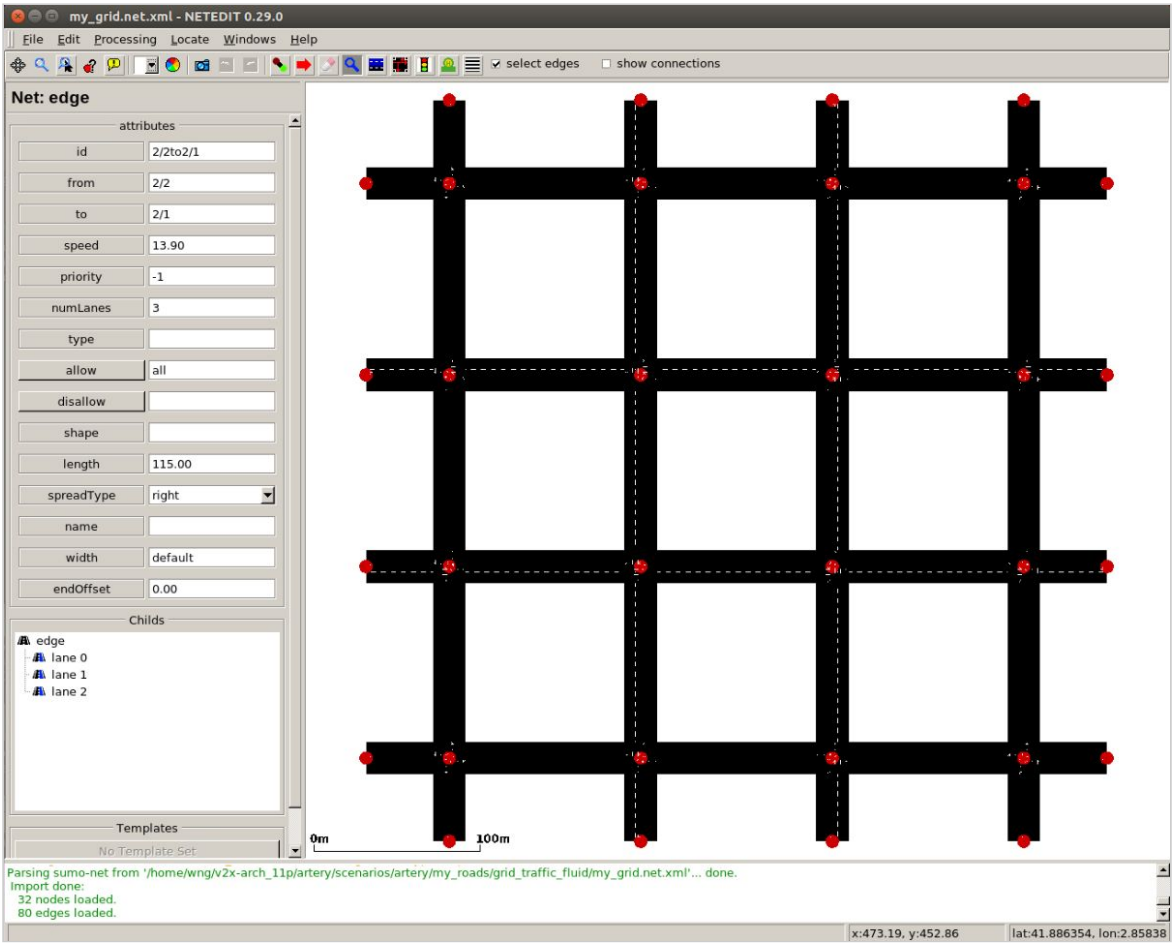


Figure 26. NETEDIT

4.3.1.2. Routes File (`*.rou.xml`)

The routes file specifies the vehicle types and routes for the vehicles in the simulation. The `vehicles type` field includes the physical properties of the vehicle, such as shape and color, as well the maximum speed and minimum gap from the vehicle ahead. Different routes are identified by their `route id`, and each of them defines the relevant edges and direction of movement of vehicles (e.g., going to the right). Moreover, a `flow` contains the following information, which control how the vehicles are inserted in the scenario and how they behave during the simulation.

- `type` - vehicle type previously defined
- `begin` - departure time of first vehicle
- `period` - insertion period of vehicles

- `end` - departure time of last vehicle
- `departLane` - lane on which the vehicle will be inserted
- `departSpeed` - speed with which the vehicle will enter the scenario
- `departPos` - position at which the vehicle will enter the scenario
- `route` - route of vehicle previously defined

```
<routes>

        <!--..................-->
        <!-- VEHICLE TYPES -->
        <!--..................-->
        <vType  id="type1"              color="green"          guiShape="passenger/sedan"          maxSpeed="33.33"              minGap="2"/>
        <vType  id="type2"              color="yellow"         guiShape="passenger/sedan"          maxSpeed="33.33"              minGap="2"/>
        <vType  id="type3"              color="blue"           guiShape="passenger/sedan"          maxSpeed="33.33"              minGap="2"/>
        <vType  id="type4"              color="grey"           guiShape="passenger/sedan"          maxSpeed="33.33"              minGap="2"/>
        <vType  id="type5"              color="red"            guiShape="passenger/sedan"          maxSpeed="33.33"              minGap="2"/>
        <vType  id="type6"              color="white"          guiShape="passenger/sedan"          maxSpeed="33.33"              minGap="2"/>

        <!--..................-->
        <!-- ROUTES for CAR -->
        <!--..................-->
        <!-- example horizontal route: "3hr" == edge 3 horitzontal to right -->
        <route id="route0hr" edges="E0"/>

        <!--..............................-->
        <!-- CAR VEHICLES nodes[16..x]...-->
        <!--..............................-->
        <!-- for perpetual flow, "end=" > simulation time in .cfg file-->
        <flow   id="flow0hr"                                   type="type2"
                        begin="0"                              end= "7200"                         period="0.001"
                        departLane="random"                    departSpeed="random"
                        departPos="random"
                        route="route0hr"/>

</routes>
```

Figure 27. SUMO routes file

4.3.1.3. Configuration File (`*.sumo.cfg`)

The configuration file specifies the associated network and routes files for a given scenario. Moreover, it is possible to configure the `step-length`, which is the granularity of the simulation and has a minimum value of 1 ms. It also corresponds to the time interval with which vehicle positions are updated.

```
<?xml version="1.0" encoding="iso-8859-1"?>

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/sumoConfiguration.xsd">

    <input>
        <net-file              value="single_road_4lanes.net.xml"/>
        <route-files           value="single_road_4lanes.rou.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="710"/>
        <step-length value="0.001"/>
    </time>

    <report>
        <no-step-log value="true"/>
    </report>

    <gui_only>
        <start value="false"/>
    </gui_only>

</configuration>
```

Figure 28. SUMO configuration file

The parameter values are given in meters (for distance), seconds (for time) and meters per second (for speed). As detailed in [35], other values may be configured, apart from those used in the project. Moreover, additional attributes may be defined in the SUMO files. This allows customizing the scenarios according to the requirements and individual goals of the simulations.

4.3.2. Physical Topologies

Different road topologies were used in the project. One of which was the highway scenario, which simulated direct line-of-sight (LOS) conditions and non-stop driving (i.e., no traffic lights, intersections). The other one was the Manhattan grid scenario, which helped in understanding the effects of walls and buildings, as well as intersections. Moreover, the project defined a statistical region in the scenarios, highlighted in red below. Statistics were only recorded in this area to eliminate border effects. For instance, less vehicles may be present at either end of the highway scenario compared to its central region, and this consequently affects the carrier sense mechanism employed by IEEE 802.11p. Although the exact coordinates of the statistical area are given below, these could be readily modified in the `omnetpp.ini` file, as discussed in Appendix B.2.

4.3.2.1. Highway Scenario

The length of the highway was 1000 m, with the statistical region corresponding to the central 400 m (i.e., 300m < x < 700 m). In Figure 29, the highway had only four lanes, with each lane having a width of 3.2 m. However, the scenario could be easily configured using NETEDIT to have varying number of lanes. In the project, 4-, 8- and 16-lane highways were used. Moreover, both unidirectional and bidirectional scenarios were created to deduce any implication on the system performance. For instance, a 4-lane bidirectional highway scenario had 2 lanes in each direction.

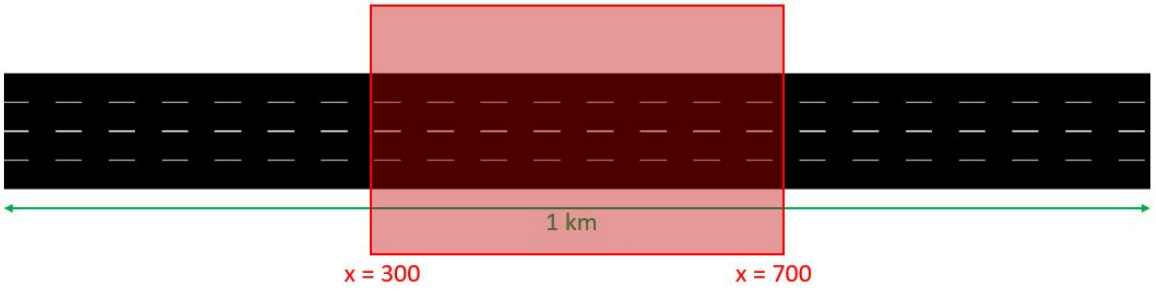


Figure 29. 1km highway scenario

4.3.2.2. Manhattan Grid Scenario

The Manhattan grid scenario measured 445 m x 445 m, with the statistical area bounded by 30 m < x < 375 m and 111.25 m < y < 333.75 m, which was an approximation of the central region of the grid. The presence of walls could also be configured in `omnetpp.ini`.

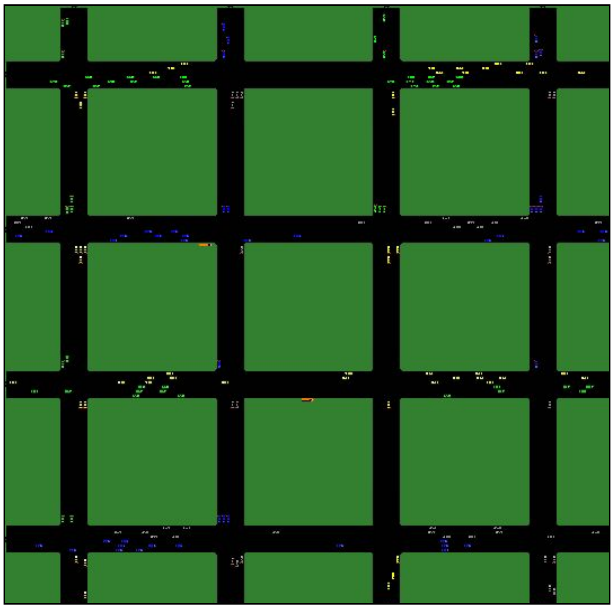


Figure 30. Manhattan grid scenario

4.3.3. Classification of Vehicle Speed and Density

Defining different vehicle speeds was an effective way to understand how the CA basic service performed in sparse and dense situations. Slow vehicle speed translated to dense environments with more vehicles being packed in the scenario. On the contrary, vehicles moving fast required greater braking distance, and thus, less vehicles fit into the same scenario, creating light vehicle density.

The following are the different factors affecting the vehicle speed in SUMO [35].

- Maximum vehicle speed (in `*.rou.xml`)
- Maximum lane speed (in `*.net.xml`)
- `speedFactor`, which is a speed multiplier (not used in the project)
- Car-following model, which defines vehicle speed in relation to the vehicle ahead

  There are different car-following models available. However, the project used the default model that is the `carFollowing-Krauss`. Basically, this model always selects the maximum speed safe enough for the vehicles to be able to stop before any collision occurs.

- `departSpeed` and `arrivalSpeed` (in `*.rou.xml`)

  `departSpeed` is the speed with which the vehicle enters the scenario, while `arrivalSpeed` is the speed when the vehicle reaches its destination or leaves the scenario.

Each sets an upper bound, but the actual speed implemented is the minimum speed resulting from all of these factors. A simple and effective way to control the speed was to assign different values for the maximum vehicle speed (`maxSpeed`) in `*.rou.xml` as explained later in this section.

The `minGap` parameter in `*.rou.xml` could also be configured to control the vehicle density. `minGap` is the empty space after the vehicle ahead. While the default value is 2.5 m, the project used 2 m to fit in more vehicles in the scenario. Moreover, `departPos` parameter in `*.rou.xml` was set to `random` to populate the scenario with vehicles faster. Configuring this to `random` allowed the vehicles to enter the scenario in any position. This was in contrast to having all the vehicles enter the scenario through the same entry point, such as from the leftmost side of the highway scenario.

In the project, the following classification was defined according to the maximum vehicle speed (`maxSpeed`) specified in `*.rou.xml`.

Classification of Speeds	Maximum Vehicle Speed [m/s]
Fast	33.33
Moderate	17.00
Slow	3.00

Table 15. Vehicle speeds

Table 16 lists down the average number of vehicles in different highway scenarios, which turned out to be the same for the unidirectional and bidirectional cases. As the highway spans 1 km in length, an approximation of the vehicle density was obtained by dividing the average number of vehicles by the number of lanes. This resulted to the number of vehicles per km per lane. Figure 31 shows the relationship between the vehicle speed and density.

Classification of Vehicle Speeds	Approximate number of vehicles in the scenario			Vehicle Density [vehicles/km/lane]
	4 Lanes	8 Lanes	16 Lanes	
Fast	145	290	586	37.00
Moderate	216	420	845	53.00
Slow	357	706	1421	89.00

Table 16. Average number of vehicles in different bidirectional highway scenarios

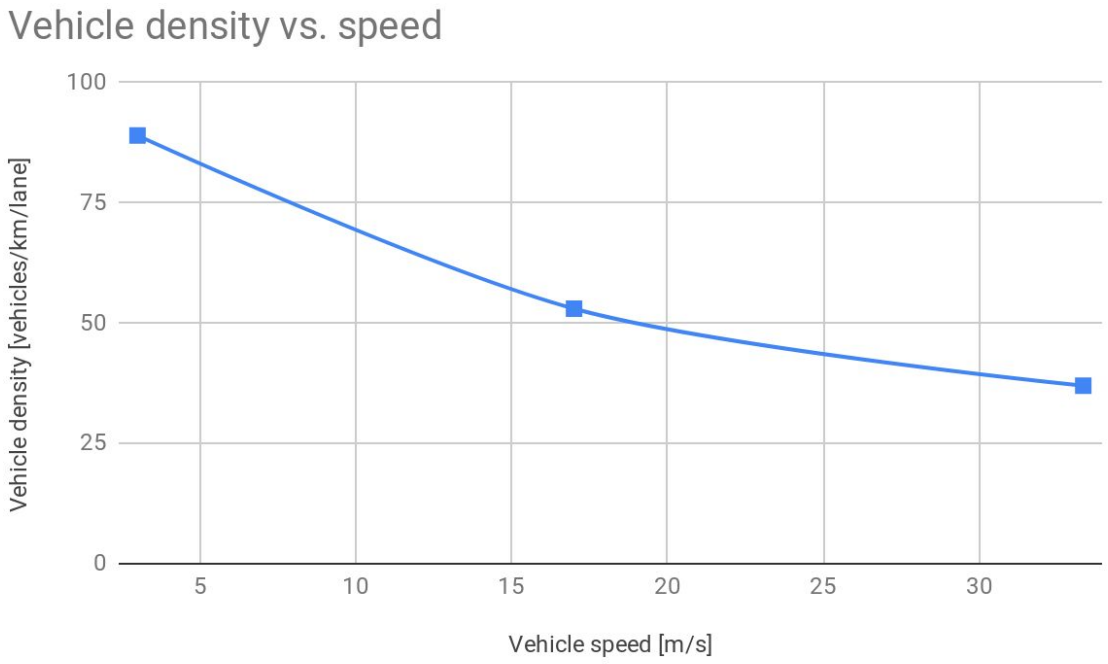


Figure 31. Relationship of vehicle density and speed

4.4. Simulation Parameters

Table 17 provides a summary of the parameters used in the simulations. These parameters could be readily configured in the `omnetpp.ini` file, as detailed in Appendix B.2.

Category	Parameter	Value
Node	Operation mode	802.11p
	Carrier frequency	5.9 GHz
	Bandwidth	10 MHz
	Channel number	180
	Modulation	BPSK
	Bitrate	6 Mbps
	Transmitter power	200 mW
	Receiver sensitivity	-95 dBm
	Energy detection	-95 dBm
	SNIR threshold	16 dB
Medium	Obstacle loss type	{dielectric, ideal, " "}

	Path loss type	Rayleigh fading
	Path loss alpha	3
	Background noise type	Isotropic scalar
	Background noise power	-110 dBm
Scenario	Topology	{bidirectional highway, unidirectional highway, grid}
	Maximum vehicle speed	{3, 17, 33.33 m/s}
	CAM message period	{100, 200, 500, 1000 ms}
	Mobility model	Krauss (default)
	Simulation time limit	{50 s (highway), 500 s (grid)}
	Warm-up period	{25 s (highway), 300 s (grid)}

Table 17. Simulation parameter values used

The parameter values used to model the individual nodes were selected based on those specified in the standards. In accordance to Table 11, a control channel (CCH) was used, with 10 MHz of bandwidth centered at 5.9 GHz, channel number of 180 and default data rate of 6 Mbps. The nodes were configured to transmit with a power of 200 mW or 23 dBm, which was well below the 33 dBm power limit. The receiver sensitivity was set to -95 dBm, with reference to the power measurements cited in the specifications of commercial V2X devices.

The radio medium was modelled using the Rayleigh fading profile, which allowed simulating highly dense urban environments without direct LOS between the communicating nodes. The corresponding alpha values for urban areas ranged from 2.5 to 3.5, from which a value of 3 was arbitrarily chosen to be used in the simulations. The physical environment allowed more realistic simulations by enabling or disabling the walls/buildings in the scenario through the obstacle loss type field. Setting the field to either dielectric or ideal enabled the walls, while leaving it blank disabled them. The properties of the walls could be configured in the `walls.xml` file.

The default road topology was the bidirectional highway scenario, although in some cases, the unidirectional highway and grid scenarios were also simulated to understand the impact of certain parameters. Depending on the scenario, the value of the warm-up period and the simulation time differed. Statistics were recorded only upon reaching the warm-up time in the simulation. For this reason, the value of the warm-up period was selected such that the scenario had reached steady-state conditions by the time the statistics were started to be recorded. The ETSI specification indicates that the CAM generation period ranges from 100 to 1000 ms [13]. Simulations were conducted while varying the transmit period to deduce its impact on performance.

5. Results

This chapter is mainly divided into two sections, presenting the results and analyses for the two subtasks of the project.

5.1. On-board/Roadside Unit

As detailed in Section 3.5, the modification of the different entities belonging to the Linux wireless architecture was central to the development of an open-source, low-cost OBU/RSU that supports IEEE 802.11p. In order to check the validity of the entire procedure, as well as the operation of the newly-developed CAM application, the system was tested using a commercial V2X platform, that is, the Cohda Wireless MK5 OBU shown in Figure 32. The test was also meant to prove the interoperability of the developed OBU/RSU with commercial devices, which was one of the goals of the project.

Figure 32. Cohda Wireless MK5 OBU [36]

5.1.1. OBU/RSU Testing

The test setup consisted of the OBU/RSU project and the commercial Cohda Wireless MK5 OBU. Both devices are capable of transmitting and receiving periodic CAMs. For the testing, one device was set to be in transmit mode while the other device was set to be in receive mode, and then their roles were switched afterwards.

In order to monitor the flow of packets in real time, the CAM applications were programmed to continuously print CAM information in the command line, as depicted in the following figures.

```
apu@loki:~/bike-app/build$ bin/bike_app -i ocb0 --gpsd-host localhost
Bike app running
Reading options...
Initialising CAM sender...
Starting runtime at 2019-Mar-12 17:36:38.166763
sent packet to ff:ff:ff:ff:ff:ff (50 bytes)
Skipping CAM, because no good position is available, yet.
Skipping CAM, because no good position is available, yet.
Skipping CAM, because no good position is available, yet.
sent packet to ff:ff:ff:ff:ff:ff (50 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
sent packet to ff:ff:ff:ff:ff:ff (99 bytes)
```

Figure 33. CAM transmitter application

```
Size of received message: 102
Parsing of received message starts now...
*****************************************************************
*                              LINK                             *
*****************************************************************
Length: 14
Source: 04:e5:48:00:00:01
Destination: ff:ff:ff:ff:ff:ff
Ether type: 8947
*****************************************************************
*                          GEONETWORKING                        *
*****************************************************************
----------Basic Header----------
Length: 4
Version: 1
Next header (0:ANY, 1:COMMON, 2:SECURED): 1
Reserved: 0
Lifetime: 80
Remaining hop limit: 1
----------Common header----------
Length: 8
Next header (0:ANY, 1:BTP_A, 2:BTP_B, 3:IPv6): 2
Reserved1: 0
Header type: 80
Traffic class: 2
Flags: 128
Payload: 48
Maximum hop limit: 1
Reserved2: 0
*****************************************************************
*                    BASIC TRANSPORT PROTOCOL                   *
*****************************************************************
----------BTP Header B----------
Length: 4
Destination port: 53511
Destination port info: 0
```

Figure 34. CAM receiver application (part 1)

```
*************************************************************
*                          APPLICATION                      *
*************************************************************
Size of CAM msg: 44
----------ItsPduHeader----------
Protocol version: 1
Message ID: 2
Station ID: 3232238231
----------CoopAwareness----------
Generation delta time: 15340
----------Cam parameters (Basic container)----------
Station type: 5
Reference position - Altitude: 0
Reference position - Longitude: 8.58829
Reference position - Latitude: 49.8559
Reference position - Position confidence ellipse - Semi-major orientation: 0
Reference position - Position confidence ellipse - Semi-major confidence: 400
Reference position - Position confidence ellipse - Semi-minor confidence: 400
----------Cam parameters (High frequency container)----------
Present: 1
Choice - Heading value: 509
Choice - Heading confidence: 127
Choice - Speed value: 13.88
Choice - Speed confidence: 4
Choice - Drive direction: 2
Choice - Longitudinal acceleration value: 0
Choice - Vehicle length value: 49
Choice - Vehicle length confidence indication: 3
Choice - Vehicle width: 18
Choice - Curvature value: -1612
Choice - Curvature confidence: 6
Choice - Curvature calculation mode: 0
Choice - Yaw rate value: 0
----------END OF CAM----------
```

Figure 35. CAM receiver application (part 2)

The CAM transmitter application in Figure 33 could be seen sending periodic CAMs with a message size of 99 bytes to the destination broadcast address `ff:ff:ff:ff:ff:ff`. In the initial part of the transmission, the application took a few seconds to retrieve the device's GPS coordinates, which, for this project, were acquired from an emulated GPS signal playing in the background (in a separate command window). The GPS coordinates were necessary because without a valid position, the application was not able to successfully transmit CAMs. Moreover, looking at the executed command, the application sent the packets through the `ocb0` interface that was configured to join and use the 5.9 GHz channel, as discussed in Section 3.5.6. This configuration was verified during the testing, where the CAMs transmitted by the APU2 using IEEE 802.11p were successfully received by the Cohda device.

