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

telecos BCN

V2X communications performance analysis using open-source simulators

Degree Thesis
In partial fulfilment of the requirements for the degree in Telecommunications Engineering

Author
Eudald Llagostera Brugarola

Advisors
Jordi Casademont Serra
Jordi Contreras Suñé

Barcelona, June 2021

Contents

List of Figures

List of Tables

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 and design an interface to communicate the Objective Modular Network Tested in C++ (OMNeT++, a network simulator simulating V2X scenarios) , with the Car Learning to Act (CARLA, an autonomous driver simulator), feeding the messages received from the OMNeT++ simulation to CARLA. This way, being the Ego vehicle more aware of their surroundings. This project also aimed to evaluate the effectiveness of the Cooperative awareness (CA) basic service through the development of an IEEE 802.11p-based V2X system simulator. The simulations were executed varying the density of vehicles and Cooperative Awareness Message (CAM)'s length in two different scenarios: the highway scenario and the Manhattan grid scenario. The performance was then assessed by analyzing the Packet Error Rate (PER), the number of messages received, and also, in the Manhattan scenario, differentiating the Line of Sight (LOS) cases. The presence of more vehicles caused higher packet losses due to increased interference and collisions probability, leading to higher PER values. When the CAM's length increased, the PER as well as the interference in the scenario increased. In the Manhattan scenarios a peak of more packets received and more interference was present in the intersections, leading to a higher PER values.

Resum

Un dels aspectes claus de la comunicació entre Vehicles cap a Tot (V2X) és el concepte de la consciència cooperativa, on l'intercanvi periòdic de l'estat de les informacions permet als vehicles ser conscients del seu entorn augmentant així la seguretat i l'eficàcia del trànsit. Aquest projecte consta de dos objectius, el primer ha estat implementar i dissenyar una interfície per comunicar l' *Objective Modular Network Tested in C++* (OMNeT++, un simulador de xarxes que permet simular escenaris V2X) amb el *Car Learning to Act* (CARLA, un simulador de conducció autònoma), transmetent els missatges simulats amb l'OMNeT++ cap al CARLA. D'aquesta manera el vehicle egocèntric és més conscient del seu entorn. El segon objectiu ha estat avaluar l'efectivitat del servei bàsic de Conciència cooperativa (CA) mitjançant un simulador IEEE 802.11p V2X. Les simulacions s'han executat variant la densitat de vehicles i les mides dels Missatges de Concència Cooperativa (CAM) en dos escenaris diferents: una autopista i una quadrícula de Manhattan. El rendiment ha estat avaluat analitzant la Tassa de Paquets Erronis (PER) i el nombre de missatges rebuts en els dos escenaris. En el cas de la quadrícula de Manhattan també s'ha diferenciat el cas de Vista Directa (LOS). La presència de més vehicles ha causat més pèrdues de paquets a causa de l'increment de la interferència i la probabilitat de col·lisions de paquets, incrementant així els valors de la PER. En el moment que s'ha augmentat la mida dels missatges CAM, la PER també ha augmentat, ja que les interferències dels escenaris han augmentant. A l'escenari de Manhattan hi ha un pic de més paquets rebuts i més interferències a les interseccions, la qual cosa comporta un increment de la PER.

Acknowledgements

I would like to express my sincere gratitude to my advisors, Dr. Jordi Casademont Serra and Mr. Jordi Contreras Suñé, for the guidance and for giving me the opportunity to learn a lot through this project. I would also like to thank Jordi Marias Parella, Dr. Juan-Luis Gorricho Moreno, for all the help they provided. Finally, I would like to thank to my family and friends for their continuous support and encouragement.

Revision history and approval record

Revision	Date	Purpose
0	30/03/2021	Document creation
1	14/06/2021	Document revision

Document Distribution List

Name	e-mail
Eudald Llagostera Brugarola	eudald.llagostera@estudiantat.upc.edu
Jordi Casademont i Serra	jordi.casademont@upc.edu

Written by:		**Reviewed and approved by:**	
Date	14/06/2021	Date	14/06/2021
Name	Eudald Llagostera Brugarola	Name	Jordi Casademont i Serra
Position	Project Author	Position	Project Supervisor

Chapter 1

Introduction

Safety still remains one of the biggest concerns in the traffic infrastructure and automobile technology. Road accidents are one of the leading causes of death with an estimated total of 1.35 million each year [1]. Vehicle-to-Everything (V2X) communications is considered to be a revolutionary technology which would revolutionize road safety. Moreover, V2X aims to increase traffic efficiency, reduce environmental impact (with efficient driving), and improving the overall transport experience.

The project was performed in the framework of the foundation i2CAT, concretely enrolled with H2020 CPSOSaware, an European project. The project works with three open-source simulators, which simulates V2X architectures and autonomous driving in different scenarios. Working with these simulators, the project aims to the development of an external interface to control the communication between the three simulators, which was a new part started from scratch. And to perform an evaluation of the performance of the Cooperative Awareness (CA) services in different scenarios, what has been done before, but not with the actual version of the simulators. Due to the new versions, the anterior work has been outdated and it's not functional with the new versions of the simulators.

1.1 Scope

This project had two main goals. One was to implement the CA basic service and evaluate its effectiveness in enabling cooperative awareness among vehicles, and the other, was to connect existing system-level simulators from the V2X and autonomous driving.

For the evaluation of the CA basic service an existing system-level IEEE 802.11p simulator was used as the foundation in this project. The simulator was then modified to enable the variation of important parameters (density, CAM's length) in order to deduce the impact on system performance, and to perform more realistic simulations under different road topologies (highway, Manhattan grid). Moreover, different metrics were designed to asses the CA basic performance.

For the simulators connection, three simulators had been connected between them and an analysis of the protocol used for connecting all simulators was performed. Then, a design of an interface to connect all the simulators was depicted and a first version to control and synchronize them was developed.

It is important to note that the experiments carried out only with the IEEE 802.11p standard, and comparisons of the different V2X technologies were out of scope of the project.

1.2 Work Plan

The project was carried out form February 2021 to June 2021, and it was divided in five work packages, as can be seen in Figure 1.1. The first month was dedicated to studying the concepts that were necessary for the implementations 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 of the simulators.

The next months were allocated to work on the two tasks. In the case of the evaluation of the CA basic service, the development of the road scenarios, and adjusting the simulator parameters to make the simulations as realistic as possible. As well as, preparing the statistical results. On the other hand, a deep analysis of the TraCI protocol with Wireshak was performed together with the design and implementation of a first version of an interceptor between the simulators.

In order to develop all the tasks, the usage of new tools was needed. A great part of the time was invested in learning programming languages as C++ for the enhancements in the Objective Modular Network Tested in C++ (OMNeT++) simulator, Python for the implementation of the interceptor and analysis of the Traffic Control Interface (TraCI) protocol, Python modules as Pandas or Matplotlib to prepare the statistic results, and SQLite to acquire basic knowledge of databases to manipulate the data that the simulations were outputting.

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 simulation parameters when unexpected results were observed. This phase was relatively time-consuming since a single batch of simulations took a few days to complete.

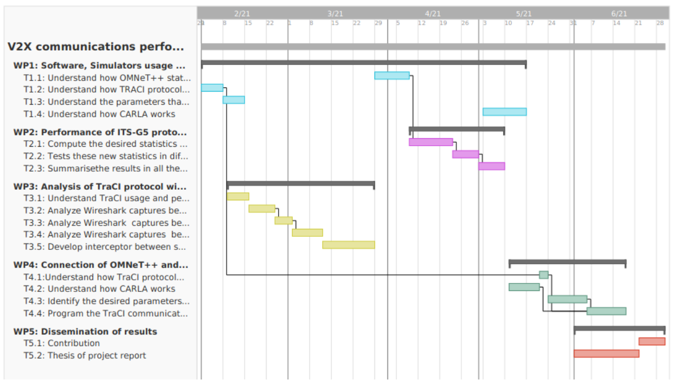


Figure 1.1: Gantt diagram

1.3 Budget

As explained in the introduction, this project was performed in the framework of the foundation i2CAT, concretely enrolled in an European project, CPSOSaware. As such, and in-depth study of its financial is out of scope. However, a very simple cost analysis can be provided.

The duration of this project has been 20 weeks, requiring one person to spend an average of 40 hours a week. This thesis has been developed with a cooperation agreement between the Polytechnic University of Catalonia (UPC) and i2CAT, with a cost of 9€/h for a total of 7200€.

Besides form man-power the basic expected hardware for a software project was needed. Including two desktop computers provided by UPC with a estimated price of 1000€ each, all the other software used was free and open source (Ubuntu 20.04 LTS, OMNeT++ 5.6.2, SUMO 1.9.2, CARLA 0.9.9, etc.).

The summary of the total budget is depicted in Table 1.1.

Concept	**Cost**
Equipment	2000€
Personal	7200€
TOTAL	9200€

Table 1.1: Summary of total budget

Chapter 2

State of Art

2.1 V2X Protocol Architecture

The C-ITS network architecture consists of different entities, or ITS station (ITS-S), communicating with each other. These are:

- Personal ITS-S – handheld devices of pedestrians
- Vehicle ITS-S – On-board Unit(OBU) mounted on vehicles
- Central ITS-S – traffic management centres
- Roadside ITS-S – Roadside Unit (RSU) or fixed traffic infrastructures

The combination of any of these entities results to different communication modes.

The ITS-S reference architecture defines the protocol stack implemented on each station. It comprises four horizontal layers along with two vertical entities [2] , as it can be seen in Figure 2.1. It is analogous to the Open System Interconnection (OSI) model, except that it extends the model to include the ITS applications, as shown in Table 2.1.

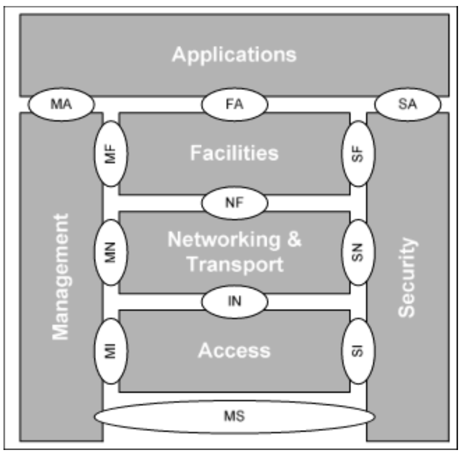


Figure 2.1: ITS-S reference architecture

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

Table 2.1: Mapping between ITS-S reference architecture and OSI model

2.1.1 Applications

ITS applications are formed by complementary ITS-S applications. A group of applications and use cases is known as the Basic Set of Applications (BSA). These use cases are categorized into the following three classes [3].

1. Active road safety: The goal of this class is to improve traffic safety by preventing road casualties.

2. Cooperative traffic efficiency: The goal of this class is to improve road traffic management, and increase traffic efficiency in terms of travel times, fuel consumption, emissions, etc.

3. Other applications: These include applications providing other services such as those for infotainment.

2.1.2 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 key role is to provide service to the ITS applications in the upper layer, and thus, the facilities are also referred to as basic service. Facilities can be grouped in two ways, according to: type of support and scope of support provided to the ITS BSA[3].

Classification of facilities according to the type of support provided:

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:

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.

On this layer, there can be found the so-known Cooperative Awareness Messages (CAM) and Decentralized Environmental Notification Message (DENM) which were proposed under the European Standardization.

2.1.3 Networking and Transport

The Basic Transport Protocol (BTP) provides and 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 GeoNetworking protocol. The way multiplexing/demultiplexing works is based on ports, which act as identifiers to distinguish different processes running on the ITS station. In the case of CAM and DENM facilities the BTP port corresponding to them are 2001 and 2002 respectively. 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 [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 packets transport, while BTP-B signals non-interactive. Moreover, they differ in packet structure, with BTP-A containing both: source and destinations ports, and BTP-B specifying only the destination port with the addition of destination port information in case of well-known ports, making clear that the BTP-B is designed for broadcast communications.

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 [5].

The GeoNetworking protocol has de following main functions:

1. Geographical addressing: A packet is sent to a destination node with a specific geographical position or to several destination nodes belonging to a geographical area.

2. Geographical forwarding: Each node maintains knowledge of the network

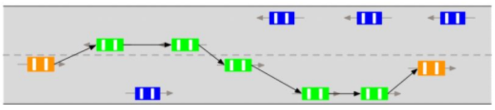
topology. When a node receives a packet, it examines the destination field, and compares the indicated geographical address to its knowledge of the network topology to make forwarding decisions.

The GeoNetworking routing employs different packet forwarding schemes depicted in Figure 2.2:

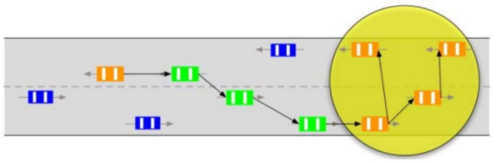
1. GeoUnicast: The packet is continuously forwarded by intermediate nodes until it reaches its destination.

2. GeoBroadcast: The packet is continuously forwarded until it reaches its destination geographical area. The nodes inside the area re-broadcast the packet, unlike the GeoAnycast.

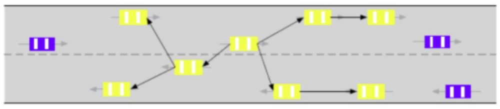
3. Topologically-scoped broadcast: The packet is continuously re-forwarded until the n-hop node.



(a) GeoUnicast



(b) GeoBroadcast



(c) Topologically-scoped broadcast

Figure 2.2: GeoNetworking routing schemes

2.1.4 Access

The ITS access layer maps to the data link and physical layers of the OSI reference model. The data link layer consists of the Medium Access Control (MAC) and the Logical Link Control (LLC) sublayers. The ITS access layer technology is termed ITS-G5. Wich is based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11 Wireless Local Area Network (WLAN) standard. IEEE 802.11p corresponds to the Physical Layer (PHY) and MAC layers and is a modification of IEEE 802.11a.

IEEE 802.11p employs an almost identical physical layer as the IEEE 802.11a.

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 and works on the 5.9 GHz band, as opposed to the 20 MHz of IEEE 802.11a and 5 GHz band, to make the signal more robust to fading and other propagation effects.

