



**UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH**

---

**Escola Tècnica Superior d'Enginyeria  
de Telecomunicació de Barcelona**

**“SIMULATION OF BASIC MULTI-HOP BROADCAST  
TECHNIQUES IN VEHICULAR Ad-Hoc NETWORKS USING  
VEINS SIMULATOR”**

**A Master's Thesis  
Submitted to the Faculty of the  
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona  
Universitat Politècnica de Catalunya  
by  
David Leonardo Renjifo Herrera**

**In partial fulfilment  
of the requirements for the degree of  
MASTER IN TELECOMMUNICATIONS ENGINEERING**

**Advisor: Mónica Aguilar Igartua  
Co-Advisor: Cristhian Iza**

**Barcelona, February 2016**



**Title of the thesis:** “Simulation of basic multi-hop broadcast techniques in Vehicular Ad-hoc Networks using VEINS simulator”.

**Author:** David Leonardo Renjifo Herrera

**Advisor:** Mónica Aguilar Igartua

**Co-advisor:** Cristhian Iza

## Abstract

This project presents to the reader an incursion into the world of smart cities and vehicles, intelligent transport systems, and vehicular ad-hoc networks (VANETs). They are conceptually analyzed, studied in terms of requirements, framework, architecture, applications and standardization.

The main objective of this project is the simulation of basic techniques of message dissemination in VANETs, this project shows a brief description of broadcast dissemination techniques. Also, it is necessary to analyze simulation of VANETs. For this reason, it presents a brief description of mobility generators, network simulators and VANETs simulators paying attention to the open source ones.

The main focus is set on the VEINS simulation framework, due to its high performance results and a bidirectional coupling between the network and the traffic simulators. From that point, the project is aimed at analyzing the basic paradigms of VEINS components (OMNeT++, SUMO and the TraCI module) and providing a study on their features.

Three techniques are simulated (*Flooding*, *Counter* and *Probability* schemes) in two different scenarios (urban and highway), the simulation results are shown focused on the amount of retransmitting nodes, percentage of reached nodes, amount of packets sent, average packet delay, and percentage of packets received.

Finally, a detailed manual is presented to the reader. This manual shows how to install the VEINS simulator and its components in both Windows and Linux systems. In addition, it shows how to create a scenario from an extracted map from OpenStreetMap.

**Keywords:** Intelligent transport systems, Vehicular ad-hoc networks, VANETs simulators, VEINS, SUMO, OMNeT++.

## Acknowledgements

Two years ago this adventure begun and I would like to thank all the people who have accompanied me during this stage of my life.

Thank God for guiding me in every moment of my life and his immeasurable love. Thank my family for their unconditional support, especially my parents Monica and Leonardo who are the best.

Thank Cristhian Iza and Monica Aguilar for the opportunity to work with them and go into world of vehicular networks.

Thank you very much!

## Table of Contents

<b>Abstract .....</b>	<b>1</b>
<b>Acknowledgements .....</b>	<b>2</b>
<b>List of Figures .....</b>	<b>8</b>
<b>List of tables .....</b>	<b>12</b>
<b>Symbols and acronyms .....</b>	<b>13</b>
<b>Chapter 1 Introduction .....</b>	<b>15</b>
1.1 Problem Statement .....	15
1.2 Objectives .....	16
1.2.1 General objective .....	16
1.2.2 Specific objectives .....	16
1.3 Contents and organization of the Master's Thesis .....	16
<b>Chapter 2 Smart cities and vehicles .....</b>	<b>19</b>
2.1 Smart City .....	19
2.1.1 Definition .....	19
2.1.2 Smart city applications .....	21
2.1.3 Construction of smart transportation .....	22
2.2 Smart Vehicles .....	22
2.2.1 Definition .....	22
2.2.2 General architecture .....	23
<b>Chapter 3 Intelligent Transport Systems .....</b>	<b>25</b>
3.1 Definition .....	25
3.2 Objectives .....	26
3.3 Framework .....	27
3.4 Classification and applications of Intelligent Transport Systems .....	27
3.4.1 Advanced Traveler Information Systems .....	28
3.4.2 Advanced Transportation Management System .....	29
3.4.3 Commercial Vehicle Operation .....	29
3.4.4 Advanced Public Transportation System .....	29
3.4.5 Advanced Vehicle Control and Safety System .....	29
<b>Chapter 4 Vehicular Ad-Hoc Networks .....</b>	<b>30</b>
4.1 Definition .....	30
4.2 Architecture .....	31