In Figure 34 and Figure 35, the contents of the packets received by the CAM receiver application were printed out in the command line and were classified according to the different ITS layers. An analysis of the CAM message fields is presented in the next section using a Wireshark capture. Same as with the transmitter, the CAMs coming from the Cohda device were received through the configured `ocb0` interface that uses the 5.9 GHz channel. At this point, the proper operation of both the CAM transmitter and receiver applications were confirmed. Moreover, being able to continuously transmit and receive using the IEEE 802.11p channel also verified the configurations previously made in the Linux wireless network. The developed OBU/RSU was proven to be functioning as expected and interoperable with a commercial V2X platform.

5.1.2. Analysis of CAM Fields using Wireshark

While displaying the CAM contents in real time using the command line provided a quick way to view them, it did not offer the functionality of recording the packets for offline analysis. For this reason, Wireshark, a well-known network packet analyzer, was used during the testing. In particular, the CAMs coming from the Cohda device and received by the developed OBU/RSU were captured, as illustrated in Figure 36. Note that a special dissector was utilized in order to decode the CAM, which was not possible with the standard Wireshark releases. This section aims to discuss and interpret the different message fields of a CAM. In addition, it verifies whether the contents are in accordance with the ETSI standards discussed in Section 2.3, where a description of the ITS layers and packet header structure is provided.

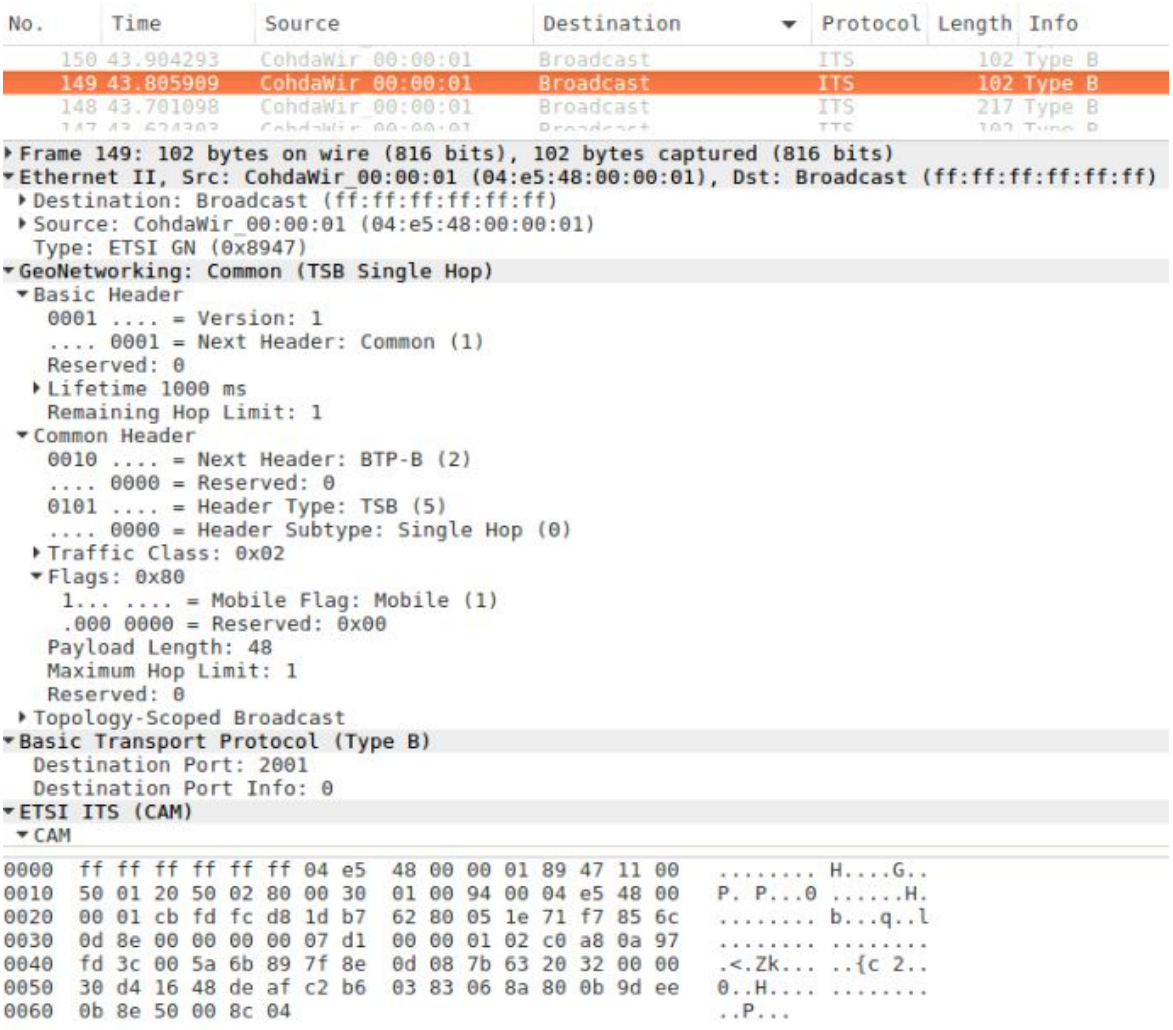


Figure 36. Wireshark capture of a CAM

Figure 36 shows the contents of a 102-byte CAM, with 14 bytes belonging to the Ethernet II field of the ITS access layer. As the CAM originated from the Cohda device, the source

field was set to the device's MAC address, while the destination was a broadcast address. The ether type was `0x8947`, specifically corresponding to ITS GeoNetworking.

Several information were included in the GeoNetworking field, which was mainly divided into the mandatory Basic and Common headers, as well as the optional Extended header. The Basic header took up 4 bytes, with its contents briefly described in Table 6. Based on the Wireshark capture, the GeoNetworking protocol version used was 1; the next header was set to 1, indicating that the Common header followed next; and the values for the lifetime and remaining hop limit were also specified.

GeoNetworking's Common header amounted to 8 bytes, and its fields are defined in Table 7. The next header was set to 2, meaning the header following GeoNetworking was BTP-B. The values of the header type and subtype could be decoded by referring to Table 8, which in this case, indicated topologically-scoped single-hop broadcast. The traffic class ID used in the data traffic prioritization had a value of 2 that pertained to CAM, as specified in Table 12. The flag indicates that the ITS-S was a mobile station. The payload length corresponded to the packet size following the GeoNetworking header. In this case, the 48 bytes belonged to the BTP and CAM (in the ITS facilities layer). As mentioned in Section 3.2, the Extended header was out of scope of the project, but further information could be found in [7].

The BTP field took up 4 bytes in total, which meant that for this specific packet, the message size of the CAM itself was 44 bytes. As specified in GeoNetworking's Common header, the BTP header type was BTP-B that included both the destination port and additional port information in its header structure. With reference to Table 4, destination port `2001` indicated that the ITS facilities layer entity in use was CAM.

Figure 37 shows the actual contents of a CAM in the ITS facilities layer during the testing. The general message structure is illustrated in Figure 12. As depicted in the Wireshark capture, the CAM was broadly grouped into two sections, the ITS PDU header and `cam` fields.

The ITS PDU header specified the version of the ITS message, which was 1 in this case. The `messageID` indicated the type of message, with the value of 2 corresponding to CAM, as listed in Table 13. The `stationID` identified the ITS-S from which the CAM originated.

The `cam` part included the CAM payload, which consisted of the message timestamp (`generationDeltaTime`) and a number of vehicle containers. The basic container is a mandatory container that specifies the type of ITS-S and the geographical position of the ITS-S when the CAM was generated. In this case, the ITS-S was a passenger car with reference to the mapping of values in Table 14. The `referencePosition` provided information about the longitude, latitude and altitude of the ITS-S.

At least one high frequency container is mandatory in each CAM, and includes dynamic or fast-changing attributes. Reference [13] specifies the mandatory fields inside this container, specifically, the heading, speed, drive direction, vehicle length and width, longitudinal acceleration, curvature, curvature calculation mode and yaw rate, which were all present in the captured CAM. The acceleration control and lateral acceleration are optional, along with the lane position, steering wheel angle, vertical acceleration, performance class and CEN DSRC tolling zone.

In addition, there are other containers that may be included, such as the low frequency and special containers. These are optional parts of the CAM and could be seen omitted in the Wireshark capture.

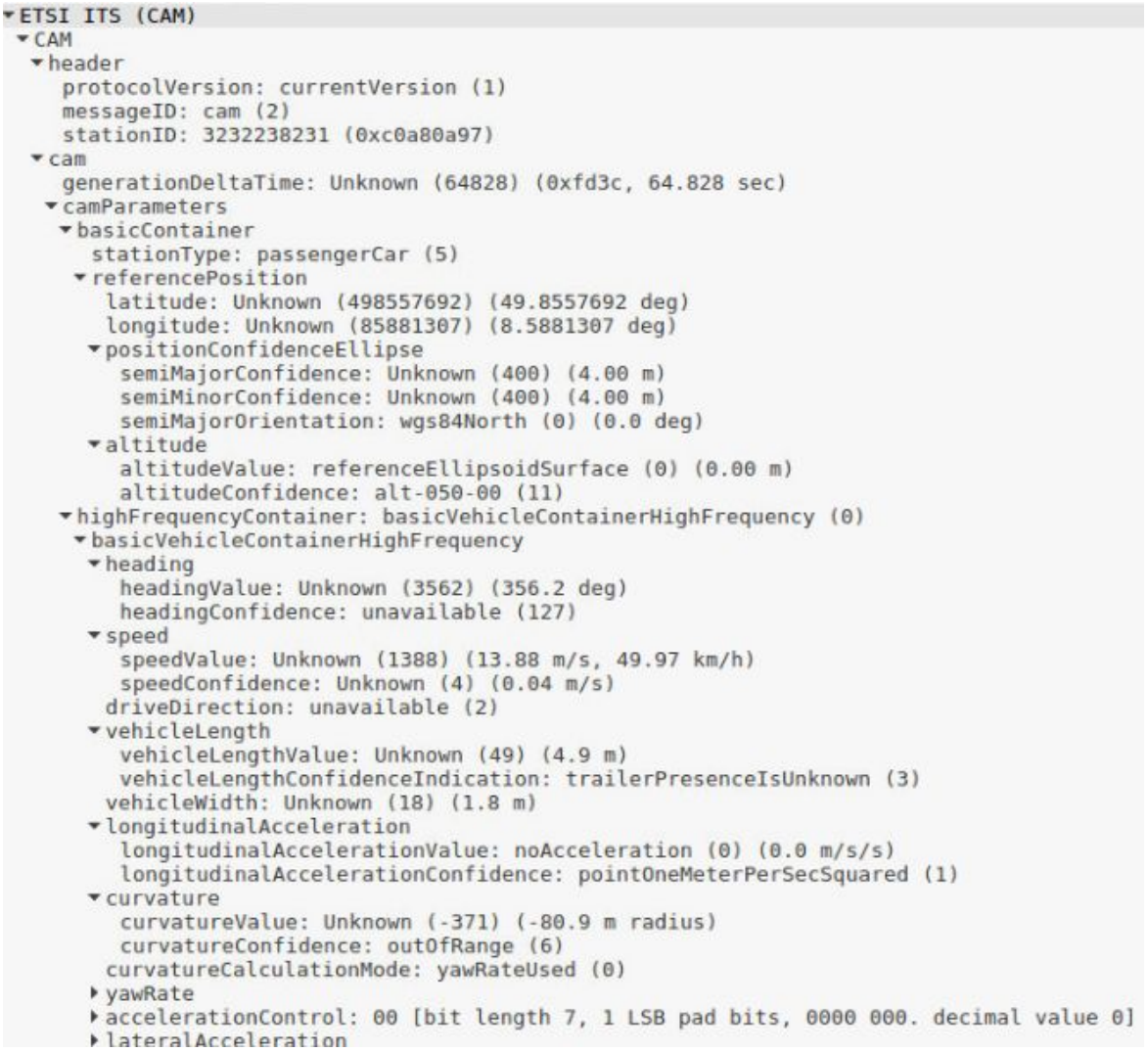


Figure 37. CAM contents in the facilities layer

5.2. IEEE 802.11p-based Simulator

This section presents the results for the experimental evaluation of the CA basic service using the modified IEEE 802.11p simulator. Its performance was studied by varying different parameters such as the lane count, vehicle speed and CAM transmit frequency. For this purpose, a number of performance metrics had been designed and employed, including the Packet Reception Ratio, position error (`deltaPosition`), distance error (`deltaDistance`), and Neighborhood Awareness Ratio. These were plotted according to the Tx-Rx distance, defined as the distance between two communicating nodes

Note that in the following sections, only selected figures, corresponding to specific scenario configurations, are included and used for analyzing the results. However, the complete set of figures are available in Appendix B.5 for reference.

5.2.1. Packet Reception Ratio

The Packet Reception Ratio (PRR) is an ITS access layer metric commonly used by the scientific community in order to evaluate link reliability. The resulting PRRs of the different scenarios were used to better understand the behavior of the upper-layer performance metrics that are discussed in the later sections. In this project, PRR was defined as follows.

$$PRR = \frac{correctly\ received\ packets}{total\ received\ packets\ above\ SNIR\ threshold} \times 100$$

A CAM is considered to be correctly received when it reaches the MAC layer without errors, in addition to satisfying the receiver sensitivity, energy detection and SNIR threshold values in the PHY layer.

5.2.1.1. Effect of Vehicle Speed

The effect of vehicle speed on the resulting PRR is depicted in Figure 38.

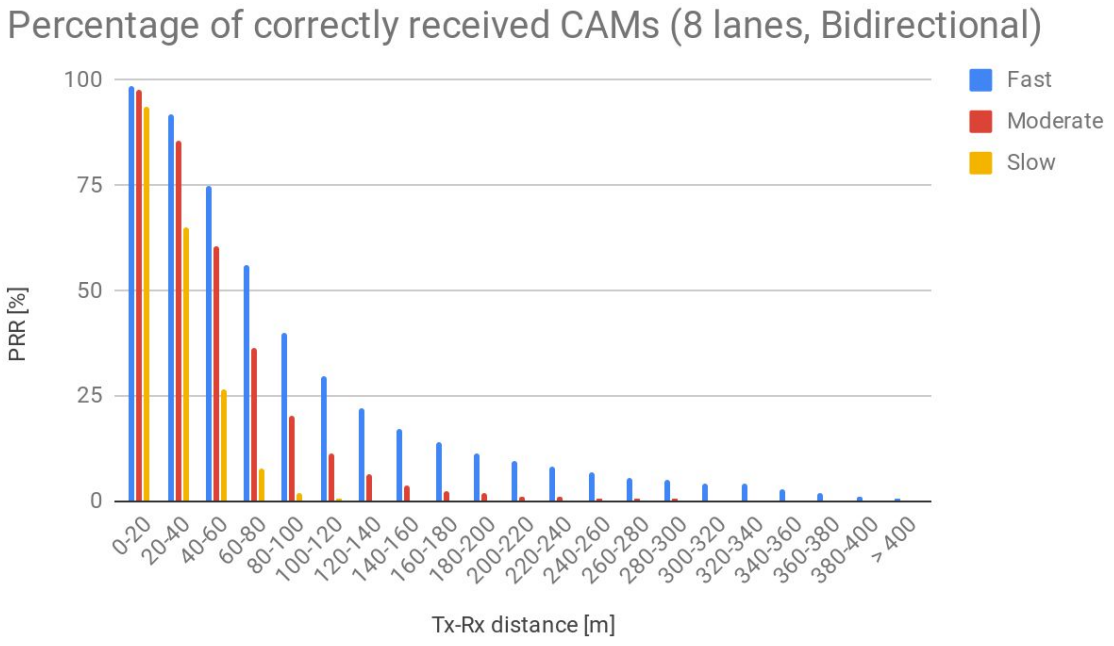


Figure 38. Effect of vehicle speed on PRR

The length of the highway was fixed to be 1 km, and as such, increasing the speed affected the spacing between the vehicles. In the fast speed scenario, the vehicles were more spread out. Since the vehicles were moving fast, a large braking distance had to be maintained between vehicles, as specified by the `Krauss` car-following model. On the other hand, the vehicles were more tightly packed in the slow speed scenario, and thus, only required a small braking distance. For instance, taking the 4-lane case in Table 16, there were 357 vehicles in the slow scenario, compared to only 145 vehicles in the fast scenario. Given these numbers, it could be inferred that the speed is related to vehicle density. The fast speed scenario corresponded to low vehicle density, while the slow speed scenario implied high density. Given a fixed number of lanes in Figure 38, it could be observed that the PRR decreased as the vehicle speed decreased. Slow-moving vehicles created a highly dense environment, with more vehicles being packed in the 1 km highway scenario. This resulted to an increase in interference and packet collisions, and more CAMs being lost, which consequently decreased the PRR. This explains why the slow scenario had much lower PRR values than the fast scenario.

5.2.1.2. Effect of Number of Lanes

The number of lanes had a significant impact on the resulting PRR, since the lane count correlated with the number of vehicles present in the scenario. Given a fixed vehicle speed, increasing the number of lanes translated to an increase in the total vehicle count (i.e., there are more vehicles in a 16-lane scenario than in a 4-lane scenario). As an example, considering the case of fast vehicle speed, there were 145 vehicles in the 4-lane scenario as opposed to the 586 vehicles in the 16-lane scenario, listed in Table 16. With more vehicles accessing the channel to periodically send their CAMs, higher interference was created and the probability of collisions increased, resulting to more packets being lost, thereby decreasing the PRR. This could be observed in Figure 39, where there was a significant difference between the PRRs of the extreme cases, the 4-lane and 16-lane scenarios, considering a fixed vehicle speed. With the 4-lane scenario having less vehicles, and thus, having lower interference and collision probability, its PRR was much higher than those of the 8-lane and 16-lane scenarios.

It could also be generally observed in Figure 38 and Figure 39 that the PRR decreased as the Tx-Rx distance increased. The transmitted signal naturally weakened as the distance increased due to the noise and propagation effects. However, even at much greater distances, there were still few packets being correctly received in some of the scenarios. This could be attributed to the random fluctuations in the Rayleigh fading model.

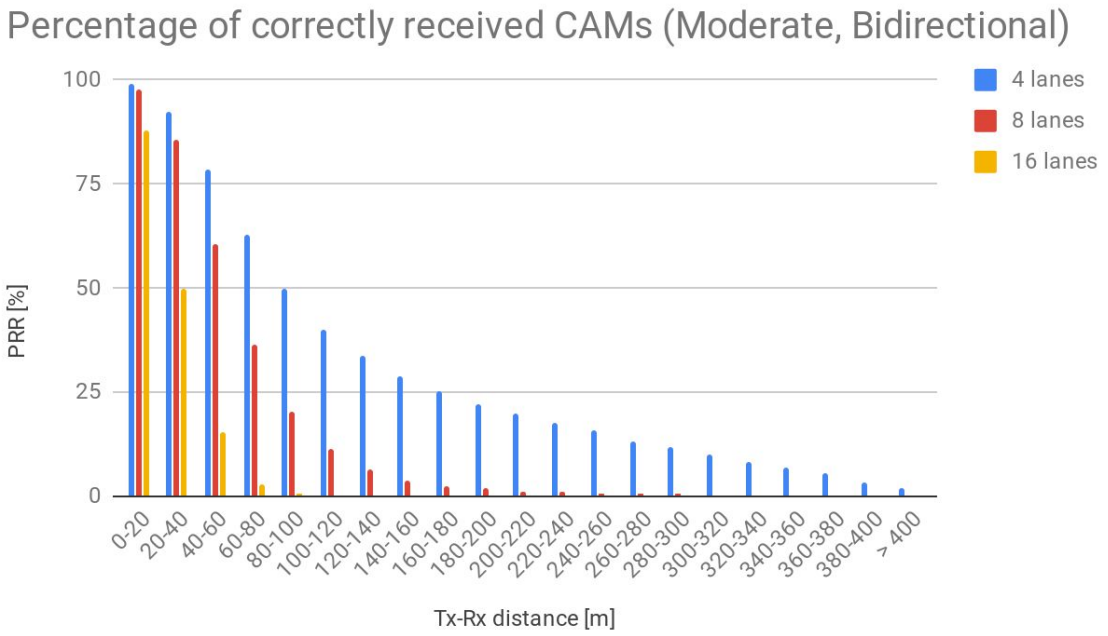


Figure 39. Effect of lane count on PRR

5.2.1.3. Effect of Traffic Flow Direction

The impact of the traffic flow direction is now studied by comparing the PRRs of a unidirectional and a bidirectional highway scenarios. The resulting PRRs are depicted in Figure 40, with both simulations having 4 lanes each and having vehicles running in fast speed.