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 services channels (SSH) are allocated [6].

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), as shown in Table 2.2.

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	Multi-hop DENM, other data traffic

Table 2.2: ITS-G5 Traffic classes

In an ITS ad hoc network, the network topology varies constantly, and 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 channel capacity. 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. The different DCC access techniques used to control the channel load is:

1. Transmit Power Control (TPC): adjust the output power to reduce the resulting interference.

2. Transmit Rate Control (TRC): adjust the time between consecutive packets, such as is increased in high density scenarios.

3. Transmit Data rate Control (TDC): adjust the transfer rate, such that it is lowered at high load scenarios.

2.2 Simulator framework overview

For the development of this project, three simulators were used: SUMO for road traffic simulations, OMNeT++ for a network simulations and CARLA for validating of autonomous driving systems.

2.2.1 SUMO

Simulation of Urban Mobility (SUMO) is an open-source, portable simulator which handles vehicular traffic simulations and its characterizations. It allows the creation of different road topologies for the simulation, such as Highway and Manhattan grid scenarios, as well as the experimentation of various mobility models. A representation of Highway scenario can be seen in Figure 2.3. Moreover, it is microscopic, as vehicles are individually modelled, and move independently through the network.

The road topologies generated for the study of the behavior of CAMs messages in Highway and Manhattan scenarios were generated through (SUMO).

To perform a simulation in SUMO mainly three kind of configuration files are needed.

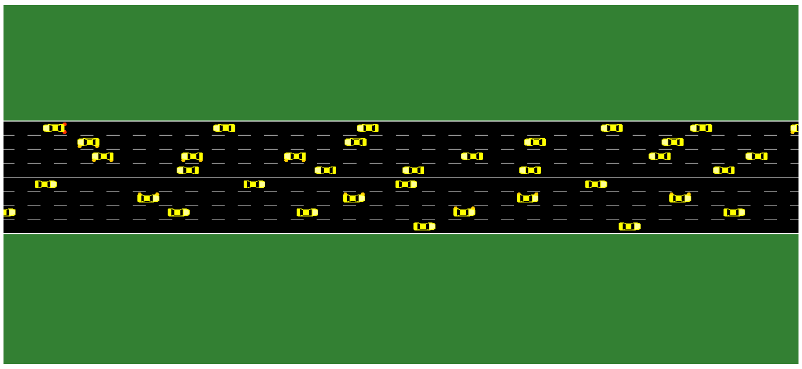


Figure 2.3: Highway scenario

Sumo Files

Each scenario is created using three SUMO files [7]. The basic files to run a simulation are the network file, the routes file and the configuration file.

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. Two edges are connected by junctions.

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, color, maximum speed and minimum gap for the vehicle ahead. Different routes are identified by their road id, and each of them defines the relevant edges and direction of movement of vehicles. Moreover, a flow contains the information to control the vehicles inserted in the scenario and how they behave during the simulation.

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 1ms. It also corresponds to the time interval with which vehicle positions are updated.

Besides these files a SUMO simulation may have additional-files. These files can include a wide range of network elements such as traffic light programs, detector definitions, variable speed signs, or they can also be used to configure simulation outputs as edge-based traffic measures or traffic light switching information.

For the creation or manipulation of SUMO XML description files, SUMO provides the following tools among others:

- Netconvert: imports digital road networks from different sources and generates road networks that can be used by other tools form the package.
- Netgenerate: generates abstract road networks that may be used by other SUMO-applications
- Netedit: a visual net editor. It can be used to create networks form scratch and to modify all aspects of existing networks.
- jtrouter: computes routes that may be used by SUMO based on traffic volumes and junction turning rations.

2.2.2 OMNeT++

The IEEE 802.11p based simulator is composed of several simulation frameworks of different functionality, as depicted in Figure 2.4.

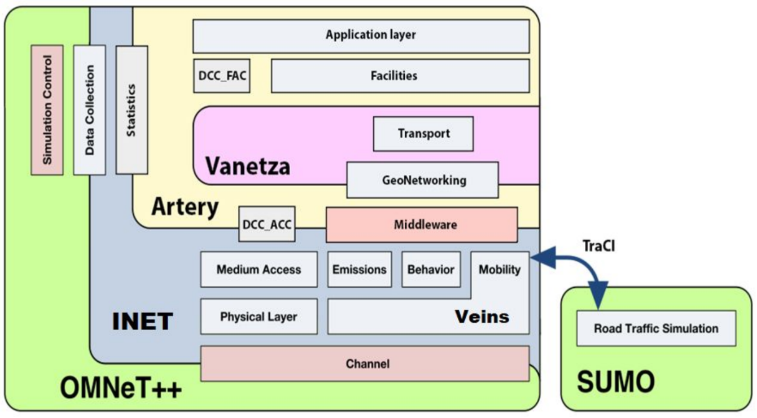


Figure 2.4: Simulation Framework Overview

The Objective Modular Network Testbed in C++ (OMNeT++) is an extensible and modular simulation library and framework for the research and development of complex distributed systems. Countless simulation models and model frameworks have been written in top of OMNeT++ by researchers in diverse areas, and vehicular networks is one of it. OMNeT++ works by assembling individual components/models to larger ones. 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 a network simulator 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 in particularly written for the OMNeT++ environment. Some of its features include models for the Internet stack (IPv4, IPv6, Transmission Control Protocol (TCP), User Datagram Protocol (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 frameworks.

Artery was originally developed as an extension of Veins, although, it could now be used independently. Artery corresponds to the application and facilities layers, which enable the generation of CAMs and DENMs. Moreover, Artery's middle ware provides common facilities to the multiple ITS-G5 services running on individual vehicles. The architecture of Artery is depicted in Figure 2.5.

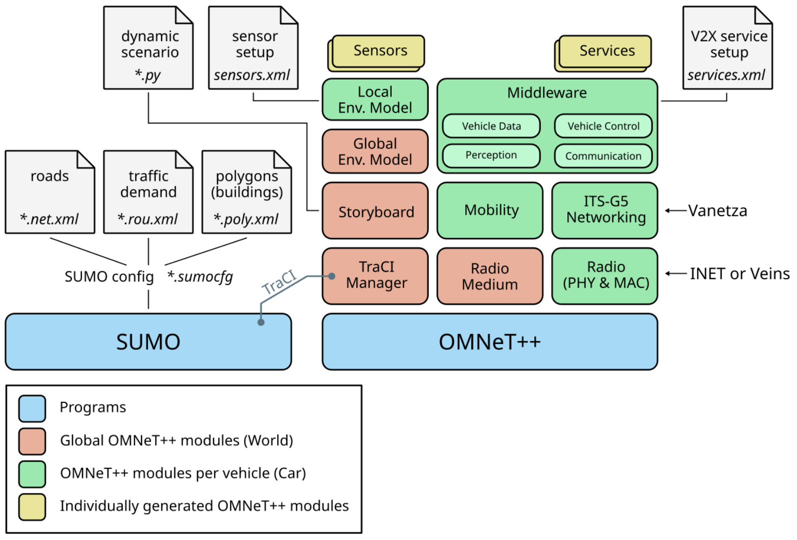


Figure 2.5: Artery Simulation Framework

Usually, two programs are running hand in hand when a simulation is running. On the left hand, there is the traffic simulator SUMO, and on the right-hand, we have the OMNeT++ runtime environment. The interaction of 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++.

In this project OMNeT++ was used for the simulation and evaluation of CA basic service. It calculates the propagation of the packets according to the parameters specified in the configuration files, and also, with the help of other tools, it computes the packets received rate in different scenarios.

2.2.3 CARLA

Car Learning to Act (CARLA) is an open simulator for urban driving. CARLA has been developed from the ground up to support training, prototyping, and validation of autonomous driving models, including both perception and control. The simulation platform supports flexible setup of sensor suites and provides signals that can be used to train driving strategies, such as GPS coordinates, speed, acceleration, and detailed data on collisions and other infractions. A wide range of environmental conditions can be specified, including weather and time of day[8]. An example of CARLA simulator is shown in Figure 2.6.

Figure 2.6: Carla Simulator

Chapter 3

Simulator Enhancement

This chapter provides an explanation of the usage of the simulators for the acquisition of statistics, including details extending its functionality and using tools for data analysis and databases manipulation.

This chapter starts with the development of the SUMO scenarios, followed by the parameters for the OMNeT++ to characterize the simulations to extract valid data. And ending with the collection of statistics with tools like SQLite and Pandas.

3.1 Sumo Scenarios

For the performance of the simulations of this project two kinds of road topologies have been used and developed with the SUMO simulator: highway and a Manhattan grid.

3.1.1 Sumo Files

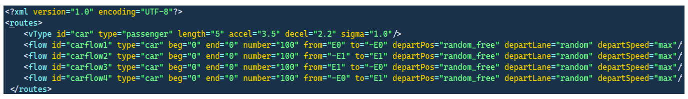
Each scenario was created using three SUMO files, which are located in *artery/scenarios/i2cat2*.

Each scenario has been created using different set of tools. For the Highway scenario network file netedit has been used. The routes file has been created manually adding as many cars as needed for the simulation. Besides, the basic files for a simulation an additional file has been used. The additional-file reroutes the vehicles routes making them drive in circles throw the highway and never exiting the simulation. This way the density of vehicles in the simulation can be controlled.

For the Manhattan grid scenario, the Network file has been created automatically with the usage of netgenerate. This tool allows to specify the grid format and the number of cells for the grid scenario. Once the network file is generated, it can be used to generate the routes file with the tool jtrouter, being able to specify the number of vehicles in the simulations and the turning ratios that were set to 25-50-25. Moreover, and additional file has been added for the Manhattan grid scenario adding polygons/buildings for every cell.

The configuration of all the simulations has been done for the vehicles to enter in the simulation as soon as possible, making them depart from any free position in the simulation, at any lane, with the maximum speed allowed, as can be seen in

Figure 3.1. This way the warm-up period from the simulation is very low, with the densest scenarios taking at most 10 seconds. The measures are valid after this period because the vehicles are already at the desired conditions for the simulation. This was a critical issue taking into account the time taken to throw a simulation.



```
<?xml version="1.0" encoding="UTF-8"?>
<routes>
    <vType id="car" type="passenger" length="5" accel="3.5" decel="2.2" sigma="1.0"/>
    <flow id="carflow1" type="car" beg="0" end="0" number="100" from="E0" to="-E0" departPos="random_free" departLane="random" departSpeed="max"/
    <flow id="carflow2" type="car" beg="0" end="0" number="100" from="-E1" to="E1" departPos="random_free" departLane="random" departSpeed="max"/
    <flow id="carflow3" type="car" beg="0" end="0" number="100" from="E1" to="-E0" departPos="random_free" departLane="random" departSpeed="max"/
    <flow id="carflow4" type="car" beg="0" end="0" number="100" from="-E0" to="E1" departPos="random_free" departLane="random" departSpeed="max"/
 </routes>
```

Figure 3.1: Highway routes file 400 vehicles

3.1.2 Physical topologies

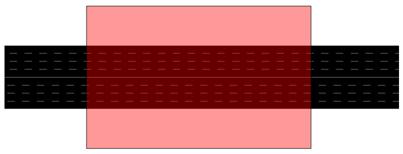
Different road topologies were used in this project. One of which was the highway scenario, which simulate direct line-of-sight (LOS) conditions and non-stop driving. The other one was the Manhattan grid scenario, which help in understanding the effects of walls and buildings, as well as intersections. Moreover, the project defined a statistical region in the scenarios, highlighted in red below, as shown in Figure 3.2. 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.

Highway Scenario

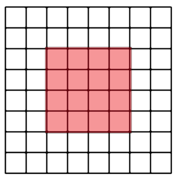
The Highway scenario measured 2000m x 25m, with the statistical area bounded by $500m < x < 1500m$, which corresponds to the central region of the highway. Each direction had 4 lanes for a total of 8 lanes. The end of one circulatory direction is connected with the beginning of the other direction, making a loop for the vehicles loaded in the simulation. This way the density of vehicles in the simulation is controlled.

Manhattan Grid Scenario

The Manhattan grid scenario measured 800m x 800m, with the statistical area bounded by $200m < x < 600m$ and $200m < y < 600m$, which corresponds to the central region of the grid. The presence of walls is configured in the file *walls.xml* and imported into the *omnetpp.ini*.



(a) Highway Scenario



(b) Manhattan grid scenario

Figure 3.2: Simulation scenarios

3.2 Simulation Parameters

Table 3.1 provides a summary of the parameters used in the simulations:

Category	Parameter	Value
Node	Operation mode	802.11p
	Carrier frequency	5.9 GHz
	Bandwidth	10 MHz
	Channel number	180
	Modulation	QPSK
	Bitrate	6 Mbps
	Transmitter power	200 mW
	Receiver sensitivity	-85 dBm
	Energy detection	-85 dBm
	SNIR threshold	4 dB
Medium	Obstacle's loss type	DialectricObstacleLoss
	Path loss type	{Nakagami Fading, Rayleigh Fading}
	Path loss factor	3
	Background noise type	IsotropicScalarbackgoundNoise
	Background noise power	-110 dBm
Scenario	Topology	{Bidirectional Highway, Manhattan grid}
	Maximum vehicle speed	33.33 m/s
	CAM message period	100 – 1000 ms
	CAM message length	250 - 500 - 750 - 1000 bytes
	Simulation time limit	200 - 300 s
	Density of vehicles	{100-200-300-400}
	Warm-up period	0-10 s

Table 3.1: Simulation parameter values

The parameter values, used to model the individual nodes, were selected based in those specified in the standards. A control channel (CCH) was used, with 10 MHz of bandwidth cantered 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 200mW or 23 dBm, which was below the 33 dBm power limit. The receiver sensitivity was set to -85 dBm, with reference to [9] tables.