4.2.1 Main Components .....	32
4.2.2 Communication architecture .....	35
4.3 Vehicular network applications .....	37
4.4 Standardization .....	38
4.4.1 Dedicated Short Range Communication .....	38
4.4.2 IEEE 1609 Standards for wireless access in vehicular environments (WAVE) .....	39
<b>Chapter 5 Message dissemination in Vehicular Ad-Hoc Networks .....</b>	<b>41</b>
5.1 VANET message dissemination .....	41
5.2 Broadcast in the link layer .....	41
5.3 Vehicular multi-hop broadcast .....	42
5.3.1 Blind flooding .....	42
5.3.2 Counter-based scheme .....	42
5.3.3 Distance-based scheme.....	43
5.3.4 Location-based Scheme .....	43
5.3.5 Neighbor knowledge scheme.....	43
5.3.6 Cluster-based scheme .....	43
5.3.7 Probability scheme .....	43
5.4 The broadcast storm problem .....	44
5.4.1 Weighted p-persistence scheme.....	44
5.4.2 Slotted 1-persistence scheme .....	45
5.4.3 Slotted p-persistence scheme .....	45
5.5 Farthest node scheme .....	46
5.6 Carry-store-forward mechanism.....	46
<b>Chapter 6 Simulation of Vehicular Ad-Hoc Networks .....</b>	<b>47</b>
6.1 Introduction .....	47
6.2 Mobility generators .....	48
6.2.1 SUMO .....	48
6.2.2 FreeSim.....	49
6.2.3 STRAW .....	49
6.2.4 VanetMobiSim.....	49
6.2.5 MOVE .....	50
6.2.6 CityMob .....	50
6.3 Network simulator .....	50
6.3.1 NS-2 .....	51
6.3.2 GloMoSim.....	51
6.3.3 SNS .....	51

6.3.4 JiST/SWANS .....	52
6.3.5 OMNeT++ .....	52
6.4 VANET simulators.....	52
6.4.1 GrooveNet.....	53
6.4.2 SWANS++.....	53
6.4.3 TraNS .....	53
6.4.4 VEINS .....	53
6.4.5 NCTUns.....	54
6.5 Selecting a convenient suit of simulators .....	54
<b>Chapter 7 VEINS Simulator .....</b>	<b>55</b>
7.1 Introduction .....	55
7.2 Network simulator: OMNeT ++.....	55
7.2.1 Model structure .....	56
7.2.2 Parallel Simulation Support.....	57
7.2.3 Simulation Model .....	57
7.2.4 Packet INET.....	58
7.2.5 MiXiM .....	58
7.3 Traffic simulator: SUMO .....	59
7.3.1 Basic paradigms.....	59
7.3.2 Car-Driver Model.....	60
7.3.3 Traffic Lights .....	61
7.3.4 Features.....	61
7.3.5 Components .....	62
7.4 TraCI .....	63
7.4.1 Sumo-launchd .....	64
<b>Chapter 8 ..... Analysis of simulation results of vehicular multi-hop broadcast techniques using VEINS .....</b>	<b>66</b>
8.1 Introduction .....	66
8.2 Urban scenario.....	67
8.3 Highway scenario .....	68
8.4 Flooding technique .....	68
8.4.1 Flooding technique in the Urban scenario .....	68
8.4.2 Flooding technique in the Highway scenario .....	71
8.4.3 Conclusions about the Flooding technique.....	74
8.5 Counter technique .....	75
8.5.1 Counter technique in the Urban scenario.....	75
8.5.2 Counter technique in the Highway scenario .....	78