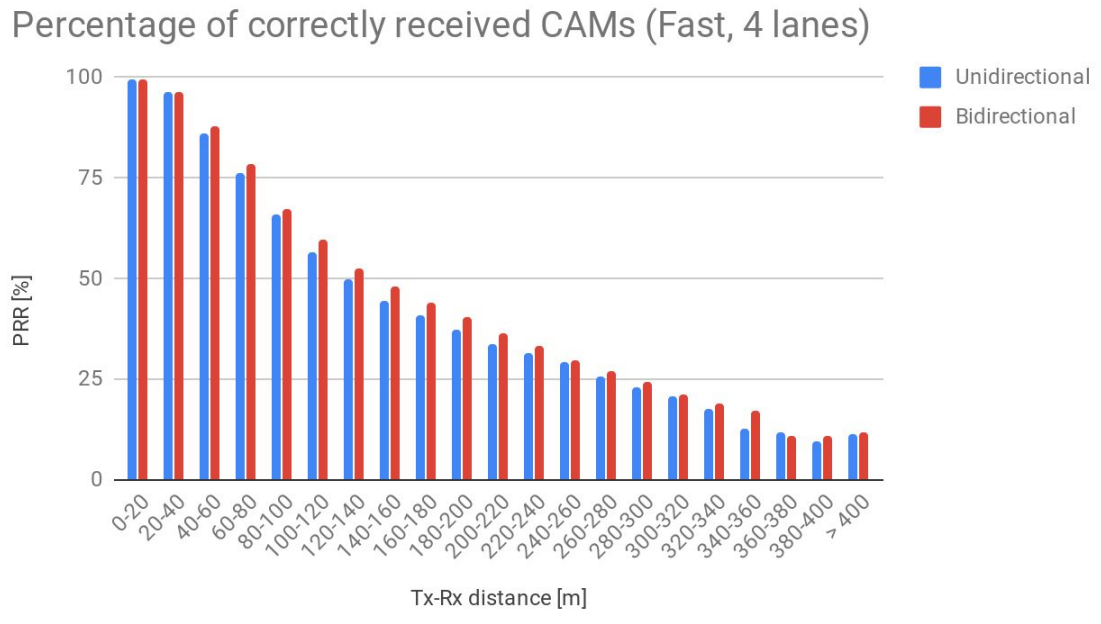


Figure 40. Effect of traffic flow direction on PRR

It could be observed that there was no significant difference between their PRRs. As IEEE 802.11p uses the CSMA/CA mechanism for transmitting CAMs, the resulting collisions were handled in the same manner, irrespective of the direction of vehicle movement. This explains why switching to the unidirectional case did not result to any substantial change in the PRR.

As the PRRs of the unidirectional and bidirectional scenarios were almost the same, the bidirectional case was henceforth set to be the default highway scenario in the succeeding analyses of the other performance metrics.

5.2.1.4. Effect of Walls

In the Manhattan grid scenario, walls can be enabled or disabled to understand their effect on PRR. The presence of walls allows for a more realistic simulation of an urban environment where actual buildings are situated, thereby affecting signal transmissions.

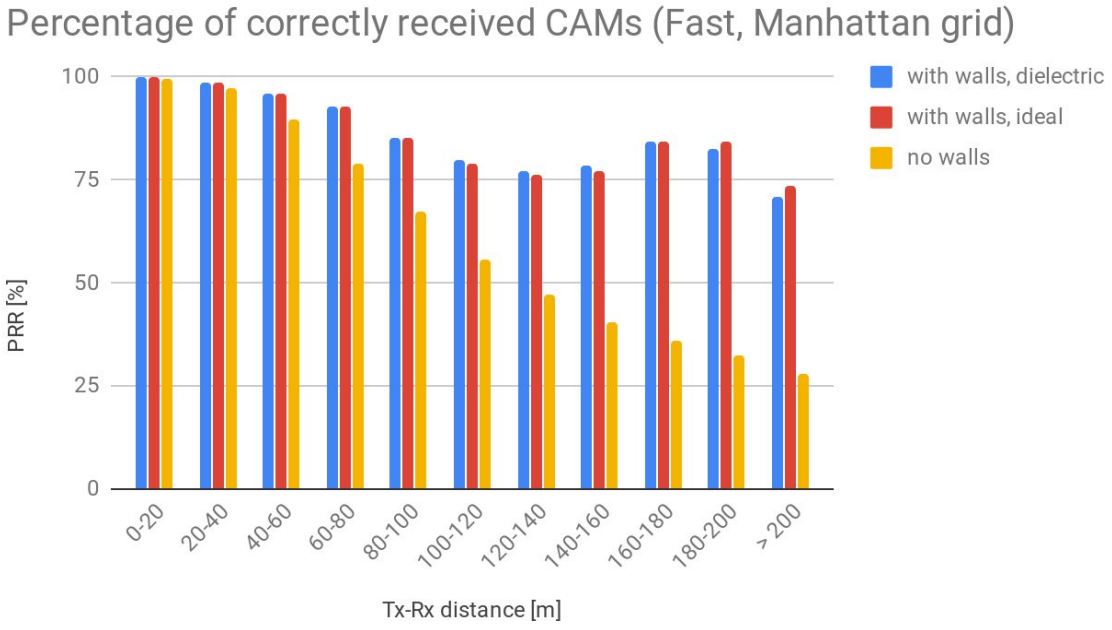


Figure 41. Effect of walls on PRR

A comparison of PRRs is presented in Figure 41 in the case of a fast speed scenario. It could be observed that disabling the walls in the grid scenario resulted to a PRR exhibiting a decreasing trend as the distance between the communicating vehicles increased. This is similar to that of the highway topology, where the transmitted signal weakened as the distance increased, due to the noise and propagation effects, besides the interference caused by neighboring vehicles. When the walls were enabled, the resulting PRR was higher than when the walls were disabled. Most of the neighboring vehicles that were supposed to cause interference (and decrease the PRR), were out of sight and hidden behind the walls. In that case, the walls blocked the interfering signals such that they did not affect the good ones, which explains why the PRR was high. The good signals were mostly those coming from vehicles located in the same street (LOS, without walls in between) and above receiver sensitivity. On the other hand, when the

walls were disabled, the interfering signals directly affected the good ones, and thus, the PRR decreased.

5.2.1.5. Effect of CAM Frequency

Figure 42 shows the resulting PRRs that correspond to the different CAM generation periods in a 4-lane, fast speed highway scenario. It could be observed that the higher the period was, the higher was the PRR. Lower periods translated to higher transmission frequencies, meaning more CAMs were sent per second. The large volume of CAMs sent led to an increase in interference and number of collisions, subsequently decreasing the PRR.

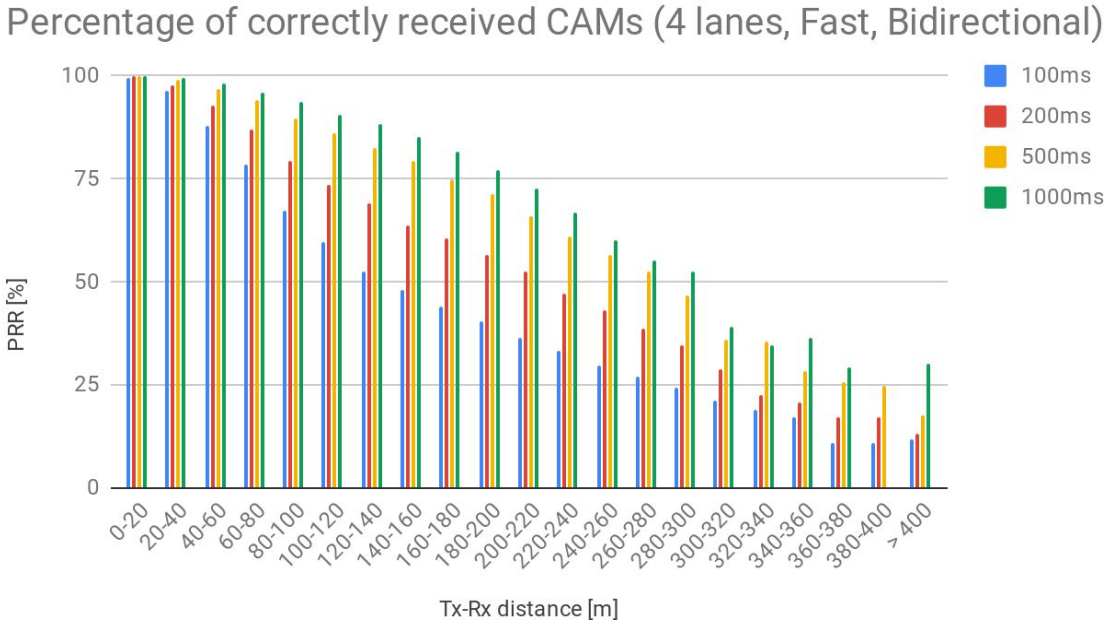


Figure 42. Effect of CAM frequency on PRR

5.2.2. Position Error

The position error or `deltaPosition` is the difference between the perceived position specified in the `lastKnownPos` database and the actual position of a neighboring vehicle in the scenario. As discussed in Section 4.2.3, `lastKnownPos` is a local database maintained by each vehicle, and acts like a local memory for storing various information about neighboring vehicles. Whenever a correct CAM is received, the vehicle updates the database entry corresponding to the transmitting vehicle.

`deltaPosition` measures how accurate the perceived positions (of neighboring vehicles) are every 100 ms, which is an important aspect to increase traffic safety through cooperative awareness among vehicles. For instance, the Rx node receives a CAM from the Tx node at time *t*, and registers the Tx node's position in its `lastKnownPos` database. Both vehicles then move, but at time *t'*, the Rx node has yet to receive a new CAM from the Tx node. For this reason, the Rx node still thinks that the Tx node is still in

the same position as when it last received a CAM from that node (perceived position). However, in reality, the Tx node had already moved to a new position (actual position). This difference between the perceived and actual positions is referred to as the position error. As such, it is ideal to have lower values of `deltaPosition`, since large ones indicate more erroneous readings on the perceived positions. Knowing where and how close surrounding vehicles are enables triggering timely warnings in cases where the distance between vehicles is less than a defined safe distance. This in turn gives enough time for vehicles to make the proper maneuver to avoid road casualties.

5.2.2.1. Effect of Vehicle Speed

Considering a fixed lane count and variable vehicle speed, as in the 4-lane highway scenario in Figure 43, it could be observed that the `deltaPosition` decreased as the vehicle speed decreased. Recall from Section 5.2.1.1 that the speed directly affected the vehicle density in the scenario. The slower the vehicles were, the denser was the scenario, since more vehicles could be packed in the 1 km highway scenario. The presence of more vehicles in the slow scenario, compared to the fast and moderate cases, translated to more interference. However, despite losing more packets, it could be observed that the position error was lowest for the slow speed scenario. This could be attributed to the fact that since the vehicles moved slower, their displacement in the scenario was small. Thus, even without correctly receiving a number of CAMs and updating the `lastKnownPos` database, the position error for the densest scenario still appeared to be low. On the contrary, when the vehicles moved fast, it follows that their displacement was large, such that missing even a single CAM led to high position error.

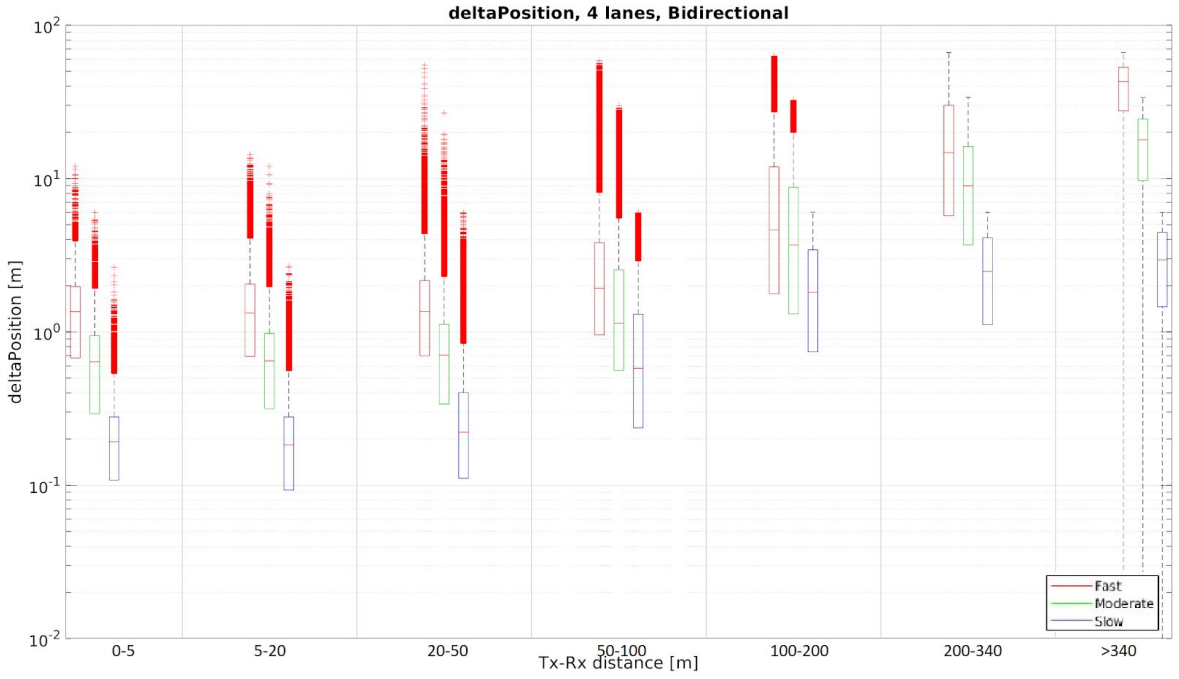


Figure 43. Effect of vehicle speed on position error

5.2.2.2. Effect of Number of Lanes

The position error is examined in the case of a moderate speed highway scenario while varying the lane count, depicted in Figure 44. As previously discussed in Section 5.2.1.2, increasing the number of lanes also increased the total number vehicles in the scenario, thereby creating higher interference and packet collisions. This explains why the `deltaPosition` of the 16-lane scenario was much greater than that of the 4-lane scenario. As there was greater interference in the case of the 16-lane scenario, more CAMs were getting lost and the corresponding `lastKnownPos` database entries were not being updated. The longer these entries were not refreshed, the higher was the resulting position error, since the neighboring vehicles continuously moved in the scenario causing greater disparity between the perceived and actual positions.

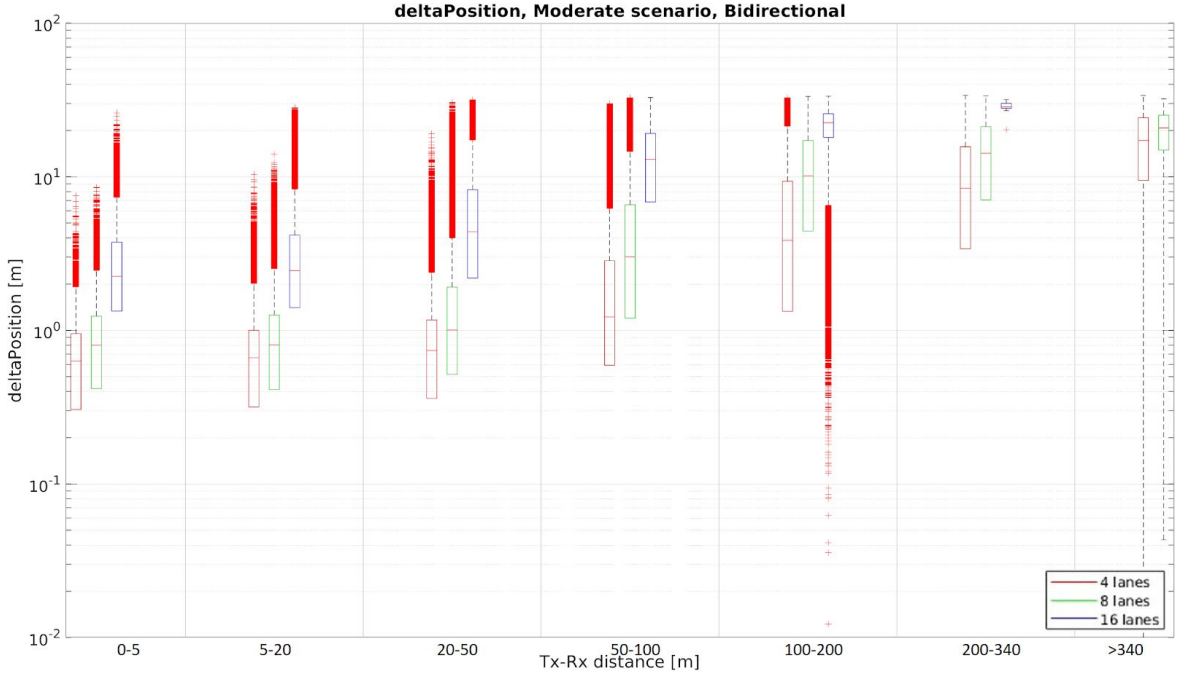


Figure 44. Effect of lane count on position error

5.2.2.3. Effect of CAM Frequency

Figure 45 illustrates the impact of the CAM transmit frequency on the position error. In the PRR analysis in Section 5.2.1.5, higher CAM transmit frequency (or lower transmit period) led to greater interference and more packets lost. However, since the CAMs were sent more frequently, the vehicles got updated more frequently as well. As such, despite the high risk of packet collisions due to high message traffic, the vehicles received enough CAMs to update their database entries, leading to lower position error. In this case, the positive impact of sending more CAMs for more frequent database updates was more dominant than the negative effect of packet collisions on the position error.

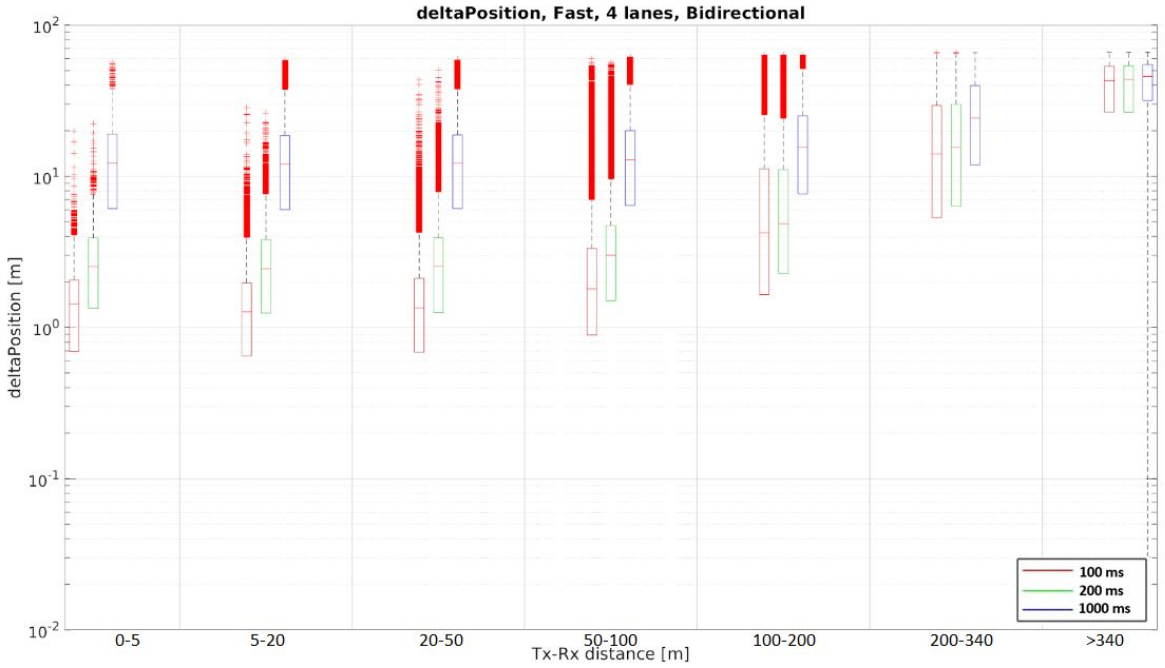


Figure 45. Effect of CAM frequency on position error

5.2.3. Distance Error

When the local database of a node is not updated (due to missed CAMs and/or due to a significant discrepancy between the time of update and the time of measuring the statistics), there would be a difference between the perceived and actual positions of known neighboring nodes, causing a position error. Instead of looking at the difference in terms of position, it is also possible to look at the difference in terms of distance. This difference is called the distance error or `deltaDistance`, as illustrated in Figure 46. It is ideal to have lower values for the `deltaDistance` since this would indicate that the vehicle is up-to-date on the true positions of its neighboring vehicles.

Figure 47 shows the `deltaDistance` for a 4-lane highway scenario, while varying the vehicle speed. As discussed in Section 5.2.1.1, slow scenarios create dense environments. The presence of more vehicles in the slow scenario translates to greater interference and collision probability, leading to more CAMs being lost. This resulted to higher distance errors compared to the fast and moderate cases. When comparing this to the `deltaPosition` in Figure 43, it could also be observed that in smaller Tx-Rx distances, the `deltaDistance` was lower than the `deltaPosition`. This behavior is also depicted in the left scenario of Figure 46. If we argue that errors in the shortest ranges are more important than errors in longer ranges, this would make `deltaDistance` a less reliable metric than position error, because `deltaDistance` gives the impression of having much smaller errors at this range.

Moreover, as the distance between the two nodes increased, the values of the `deltaDistance` in Figure 47 approaches the values of `deltaPosition` in Figure 43. This behavior is illustrated in the right scenario of Figure 46. As both `deltaPosition`

and `deltaDistance` resulted in more or less the same values in high Tx-Rx distances, it could be said that `deltaDistance` is redundant.

Therefore, it has been decided that the `deltaDistance` statistic would not be explored any further, as `deltaPosition` was decided to be the more insightful metric.