The road topologies were the bidirectional highway scenario and Manhattan grid scenario. Depending on the scenario the warm-up period and the simulation time differed. With the simulations the density of vehicles varies for low density (e.g., 100 vehicles) with high density (e.g., 400 vehicles). For the Highway scenario the radio medium was modelled using the Nakagami fading profile, adapting the shape factor m with equation 3.1 which is a good approximation of the signal fading in highway scenarios [10]. The corresponding path loss factor value for highway areas ranged between 2 and 3.5, from which a value of 3 was arbitrarily chosen to be used in the simulations.

$$m(d) = 2.7 \cdot e^{-0.01(d-1.0)} + 1.0 \tag{3.1}$$

Moreover, the different density of vehicles in the highway scenarios has been chosen according to different levels of service of a motorway. From a low level of service A (100 vehicles) to high level of service E (400 vehicles) in the simulation at the same time [11].

Level of Service	Maximum Density (pc/km/lane)
A	7
B	11
C	16
D	22
E	28
F	>28

Table 3.2: Highway Level of Service

In Table 3.2 the density of vehicles is characterized with the pc/km/lane, where *pc* is passenger cars, per kilometer, and per *lane*. The low levels of service (A , B) describes a free flow operations, vehicles are almost completely unimpeded in their ability to maneuver. Increasing the Level of service the vehicles have more constraints and less freedom to maneuver, until they reach the last level of services (E, F), where the vehicles don't have usable gaps in the traffic stream, and they can not drive at maximum speed.

For the Manhattan grid scenario, 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 path loss factor value 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 the walls/buildings in the scenario. The properties of the walls could be configured in the *walls.xml* file and how the walls behave can be adjusted with the DialectricObstacleLoss or IdealObstacleLoss parameter in the *omnetpp.ini*. In the simulations the DialectricObstaclesLoss was used.

The performance of DCC was out of scope of this project. The simulation has been set with the DCC disabled in *omentpp.ini*. The other parameter tested in this project is the length of CAMs. To do so, the length of the message vary between 250 and 1000 bytes.

3.3 Statistical Recording

In order to make statistics form the data recorded with the simulations, tools for data analysis were used: SQLite to manipulate the database, Python Pandas to compute some values needed for the histograms and Matplotlib to represent all the graphics correctly.

After a simulation, OMNeT++ stores all the results of the simulations in output files in form of scalar results and vector results. To analyse these results OMNeT++ has an IDE able to represent this results. However, the statistics desired for this project are not easily made with it. In order to make the desired statistics with only OMNeT++ a modification in the source code of the simulator would have been needed. Moreover, for computing another statistic for the same simulation, the simulation would have to be set and thrown again.

Instead of only working with OMNeT++ other tools were used to manipulate the data stored in the output-files of the simulation. The data was organized in new tables and stored in a database. This way, the data was correctly organized for computing statistics and it can be used as many times as needed.

3.3.1 SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite reads and writes directly to ordinary disk files, and a complete database is stored in a single disk file [12].

OMNeT++ allows to store the data in an SQLite file disk format. With the data stored from the simulation the desired statistics could not be made for the need of some parameters, for this reason, SQLite is used. With SQLite data from the simulation can be queried and stored in a new table with all the needed parameters to compute the statistics. Moreover, once the data is stored in the new table it can be queried filtering any parameters, which allows us, for example to query only the messages received in the center of the scenarios.

For example, as can be seen in Figure 3.3, a table to compute the Packet Error Rate in function of the distance can be obtained.

3.3.2 Pandas

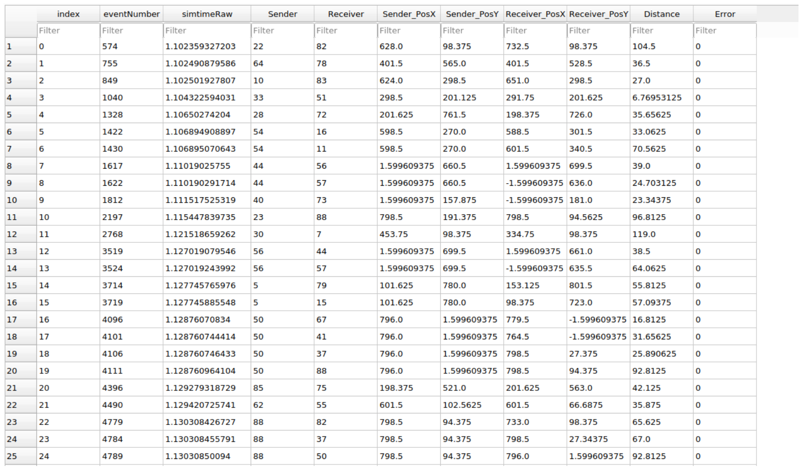
Pandas is an open-source Python package that is most widely used for data science/-data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem, and is typically included in every Python distribution [13].

Moreover, Pandas makes simple to do many of the time consuming, repetitive tasks associated with working with data. Including: data cleaning, data fill, data normalization, merges and joins, data visualizations, statistical analysis, data inspection, etc.

Pandas is a useful tool for this project, it enables to fill the tables of the databases with the missing data, and to compute the desired statistics with an easy and efficient way, and at the same time it allows to represent and visualize the desired statistics. Moreover, Pandas supports the integration with many file formats or data sources, and one of them is SQLite, which were used in the project.

For plotting the data Pandas uses the power of Matplotlib, a powerful python library for creating static, animated and interactive visualization. Matplotlib is a cross-platform, data visualization and graphical plotting library for python and its numerical extension Numpy. It offers a viable open-source alternative to MATrix LABoratory (MATLAB).

With the usage of this tools and with the data provided form OMNeT++: the position of all the vehicles every 100 ms, the time of the received packets as well as the source and destination vehicles, a linear interpolation to calculate the actual position in the moment of receiving the packet was performed with Pandas. Having the new data computed, the distance between cars was calculated for the statistic results.



	index	eventNumber	simtimeRaw	Sender	Receiver	Sender_PosX	Sender_PosY	Receiver_PosX	Receiver_PosY	Distance	Error
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	0	574	1.102359327203	22	82	628.0	98.375	732.5	98.375	104.5	0
2	1	755	1.102490879586	64	78	401.5	565.0	401.5	528.5	36.5	0
3	2	849	1.102501927807	10	83	624.0	298.5	651.0	298.5	27.0	0
4	3	1040	1.104322594031	33	51	298.5	201.125	291.75	201.625	6.76953125	0
5	4	1328	1.10650274204	28	72	201.625	761.5	198.375	726.0	35.65625	0
6	5	1422	1.106894908897	54	16	598.5	270.0	588.5	301.5	33.0625	0
7	6	1430	1.106895070643	54	11	598.5	270.0	601.5	340.5	70.5625	0
8	7	1617	1.11019025755	44	56	1.599609375	660.5	1.599609375	699.5	39.0	0
9	8	1622	1.110190291714	44	57	1.599609375	660.5	-1.599609375	636.0	24.703125	0
10	9	1812	1.111517525319	40	73	1.599609375	157.875	-1.599609375	181.0	23.34375	0
11	10	2197	1.115447839735	23	88	798.5	191.375	798.5	94.5625	96.8125	0
12	11	2768	1.121518659262	30	7	453.75	98.375	334.75	98.375	119.0	0
13	12	3519	1.127019079546	56	44	1.599609375	699.5	1.599609375	661.0	38.5	0
14	13	3524	1.127019243992	56	57	1.599609375	699.5	-1.599609375	635.5	64.0625	0
15	14	3714	1.127745765976	5	79	101.625	780.0	153.125	801.5	55.8125	0
16	15	3719	1.127745885548	5	15	101.625	780.0	98.375	723.0	57.09375	0
17	16	4096	1.12876070834	50	67	796.0	1.599609375	779.5	-1.599609375	16.8125	0
18	17	4101	1.128760744414	50	41	796.0	1.599609375	764.5	-1.599609375	31.65625	0
19	18	4106	1.128760746433	50	37	796.0	1.599609375	798.5	27.375	25.890625	0
20	19	4111	1.128760964104	50	88	796.0	1.599609375	798.5	94.375	92.8125	0
21	20	4396	1.129279318729	85	75	198.375	521.0	201.625	563.0	42.125	0
22	21	4490	1.129420725741	62	55	601.5	102.5625	601.5	66.6875	35.875	0
23	22	4779	1.130308426727	88	82	798.5	94.375	733.0	98.375	65.625	0
24	23	4784	1.130308455791	88	37	798.5	94.375	798.5	27.34375	67.0	0
25	24	4789	1.13030850094	88	50	798.5	94.375	796.0	1.599609375	92.8125	0

Figure 3.3: Packets Error Rate table with SQLite

Chapter 4

EVI Ego-vehicle Interface

This chapter provides an overview of the communication flow between simulators via the TraCI protocol. Depicting the communication between OMNeT++, CARLA and SUMO simulators.

Moreover, this chapter presents a design and development of an Ego-vehicle Interface (EVI).

4.1 Connection between simulators (TraCI)

Traffic Control Interface (TraCI) gives access to a running road traffic simulation allowing to retrieve values of simulated objects and to manipulate their behaviour on-line.

This tool, provided from SUMO, is used for the other simulators: OMNeT++ and CARLA. They use it to communicate with SUMO and send information as the position, velocity or heading angle of the vehicles. When the three simulators are working together, SUMO is the one controlling and synchronizing the three simulators via the TraCI Protocol.

4.1.1 TraCI Protocol

TraCI uses a TCP based client/server architecture to provide access to SUMO. Thereby, SUMO acts as a server that is started with additional command-line options: –remote-port $< INT >$ where $< INT >$ is the specified port where SUMO is listening on for incoming connections [14].

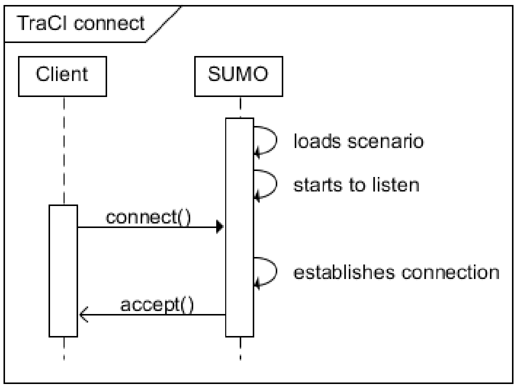
When started with the –remote-port option, SUMO only prepares the simulation and walls and waits for all the external applications to connect and take over the control as shown in figure 4.1a.

After starting SUMO, clients connect to it by setting up a TCP connection to the appointed SUMO port. TraCI supports multiple clients and executes all commands of a client in a sequence until it issues the *Simulation Step* command.

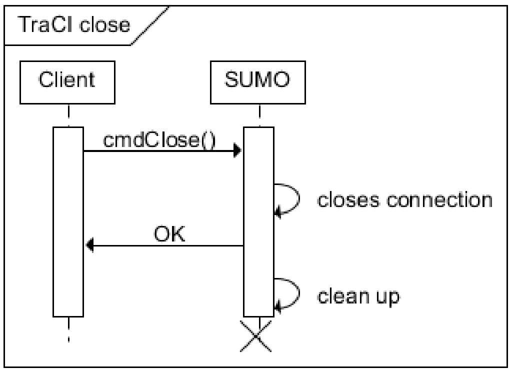
The client applications send commands to SUMO to control the simulation run, to influence single vehicle's behavior or to ask for environmental details. SUMO answers with a *Status-response* and additional results depending on the given command.

The client has to trigger each simulation step in SUMO using the *Simulation step* command. The simulation will advance to the next step once all clients have sent the *Simulation Step* command.

The client is responsible for shutting down the connection using the *Close* command. When all the clients issued a *Close* command, the simulation will end, freeing all resources as shown in figure 4.1b.



(a) establishing connection



(b) closing connection

Figure 4.1: TraCI SUMO connection

A TCP message acts a as a container for a list of commands or results. Therefore, each TCP message consists of a small header, that gives the overall message size, and a set of commands that are put behind it. The length and identifier of each command is placed in front of the command as shown in Table 4.1.

0 7	8 15
Message Length Including this header	
(Message Length, continued)	
Length	Identifier
Command_0 content	
...	
Length	Identifier
Command_n-1 content	

Table 4.1: TCP message container

4.1.2 Multiple simulator connection

TraCI allows multiple clients connections. One SUMO simulation can be accessed and modified by multiple clients. In the case of this project, a SUMO simulation is shared at the same time with OMNeT++ and CARLA.

The number of clients which can connect is given as an additional option –num-clients for a SUMO simulation. In order to have a predefined execution order, every client should issue a *Set Order* command before the first *Simulation Step* command. The *Set Order* command assigns a number to the client, the lower number having the preference.

The way that TraCI handles the multiple clients commands is one at a time. The client with the lowest number in the *Set Order* list is the one that TraCI handles first. Once the first client has sent all the commands until the *Simulation Step* command, TraCI is only responding to its commands. When TraCI receives the *Simulation Step* command it handles the next client of the *Set Order* list, until it sends the simulation step command. It works this way until clients close the connection. In figure 4.2 a chart depicting the communication flow between the three simulators is shown. This chart was created using the information of packets captured with Wireshark.

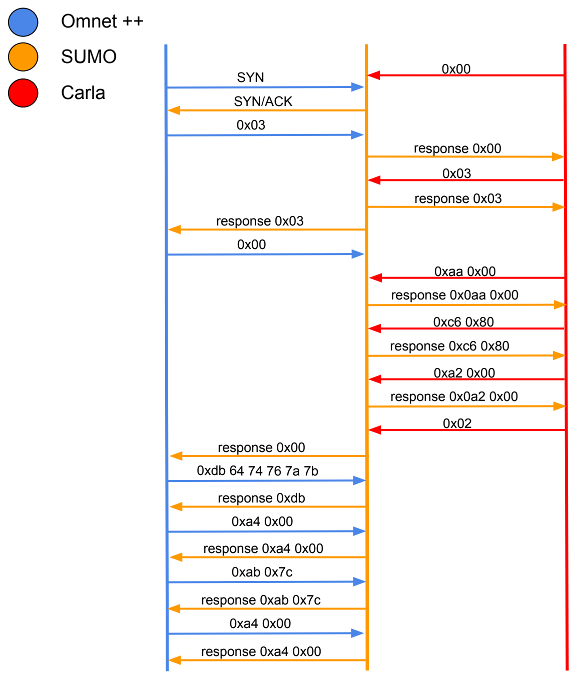


Figure 4.2: OMNeT++, CARLA and SUMO communication flow

A more accurate analysis of how TraCI works with a multiple client connection can be seen in Appendix A.