8.5.3 Conclusions about Counter technique .....	80
8.6 Probability technique.....	80
8.6.1 Probability technique in the Urban scenario .....	81
8.6.2 Probability technique in the Highway scenario .....	84
8.6.3 Conclusions about Probability technique .....	86
8.7 Comparison between Flooding, Counter and Probability techniques.....	87
8.7.1 Comparison in the Urban scenario.....	87
8.7.2 Comparison in the Highway scenario.....	89
<b>Chapter 9 Conclusions and future work .....</b>	<b>92</b>
9.1 Conclusions .....	92
9.2 Future work.....	93
<b>References .....</b>	<b>94</b>
<b>ANNEXS.....</b>	<b>96</b>
<b>A. How install Virtual Box and create a virtual machine with Linux Ubuntu System.....</b>	<b>96</b>
A.1 How to create a virtual machine in Virtual Box .....	99
<b>B. Installation of SUMO, OMNeT++ and Veins simulator .....</b>	<b>116</b>
B.1 Get the source code for the programs .....	116
B.1.1 Get the source code for SUMO .....	116
B.1.2 Get the source code for OMNeT++ .....	116
B.1.3 Get the source code for VEINS .....	116
B.2 Installation in Linux .....	116
B.2.1 Installation of packages in Linux Ubuntu .....	116
B.2.2 Installation of SUMO .....	119
B.2.3 Installation of OMNeT++ .....	121
B.3 Installation in Windows .....	123
B.3.1 Installation of SUMO .....	123
B.3.2 Installation of OMNeT .....	124
B.4 Installation of Veins .....	126
B.4.1 Import VEINS in OMNeT++ IDE .....	126
B.4.2 Run the Veins demo scenario.....	130
<b>C. Example of simulation in Veins .....</b>	<b>133</b>
C.1 Importing networks and generation of routes in SUMO .....	133
C.1.1 OpenStreetMap.....	133
C.1.2 Get a map from OpenStreetMap .....	133
C.1.3 Preparation the map for use in SUMO.....	135
C.1.4 Generation of routes in SUMO.....	138

C.2 Prepare files before simulating.....	139
C.3 Running the simulation.....	142
<b>D. Source code of files used in the simulations.....</b>	<b>148</b>
D.1 Implementation of omnetpp .....	148
D.2 Implementation of FranciscoScenario .....	151
D.3 Implementation of Counter technique.....	152
D.3.1 Counter.cc .....	152
D.3.2 Counter.h .....	155
D.3.3 Counter.ned .....	156
D.4 Implementation of Flooding technique .....	157
D.4.1 Flooding.cc .....	157
D.4.2 Flooding.h.....	159
D.4.3 Flooding.ned.....	160
D.5 Implementation of Probability technique .....	160
D.5.1 Probability.cc.....	160
D.5.2 Probability.h .....	163
D.5.3 Probability.ned .....	164
D.6 Implementation of FranciscoStatistics.....	164
D.6.1 FranciscoStatistics.cc.....	164
D.6.2 FranciscoStatistics.h .....	165
D.6.3 FranciscoStatistics.ned .....	166

## List of Figures

Figure 2.1 Technical architecture diagram of smart city [6].	20
Figure 2.2 A smart car's onboard instrumentation [9].	23
Figure 2.3 General architecture of a smart vehicle [10].	23
Figure 3.1 Intelligent transport systems [12].	26
Figure 4.1 WLAN/Cellular architecture [14].	31
Figure 4.2 Ad Hoc architecture [14].	32
Figure 4.3 Hybrid architecture [14].	32
Figure 4.4 VANETs system domain [17].	34
Figure 4.5 C2C-CC reference architecture [17].	35
Figure 4.6 key functions of each communication type [17].	36
Figure 4.7 Communications in vehicle networks [18].	37
Figure 4.8 Overview of the WAVE stack and associated standards [2].	39
Figure 5.1 Weighted $p$ -persistence [20].	45
Figure 5.2 Slotted 1-persistence scheme [20].	45
Figure 5.3 Slotted $p$ -persistence scheme [20].	46
Figure 6.1 Classification of VANETs simulators [23].	48
Figure 7.1 Model Structure in OMNeT++ [25].	57
Figure 7.2 Logical Architecture of the OMNeT++ Parallel Simulation kernel [25].	57
Figure 7.3 Logical Architecture of an OMNeT++ Simulation Program [25].	58
Figure 7.4 Multimodality [29].	59
Figure 7.5 The different traffic flow model [30].	60
Figure 7.6 Scheme of applications in SUMO.	62
Figure 7.7 Establishing a connection to SUMO [33].	63
Figure 7.8 Closing a connection to SUMO.	64
Figure 7.9 VEINS, simulation model [34].	64
Figure 7.10 Lifecycle management functionality of sumo-launchd [35].	65
Figure 8.1 Urban scenario. Part of the Eixample district. Barcelona, Spain.	67
Figure 8.2 Highway scenario. Part of the C-32 Highway. Barcelona, Spain.	68
Figure 8.3 The classical Flooding technique. Number of retransmitting nodes. CI 90%. Urban scenario.	69
Figure 8.4 The classical Flooding technique. Sent packets. CI 90%. Urban scenario.	69
Figure 8.5 The classical Flooding technique. Percentage of received packets. CI 90%. Urban scenario.	70
Figure 8.6 The classical Flooding technique. Percentage of lost packets. CI 90%. Urban scenario.	70
Figure 8.7 The classical Flooding technique. Delay. CI 90%. Urban scenario.	71
Figure 8.8 The classical Flooding technique. Percentage of reached nodes. CI 90%. Urban scenario.	71
Figure 8.9 The classical Flooding technique. Number of retransmitting nodes. CI 90%. Highway scenario.	72
Figure 8.10 The classical Flooding technique. Sent packets. CI 90%. Highway scenario.	72
Figure 8.11 The classical Flooding technique. Percentage of received packets. CI 90%. Highway scenario.	73