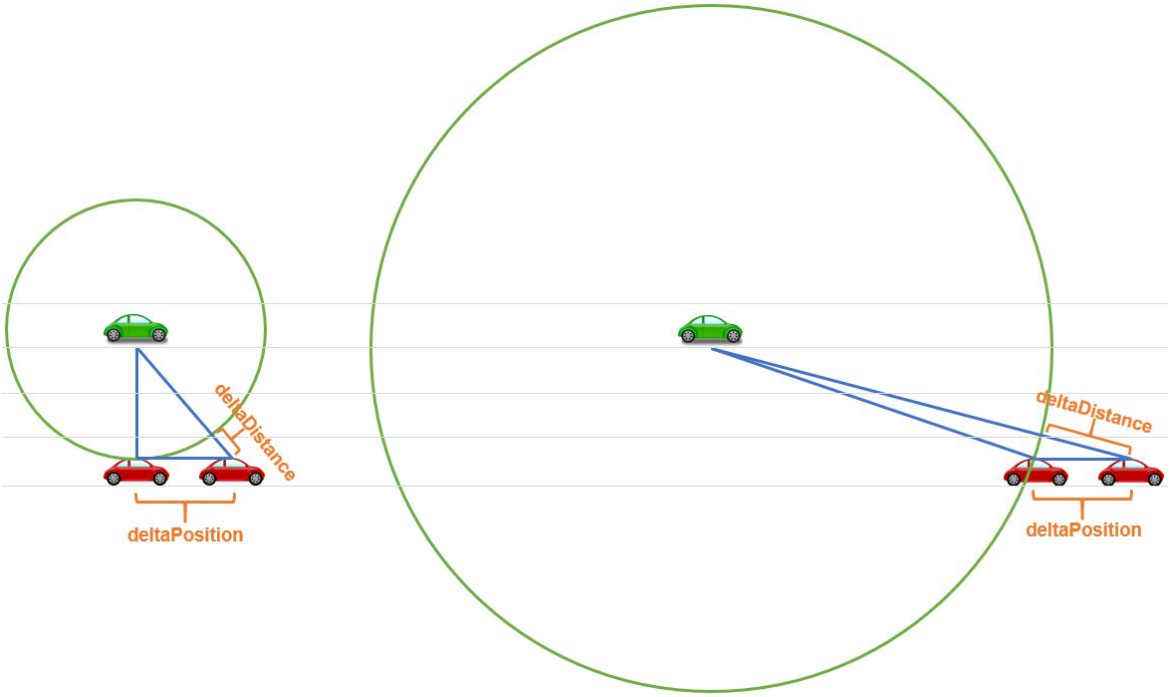


Figure 46. `deltaPosition` and `deltaDistance` metrics

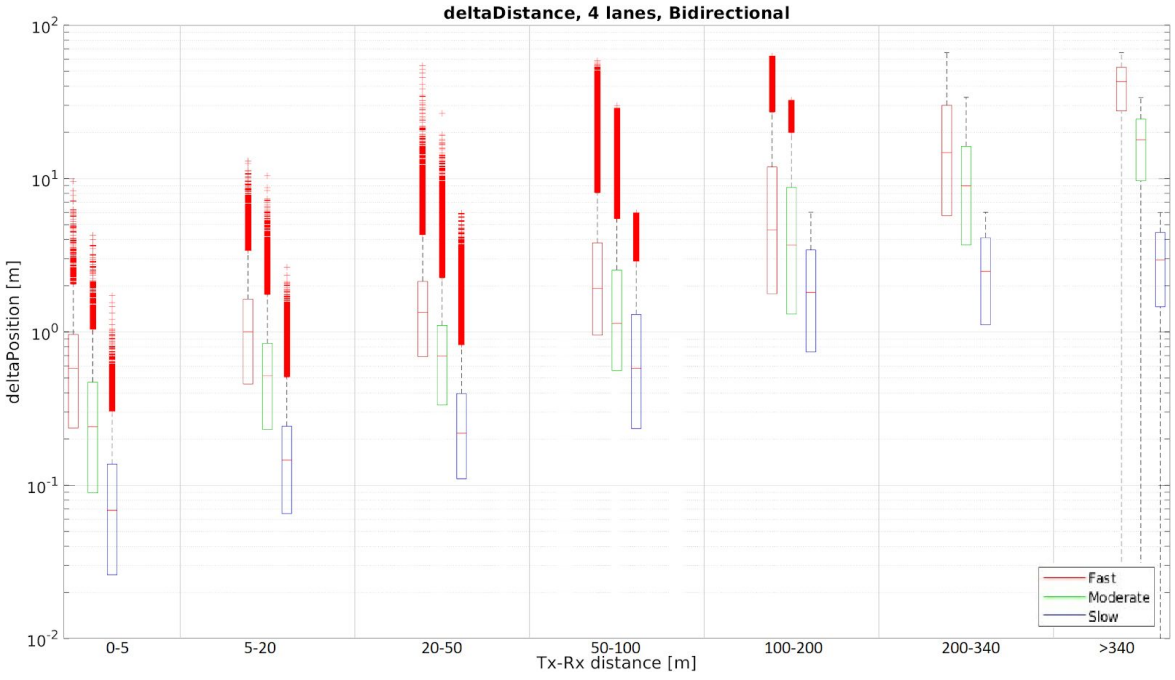


Figure 47. Distance error in a moderate speed highway scenario

5.2.4. Neighborhood Awareness Ratio

The Neighboring Awareness Ratio (NAR) is a measure of the cooperative awareness achieved by the vehicles in a scenario. This statistic is taken every 100 ms, and is computed as follows.

$$NAR = \frac{perceived\ number\ of\ vehicles}{actual\ number\ of\ vehicles} \times 100$$

The perceived number of vehicles corresponds to the number of entries in the `lastKnownPos` database, with each entry representing a distinct neighboring vehicle in the scenario. To ensure the validity of the database entries, recall that an expiry time was defined to be 2 seconds. Thus, after 2 seconds (20 missing CAMs) that an entry has not been updated, it will be automatically removed from the database. On the other hand, the actual number of vehicles is taken from the `GlobalMapper`, which maintains a global database that contains various information about all the vehicles in the scenario, including their actual position at any given time.

5.2.4.1. Effect of Vehicle Speed

The impact of vehicle speed on NAR is depicted in Figure 48, where the lane count was fixed to 8 for a highway scenario.

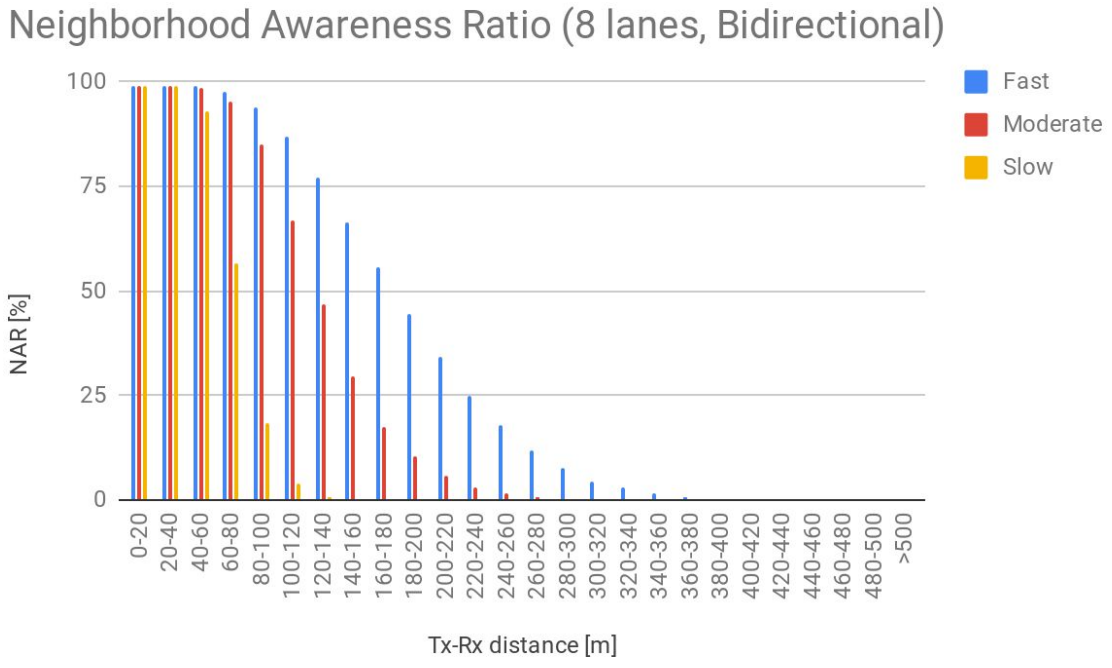


Figure 48. Effect of vehicle speed on NAR

As previously discussed, slow vehicle speed translates to denser environments, with more vehicles being packed in the highway scenario. Since NAR acknowledges the presence of vehicles based on whether they are registered in the `lastKnownPos` database, the inability to update this database due to CAMs being lost to interference and collisions, directly affects the calculation of NAR. This explains why the resulting NAR of the slow speed scenario was much lower than that of the sparser scenarios, as the high level of interference negatively impacted NAR.

5.2.4.2. Effect of Number of Lanes

Fixing the vehicle speed, Figure 49 illustrates the effect of the number of lanes on NAR.

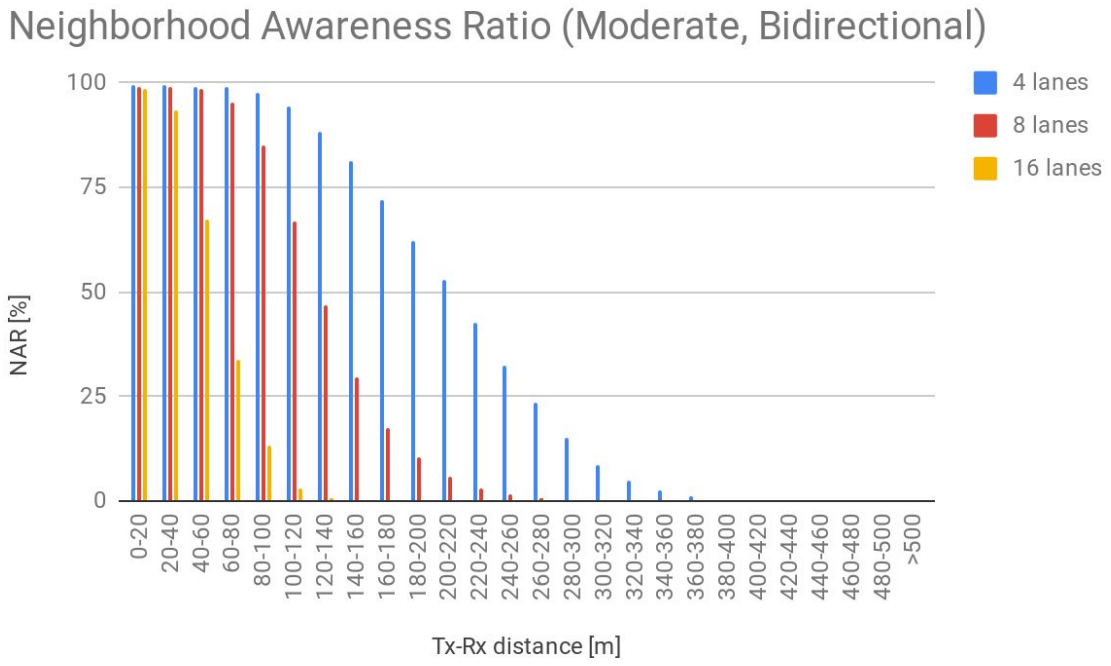


Figure 49. Effect of lane count on NAR

Increasing the number of lanes meant increasing the number of vehicles as well. The presence of more vehicles in the 16-lane scenario created higher interference and collision probability compared to the 4-lane and 8-lane scenarios. This in turn caused more CAMs to be lost, and after a certain period of not being able to update the local database, the corresponding entries were automatically removed, thereby impacting the NAR calculation.

5.2.4.1. Effect of CAM Frequency

Figure 50 shows the effect of CAM frequency on NAR in a 4-lane, fast speed highway scenario. It could be observed that decreasing the CAM transmit period resulted to higher NAR values. As with the observations made in the effect of CAM frequency on position error, it seemed that increasing the frequency translated to vehicles becoming more aware of their surroundings through more frequent CAM exchanges and database

updates. However, further increases in frequency caused the NAR values to start worsening, which is the same trend observed in [21]. That is, increasing the CAM transmit frequency improved cooperative awareness only up to a certain point, after which, further increases in frequency started to have a negative impact on NAR. This was because in situations with sufficiently large CAM traffic, the detrimental effects of interference become more dominant over the beneficial effects of having frequent updates. This contributed to the decrease in NAR.

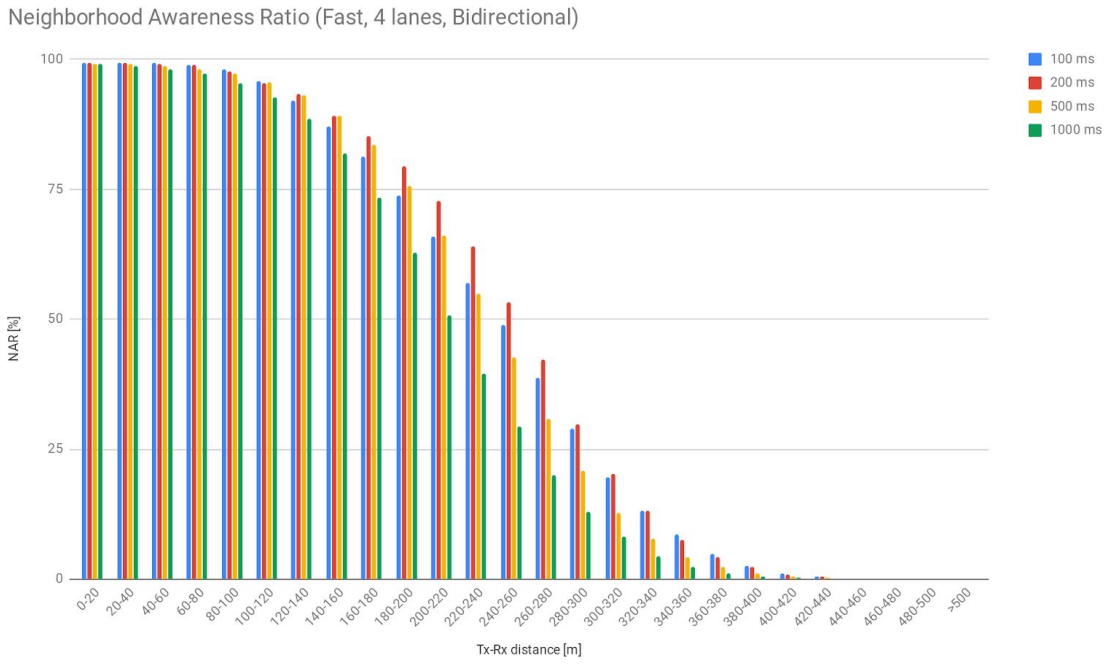


Figure 50. Effect of CAM frequency on NAR

6. Budget

The APU2 device and the WLE200NX wireless module for the OBU/RSU project were acquired from the i2CAT Foundation. As for the simulator project, a desktop PC was provided by UPC. All the software programs utilized were open-source and incurred no additional costs.

In terms of manpower, as the duration of the project was from September 2018 to mid-May 2019 (34 weeks), the total time spent in developing the project assuming an 8-hour workday is 1360 hours.

7. Conclusions and Future Development

A low-cost OBU/RSU that implemented the ETSI C-ITS protocol stack was successfully developed using an open-source software. In particular, the testbed implemented the CA basic service, so that it was capable of periodically transmitting and receiving CAMs. An important aspect of the task was determining which hardware to use such that it was configurable to operate in the 5.9 GHz band. This functionality was supported by a number of Atheros wireless cards that use the ATH9K Linux kernel driver and also required the PCI interface. While the initial plan was to use the Raspberry Pi 3 Model B+, it was later concluded that it was not suitable, since it did not support PCI interfaces. On the other hand, the combination of the APU2 hardware and the WLE200NX wireless module met all the requirements, and was therefore used to develop the OBU/RSU. In order to tune the testbed to work using the IEEE 802.11p channel, several modifications were made in the Linux kernel driver and in the user-space entities. These included configuring it in OCB operation mode and adding the ITS-G5 frequencies and power levels to the wireless regulatory database. The proper operation and interoperability of the developed OBU/RSU were verified by testing it with a commercial V2X device. Moreover, the contents of the CAM were analyzed to be compliant with the ETSI standards.

An existing IEEE 802.11p-based simulator was modified to include additional functionalities and simulate different scenarios. Parameters including the vehicle speed, number of lanes and CAM transmit frequency, were varied to evaluate the CA basic service using different performance metrics such as the PRR, position error and NAR.

Due to the usage of the Krauss car following model, changing the vehicle speeds affected the vehicle traffic density of the simulations. Slow-moving vehicles only required small braking distances, thus allowing more vehicles to fit into the scenario, and consequently creating dense environments. The presence of more vehicles resulted to higher interference and collision probability as more nodes attempt to access the channel to send their periodic CAMs, ultimately decreasing the PRR values. The computation of NAR relied on the local database maintained in each vehicle, which was only updated whenever a CAM is correctly received. Since more CAMs were lost in the slow speed scenarios, this also led to lower NAR values. An interesting result was that the densest cases had the lowest position errors. However, this was due to the fact that given the vehicles moved slowly, it followed that their displacement in the scenario was small. As such, even without receiving a number of CAMs, the position errors were low.

Increasing the number of lanes also increased the vehicle count in the scenario. As previously discussed, this created interference and collisions, which resulted to more packets being lost and the local database not being updated. This led to lower PRR and NAR values. Higher lane count yielded higher position error, since the error was computed taking into account the perceived position specified in the local database. As such, the longer the database was not updated, the larger was the position error.

Higher CAM frequency meant that the CAMs were transmitted more frequently. This created a higher collision probability, which consequently decreased the PRR. In the case of the position error, it was observed that higher transmit frequencies translated to lower errors. The positive impact of sending more CAMs and updating the database more

frequently was more prevalent than the negative effect of high risk of packet collisions. Increasing the transmit frequency caused an increase in the resulting NAR values up to a certain point, after which, further increases in frequency had a negative impact on the NAR. This was because the detrimental effects of interference overpower the beneficial effects of having frequent CAM updates when the message traffic is sufficiently high, thereby decreasing the NAR.

There are a number of recommendations for continuing with this research study. For one, the development of a low-cost, open-source OBU/RSU encourages carrying out future field trials without the need for expensive hardware. As the testing done in the project was confined to the laboratory, it would be interesting to see how the developed OBU/RSU performs in real-world settings. Moreover, while the project mainly tackles the CA basic service, it could also be extended to transmit and receive other types of V2X messages, such as DENM, SPAT, IVI, etc.

In the case of the system simulator, other road topologies could be tested, since the project only dealt with highway and grid scenarios. For instance, SUMO allows importing real-world maps, so that it is possible to study the performance of the CA basic service in more realistic environments.

Bibliography

[1] World Health Organization. Road traffic injuries [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries. [Accessed: May 12, 2019].

[2] i2CAT. [Online]. Available: https://www.i2cat.net/. [Accessed: May 12, 2019].

[3] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Communications Architecture" ETSI EN 302 665 V1.1.1 (2010-09).

[4] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 1: Functional Requirements" ETSI TS 102 637-1 V1.1.1 (2010-09).

[5] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 5: Transport Protocols; Sub-part 1: Basic Transport Protocol" ETSI EN 302 636-5-1 V1.2.1 (2014-08).

[6] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 1: Requirements" ETSI EN 302 636-1 V1.2.1 (2014-04).

[7] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1: Media-Independent Functionality" ETSI EN 302 636-4-1 V1.3.1 (2017-08).

[8] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Vehicular Communications; Geographical Area Definition" ETSI EN 302 931 V1.1.1 (2011-07).

[9] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transportation Systems operating in the 5 GHz frequency band" ETSI EN 302 663 V1.2.1 (2013-05).

[10] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 2: Media-dependent functionalities for ITS-G5" ETSI TS 102 636-4-2 V1.1.1 (2013-10).

[11] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Decentralized Congestion Control Mechanisms for Intelligent Transport Systems operating in the 5 GHz range; Access layer part" ETSI TS 102 687 V1.2.1 (2018-04).

[12] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 3: Network Architecture" ETSI EN 302 636-3 V1.2.1 (2014-12).

[13] European Telecommunications Standards Institute (ETSI). “Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service” ETSI EN 302 637-2 V1.3.1 (2014-09).

[14] European Telecommunications Standards Institute (ETSI). “Intelligent Transport Systems (ITS); Users and applications requirements; Part 2: Applications and facilities layer common data dictionary” ETSI TS 102 894-2 V1.3.1 (2018-08).

[15] European Telecommunications Standards Institute (ETSI). “Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service” ETSI EN 302 637-3 V1.2.1 (2014-09).

[16] R. Lisovy, M. Sojka, and Z. Hanzalek. “IEEE 802.11p Linux Kernel Implementation” [Online]. Available: https://rtime.felk.cvut.cz/publications/public/ieee80211p_linux_2014_final_report.pdf. [Accessed: May 12, 2019].

[17] J. F. Pastrana. “802.11p standard and V2X applications on commercial Wi-Fi cards” [Online]. Available: https://github.com/jfpastrana/802.11p/blob/master/Documentation/802.11p_standard_and_V2X_applications.pdf. [Accessed: May 12, 2019].

[18] I. Mavromatis, A. Tassi, R. Piechocki, and A. Nix. A City-Scale ITS-G5 Network for Next-Generation Intelligent Transportation Systems: Design Insights and Challenges. (2018).

[19] N. Vivek, P. Sowjanya, B. Sunny and S. V. Srikanth, "Implementation of IEEE 1609 WAVE/DSRC stack in Linux," *2017 IEEE Region 10 Symposium (TENSYMP)*, Cochin, 2017, pp. 1-5.

[20] A. Abunei, C. Comşa and I. Bogdan, "Implementation of a Cost-effective V2X hardware and software platform," *2016 International Conference on Communications (COMM)*, Bucharest, 2016, pp. 367-370.

[21] M. Boban and P. M. d'Orey, "Exploring the Practical Limits of Cooperative Awareness in Vehicular Communications," in *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3904-3916, June 2016.

[22] J. Santa, F. Pereñíguez, A. Moragón, and A. Skarmeta, “Experimental evaluation of CAM and DENM messaging services in vehicular communications”, Transportation Research Part C: Emerging Technologies, Volume 46, 2014, Pages 98-120.

[23] Vanetza. [Online]. Available: https://github.com/riebl/vanetza. [Accessed: May 12, 2019].

[24] Raspberry Pi. [Online]. Available: https://www.raspberrypi.org/. [Accessed: May 12, 2019].

[25] Raspberry Pi Configuration. [Online]. Available: https://docs.emlid.com/navio2/common/ardupilot/configuring-raspberry-pi/. [Accessed: May 12, 2019].

[26] Atheros 802.11n PCI/PCI-E devices. [Online]. Available: https://wiki.debian.org/ath9k. [Accessed: May 12, 2019].

[27] APU2 Platform. [Online]. Available: https://www.pcengines.ch/apu2.htm. [Accessed: May 12, 2019].

[28] Dual Band 2x2 MIMO 802.11n Mini PCIe Module. [Online]. Available: https://compex. com.sg/wp-content/uploads/2018/10/wle200nx-v1.3-wh-192.pdf. [Accessed: May 12, 2019].

[29] Linux Wireless. [Online]. Available: https://wireless.wiki.kernel.org/en/users. [Accessed: May 12, 2019].

[30] The Linux Kernel Archives. [Online]. Available: https://www.kernel.org/. [Accessed: May 12, 2019].

[31] OMNeT++. [Online]. Available: https://omnetpp.org/. [Accessed: May 12, 2019].

[32] Veins. [Online]. Available: https://veins.car2x.org/. [Accessed: May 12, 2019].

[33] SUMO. [Online]. Available: https://sumo.dlr.de/. [Accessed: May 12, 2019].

[34] Artery. [Online]. Available: https://github.com/riebl/artery. [Accessed: May 12, 2019].

[35] SUMO - Wiki. [Online]. Available: https://sumo.dlr.de/wiki/. [Accessed: May 12, 2019].