4.2 EVI

One of the purposes of this project was to develop an Ego-Vehilce interface (EVI). Being able to pass the information received in OMNeT++, of V2X simulations, to the Ego-Vehicle allocated in CARLA. For this thesis, the complete development of the EVI is out of scope. The final development will be performed with work hours in i2CAT.

The first approach to design the EVI was to design and interceptor between the simulators, controlling the flow of messages passed between them.

Subsequently, and architecture design is explained and an overview of the desired output is shown.

4.2.1 Interceptor

The interceptor was aimed to intercept the communication between three simulators: OMNeT++, CARLA and SUMO. OMNeT++ and CARLA communicate with SUMO via TraCI, as seen previously in 4.1.2. SUMO controls the communication flow, via TraCI protocol depending on the *Set Order* command. An schema of the architecture of the interceptor is depicted in Figure 4.3.

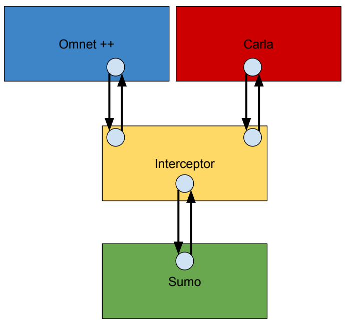


Figure 4.3: Schema of three simulator interceptor

The development of this interceptor has been done using Python, with the help of multi-threading. One thread for each of the communications channels: two for OMNeT++ and SUMO and two more for CARLA and SUMO. The synchronization of the threads has been done with events.

Moreover, a new class, Globals, has been created to store the global variables and assure the correct access to these variables. This class stores the global variables and provides the getters and setters to modify these safely.

With this interceptor the communication between the simulators was captured and with it, the first step to develop the EVI was accomplished. The code can be seen in Appendix C.

4.2.2 EVI architecture

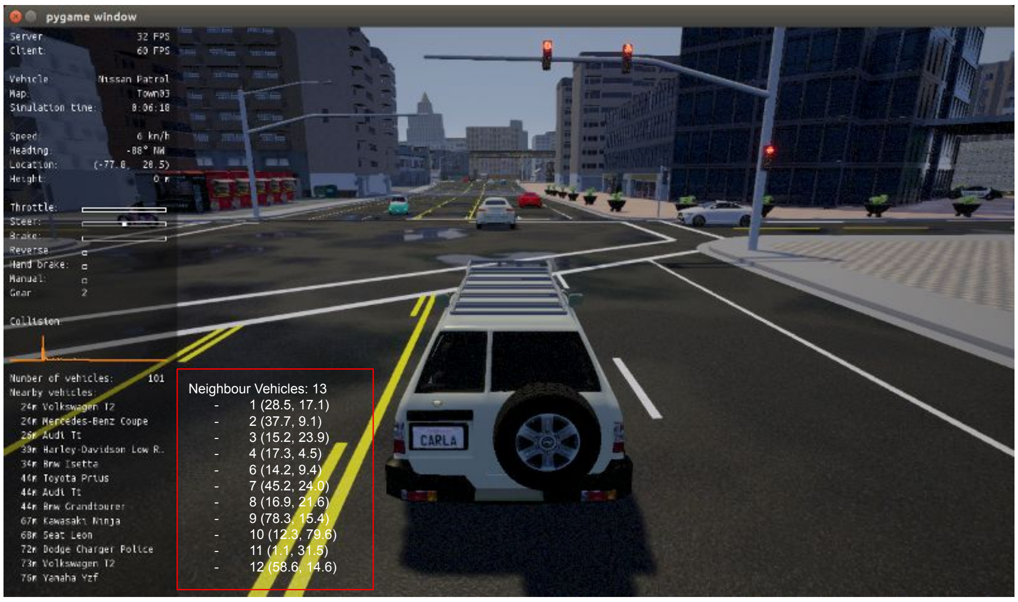
The following step in the development of this project is designing the EVI architecture. Based on the Interceptor approach explained before, an enchantment to communicate OMNeT++ and CARLA is needed. The project aims for OMNeT++ to feed CARLA with information received form the V2X protocols. Only a one-direct communication flow is needed from OMNeT++ to CARLA.

On one hand, the OMNeT++ simulator needs to be able to identify the EGO vehicle that CARLA controls. Once the vehilce has been identified, every time it receives a packet correctly in OMNeT++ it needs to pass the information to CARLA simulator using the interceptor.

On the other hand, CARLA simulator receives the information passed by the interceptor and deals with it as any other sensor in CARLA. This way the EGO vehicle has the information provided by V2X protocols. For example, the positions broadcasted by the CAMs messages of the neighbour vehicles.

Taking this approach, the information fed to CARLA can be controlled to optimize the simulators performance, and the synchronization between them.

The development of this architecture will be performed in the future work hours in i2CAT. It is still in process by the time to deliver this document. The desired output of this architecture can be seen in figure 4.4.



```
Server:                32 FPS
Client:                60 FPS

Vehicle:         Nissan Patrol
Map:                    Town03
Simulation time:       0:06:10

Speed:                  6 km/h
Heading:               -88° NW
Location:        (-77.0,  28.5)
Height:                    0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:       2

Collision:

Number of vehicles:        101
Nearby vehicles:
  24m Volkswagen T2
  24m Mercedes-Benz Coupe
  26m Audi Tt
  30m Harley-Davidson Low R.
  34m Bmw Isetta
  44m Toyota Prius
  44m Audi Tt
  44m Bmw Grandtourer
  67m Kawasaki Ninja
  68m Seat Leon
  72m Dodge Charger Police
  73m Volkswagen T2
  76m Yamaha Yzf
```

```
Neighbour Vehicles: 13
    -     1 (28.5, 17.1)
    -     2 (37.7, 9.1)
    -     3 (15.2, 23.9)
    -     4 (17.3, 4.5)
    -     6 (14.2, 9.4)
    -     7 (45.2, 24.0)
    -     8 (16.9, 21.6)
    -     9 (78.3, 15.4)
    -     10 (12.3, 79.6)
    -     11 (1.1, 31.5)
    -     12 (58.6, 14.6)
```

Figure 4.4: Desired output in Carla simulator

A real time position of the neighbour vehicles can be seen for the EGO vehicle thanks to the CAM's received correctly.

Chapter 5

Results

This section presents the results for the experimental evaluation of CA basic service using the modified IEEE 802.11p simulator. Its performance is studied by varying different parameters such as vehicle density, packet length. The plots were made according to the Tx-Rx distance, defined as the euclidean distance between two communicating nodes. In the case of the Manhattan grid scenario, the propagation model applied for the presence of buildings is the Dielectric Obstacle Loss, which attenuates the signal as it passes through the building. All the graphics shown in this chapter and others can be seen in Appendix B.

5.1 Simulated scenarios

This section compares a Highway scenario with Manhattan grid scenario. Different simulations, varying the number of vehicles (100-200-300-400) in each scenario, have been run. Nevertheless, in order to compare both scenarios, the road surface should be taken into account. The highway scenario has a surface of $48000m^2$ and the Manhattan grid scenario has a surface of $83458m^2$. Taking this into account, the density of vehicles in both scenarios can be computed, as shown in Table 5.1.

Number of Vehilces	Density Highway scenario (pc/km/lane)	Density Manhattan grid scenario (pc/km/lane)
100	6.25	3.6
200	12.5	7.2
300	18.75	10.8
400	25	14.4

Table 5.1: Density of vehicles

5.2 Packet Error Rate

Before explaining how the Packet Error Rate (PER) is computed, an explanation of how OMNeT++ computes the received packets is needed. OMNeT++ computes the received packets in the following way: first, it checks if the power of the packet is above the receiver sensitivity, discarding all the packets below. Afterward, it considers as erroneous all the packets which didn't pass the Signal to Noise Ratio (SNR) threshold, it does it in a deterministic way. For the last step, it computes the probability of receiving the packet correctly, depending on the values of the Signal to Noise and Interference Ratio (SNIR). The packets are considered to be received when they have passed the sensitivity threshold. After this, they can be an erroneous one or a correct one depending on SNR and SNIR values.

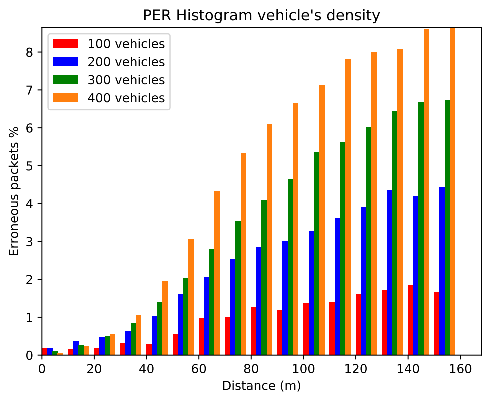
Packet Error Rate (PER) is used to test the performance of an access terminal's receiver. PER is the ratio, in percent, of the number of packets not successfully received by the access terminal. The resulting PERs of the different scenarios are used to better understand the behavior of the CA basic service. In this project the PER is defined as follows:

$$PER = \frac{Erroneous\, received\, packets}{Total\, received\, packets} \cdot 100 \tag{5.1}$$

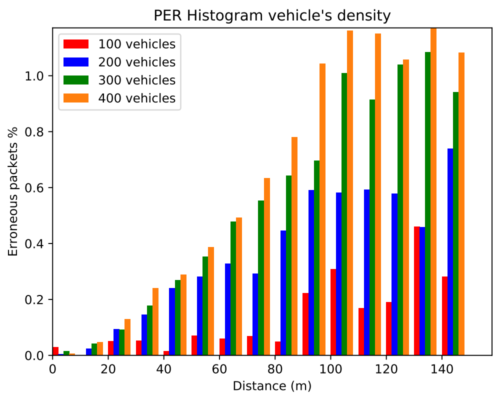
A packet is considered to be correctly received if it successfully arrived at the MAC layer. The Total packets in the denominator of Equation 5.1, refer to the packets which passed the sensitivity threshold of the receiver, and the Erroneous received packets refer to the ones which didn't pass SNR and SNIR thresholds.

5.2.1 Effect of Vehicle Density

The length of the highway was fixed to be 2 km, and as such, increasing the number of vehicles makes them closer together. This resulting in an increase in interference, packet collisions, and more CAM's being lost, which consequently increases the PER, as can be observed in 5.1a. It can be observed an increase in the PER when the vehicles in the simulation or the distance between vehicles increases because more packet collisions can occur.



(a) Highway PER Histogram



(b) Manhattan PER Histogram

Figure 5.1: PER Histograms, with variation of vehicles density. CAM's length: 250 bytes. Simulation time: 200 s (Highway), 300 s (Manhattan).

The same effect can be seen in the Manhattan grid scenario in Figure 5.1b. As more vehicles are loaded in the simulation the PER increases, but not as clean and regular as the Highway scenario, due to the presence of buildings and intersections, which are located every 100 m in the street.

In order to observe this effect with more detail, Figures 5.2 and 5.3a can be observed. In these charts, it can be observed that the packets received in the Manhattan scenario don't decrease fluidly with the distance. In the distance of 100 m, it holds receiving the same packets as the anterior distances and then, at 110 m, it has a drop of packets received.

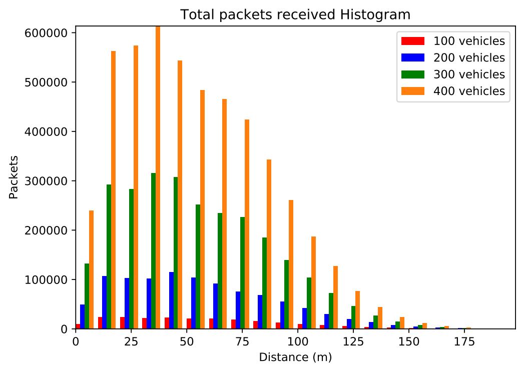
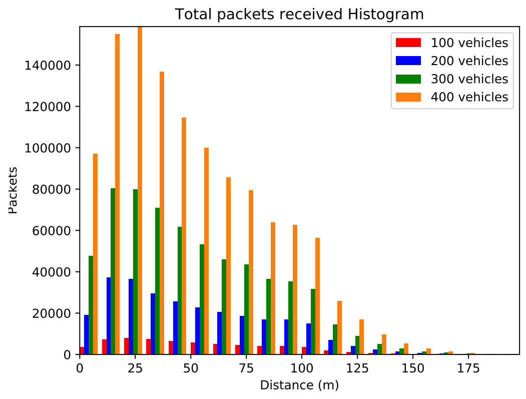
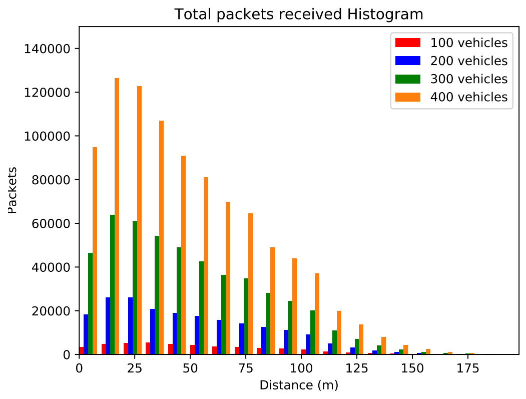


Figure 5.2: Highway Packets Received Histogram, with variation of vehicles density. CAM's length: 250 bytes. Simulation time: 200s.



(a) Manhattan total packets received



(b) Manhattan LOS packets received

Figure 5.3: Manhattan Total and LOS packets received Histogram CAM's length: 250 bytes. Simulation time: 300 s.

Besides this effect, it's observable that the messages received decrease as the distance between cars increases, and the number of messages received is higher if more vehicles are in the simulation, as more nodes are broadcasting CAM's messages. Nonetheless, having the same number of vehicles in the Manhattan and Highway scenarios, the number of packets, which passed the sensitivity of the receiver, is much lower in the Manhattan grid scenario, although the simulation time is higher (300s instead of 200s). This fact is because of the presence of buildings, which attenuates the signal in the Manhattan grid scenario.

As noticed before, in the Manhattan grid scenario, effects of the buildings and intersections can be observed in every chart. The buildings attenuate the signal, discarding the majority of packets for not reaching the sensitivity threshold of the receiver. However, some packets, which pass throw buildings, can reach it , as can be seen comparing Figures 5.3a and 5.3b. These charts are from the same simulation results: one with all the packets received, and the other filtering only the packets received in Line of Sight (LOS). The graphics follow the same pattern having the chart with all the packets received an offset of the packets received in Non Line of Sight (NLOS).

However, another effect can be noticed. The presence of intersections in the Manhattan grid scenario triggers an effect of having some peaks in the 100 m distance: more packets received (shown in Figure 5.3a) and higher PER (shown in Figure 5.1b). When a vehicle is situated in an intersection, it receives more packets in LOS from every distance. Concretely, in the distance of 100 m, the vehicle receives messages from vehicles which are not in LOS and are close to an intersection, as shown in Figure 5.4. These packets use to have bad transmitting conditions, as if they were at 140 m, because of being behind a building. This effect disappears when the distance increases, because the vehicles behind the building have a higher attenuation, therefore they don't reach the sensitivity threshold of the receiver.

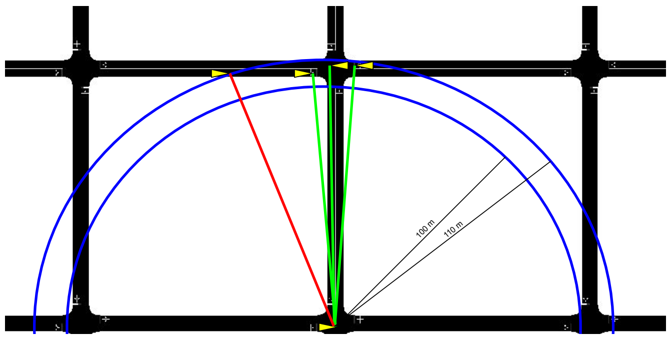


Figure 5.4: 100 m Intersection effect

With the filter of the LOS, the effect of the intersection disappears as well. Because all the packets received from behind the buildings are not considered.

5.2.2 Effect of CAM's length

The length of the CAM's message has a significant impact on the resulting PER. Given a higher length for the CAM the time to have an interference is higher as well, resulting in an increase in packet collisions, which consequently causes an increase in erroneous packets as can be seen in Figure 5.5. It can also be observed an increment in the PER as the distance increases.

For the Manhattan grid scenario, the length of the messages affects the same way, the longest the packet is more erroneous packets appear, as shown in Figure 5.6a. Although, it differs from the Highway scenario because the graphic does not increase uniformly. This is due to the presence of intersections and buildings, which generates the same effects as before.

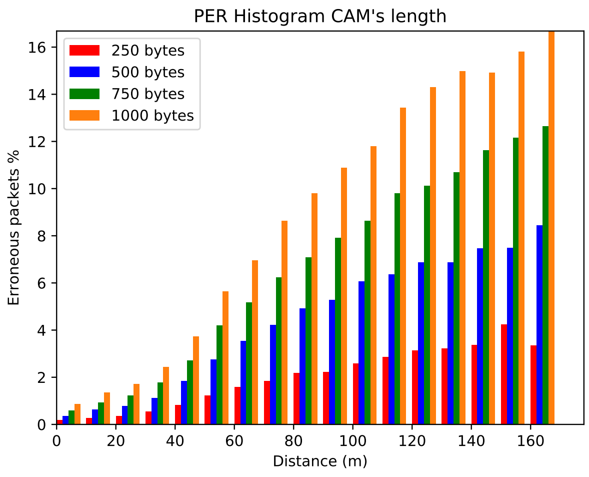
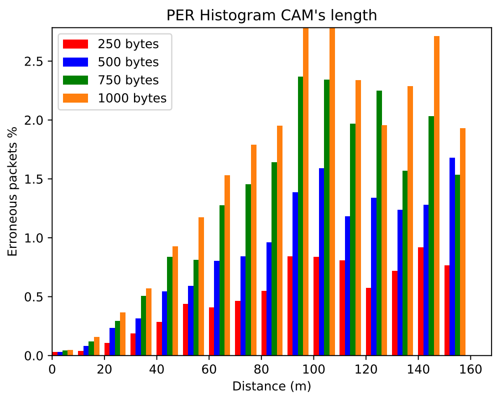
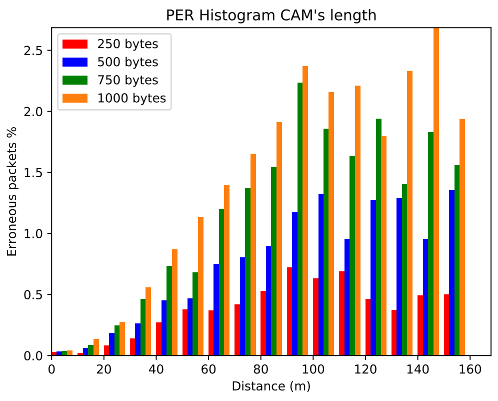


Figure 5.5: Highway PER Histogram, with variation of CAM's length. Total vehicles: 200. Simulation time: 200 s. Density of vehicles: 12.5 pc/km/lane.



(a) Manhattan PER



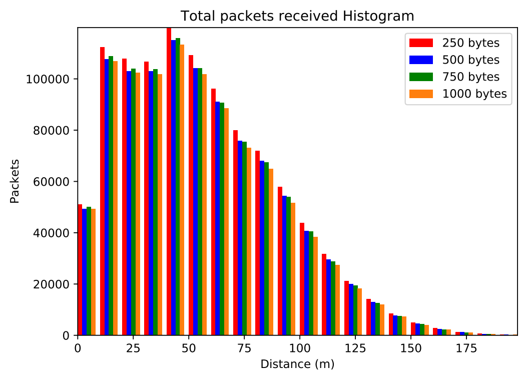
(b) Manhattan LOS PER

Figure 5.6: Manhattan PER Histograms, with variation of CAM's length. Total vehicles: 200. Simulation time: 200 s. Density of vehicles: 7.2 pc/km/lane.

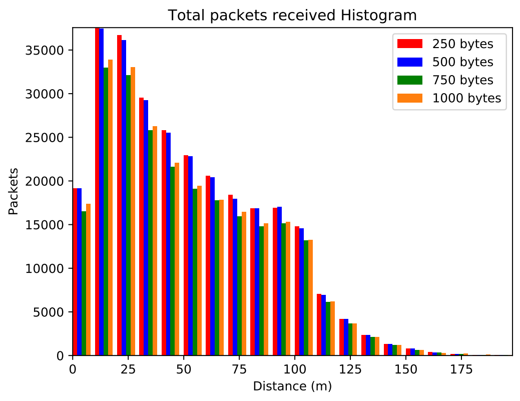
Observing the PER Histograms in Figures 5.6a and 5.6b, it is shown that the peaks of PER in the 100 m distance are reduced when the vehicles behind the buildings are filtered. However, due to the width of intersections a minimum effect of irregularity in the chart is still present, is not as uniform as the Highway scenario yet.

The number of packets that passed the sensitivity threshold of the receivers is really similar independently of the length of the CAM's as can be seen in Figure 5.7a. The first 10 m have fewer packets received, as the vehicle size and the security distance between vehicles takes the first meters of space, resulting in fewer cars close to the receiver. The range where more packets passed sensitivity is between 10 m and 70 m. Afterwards, it decreases exponentially with the distance according to the propagation effects.

In the Manhattan grid scenario, the same effect happens, but because of the presence of walls and intersections the number of packets is much lower and there is a discontinuous jump at 100 m distance, as shown in Figure 5.7b. It occurs form the same effect explained in section 5.2.1.



(a) Highway scenario



(b) Manhattan grid scenario

Figure 5.7: Total Packets received CAM length varying
Total vehicles: 200. Simulation time: 200 s (Highway), 300 s (Manhattan). Density of vehicles: 12.5 pc/km/lane (Highway), 16.9 pc/km/lane.

Chapter 6

Conclusions and future work

An existing IEEE 802.11p-based simulator was modified to perform simulations in different scenarios, varying critical parameters as vehicle density or packet lengths in order to evaluate the CA basic service.

The presence of more vehicles resulted in higher interference and collision probability as more nodes attempt to access the channel to send their periodic CAMs, ultimately increasing PER values. The packets arrived up to 175 m of distance with the simulation parameters set in Section 3.2. As more distance between vehicles higher the PER values and fewer packets passed the sensitivity threshold. In the Manhattan grid scenario, the presence of intersections and buildings was reflected, with fewer packets received being the majority in LOS, and the appearance of peaks in the 100 m distance corresponding to the distance between intersections.

The length of the CAMs affected significantly the PER values, due to the higher interference generated for each packet and therefore the collision probability incremented. As longer the CAM higher the PER.

A design and first implementation of an EVI was successfully developed. Starting with an analysis with Wireshark of the TraCI protocol which interconnects the simulators, and following with the implementation of an interceptor, which captures the messages passed between simulators. Finishing with a design of the following steps to follow in the implementation of the EVI.

There are several recommendations for continuing with this research study. For one, continue with the development of the EVI, which will be performed in the following months.

In the case of the system simulators, other road topologies could be tested, since the project only dealt with highway and grid scenarios. For instance, SUMO allows importing real-world maps. Or testing new metrics with OMNeT++ simulator, as Neighborhood Awareness Ratio.

Appendices

Appendix A

TraCI analysis with Wireshark

A.1 OMNeT++ and SUMO

A.1.1 Command List

Command	Description
0x02	Simulation step
0xab	Simulation value retrieval
0xab 0x7c	Network bounding box
0xab 0x7b	delta T, length of simulation step
0xab 0x82	Position conversion
0xab 0x7d	Number of vehicles in the network, plus the ones waiting to start.
0xa4	Vehicle value retrieval
0xa4 0x00	Id list, list of vehicles in the simulation
0xa4 0x4f	Type id
0xa4 0x40	Speed
0xa4 0x42	Position
0xa4 0x43	Angle
0xa4 0x49	vClass, permissions of the class
0xd4	Subscription vehicle value
0xd4 0x40	Subscribe speed
0xd4 0x42	Subscribe position
0xd4 0x43	Subscribe angle
0xdb	Subscription simulation value
0xdb 0x66	Current simulation time
0xdb 0x74	Ids of departed vehicles
0xdb 0x76	Ids of vehicles starting to teleport
0xdb 0x7a	Ids of arrived vehicles
0xdb 0x7b	Delta T, returns the length of the simulation step

Table A.1: OMNeT++ and SUMO commands

A.1.2 Communication flow

The communication flow commands between the two simulators was the following:

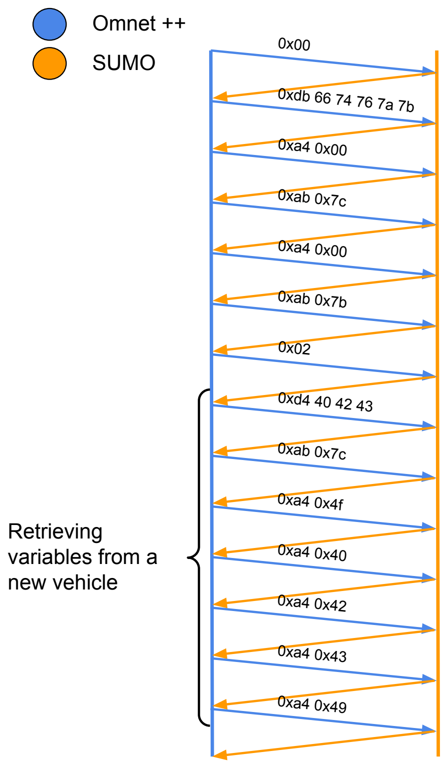


Figure A.1: OMNeT++ and SUMO communication flow 1

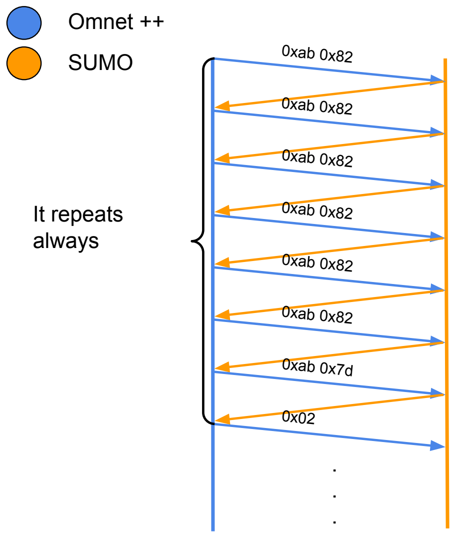


Figure A.2: OMNeT++ and SUMO communication flow 2

The figures show the command flow of the simulators, as we can see when a new vehicle enters the simulation a list of commands is passed every time. This commands consists in subscribing the new vehicle in order to retrieve the desired values in every simulation step and after that, acquire values that only are need once, (e.g., the vehicle type, vehicle permissions).