[36] Cohda Wireless. [Online]. Available: https://cohdawireless.com/. [Accessed: May 12, 2019]

[37] J. Wallen. “How to Compile a Linux Kernel”. [Online]. Available: https://www.linux. com/learn/intro-to-linux/2018/4/how-compile-linux-kernel-0. [Accessed: May 12, 2019].

[38] Linux IEEE 802.11p How To. [Online]. Available: https://ctu-iig.github.io/802.11p-linux/. [Accessed: May 12, 2019].

Appendices

A. OBU/RSU Configuration Guide

A.1. CAM Receiver Application Quick Start Guide

This quick start guide describes how to set up and run the CAM receiver application. Prior to performing the steps below, it is necessary to obtain a copy of the CAM receiver application from the i2CAT Foundation.

1. Install the Vanetza requirements.

   Python 3, pip3 and CMake from pip (the last step could require several minutes or even hours):

   ```
   $ sudo apt install python3 python3-pip
   $ sudo pip3 install --upgrade pip
   $ sudo pip install scikit-build
   $ sudo python3 -m pip install cmake
   ```

   Boost, GeographicLib, Crypto++, git:

   ```
   $ sudo apt install g++ libcrypto++-dev libgeographic-dev libboost-date-time-dev
   libboost-program-options-dev libboost-serialization-dev libboost-system-dev git -y
   ```

2. Build Vanetza (last step could require several minutes).

   ```
   $ git clone https://github.com/riebl/vanetza
   $ cd vanetza/
   $ mkdir build && cd build
   $ cmake ..
   $ make
   ```

3. Install the `socktap` example application dependencies.

   ```
   $ sudo apt install libgps-dev gpsd-clients python-gps -y
   ```

4. Build `socktap`.

   ```
   $ cmake -D BUILD_SOCKTAP=ON ..
   $ make
   ```

5. Build the `cam-rcv` application and proper permissions to binary.

   ```
   $ cd <project_folder>
   $ mkdir build && cd build
   $ cmake ..
   $ make
   $ sudo setcap cap_net_raw+ep bin/cam-rcv
   ```

6. Run the `cam-rcv` application with the desired network interface.

```
$ bin/cam-rcv -i <network_interface> --gpsd-host localhost
```

Example

Using the loopback interface:

```
$ bin/cam-rcv -i lo --gpsd-host localhost
```

Using the OCB interface:

```
$ bin/cam-rcv -i ocb0 --gpsd-host localhost
```

Note that the `cam-rcv` application requires the GPS functionality. One option is to emulate a GPS signal as described in Appendix A.2. In this case, the recorded GPS file is played in the background (in a different terminal), while running the `cam-rcv` application.

A.2. Emulation of GPS Signal

In the case that a real GPS signal is not available, an alternative way is to emulate it using the `gpspipe` and `gpsfake` tools.

In order to record a real GPS session, follow these steps:

1. Ensure that the `gpsd` daemons are running.

```
$ systemctl is-active gpsd ; systemctl is-active gpsd.socket
```

Both should be in `active` status. If not, try to start them.

```
$ sudo systemctl restart gpsd && sudo systemctl restart gpsd.socket
```

2. Start recording.

```
$ gpspipe -r | tee <gps_recorded_file>.gps
```

3. Stop recording by pressing `Ctrl + C`.

4. Save the newly created GPS file.

In order to reproduce any session, follow these steps:

1. Ensure that the `gpsd` daemons are NOT running.

```
$ systemctl is-active gpsd ; systemctl is-active gpsd.socket
```

Both should be in `inactive` status. If not, try to stop them.

```
$ sudo systemctl stop gpsd && sudo systemctl stop gpsd.socket
```

2. Start playing the recorded file.

```
$ gpsfake -c 0.5 <gps_recorded_file>.gps
```

You could check the output of the recorded file with the `gpsmon` or `xgps` tools (in a different terminal).

3. Stop playing by pressing `Ctrl + C`.

A.3. Setting up an Ad Hoc Network

An ad hoc network is necessary to enable wireless communication between devices, including the Raspberry Pis. The following steps are required to successfully configure such network.

1. For both Raspberry Pis, go to `/etc/network/` and save a copy of the `interfaces` file.

```
$ sudo cp /etc/network/interfaces /etc/network/interfaces-orig
```

2. Modify the `interfaces` file with reference to the following lines. Note that the only difference for the two devices is their assigned IP address.

```
$ sudo nano /etc/network/interfaces
```

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

iface eth0 inet dhcp

auto wlan0
iface wlan0 inet static
address 192.168.1.1
netmask 255.255.255.0
wireless-channel 1
wireless-essid PiAdhoc
wireless-mode ad-hoc
```

Figure 51. `interfaces` of Raspberry Pi #1

```
source-directory /etc/network/interfaces.d
```

```
auto lo
iface lo inet loopback

iface eth0 inet dhcp

auto wlan0
iface wlan0 inet static
address 192.168.1.2
netmask 255.255.255.0
wireless-channel 1
wireless-essid PiAdhoc
wireless-mode ad-hoc
```

Figure 52. `interfaces` of Raspberry Pi #2

3. Stop or disable the `dhcp` service in both devices.

   ```
   $ sudo systemctl stop dhcpcd.service
   ```

4. Reboot both devices. Ping each other to verify the ad hoc network configuration.

A.4. Setting up the Linux Kernel

The following procedure details how the Linux kernel is set up and configured [37].

1. Download the Linux kernel from [30]. This project uses kernel version 4.20.7.

2. Install the software dependencies.

   ```
   $ sudo apt-get install git fakeroot build-essential ncurses-dev xz-utils libssl-dev
   bc flex libelf-dev bison
   ```

3. Extract the downloaded Linux kernel source file.

   ```
   $ tar xf linux-4.20.7.tar.xz
   ```

4. Modify the ATH9K driver source codes, as described in Appendix A.5.

5. Configure which kernel modules to include.
   a. Copy the configuration file of the currently running kernel.

      ```
      $ cp /boot/config-$(uname -r) .config
      ```

   b. By running the following command, the kernel configuration menu will be launched, where kernel modules can be enabled/disabled. The configuration used in the project is specified in Appendix A.6.

      ```
      $ make menuconfig
      ```

6. Compile the kernel.

```
$ make
```

7. Install the kernel modules previously enabled.

```
$ make modules_install
```

8. Install the kernel.

```
$ sudo make install
```

9. Enable the kernel for boot, and then restart the system.

```
$ sudo update-initramfs -c -k 4.20.7
$ sudo update-grub
```

A.5. Modifications on the ATH9K Driver Source Codes

It is necessary to modify some of the ATH9K driver source codes to fully implement IEEE 802.11p on a Linux system [17]. The changes are written in red.

```
if (is_scanning ||
(ah->opmode != NL80211_IFTYPE_STATION &&
       ah->opmode != NL80211_IFTYPE_OCB &&
       ah->opmode != NL80211_IFTYPE_ADHOC)) {
/*

* If we're scanning or in AP mode, the defaults (ini)
```

Figure 53. drivers/net/wireless/ath/ath9k/ani.c

```
static const struct ieee80211_channel ath9k_5ghz_chantable[] = {
       ...
       CHAN5G(5850, 38), /* Channel 170 */
       /* ITS-G5B */
       CHAN5G(5855, 39), /* Channel 171 */
       CHAN5G(5860, 40), /* Channel 172 */
       CHAN5G(5865, 41), /* Channel 173 */
       CHAN5G(5870, 42), /* Channel 174 */
       /* ITS-G5A */
       CHAN5G(5875, 43), /* Channel 175 */
       CHAN5G(5880, 44), /* Channel 176 */
       CHAN5G(5885, 45), /* Channel 177 */
   CHAN5G(5890, 46), /* Channel 178 */
       CHAN5G(5895, 47), /* Channel 179 */
       CHAN5G(5900, 48), /* Channel 180 */
       CHAN5G(5905, 49), /* Channel 181 */
       /* ITS-G5D */
       CHAN5G(5910, 50), /* Channel 182 */
       CHAN5G(5915, 51), /* Channel 183 */
       CHAN5G(5920, 52), /* Channel 184 */
       CHAN5G(5925, 53), /* Channel 185 */
};
```

Figure 54. drivers/net/wireless/ath/ath9k/common-init.c

```
#define ATH9K_RSSI_BAD          -128

#define ATH9K_NUM_CHANNELS      54

/* Register read/write primitives */
#define REG_WRITE(_ah, _reg, _val) \
```

Figure 55. drivers/net/wireless/ath/ath9k/hw.h

```
   ath9k_hw_setopmode(ah);

       ctx->switch_after_beacon = false;
       if ((iter_data.nstations + iter_data.nadhocs + iter_data.nmeshes + iter_data.nocbs)
> 0)
               ah->imask |= ATH9K_INT_TSFOOR;
       else {
               ah->imask &= ~ATH9K_INT_TSFOOR;
               if (iter_data.naps == 1 && iter_data.beacons)
                       ctx->switch_after_beacon = true;
```

Figure 56. drivers/net/wireless/ath/ath9k/main.c

```
/* We allow IBSS on these on a case by case basis by regulatory domain */
#define ATH9K_5GHZ_5150_5350    REG_RULE(5150-10, 5350+10, 80, 0, 30,\
                                        NL80211_RRF_NO_IR)
#define ATH9K_5GHZ_5470_5925    REG_RULE(5470-10, 5925+10, 80, 0, 30,\
                                        NL80211_RRF_NO_IR)
#define ATH9K_5GHZ_5725_5925    REG_RULE(5725-10, 5925+10, 80, 0, 30,\
                                        NL80211_RRF_NO_IR)

#define ATH9K_2GHZ_ALL          ATH9K_2GHZ_CH01_11, \
                                ATH9K_2GHZ_CH12_13, \
                                ATH9K_2GHZ_CH14

#define ATH9K_5GHZ_ALL          ATH9K_5GHZ_5150_5350, \
                                ATH9K_5GHZ_5470_5925

/* This one skips what we call "mid band" */
#define ATH9K_5GHZ_NO_MIDBAND   ATH9K_5GHZ_5150_5350, \
                                ATH9K_5GHZ_5725_5925

/* Can be used for:
* 0x60, 0x61, 0x62 */

//--------------------------------------------------------------------

     if (reg->country_code == CTRY_DEFAULT &&
           regdmn == CTRY_DEFAULT) {
              printk(KERN_DEBUG "ath: EEPROM indicates default "
                      "country code should be used\n");
              reg->country_code = CTRY_SPAIN;
       }
```

Figure 57. drivers/net/wireless/ath/regd.c

A.6. Configuring the Kernel Configuration Menu

Different kernel modules can be enabled/disabled in the kernel configuration menu. It is important to enable `Verbose OCB debugging` [17].

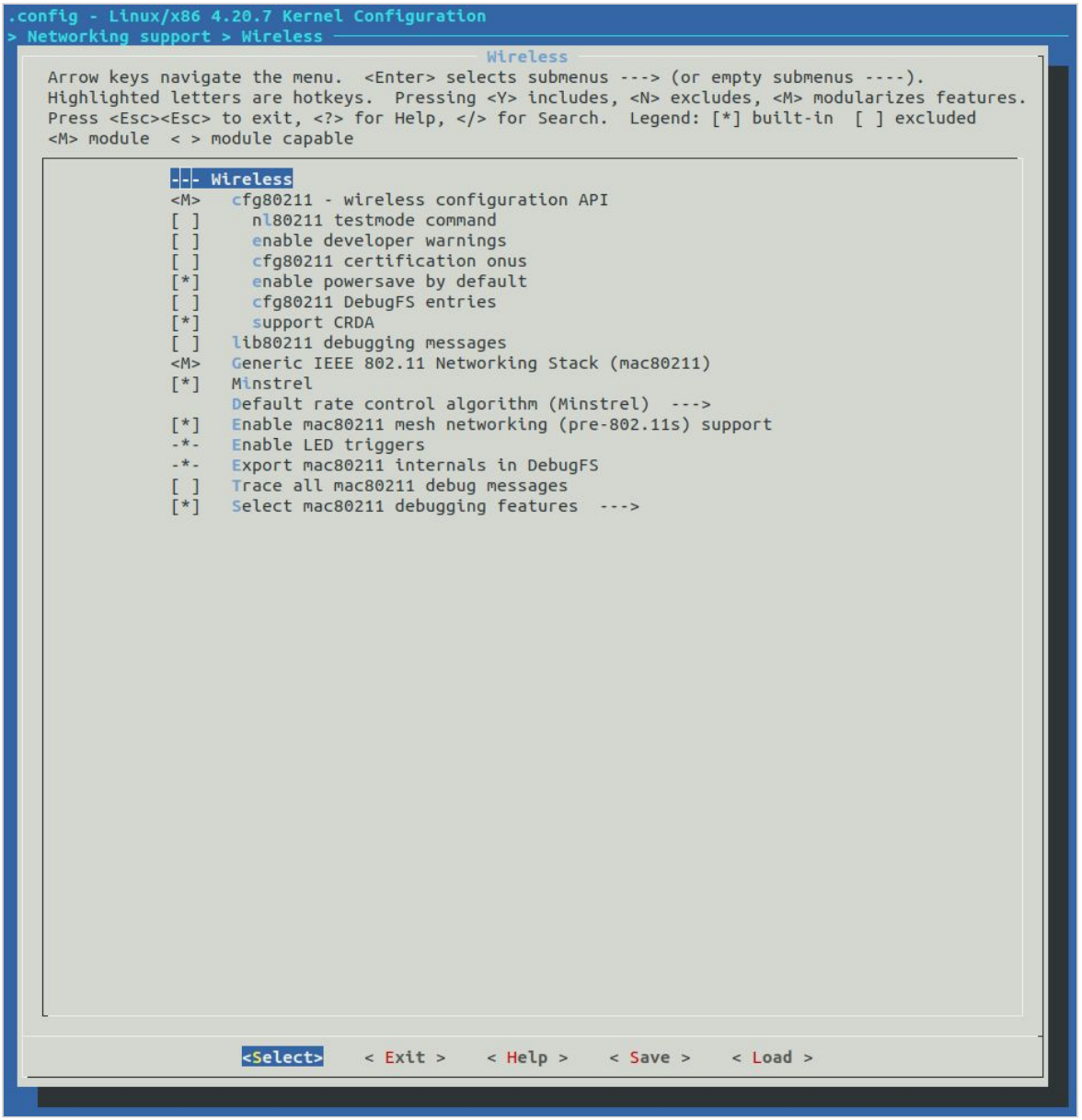
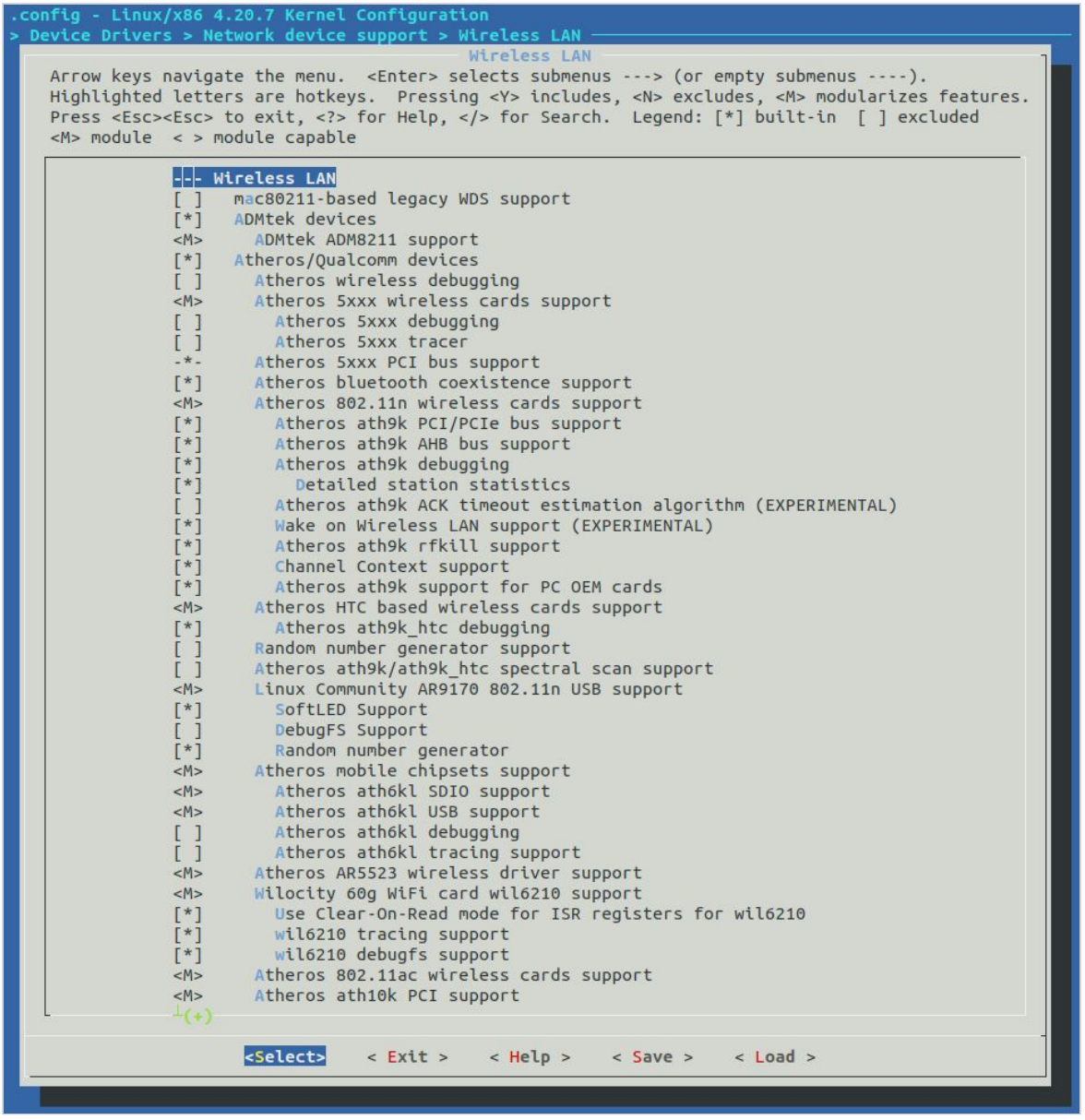


Figure 58. `Networking support > Wireless`



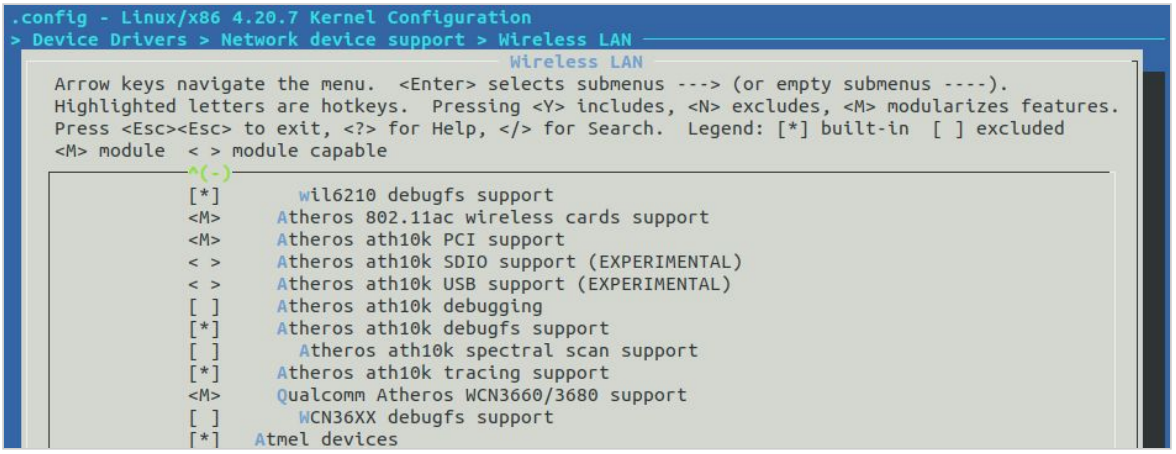


Figure 59. `Device Drivers > Network device support > Wireless LAN`

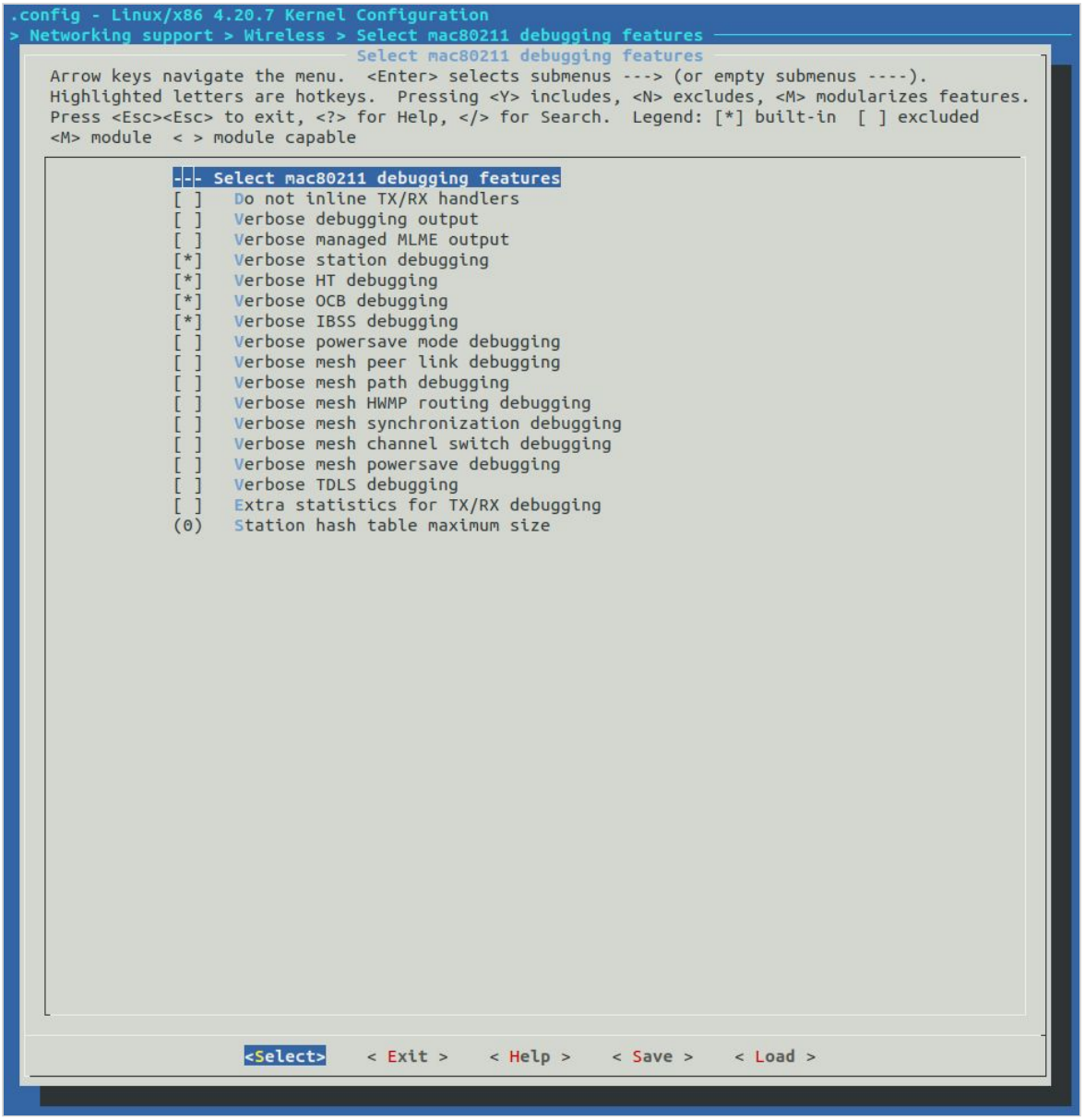


Figure 60. `Networking support > Wireless > Select mac80211 debugging features`

A.7. Setting up the `iw`

The following procedure details how the `iw` is set up and verified to support the OCB mode [38].

1. Install the software dependencies.

```
$ sudo apt-get install pkg-config libnl-genl-3-dev
```

2. Clone the `iw` repository.

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/jberg/iw.git
```

3. Build `iw`.

```
$ cd iw
$ make
```

4. Install `iw`.

```
$ sudo PREFIX=/ make install
```

5. Check if the `iw` version used supports the OCB mode.

```
$ /sbin/iw | grep -i ocb
```

Expected output:

```
dev <devname> ocb leave
dev <devname> ocb join <freq in MHz> <5MHZ|10MHZ>[fixed-freq]
```

A.8. Setting up the `wireless-regdb`

The following procedure details how the `wireless-regdb` is set up and configured to support the ITS-G5 frequencies [38].

1. Install the software dependencies.

```
$ sudo apt-get install python-m2crypto
```

2. Clone the `wireless-regdb` repository.

```
$ git clone --branch its-g5_v1
https://github.com/CTU-IIG/802.11p-wireless-regdb.git
```

3. Modify `db.txt` accordingly. In the RSU project, the ITS-G5 channels and transmission power values are added under `country ES` in `db.txt`.

```
country ES: DFS-ETSI
        (2400 - 2483.5 @ 40), (100 mW)
        (5150 - 5250 @ 80), (100 mW), NO-OUTDOOR
        (5250 - 5350 @ 80), (100 mW), NO-OUTDOOR
        (5470 - 5725 @ 80), (500 mW), DFS
        # For ITS-G5 evaluation
        (5840 - 5935 @ 10), (30)
        # 60 gHz band channels 1-4, ref: Etsi En 302 567
        (57240 - 65880 @ 2160), (40), NO-OUTDOOR
```

Figure 61. `802.11p-wireless-regdb/db.txt`

4. Build `wireless-regdb`.

```
$ cd 802.11p-wireless-regdb
$ make
```

5. Install `wireless-regdb`.

```
$ sudo make install PREFIX=/
```

6. Reboot the system for the changes in `db.txt` to take effect.

A.9. Setting up the `CRDA`

The following procedure details how the `CRDA` is set up to implement the wireless regulatory domain [38].

1. Install the software dependencies.

```
$ sudo apt-get install python-m2crypto libgcrypt11-dev
```

2. Clone the `CRDA` repository.

```
$ git clone --branch its-g5_v1 https://github.com/CTU-IIG/802.11p-crda.git
```

3. Copy the public keys installed by `wireless-regdb`.

```
$ cd 802.11p-crda
$ cp /lib/crda/pubkeys/$USER.key.pub.pem pubkeys/
```

4. Build `CRDA`.

```
$ make REG_BIN=/lib/crda/regulatory.bin
```

5. Install `CRDA`.

```
$ sudo make install PREFIX=/ REG_BIN=/lib/crda/regulatory.bin
```

6. Test `CRDA` and the generated `regulatory.bin`.

```
$ sudo /sbin/regdbdump /lib/crda/regulatory.bin | grep -i ocb
```

Expected output:

```
country 00: invalid
(5850.000 - 5925.000 @ 20.000), (20.00), NO-CCK, OCB-ONLY
```

A.10. Setting up the OBU Interface and IEEE 802.11p Channel

The following procedure details how to configure a wireless interface for OCB mode and join an IEEE 802.11p channel [17].

1. Bring down the wireless interface (for example, `wlan0`).

```
$ sudo ip link set wlan0 down
```

2. Add an OCB interface (`ocb0`).

```
$ sudo iw dev wlan0 interface add ocb0 type ocb
```

3. Set the OCB mode.

```
$ sudo iw dev ocb0 set type ocb
```

4. Bring down the OCB interface.

```
$ sudo ip link set ocb0 down
```

5. Set the wireless regulatory domain to `ES`, and verify afterwards.

```
$ sudo iw reg set ES
$ sudo iw reg get
```

6. Bring up the OCB interface.

```
$ sudo ip link set ocb0 up
```

7. Join an IEEE 802.11p channel. In this case, this is the Control Channel (CCH), with center frequency 5900 MHz, channel number 180 and channel spacing 10 MHz.

```
$ sudo iw dev ocb0 ocb join 5900 10MHZ
```

8. Verify the setup. The expected output is explained in Section 3.5.6.

```
$ sudo iw dev | iwconfig
```

B. IEEE 802.11p Simulator Installation Guide and User Manual

B.1. Installation Procedure

The IEEE 802.11p simulator utilizes several simulation frameworks of different functionality. This section details the framework versions and installation procedure to create the simulator environment. The Linux distribution used is Ubuntu 16.04 (64-bit).

1. Prepare the software packages.

   Extract the `Simulador.zip` file. Make a copy of the `v2x-arch_11p` folder and place it in the home directory. This folder will contain all the files related to the simulator. Note that for the following installation steps, OMNeT++, SUMO and Artery must be at the same folder hierarchy level inside `v2x-arch_11p`.

2. Install OMNeT++.
   a. Download the OMNeT++ installer (`omnetpp-5.1.1-src-linux.tgz`) from the OMNeT++ download website [31]. Save it in the `v2x-arch_11p` folder.

   b. Extract and proceed with the installation by entering the following commands.

   ```
   v2x-arch_11p$ tar xvfz omnetpp-5.1.1-src-linux.tgz
   v2x-arch_11p$ cd omnetpp-5.1.1
   v2x-arch_11p/omnetpp-5.1.1$ ./configure
   v2x-arch_11p/omnetpp-5.1.1$ make
   ```

3. Install SUMO.
   a. Download the SUMO installer (`sumo-src-0.29.0.tar.gz`) from the SUMO download website [33]. Save it in the `v2x-arch_11p` folder.

   b. Extract and proceed with the installation by entering the following commands.

   ```
   v2x-arch_11p$ tar xvfz sumo-src-0.29.0.tar.gz
   v2x-arch_11p$ cd sumo-0.29.0
   v2x-arch_11p/sumo-0.29.0$ ./configure
   v2x-arch_11p/sumo-0.29.0$ make
   ```

   c. Enter the following command.

   ```
   v2x-arch_11p$ . setenv
   ```

   d. Update the following line in `/home/wng/.profile`, then restart the terminal for the change to take effect.

   ```
   PATH="$HOME/bin:$HOME/.local/bin:/home/wng/v2x-arch_11p/omnetpp-5.1.1/bin:$H
   OME/.local/bin:/home/wng/v2x-arch_11p/sumo-0.29.0/bin:$PATH"
   ```

4. Install Vanetza, Veins and INET.

   The installers of the external project dependencies (Vanetza, Veins and INET) are already included in the `Simulador.zip` file. These dependencies can be built all at once by entering the following command.

   ```
   v2x-arch_11p/artery$ make all
   ```

5. Install Artery.
   a. The Artery installer is also included in the `Simulador.zip` file. Enter the following commands to proceed with the installation.

      ```
      v2x-arch_11p/artery$ mkdir build
      v2x-arch_11p/artery$ cd build
      v2x-arch_11p/artery/build$ cmake ..
      v2x-arch_11p/artery/build$ cmake --build .
      ```

   b. Update the following line in `run_ARTERY_cmdenv_from_sublime.cmd` and `run_ARTERY_guienv_from_sublime.cmd`, which are both located in the `v2x-arch_11p/artery/scenarios/artery/` directory.

      ```
      MY_NED_PATH="$HOME"
      ```

B.2. `omnetpp.ini` Configuration File

The main configuration file for the IEEE 802.11p-based simulator is the `omnetpp.ini` in `/home/wng/v2x-arch_11p/artery/scenarios/artery`.

The `omnetpp.ini` includes different parameters that may be configured prior to executing the scenario. More importantly, it allows the user to modify the parameters easily without having to go through the source code. Being able to change and experiment on the parameters helps in understanding their impact on the simulation results. The contents of `omnetpp.ini` are divided into sections as described below.

B.2.1. Simulation General Settings

The general settings for the simulation is shown in Figure 62. Both the `num-rngs` and `seed-0-mt` are used for randomization. The succeeding parameters are helpful when debugging, however, it is advisable to set them to `false` to make the simulation faster. `debug-on-errors` creates a breakpoint when an error occurs during runtime to determine the possible location and cause of the problem. `record-eventlog` creates a event log file, which can be analyzed using the sequence chart tool to illustrate how a message is routed between the nodes in the network.

```
##########################################################################
# simulation general settings
##########################################################################

network                 = artery.inet.World
num-rngs                = 10
seed-0-mt               = 532569 # 32-bit
debug-on-errors         = false
print-undisposed        = false
**.annotations.draw     = false
**.debug                = false
**.coreDebug            = false
record-eventlog         = false
```

Figure 62. Simulation general settings

B.2.2. Run Environment Settings

The run environment settings in Figure 63 include parameters specific to simulating either in command line mode (cmdenv) or graphical mode (qtenv). Enabling `cmdenv-express-mode` ensures minimal status updates on the console, while `cmdenv-status-frequency` configures the frequency of writing the status to the console.

```
##########################################################################
# run environment settings
##########################################################################

# if build.system       = ARTERY_GUIenv, file: 'run_ARTERY_guienv_from_sublime.cmd'
# if build.system       = ARTERY_cmdenv, file: 'run_ARTERY_cmdenv_from_sublime.cmd'
**.cmdenv-log-level     = off
cmdenv-express-mode     = true
cmdenv-autoflush        = true
cmdenv-status-frequency = 25s
```

Figure 63. Run environment settings

B.2.3. Statistics Settings

The statistics settings in Figure 64 specify parameters that define how the statistics are collected. Depending on the configuration, scalar and/or vector statistics are recorded during the simulation. The scalar statistics are simple measurement values that are recorded in a scalar result file (`*.sca`). On the other hand, the vector statistics are values recorded with time and are saved in a vector result file (`*.vec`). However, as vector statistics take up a lot of disk space, it is advisable to disable it unless needed. Both the scalar and vector result files are used as inputs to an analysis file (`*.anf`). Moreover, the filename of the result file can be configured.

```
##########################################################################
# statistics settings
##########################################################################

*.SystemMonitor.outputFileName  = "${resultdir}/${experiment}/${measurement}-${repetition}"
*.SystemMonitor.warmupTime      = ${warmupTime}

output-scalar-file              = ${resultdir}/${experiment}/${measurement}-${repetition}.sca
output-vector-file              = ${resultdir}/${experiment}/${measurement}-${repetition}.vec
**.param-record-as-scalar       = false
**.scalar-recording             = true
**.vector-recording             = false
```

Figure 64. Statistics settings

B.2.4. SUMO Settings

The SUMO settings in Figure 65 ensures that the simulation continues even if no vehicle is present in the scenario.

```
##########################################################################
# SUMO settings
##########################################################################

*.traci.core.version            = -1
*.traci.launcher.typename       = "PosixLauncher"
*.traci.mapper.rng-0            = 1
*.traci.core.selfStopping       = false
```

Figure 65. SUMO settings

B.2.5. Nodes Settings

The Nodes settings in Figure 66 includes the configurations for the physical, MAC and application layers of each vehicle (or node).

In the physical layer settings, the IEEE 802.11p parameters such as carrier frequency, bandwidth and bitrate are specified. In addition, this is where transmit power, receiver sensitivity, energy detection and SNIR threshold are configured, which ultimately affect the communication range. In the application layer, the update interval and jitter parameters are set to introduce randomness and reduce the possibility of collisions, such as in the case when the nodes may have been synchronized. The boundaries of the statistical region, or the area in the scenario where statistics are recorded, could be defined.

```
##########################################################################
# NODES settings, NIC, MAC, PHY, APP
##########################################################################

# energyDetection (eD) --> for CCA, default value in standard <-82,-62> dBm
# if radio's sensitivity (sensi) is better than eD --> eD = sensi
# for Cohda MK5 radios --> CohdaMobility MK5 Module Datasheet
# antenna.gain as default because huge manufacturer variability

*.node[*].wlan[*].typename                              = "VanetNic"
*.node[*].wlan[*].radioType                             = "Ieee80211ScalarRadio"
*.node[*].wlan[*].macType                               = "Ieee80211Mac"
*.node[*].wlan[*].mgmtType                              = "Ieee80211MgmtAdhoc"
*.node[*].wlan[*].opMode                                = "p"
*.node[*].wlan[*].bitrate                               = 6 Mbps

*.node[*].wlan[*].radio.displayCommunicationRange       = true
*.node[*].wlan[*].radio.displayInterferenceRange        = true
*.node[*].wlan[*].radio.receiver.ignoreInterference     = false
*.node[*].wlan[*].radio.bandName                        = "5 GHz"
*.node[*].wlan[*].radio.bandwidth                       = 10 MHz
*.node[*].wlan[*].radio.channelNumber                   = 180
*.node[*].wlan[*].radio.carrierFrequency                = 5.9 GHz
*.node[*].wlan[*].radio.transmitter.power               = 200 mW
*.node[*].wlan[*].radio.transmitter.bitrate             = 6 Mbps
*.node[*].wlan[*].radio.receiver.sensitivity            = -95 dBm
*.node[*].wlan[*].radio.receiver.energyDetection        = -95 dBm
*.node[*].wlan[*].radio.receiver.snirThreshold          = 16 dB

*.node[*].wlan[*].mac.rateSelection.modeSet             = "p"
*.node[*].wlan[*].mac.rateSelection.dataBitrate         = 6 Mbps
*.node[*].wlan[*].mac.rateSelection.multicastBitrate    = 6 Mbps

# Definition of Statistical Region
*.node[*].wlan[*].mac.rx.statRegionHighwayStart         = 300
*.node[*].wlan[*].mac.rx.statRegionHighwayEnd           = 700
*.node[*].wlan[*].mac.rx.statRegionGridStartX           = 70
*.node[*].wlan[*].mac.rx.statRegionGridEndX             = 375
*.node[*].wlan[*].mac.rx.statRegionGridStartY           = 111.25
*.node[*].wlan[*].mac.rx.statRegionGridEndY             = 333.75

*.node[*].middleware.updateInterval                     = 0.1 s
*.node[*].middleware.my_jitter                          = 0.5 s
*.node[*].middleware.datetime                           = "2017-06-01 12:35:00"
**.assumeNonUrbanEnvironment                            = true
```

Figure 66. Nodes settings

B.2.6. Medium Settings

The Medium settings in Figure 67 include configurations related to the radio channel. In the example, the background noise power is set to -110dBm, while the path loss is modeled using Rayleigh Fading with alpha set to 3 for an urban environment. Moreover, the presence and impact of physical walls can be configured. The properties of the walls, such as the shape, position and material, are specified in `walls.xml`. The presence of walls can be enabled by either setting `DielectricObstacleLoss` or `IdealObstacleLoss` for the `obstacleLossType` and disabled by leaving this field

blank. `DielectricObstacleLoss` causes partial absorption of signal when passing through the walls, while `IdealObstacleLoss` fully blocks the passage of signal.

```
##########################################################################
# MEDIUM settings (./extern/inet/src/inet/physicallayer/)
##########################################################################

World.radioMedium.obstacleLossType              = "DielectricObstacleLoss"
World.radioMedium.propagationType               = "ConstantTimePropagation"
World.radioMedium.analogModelType               = "ScalarAnalogModel"
World.radioMedium.backgroundNoiseType           = "IsotropicScalarBackgroundNoise"
World.radioMedium.backgroundNoise.power         = -110 dBm
World.radioMedium.pathLossType                  = "RayleighFading"
World.radioMedium.pathLoss.alpha                = 3


# "obstacleloss" folder: DielectricObstacleLoss, IdealObstacleloss
# "propagation"  folder: ConstantSpeedPropagation, ConstantTimePropagation
# "analogmodel"  folder: ScalarAnalogModel, dimensionalAnalogModel
# "pathloss"     folder: TwoRayGroundReflection, TwoRayInterference
#                        FreeSpacePathLoss, RayleighFading, RicianFading,
#                        LogNormalShadowing, NakagamiFading
# typical alpha  values: freespace 2, urban 2.5-3.5, shadowed urban 3-5,
#                        inbuilding LoS 1.7, inbuilding nLos 4-6
# value of alpha affects both BLUE and GREY circles


# if obstacleLossType <> ""
World.physicalEnvironment.config                = xmldoc("my_roads/walls.xml")
World.physicalEnvironment.groundType            = "FlatGround"
World.physicalEnvironment.ground.elevation      = 0 m
```

Figure 67. Medium settings

B.2.7. Scenarios Settings

Finally, the Scenarios settings are shown in Figure 68. This section includes configurations that are specific to certain scenarios. Some example fields are the simulation time, warm-up time and CAM transmit period. `Warmup-period` defines the time from which the statistics are started to be recorded, such as when the scenario has reached steady-state conditions.

```
##############################################################################
# SCENARIOS settings
##############################################################################

repeat                                          = 1     # change rng, affects num of runs (-r)
# from Cohda MK5 DS --> The packet error rate (PER)
# is less than 10% at a PSDU length of 1,000 octets for these input levels


# ============================================================
# scenario GRID
# ============================================================
[Config grid]
        sim-time-limit                          = 500s
        warmup-period                           = ${warmupTime = 300}s
        *.traci.launcher.sumo                   = "sumo-gui"
        *.node[*].middleware.services           = xmldoc("services_CA.xml")
        **.globalListener.**.vector-recording   = true
        **.globalMapper.**.vector-recording     = true
        *.node[5].**.vector-recording           = true
        *.node[8].**.vector-recording           = true
        *.node[13..16].**.vector-recording      = true
        experiment-label                        = ${configname}


        # ----------------------------------------
        # num. vehicles effect, w/ or w/o buildings
        # ----------------------------------------
        [Config grid-traffic_fluid-period]
                extends                         = grid
                measurement-label               = fluid-${iterationvars}
                **.cam_msg_period               = ${period = 100, 200, 500, 1000}
                *.traci.launcher.sumocfg        = "my_roads/grid_traffic_fluid/my_grid.sumo.cfg"
```

Figure 68. Scenario settings

B.3. Running a Scenario

There are two ways to run a scenario: (1) using the command line and (2) using the graphical user interface (GUI). The former is expected to take less time to finish; however, it is not possible to visualize the flow of packets in the OMNeT++ GUI.

To execute a scenario using the command line, enter the following command.

```
v2x-arch_11p/artery/scenarios/artery$ ./run_ARTERY_cmdenv_from_sublime.cmd
```

The scenario to be executed is configured in `declare -a configs_to_test` of `run_ARTERY_cmdenv_from_sublime.cmd`.

To run a scenario using the OMNeT++ GUI, enter the following command.

```
v2x-arch_11p/artery/scenarios/artery$ ./run_ARTERY_guienv_from_sublime.cmd
```

The scenario to be executed and the message period are selected through the pull-down menu of the OMNeT++ GUI.

For both cases, it is necessary to update the following line in `omnetpp.ini` to start the SUMO GUI. The SUMO GUI enables the visualization of the movement of vehicles during simulation.

```
*.traci.launcher.sumo = "sumo-gui"
```

B.4. Result and Analysis Files

The scalar (`*.sca`) and vector (`*.vec`) statistics may be recorded depending on the settings indicated in the `omnetpp.ini` (i.e., if they are enabled). The result files are saved in `v2x-arch_11p/artery/scenarios/artery`. These are used as inputs to the analysis file (`*.anf`), which is created by double-clicking either of the two result files in the OMNeT++ IDE.

The analysis file is used by the built-in Result Analysis tool of the OMNeT++ IDE. As illustrated in Figure 69, this tool offers several options for processing and visualizing the results, including customization of scalar/vector/histogram plots.