A.2 OMNeT++ CARLA and SUMO

A.2.1 Command List

Command	Description
0x02	Simulation step
0x03	Set Order
0xaa	Edge value retrieval
0xaa 0x00	Id list, list of edges in the simulation
0xab	Simulation value retrieval
0xab 0x7c	Network bounding box
0xab 0x7b	Delta T, length of Simulation step
0xab 0x82	Position conversion
0xab 0x7d	Number of vehicles in the network, plus the ones waiting to start.
0xa2	Get traffic lights variable
0xa2 0x00	Id list, returns a list of all traffic light id's in the scenario
0xa4	Vehicle value retrieval
0xa4 0x00	Id list, list of vehicles in the simulation
0xa4 0x4f	Type id
0xa4 0x40	Speed
0xa4 0x42	Position
0xa4 0x43	Angle
0xa4 0x49	vClass, permissions of the class
0xc6	Change route state
0xc6 0x80	Add a new route
0xd4	Subscription vehicle value
0xd4 0x32	Lateral speed
0xd4 0x36	Slope in degrees
0xd4 0x39	Position 3D
0xd4 0x40	Subscribe speed
0xd4 0x42	Subscribe position
0xd4 0x43	Subscribe angle
0xd4 0x44	Length of the vehicle
0xd4 0x45	Vehicle color
0xd4 0x49	vClass, permissions class of this vehicle
0xd4 0x4d	Width of the vehicle
0xd4 0x4f	Type of vehicle id
0xd4 0x5b	Signal states
0xd4 0xbc	Height of the vehicle

Command	Description
0xdb	Subscription simulation value
0xdb 0x66	Current simulation time
0xdb 0x74	Ids of departed vehicles
0xdb 0x76	Ids of vehicles starting to teleport
0xdb 0x7a	Ids of arrived vehicles
0xdb 0x7b	Delta T, returns the length of the simulation step

Table A.2: OMNeT++, CARLA and SUMO commands

A.2.2 Communication flow

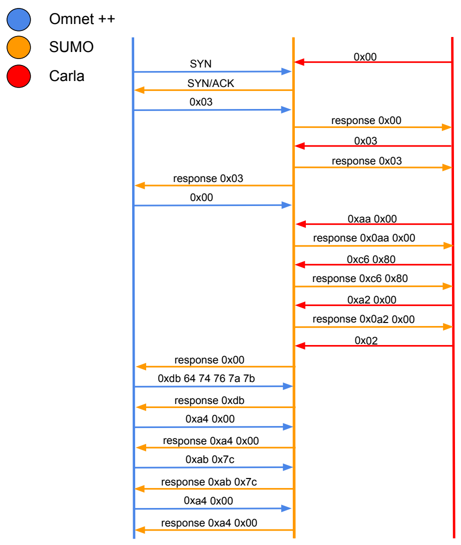


Figure A.3: OMNeT++, CARLA and SUMO communication flow 1

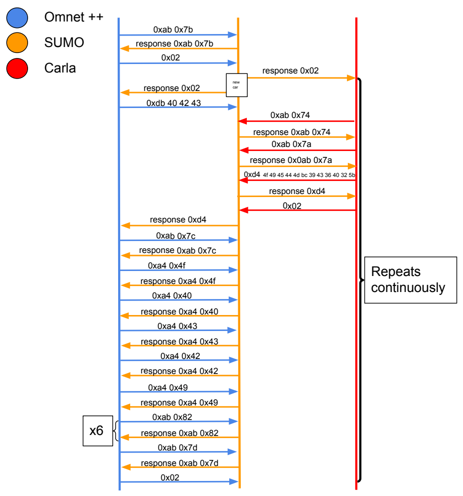


Figure A.4: OMNeT++, CARLA and SUMO communication flow 2

This example differs from the other one because a new simulator was added to the communication. SUMO is communicating with OMNeT++ and CARLA at the same time. To do so, there is a preference order to communicate, which is settled with the command 0x03 Set Order. In this case, CARLA has the preference.

Once the connection is done to both simulators and the order is set, the communication follows doing all the CARLA commands until it sends a 0x02 Simulation Step. At this moment the communication with OMNeT++ starts. When OMNeT++ finishes his first list of commands and sends the 0x02 Simulation Step command SUMO sends the response to this command to both simulators and starts communicating with CARLA again, until the simulation step is sent. This communication flow persists until the communication is finished.

In the first part of the communication before the first simulation step, both simulators: CARLA and OMNeT++, are asking for variables to configure their simulation. The OMNeT++ simulator asks for the same parameters as in the previous example. CARLA asks for the edges, the traffic lights, and it sets a new route for the simulation.

After the first simulation step, OMNeT++ and CARLA ask for the needed variables of the new vehicles in the simulations, and after retrieving this variables they subscribe the new vehicles. OMNeT++ sends the same commands explained in the previous example and CARLA asks for ID's of arrived and departed vehicles and subscribes a list of variables of the new vehicles.