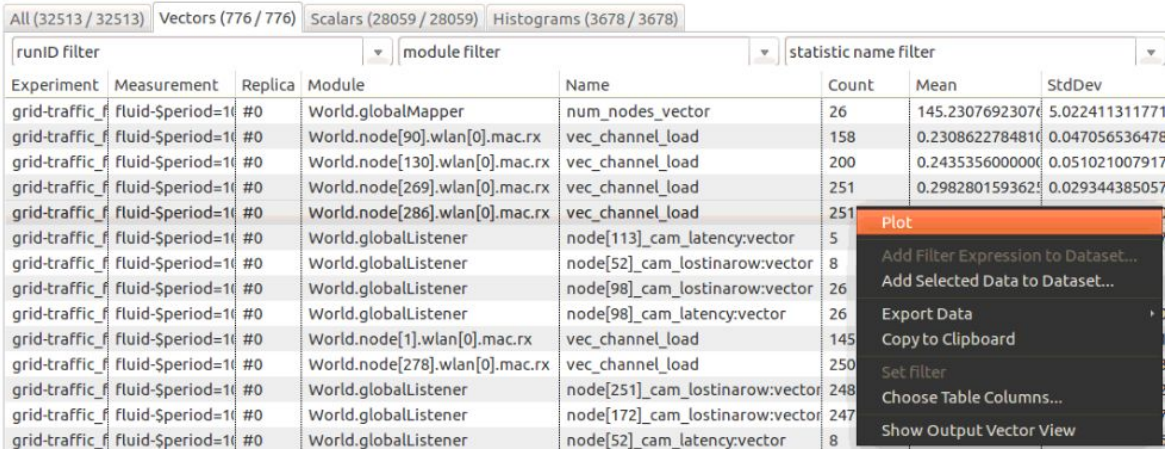
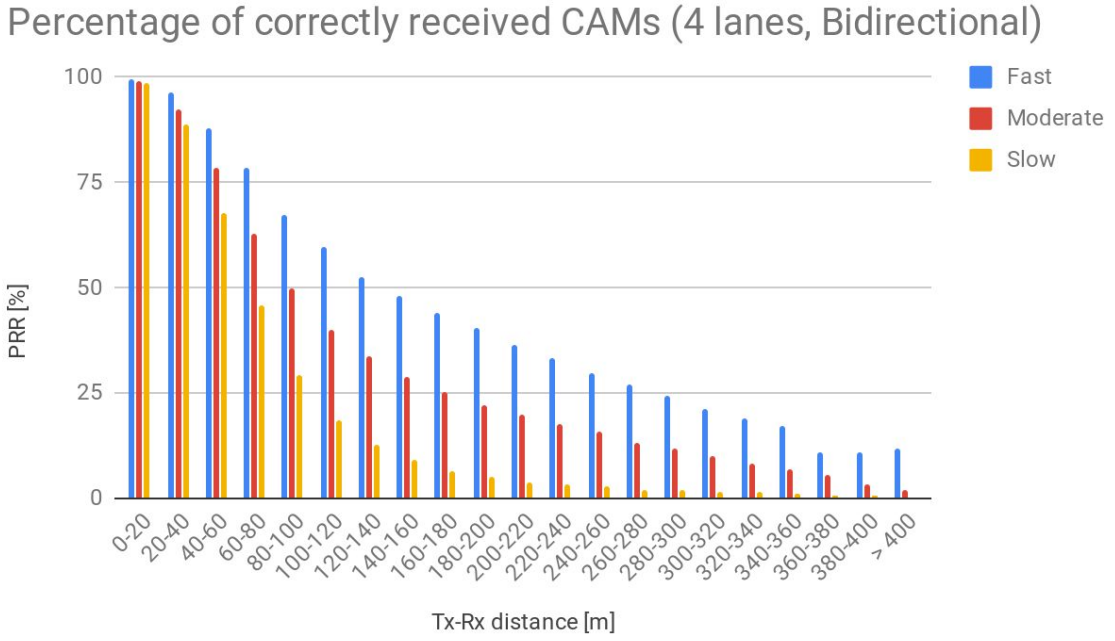


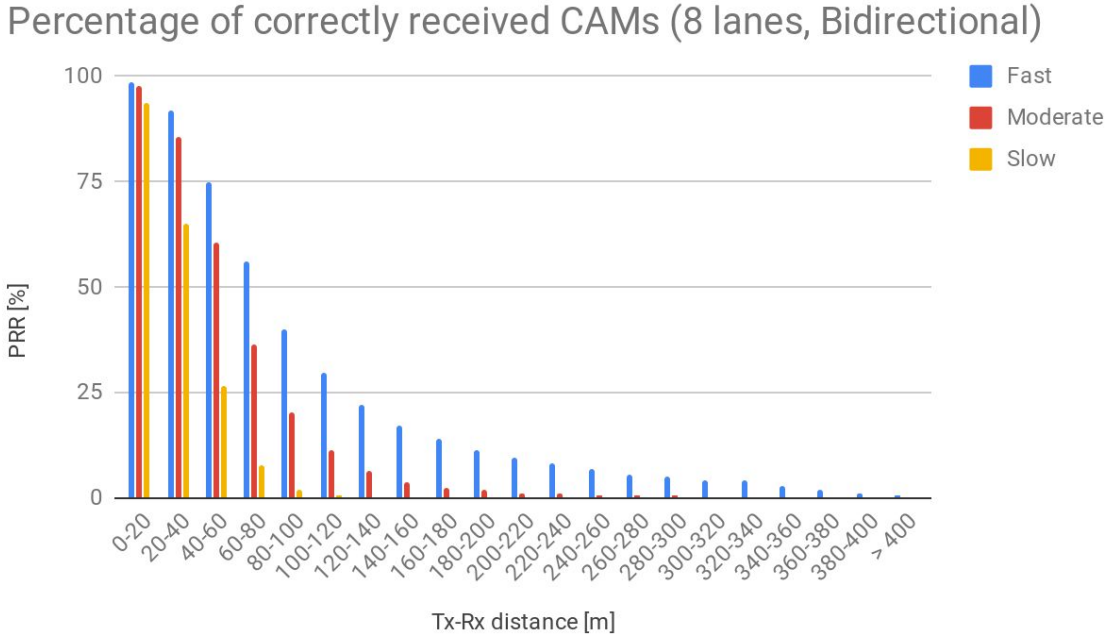
Figure 69. Result analysis tool

B.5. Complete Set of Figures

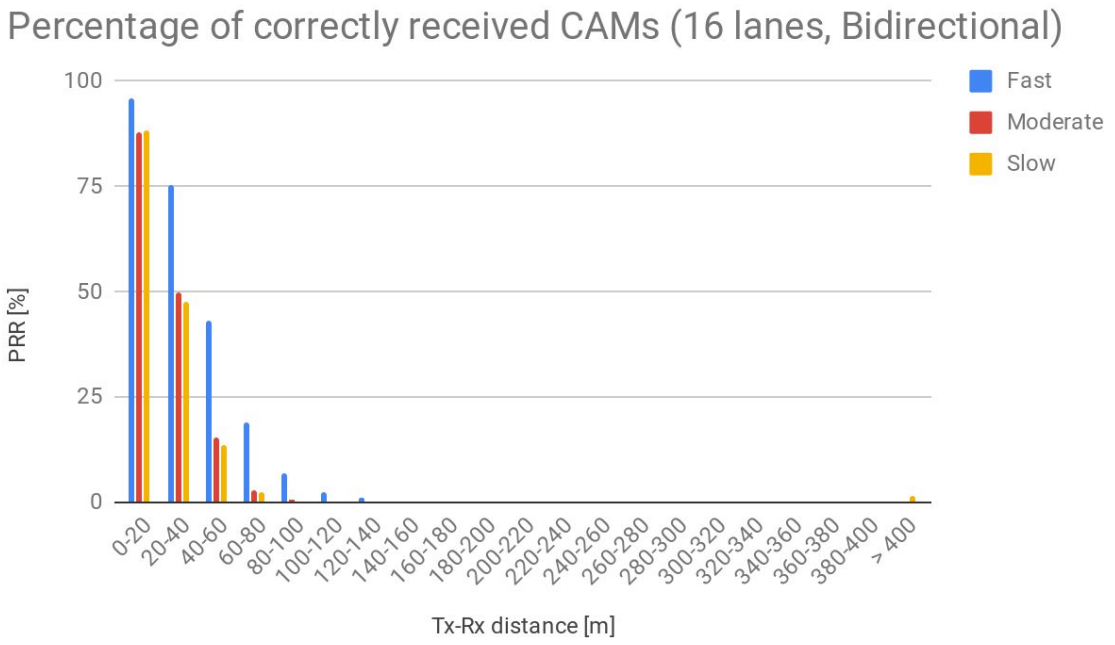
B.5.1. Packet Reception Ratio



(a) 4 lanes

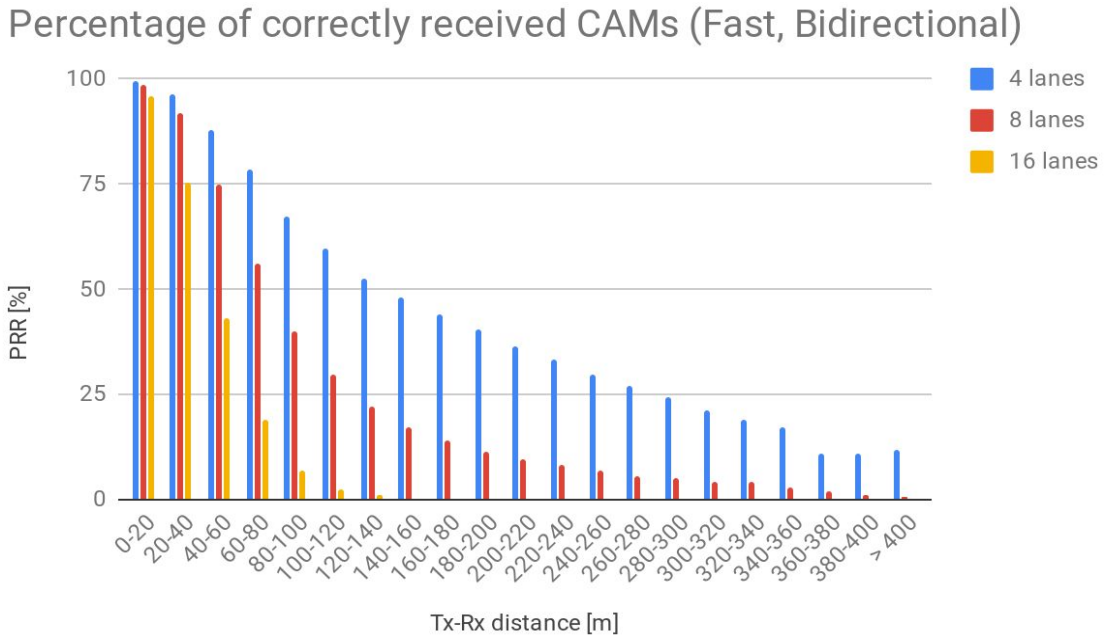


(b) 8 lanes

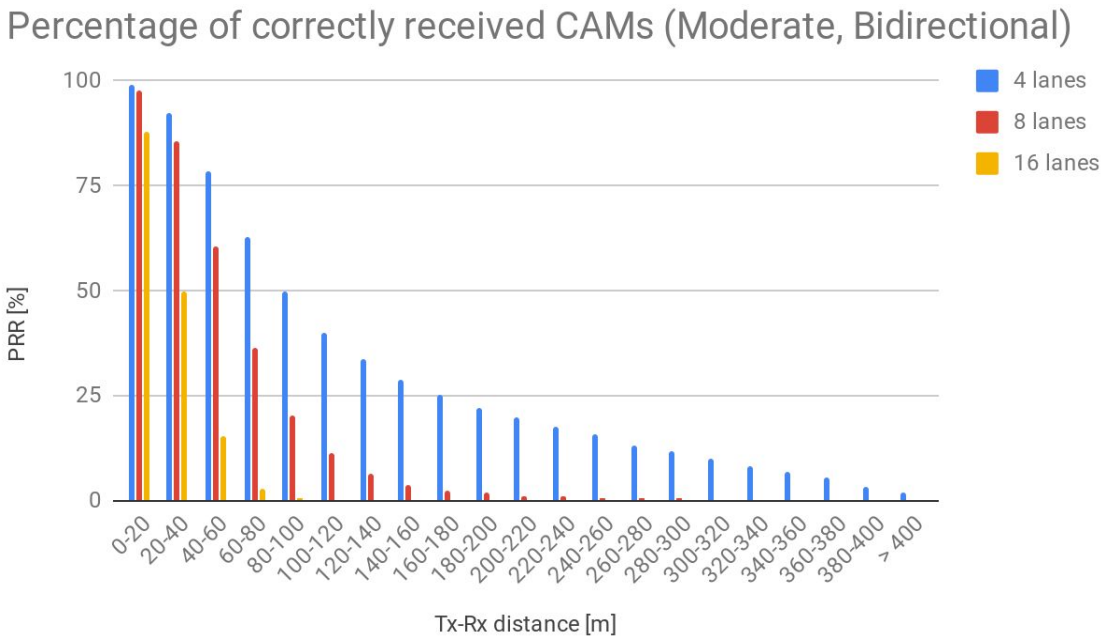


(c) 16 lanes

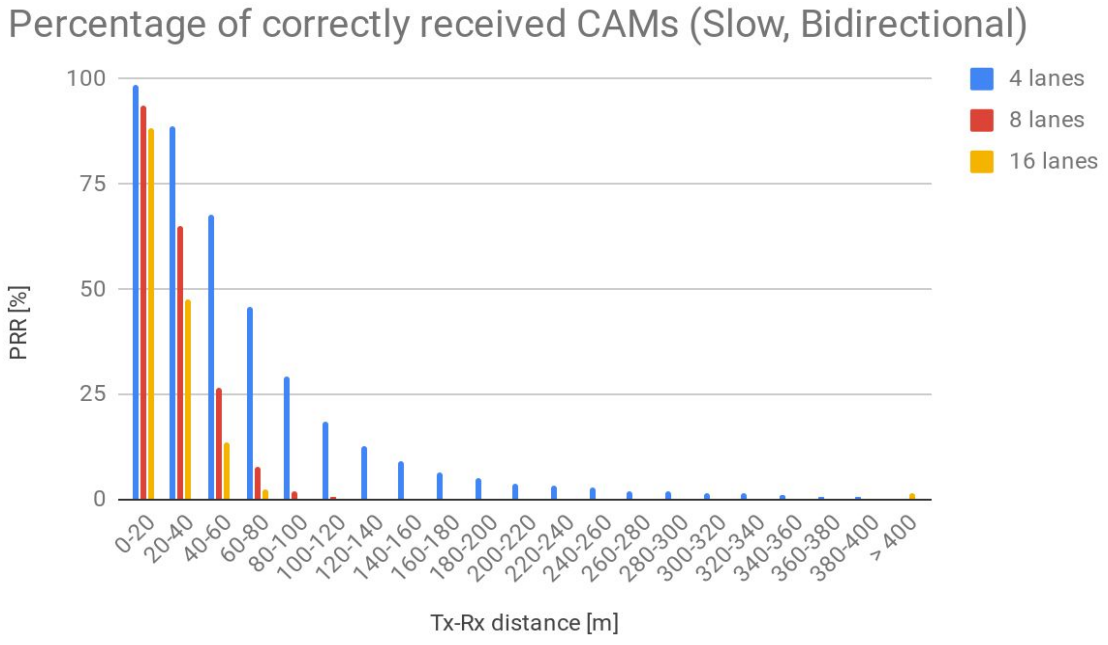
Figure 70. PRR of a bidirectional highway scenario with variable speed