Appendix B

Complete Set of Figures

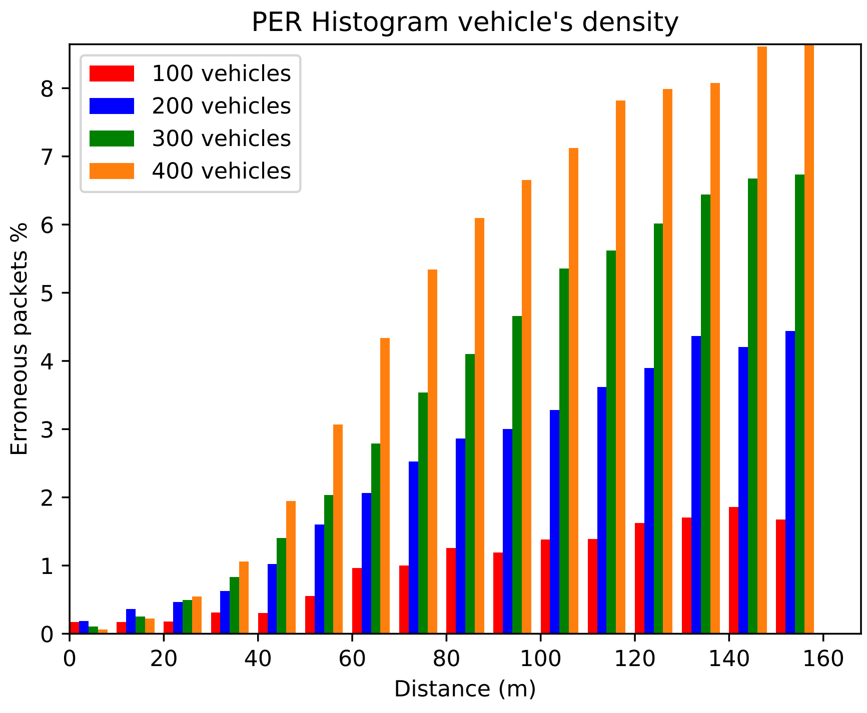


Figure B.1: Highway PER Histogram vehicle density

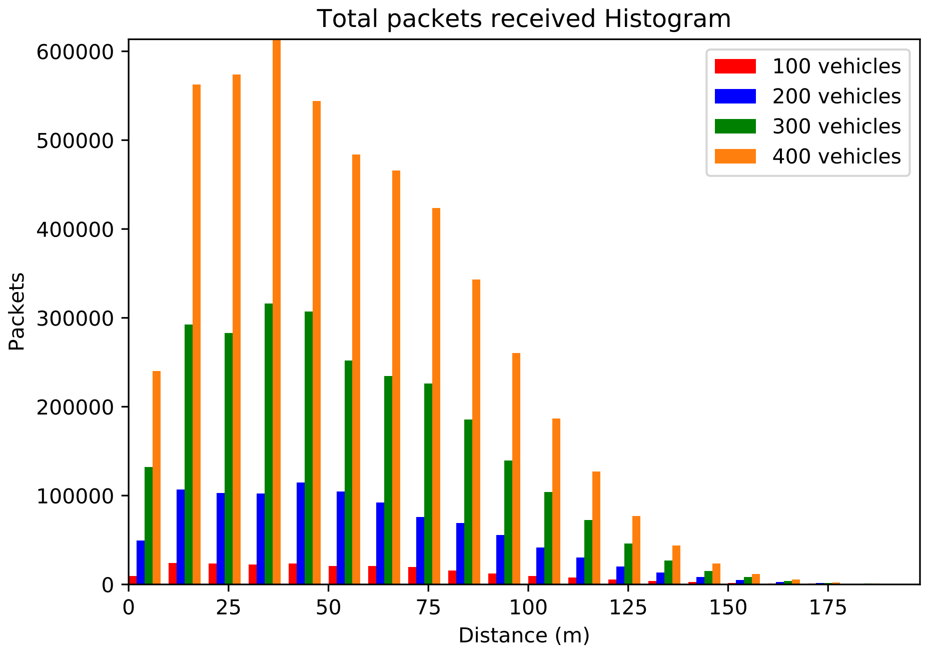


Figure B.2: Highway Packets Received Histogram vehicle density

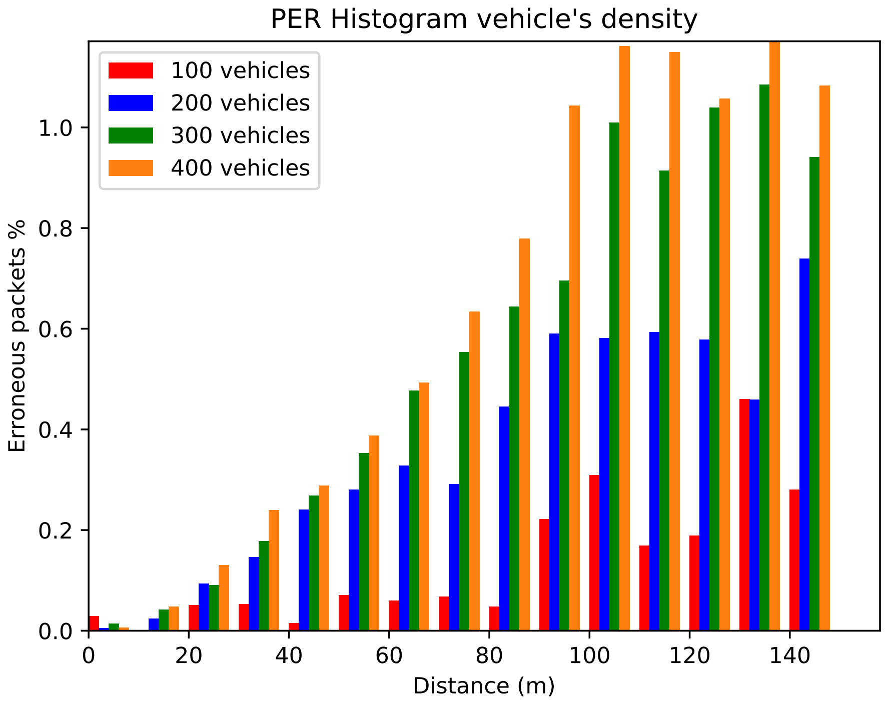


Figure B.3: Manhattan PER Histogram vehicle density

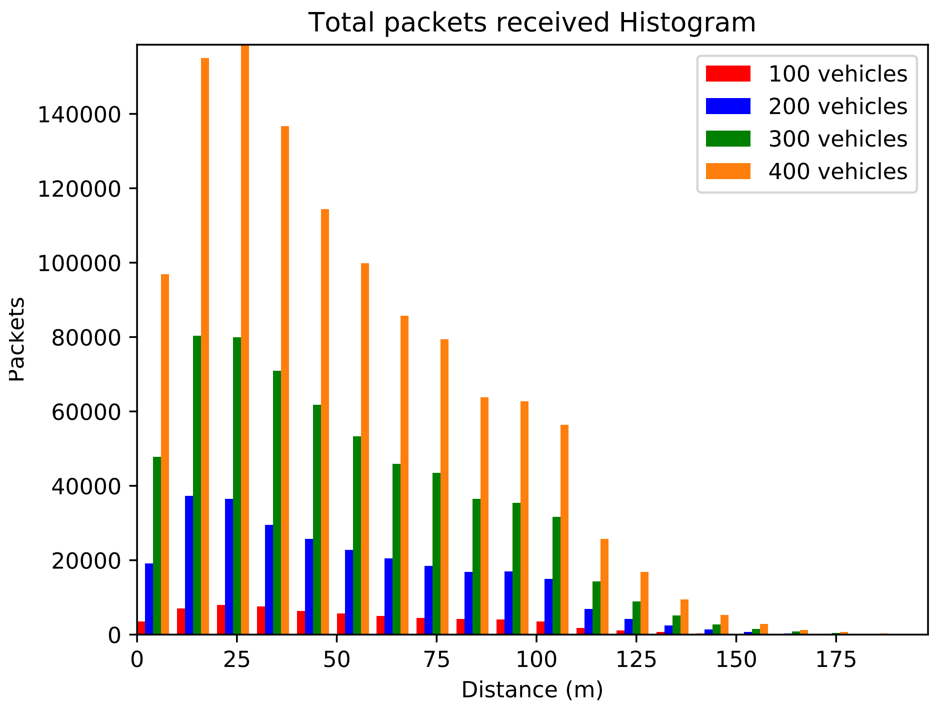


Figure B.4: Manhattan Packets Received Histogram vehicle density

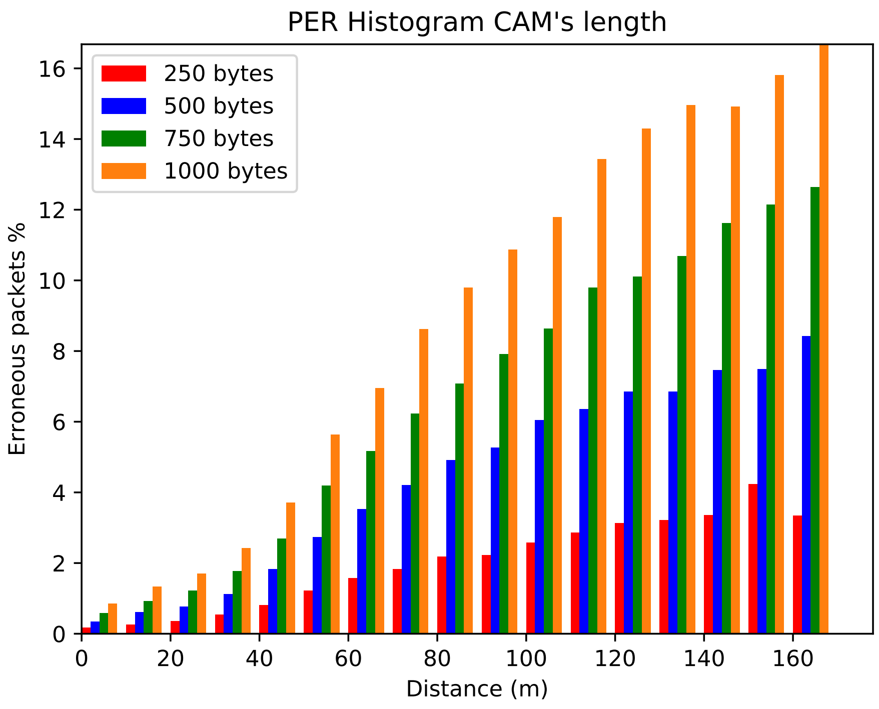


Figure B.5: Highway PER Histogram CAM's length

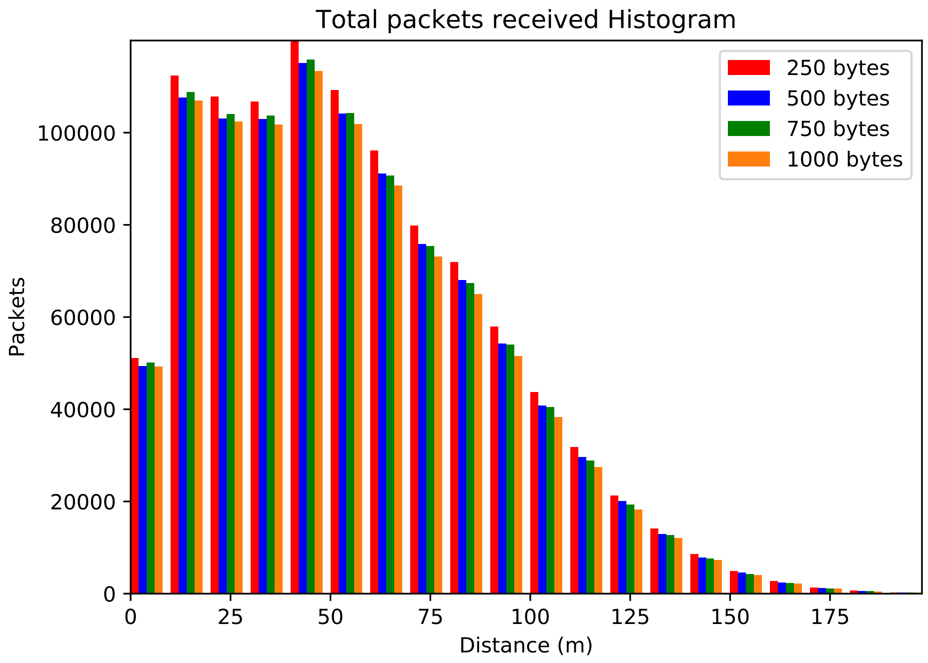


Figure B.6: Highway Packets Received Histogram CAM's length

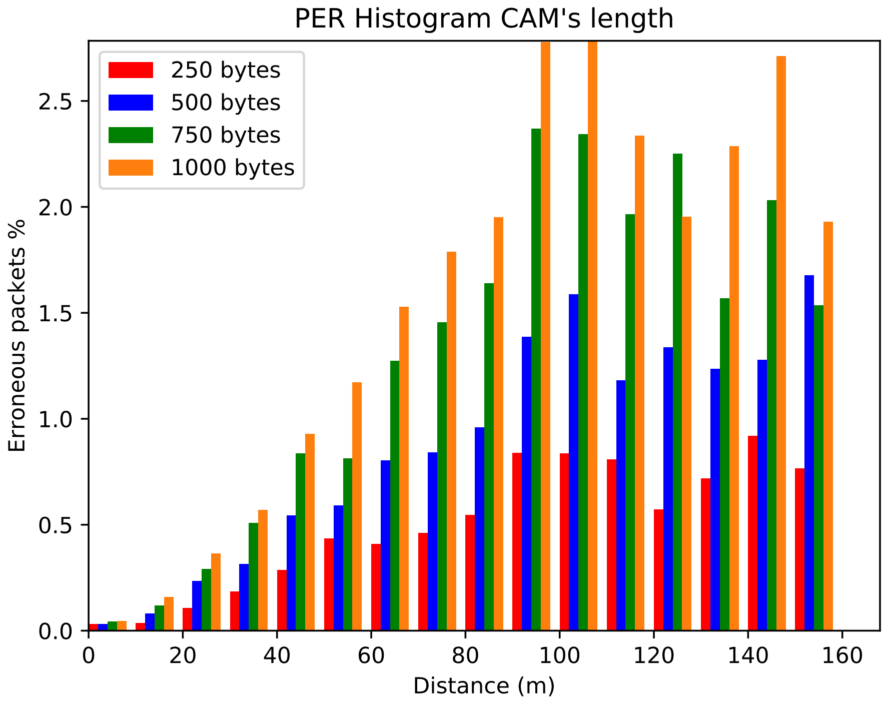


Figure B.7: Manhattan PER Histogram CAM's length

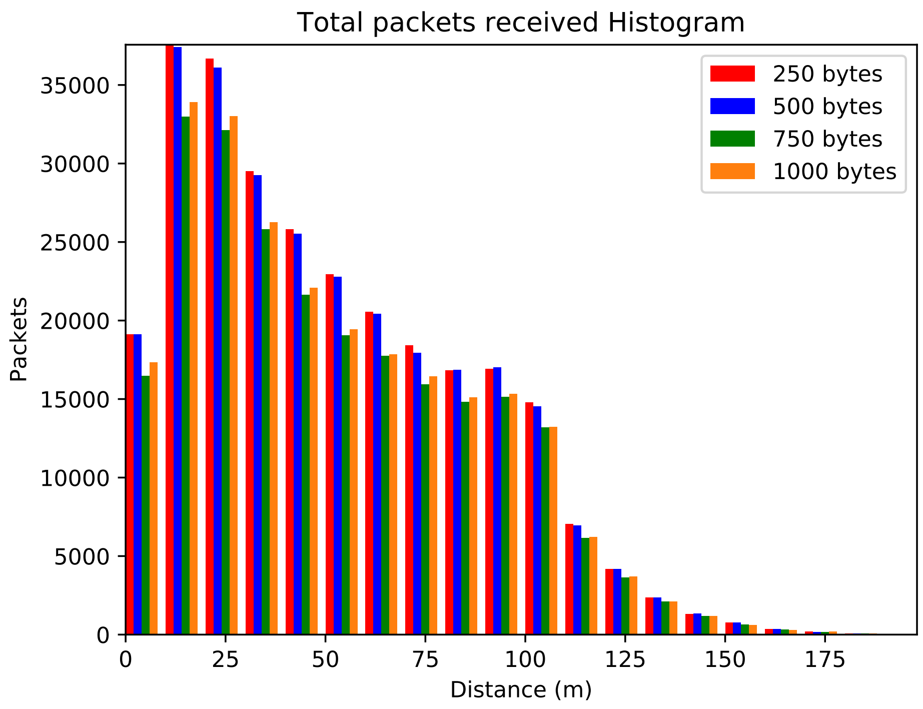


Figure B.8: Manhattan Packets Received Histogram CAM's length

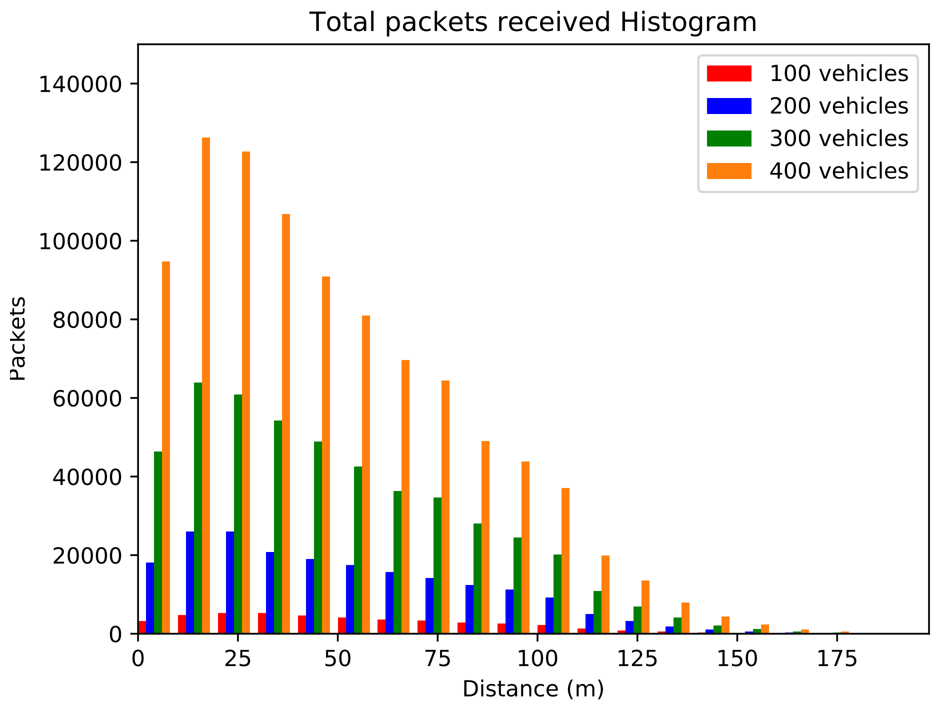


Figure B.9: Manhattan LOS Packets Received Histogram density varying

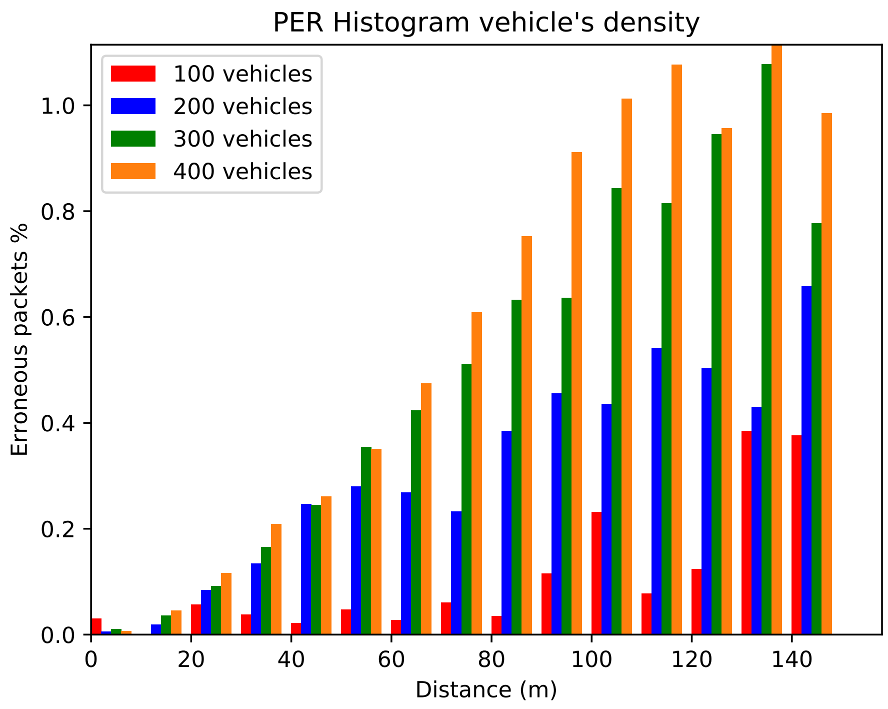


Figure B.10: Manhattan LOS PER Histogram density varying

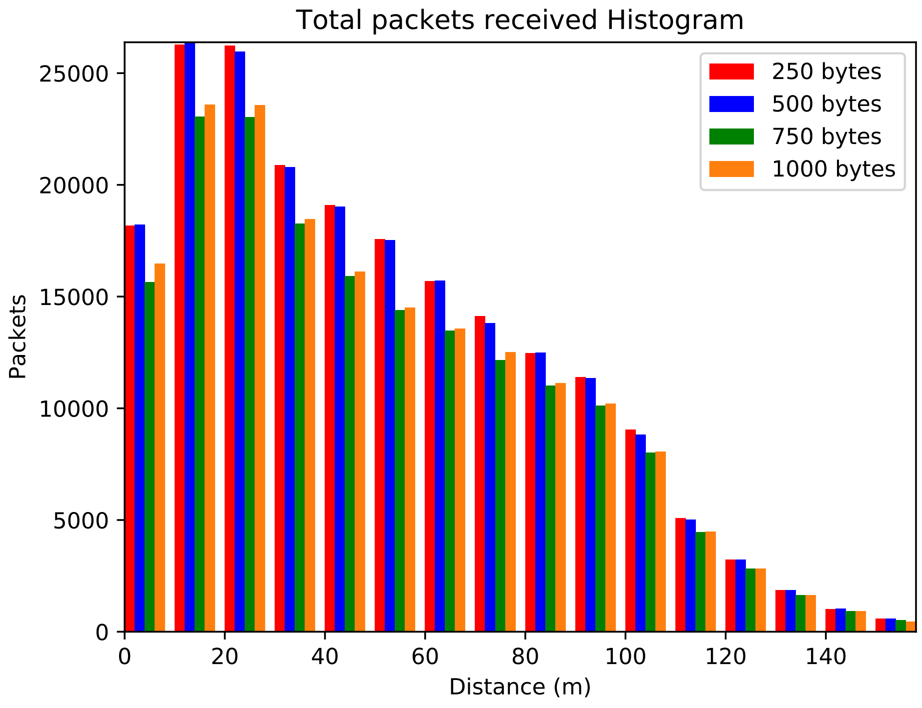


Figure B.11: Manhattan LOS Packets Received Histogram CAM's length

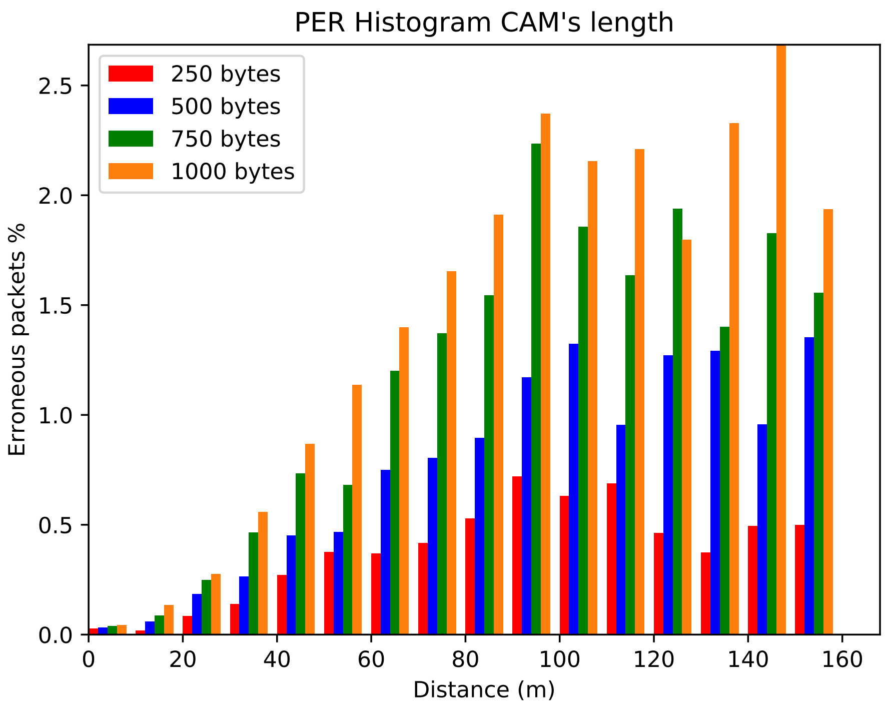


Figure B.12: Manhattan LOS PER Histogram CAM's length

Appendix C

Interceptor Code

C.1 Interceptor.py

```
import threading
import socket
import global_variables
import argparse


class MyThread(threading.Thread):
    '''Extends Thread to be used with sockets'''
    def __init__(self, name, function, host, port, event1, event2, gl_var):
        super().__init__()
        self.name = name
        self.function = function
        self.host = host
        self.port = port
        self.event1 = event1
        self.event2 = event2
        self.gl_var = gl_var

    def run(self):
        self.function(self.host, self.port, self.event1, self.event2, self.gl_var)

def omnet_server_socket(host, port, event_o, event_s, gl_var):
    '''Listens for connections and creates a socket to maintain a communication'''
    # Creates a server socket and listens
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, port))
        s.listen()
        conn, addr = s.accept()

        # Receives the command form Oment++ and waits for the response.
        # Trigegrs the corresponding events to manitain the threads synchronized
        with conn:
            print('Connected_to', addr)
            while True:
                try:
                    gl_var.command_omnet = conn.recv(gl_var.mtu)
                    event_o.set()

                    # Closes the socket when no data is received
                    if not gl_var.command_omnet:
                        break

                    # Waits until Sumo responds to the command.
```

```
                        event_s.wait()
                        conn.sendall(gl_var.command_sumo_omnet)
                        event_s.clear()

                    except KeyboardInterrupt:
                        conn.close()

def carla_server_socket(host, port, event_c, event_s, gl_var):
    '''Listens for connections and creates a socket to maintain a communication'''
    # Creates a server socket and listens
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, port))
        s.listen()
        conn, addr = s.accept()

        # Receives the command form Carla and waits for the response.
        # Triggers the corresponding events to maintain the threads synchronized
        with conn:
            print('Connected to', addr)
            while True:
                try:
                    gl_var.command_carla = conn.recv(gl_var.mtu)
                    event_c.set()

                    # Closes the socket when no data is received
                    if not gl_var.command_carla:
                        break

                    # Waits until the Sumo responses the command
                    event_s.wait()
                    conn.sendall(gl_var.command_sumo_carla)
                    event_s.clear()

                except KeyboardInterrupt:
                    conn.close()

def sumo_omnet_client_socket(host, port, event_o, event_s, gl_var):
    '''Client socket that connects to Sumo'''

    # Connects to Sumo
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((host,port))

        # Waits to receive the command from Omnet++ and sends it to Sumo.
        # Receives the response from Sumo and triggers an event.
        while True:
            try:
                # Waits until a command form Oment++ is received
                event_o.wait()
                s.sendall(gl_var.command_omnet)
                gl_var.command_sumo_omnet = s.recv(gl_var.mtu)

                # Triggers the events for proceed the communication
                event_o.clear()
                event_s.set()
                # Close the connection if no data is received
                if not gl_var.command_sumo_omnet:
                    break

            except KeyboardInterrupt:
                s.close()

def sumo_carla_client_socket(host, port, event_c, event_s, gl_var):
    '''Client socket that connects to Sumo'''
    # Connects to Sumo
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((host, port))

        # Waits to receive the command from Carla and sends it to Sumo.
        # Receives the response from Sumo and triggers an event.
        while True:
            try:
                # Waits until a command form Carla is received
                event_c.wait()
                s.sendall(gl_var.command_carla)
                gl_var.command_sumo_carla = s.recv(gl_var.mtu)

                # Triggers the events for proceed the communication
                event_c.clear()
                event_s.set()

                # Close the connection if no data is received
                if not gl_var.command_sumo_carla:
                    break
            except KeyboardInterrupt:
                s.close()


if __name__ == '__main__':
    # Parses the global variables to be modifiables from the user
    parser = argparse.ArgumentParser(
        description = 'Intercepts␣the␣communication␣between␣3␣simulators')
    parser.add_argument(
        '--host', type = str,
        help = 'Host␣where␣you␣try␣to␣connect␣(default␣=␣127.0.0.1)',
        default  = '127.0.0.1')
    parser.add_argument(
        '--port_omnet', type = int,
        help = 'Port␣where␣Omnet++␣client␣is␣trying␣to␣connect␣(default␣=␣8998)',
        default = 8998)
    parser.add_argument(
        '--port_sumo', type = int,
        help = 'Port␣where␣Sumo␣is␣waiting␣for␣a␣connection␣(default␣=␣9889)',
        default = 9889)
    parser.add_argument(
        '--port_carla', type = int,
        help = 'Port␣where␣Carla␣client␣is␣trying␣to␣connect␣(default␣=␣9999)',
        default = 9999)
    parser.add_argument(
        '--mtu', type = int,
        help = 'Size␣of␣the␣MTU␣of␣the␣protocol␣(default␣=␣66000)',
        default = 66000)
    args = parser.parse_args()

    # Initialize the global variables
    gl_var = global_variables.Globals(args.host,
        args.port_omnet, args.port_sumo, args.port_carla, args.mtu)

    # Creates the synchronization events
    e_omnet = threading.Event()
    e_sumo_o = threading.Event()
    e_carla = threading.Event()
    e_sumo_c = threading.Event()

    # Creates the threads
    t_omnet = MyThread('Omnet_socket', omnet_server_socket, gl_var.host,
        gl_var.port_omnet, e_omnet, e_sumo_o, gl_var)
    t_carla = MyThread('Carla_socket', carla_server_socket, gl_var.host,
        gl_var.port_carla, e_carla, e_sumo_c, gl_var)
    t_sumo_o = MyThread('Sumo_socket_from_omnet', sumo_omnet_client_socket,
        gl_var.host, gl_var.port_sumo, e_omnet, e_sumo_o, gl_var)
    t_sumo_c = MyThread('SUmo_socket_from_carla', sumo_carla_client_socket,
        gl_var.host, gl_var.port_sumo, e_carla, e_sumo_c, gl_var)

    # Starts the threads
    t_omnet.start()
    t_carla.start()
    t_sumo_o.start()
    t_sumo_c.start()


    t_omnet.join()
    t_carla.join()
    t_sumo_o.join()
    t_sumo_c.join()
```

C.2 Globals.py

```
from threading import Lock

class Globals:
    def __init__(self, host, port_omnet, port_sumo, port_carla, mtu):
        self._command_sumo_omnet = b''
        self._command_omnet = b''
        self._command_carla = b''
        self._command_sumo_carla = b''
        self._mtu = mtu
        self._host = host
        self._port_omnet = port_omnet
        self._port_sumo = port_sumo
        self._port_carla = port_carla
        self.lock = Lock()


    @property
    def command_omnet(self):
        return self._command_omnet

    @property
    def command_carla(self):
        return self._command_carla

    @property
    def command_sumo_omnet(self):
        return self._command_sumo_omnet

    @property
    def command_sumo_carla(self):
        return self._command_sumo_carla

    @property
    def mtu(self):
        return self._mtu

    @property
    def host(self):
        return self._host

    @property
    def port_omnet(self):
        return self._port_omnet

    @property
    def port_sumo(self):
        return self._port_sumo

    @property
    def port_carla(self):
        return self._port_carla

    @command_omnet.setter
    def command_omnet(self, received):
        self.lock.acquire()
        self._command_omnet = received
        self.lock.release()

    @command_carla.setter
    def command_carla(self, received):
        self.lock.acquire()
        self._command_carla = received
        self.lock.release()

    @command_sumo_omnet.setter
    def command_sumo_omnet(self, received):
        self.lock.acquire()
        self._command_sumo_omnet = received
        self.lock.release()

    @command_sumo_carla.setter
    def command_sumo_carla(self, received):
        self.lock.acquire()
        self._command_sumo_carla = received
        self.lock.release()

    @mtu.setter
    def mtu(self, value):
        self._mtu = value

    @host.setter
    def host(self, host):
        self._host = host

    @port_omnet.setter
    def port_omnet(self, port):
        self._port_omnet = port

    @port_sumo.setter
    def port_sumo(self, port):
        self._port_sumo = port

    @port_carla.setter
    def port_carla(self, port):
        self._port_carla = port

    def __str__(self):
        return f'''Command_omnet is {self.command_omnet}.
Command_carla is {self.command_carla}.
Command_sumo_omnet is {self.command_sumo_omnet}.
Command_sumo_carla is {self.command_sumo_carla}.
MTU is {self.mtu}.
HOST is {self.host}.
PORT_OMNET is {self.port_omnet}.
PORT_SUMO is {self.port_sumo}.
PORT_CARLA is {self.port_carla}.'''


if __name__ == '__main__':
    # Test of functionality
```

```
variables = Globals('127.0.0.1', 8998, 9889, 9999, 66000)
print(variables)
variables.command_omnet = b'\x00\x00\x02'
variables.command_carla = b'\x01\x02'
variables.command_sumo_omnet = b'\x00\x12'
variables.command_sumo_carla = b'\x00\xab'
variables.mtu = 1500
print(variables)
```

Bibliography

[1] World Health Organization. *Road traffic injuries*. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries. (accessed: 03.06.2021).

[2] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Communications Architecture". In: *ETSI EN 302 665 V1.1.1* (2010).

[3] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); Part1: Functional Requirements". In: *ETSI TS 102 637-1 V1.1.1* (2010).

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

[5] European Telecommunications Standards Institute (ETSI). "Intelligent Transport Systems (ITS); GeoNetworking; Part 1: Requirements". In: *ETSI EN 302 636-5-1 V1.2.1* (2014).

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

[7] German Aerospace Center (DLR). *SUMO - Wiki*. URL: https://sumo.dlr.de/docs/. (accessed: 20.05.2021).

[8] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[9] Alessandro Bazzi et al. "Survey and Perspectives of Vehicular Wi-Fi versus Sidelink Cellular-V2X in the 5G Era". In: *Future Internet* 11.6 (2019). ISSN: 1999-5903. DOI: 10.3390/fi11060122. URL: https://www.mdpi.com/1999-5903/11/6/122.

[10] Henrik Schumacher and Tchouankem Hugues Narcisse. "Highway Propagation Modeling in VANETs and Its Impact on Performance Evaluation". In: Mar. 2013. DOI: 10.1109/WONS.2013.6578344.

[11] Marco Morgante. “Motorway design guide: Capacity and flow analysis”. In: (2017).

[12] *SQLite - Wiki.* URL: `https://www.sqlite.org/about.html`. (accessed: 30.04.2021).

[13] *Pandas - Wiki.* URL: `https://pandas.pydata.org/docs/index.html`. (accessed: 10.05.2021).

[14] German Aerospace Center (DLR). *SUMO - Wiki - TraCI Protocol.* URL: `https://sumo.dlr.de/docs/TraCI/Protocol.html`. (accessed: 20.05.2021).

Acronyms

AC Access Category. 8

BSA Basic Set of Applications. 5

BTP Basic Transport Protocol. 6

C-ITS Cooperative Intelligent Transport Systems. 4

CA Cooperative Awareness. vii, viii, 1, 2, 12, 24, 25, 30

CAM Cooperative Awareness Message. v, vii, viii, 1, 6, 8, 9, 11, 15, 17, 23, 25–27, 29, 30

CARLA Car Learning to Act. v–viii, 3, 9, 12, 19, 21–23, 35, 36

CCH Control Channel. 8, 15

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance. 8

DCC Decentralyzed Congestion Control. 8, 17

DENM Decentralized Environmental Notification Message. 6, 8, 11

EDCA Enhanced Distributed Coordination Access. 8

EVI Ego-Vehicle Interface. 19, 22, 23, 30

GPS Global Positioning System. 12

IDE Integrated Development Environment. 17

IEEE Institute of Electrical and Electronics Engineers. 7, 8, 10, 11, 24, 30

ITS Intelligent Transport Systems. 4–8

ITS-S ITS station. v, vi, 4, 5

LLC Logical Link Control. 7

LOS Line-Of-Sight. vii, viii, 14, 16, 27, 30

MAC Medium Access Control. 7, 8

MATLAB MATrix LABoratory. 18

NLOS Non Line of Sight. 27

OBU On-Board Unit. 4

OMNeT++ Objective Modular Network Testbed in C++. v–viii, 2, 3, 9, 11–13, 17–19, 21–24, 30, 32, 33, 35, 36

OSI Open System Interconnection. vi, 4, 5, 7

PER Packet Error Rate. vii, viii, 24–28, 30

PHY Physical Layer. 7

RSU Roadside Unit. 4

SNIR Signal to Noise and Interference Ratio. 25

SNR Signal to Noise Ratio. 25

SQL Structured Query Language. 17, 18

SUMO Simulation of Urban Mobility. v, vi, 3, 9–13, 19–22, 30, 32, 33, 35, 36

TCP Transmission Control Protocol. vi, 11, 19, 20

TDC Transmit Data rate Control. 8

TPC Transmit Power Control. 8

TraCI Traffic Control Interface. v, 2, 12, 19–22, 30

TRC Transmit Rate Control. 8

UDP User Datagram Protocol. 11

UPC Polytechnic University of Catalonia. 3

V2X Vehicle to Everything. vii, viii, 1, 2, 22, 23

WLAN Wireless Local Area Network. 7