(a) Fast



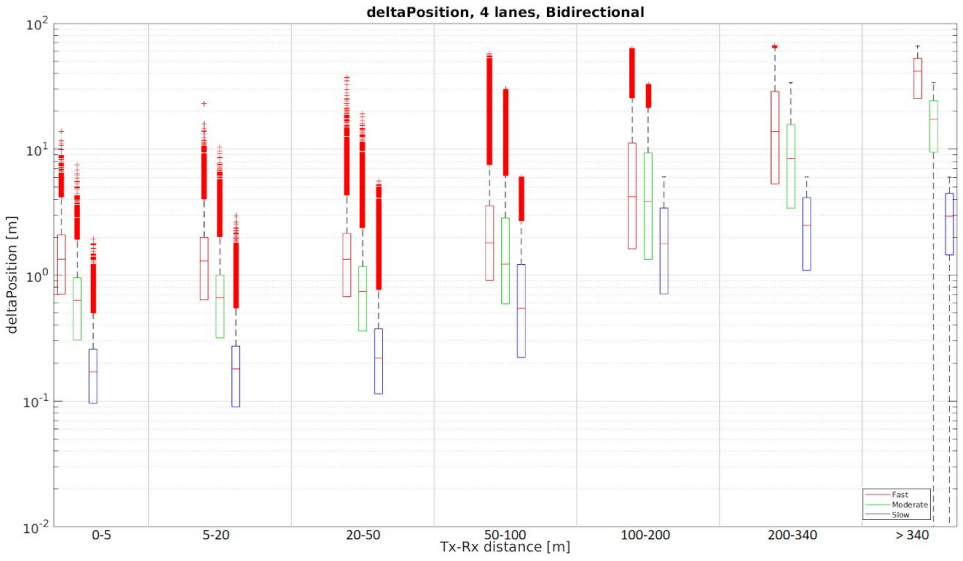
(b) Moderate



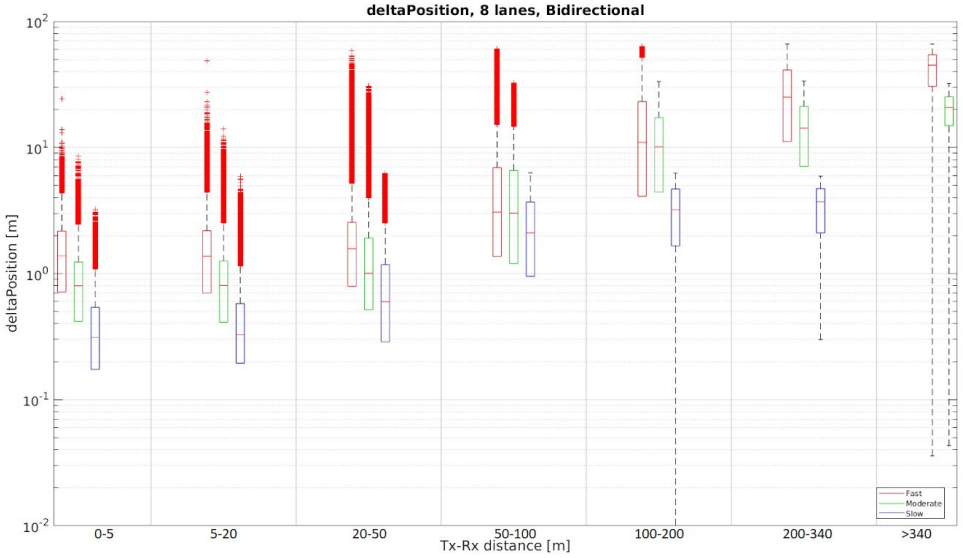
(c) Slow

Figure 71. PRR of a bidirectional highway scenario with variable lane count

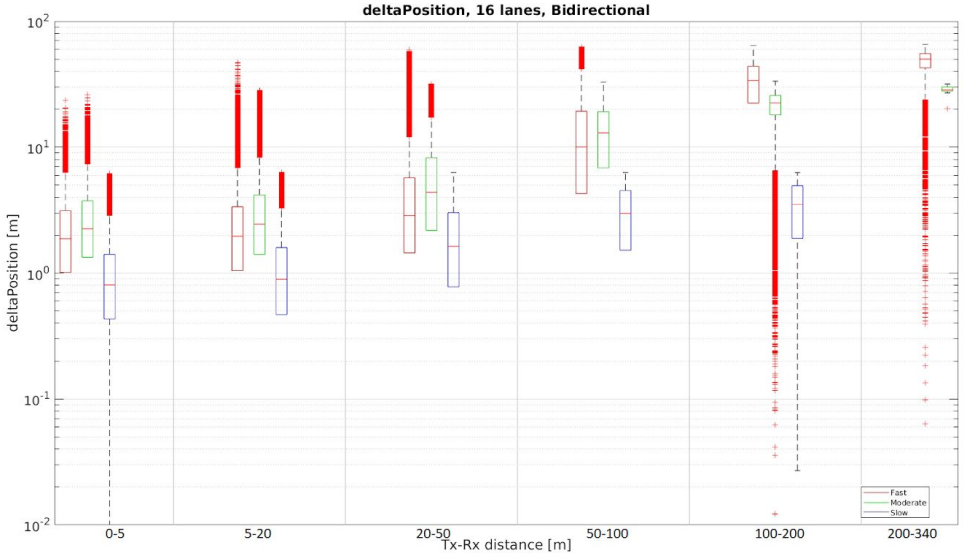
B.5.2. Position Error



(a) 4 lanes

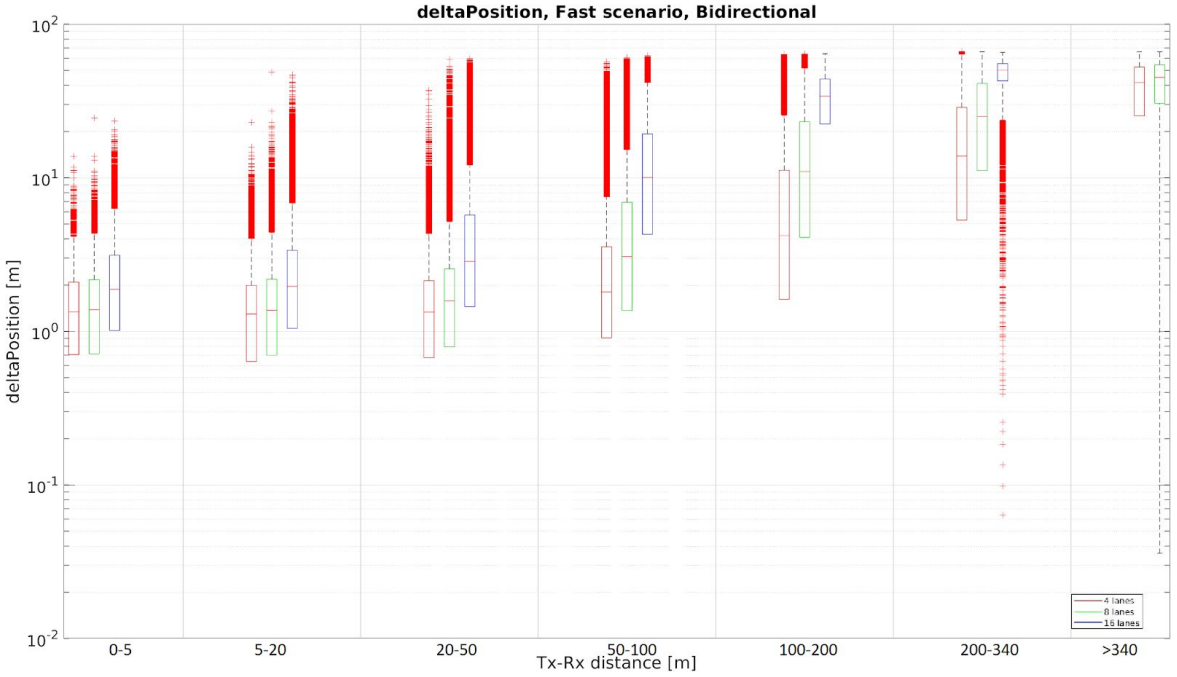


(b) 8 lanes

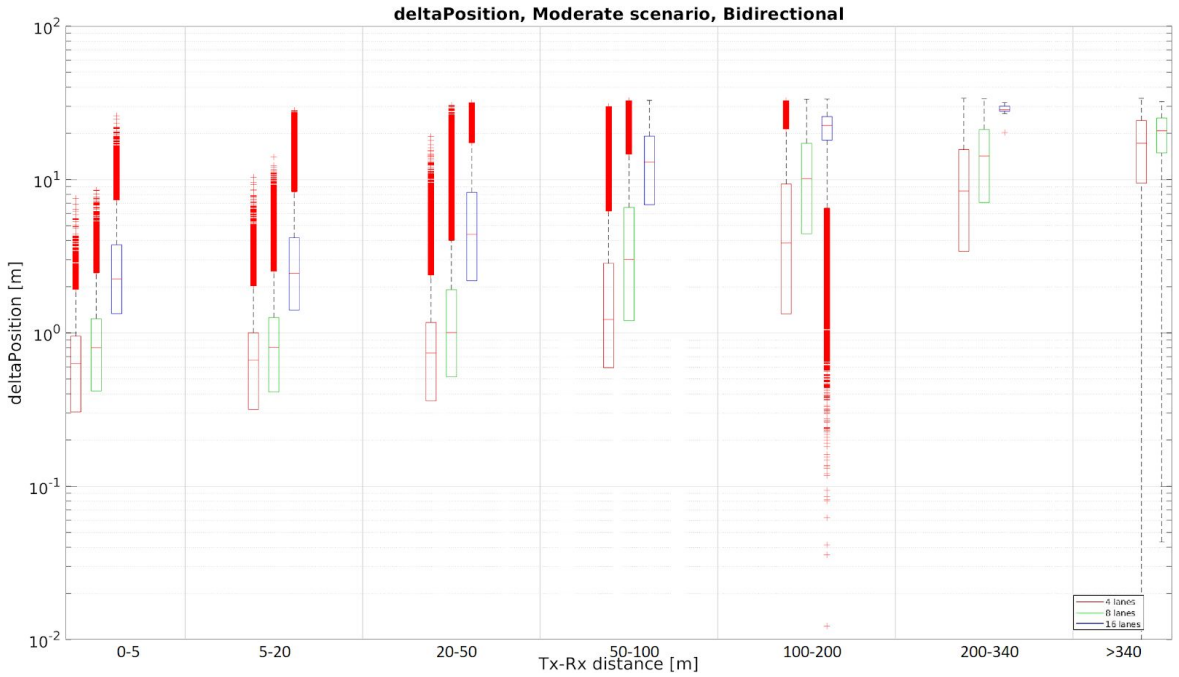


(c) 16 lanes

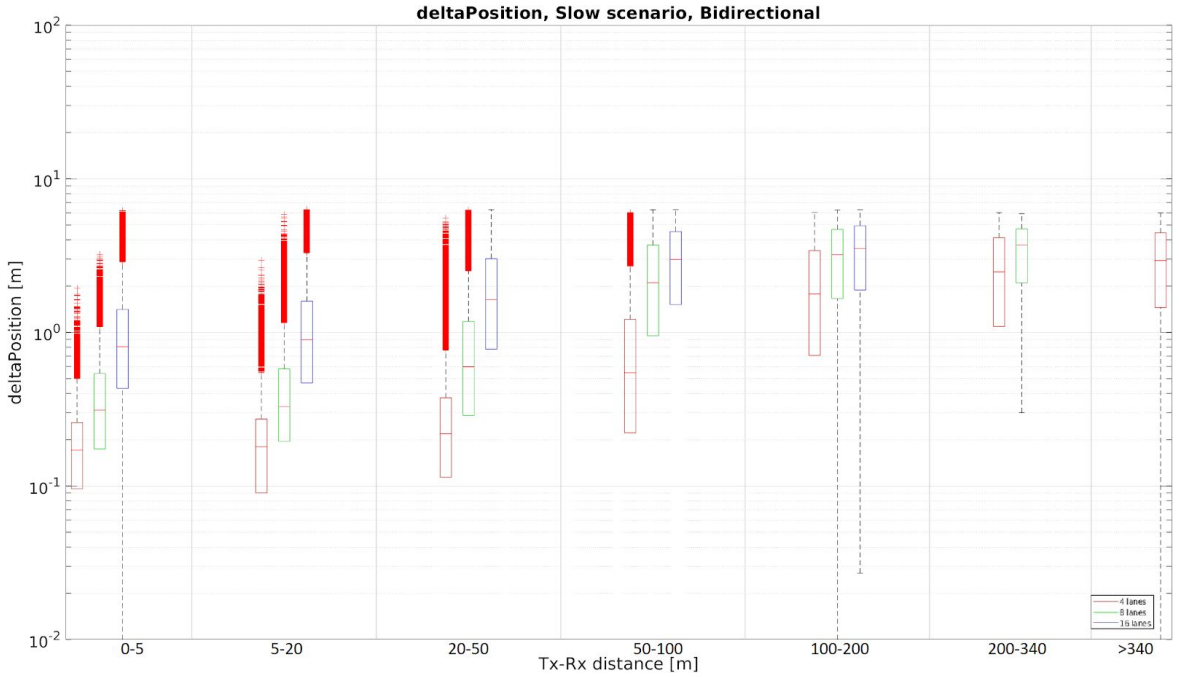
Figure 72. Position error of a bidirectional highway scenario with variable speed



(a) Fast



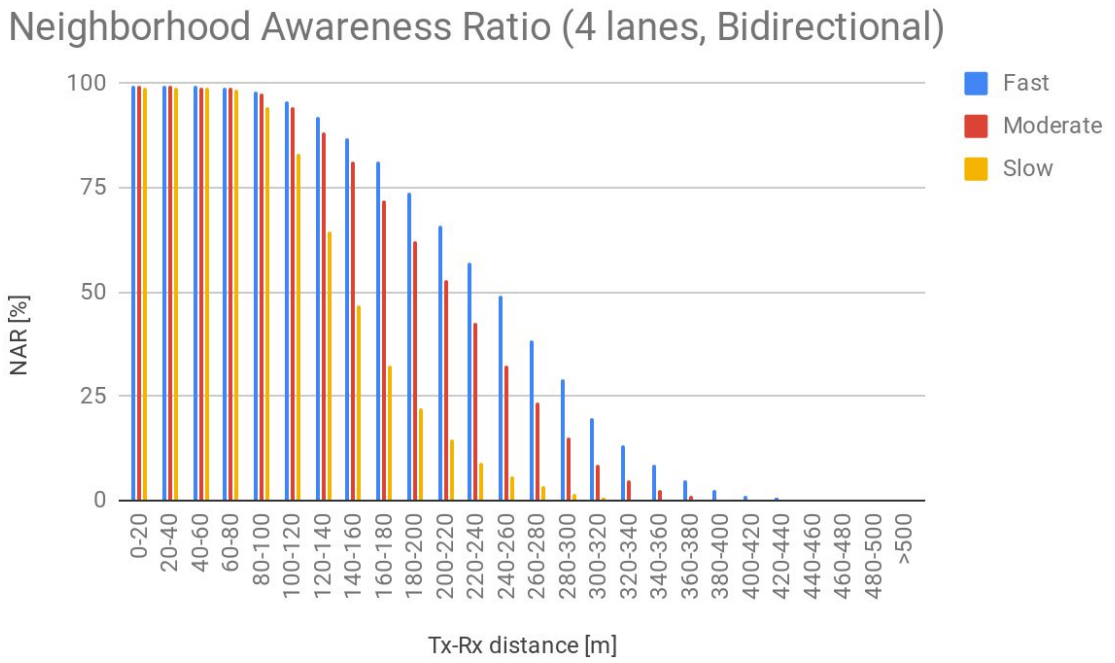
(b) Moderate



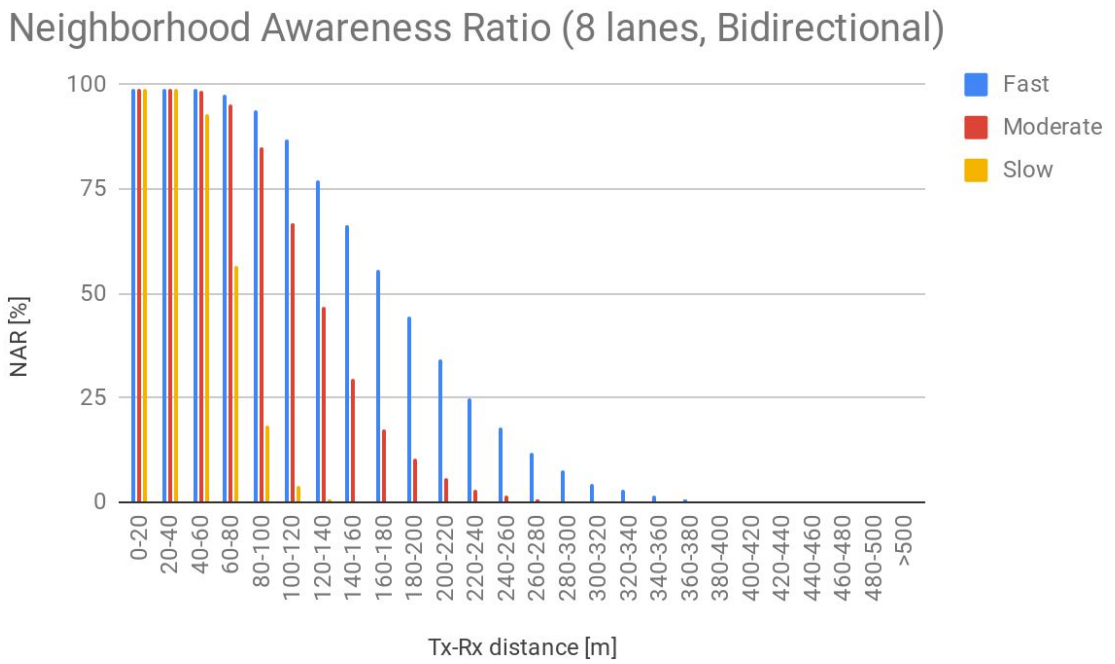
(c) Slow

Figure 73. Position error of a bidirectional highway scenario with variable lane count

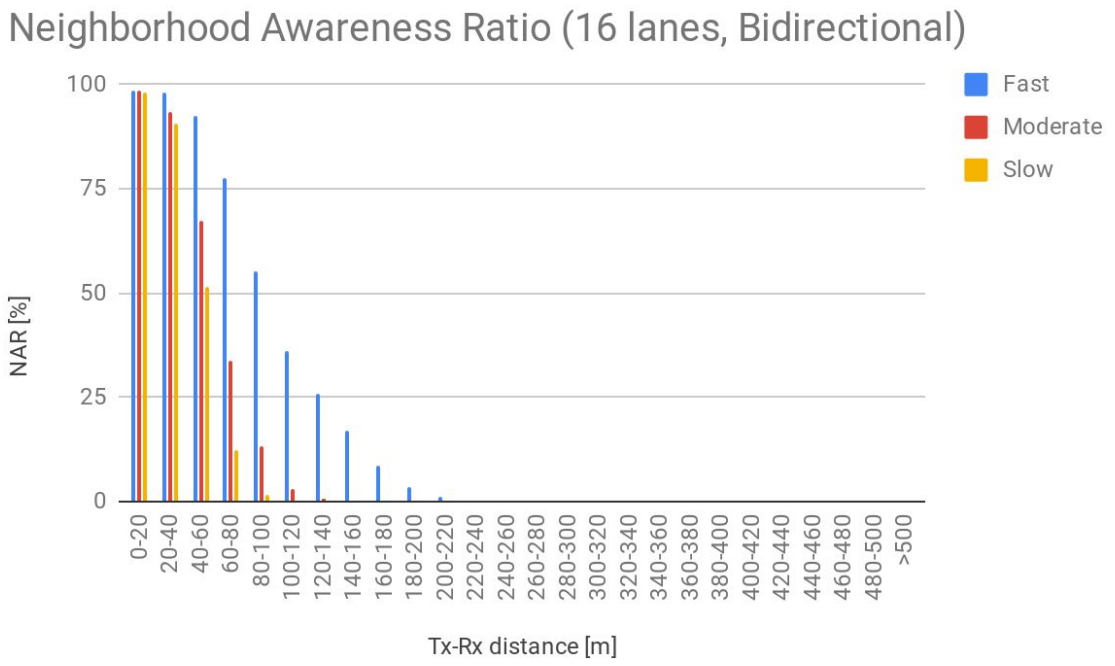
B.5.3. Neighborhood Awareness Ratio



(a) 4 lanes

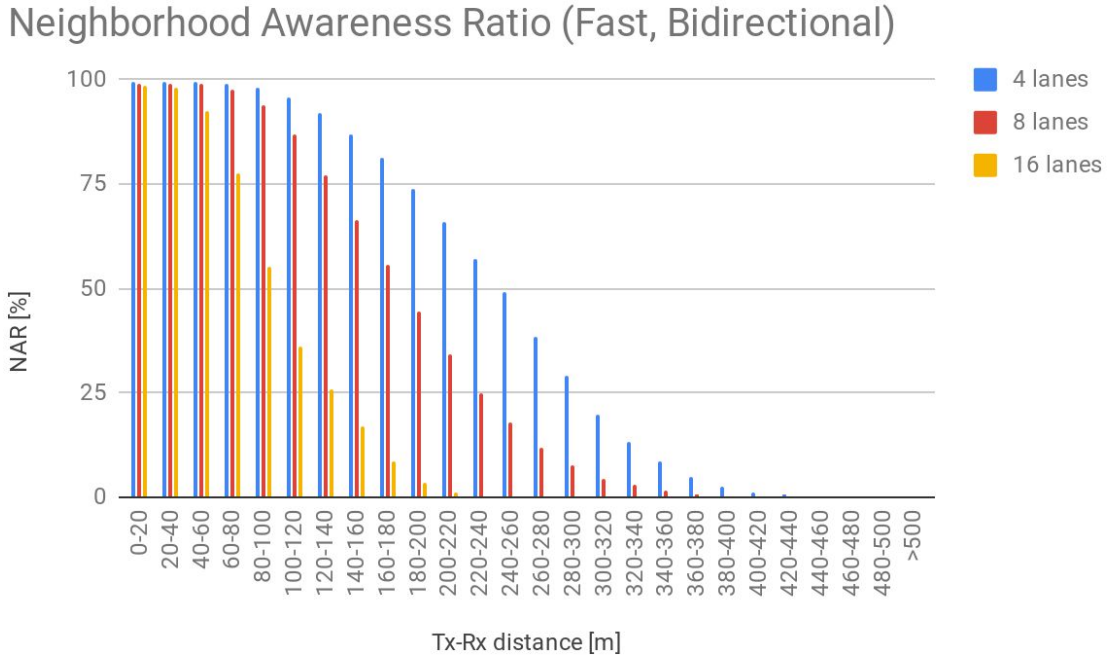


(b) 8 lanes

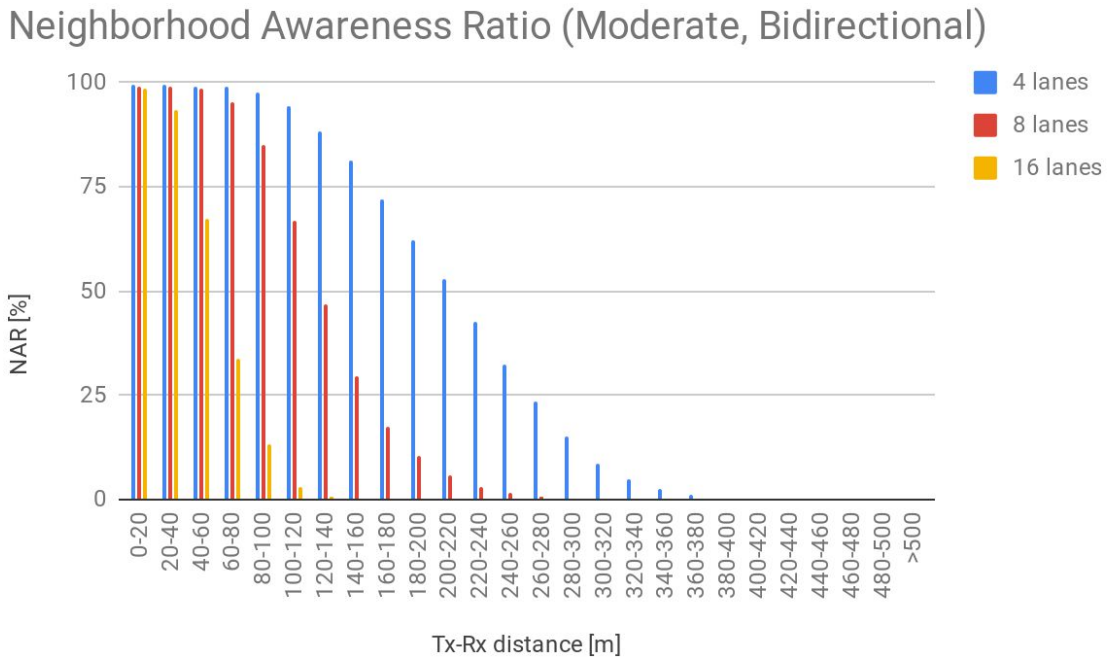


(c) 16 lanes

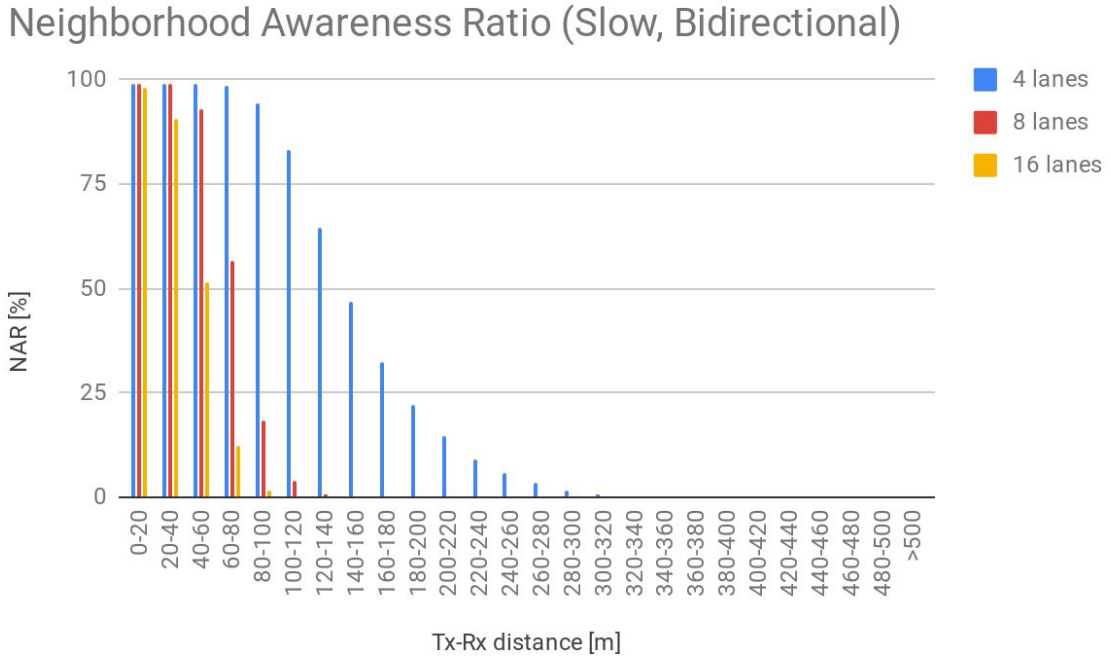
Figure 74. NAR of a bidirectional highway scenario with variable speed



(a) Fast



(b) Moderate



(c) Slow

Figure 75. NAR of a bidirectional highway scenario with variable lane count

Glossary

3GPP - 3rd Generation Partnership Project

ASN.1 - Abstract Syntax Notation One

BRAN - Broadband Radio Access Network

BSA - Basic Set of Applications

BSS - Basic Service Set

BTP - Basic Transport Protocol

CA - Cooperative Awareness

CAM - Cooperative Awareness Message

C-ITS - Cooperative ITS

CLI - Command-Line Interface

ComS - Communities Services

CoNa - Co-operative Navigation

CRDA - Central Regulatory Domain Agent

CSM - Cooperative Speed Management

CSMA/CA - Carrier Sense Multiple Access with Collision Avoidance

CSV - Comma Separated Values

D2D - Device-to-Device

DCC - Decentralized Congestion Control

DEN - Decentralized Environmental Notification

DENM - Decentralized Environmental Notification Message

DSRC - Dedicated Short-Range Communications

EDCA - Enhanced Distributed Coordination Access (EDCA)

ETSI - European Telecommunication Standards Institute

GAC - Geographically-scoped Anycast

HF - High-frequency

HMI - Human Machine Interface

HST - Header Subtype

HT - Header Type

IDE - Integrated Development Environment

IEEE - Institute of Electrical and Electronics Engineers

IP - Internet Protocol

ITS - Intelligent Transport Systems

ITSC - ITS Communication

ITS-S - ITS Station

LBS - Location Based Services

LCM - Life Cycle Management

LF - Low-frequency

LLC - Logical Link Control

LOS - Line-Of-Sight

LS - Location Service

MAC - Medium Access Control

MATLAB - MATrix LABoratory

MHL - Maximum Hop Limit

MIB - Management Information Base

MPDU - MAC Protocol Data Unit

NED - NEtwork Description

NH - Next Header

OBU - On-Board Unit

OCB - Outside the Context of a BSS

OFDM - Orthogonal Frequency Division Multiplexing

OFDMA - Orthogonal Frequency-Division Multiple Access

OMNeT++ - Objective Modular Network Testbed in C++

OSI - Open Systems Interconnection

PC - Personal Computer

PDU - Protocol Data Unit

PHY - Physical Layer

PL - Payload

PoE - Power-over-Ethernet

PPDU - Physical Protocol Data Unit

PRR - Packet Reception Ratio

PSDU - Physical layer Service Data Unit

QoS - Quality of Service

RSU - Roadside Unit

SAM - Service Announcement Message

SHB - Single-Hop Broadcast

SPAT - Signal Phase And Timing

SUMO - Simulation of Urban MObility

TC - Traffic Class

TCP - Transmission Control Protocol

TDC - Transmit Data rate Control

TPC - Transmit Power Control

TraCI - Traffic Control Interface

TRC - Transmit Rate Control

TSB - Topologically-Scoped Broadcast

UDP - User Datagram Protocol

V2G - Vehicle-to-Grid

V2I - Vehicle-to-Infrastructure

V2N - Vehicle-to-Network

V2P - Vehicle-to-Pedestrian

V2V - Vehicle-to-Vehicle

V2X - Vehicle-to-Everything

Veins - Vehicles in network simulation

VRU - Vulnerable Road User

WLAN - Wireless Local Area Network