Figure 8.12 The classical Flooding technique. Percentage of lost packets. CI 90%. Highway scenario. ....	73
Figure 8.13 The classical Flooding technique. Delay. CI 90%. Highway scenario. ....	74
Figure 8.14 The classical Flooding technique. Percentage of reached nodes. CI 90%. Highway scenario. ....	74
Figure 8.15 The Counter technique. Number of retransmitting nodes. CI 90%. Urban scenario. ....	76
Figure 8.16 The Counter technique. Sent packets. CI 90%. Urban scenario. ....	76
Figure 8.17 The Counter technique. Delay. CI 90%. Urban scenario. ....	77
Figure 8.18 The Counter technique. Percentage of reached nodes. CI 90%. Urban scenario. ....	77
Figure 8.19 The Counter technique. Number of retransmitting nodes. CI 90%. Highway scenario. ....	78
Figure 8.20 The Counter technique. Sent packets. CI 90%. Highway scenario. ....	79
Figure 8.21 The Counter technique. Delay. CI 90%. Highway scenario. ....	79
Figure 8.22 The Counter technique. Percentage of reached nodes. CI 90%. Highway scenario. ....	80
Figure 8.23 The Probability technique. Number of retransmitting nodes. CI 90%. Urban scenario. ....	82
Figure 8.24 The Probability technique. Sent packets. CI 90%. Urban scenario. ....	82
Figure 8.25 The Probability technique. Delay. CI 90%. Urban scenario. ....	83
Figure 8.26 The Probability technique. Percentage of reached nodes. CI 90%. Urban scenario. ....	84
Figure 8.27 The Probability technique. Number of retransmitting nodes. CI 90%. Highway scenario. ....	85
Figure 8.28 The Probability technique. Sent packets. CI 90%. Highway scenario. ....	85
Figure 8.29 The Probability technique. Delay. CI 90%. Highway scenario. ....	86
Figure 8.30 The Probability technique. Percentage of reached nodes. CI 90%. Urban scenario. ....	86
Figure 8.31 Number of retransmitting nodes for each technique in the Urban scenario. ....	88
Figure 8.32 Percentage of reached nodes for each technique in the Urban scenario. ....	88
Figure 8.33 Delay for each technique in the Urban scenario. ....	89
Figure 8.34 Number of retransmitting nodes for each technique in the Highway scenario. ....	90
Figure 8.35 Percentage of reached nodes for each technique in the Highway scenario. ....	91
Figure 8.36 Delay for each technique in the Highway scenario. ....	91
Figure A.1 Beginning of the installation of the Virtual Box. ....	96
Figure A.2 Select the destination folder. ....	97
Figure A.3 Create shortcuts. ....	97
Figure A.4 Warning about network interfaces. ....	98
Figure A.5 Install the Virtual Box. ....	98
Figure A.6 Installation is completed successfully. ....	99
Figure A.7 Create a new virtual machine. ....	99
Figure A.8 Select the Operating System of the virtual machine. ....	100
Figure A.9 Select the amount of RAM memory. ....	100
Figure A.10 Create a virtual hard drive. ....	101
Figure A.11 Select the type of the virtual hard drive. ....	101
Figure A.12 Select the type of storage in the virtual hard drive. ....	102
Figure A.13 Location and size of the virtual hard drive. ....	102
Figure A.14 Run the virtual machine. ....	103
Figure A.15 Select the startup disk. ....	103
Figure A.16 Choose the folder with the ISO of Ubuntu. ....	104
Figure A.17 Run the ISO of Ubuntu in the virtual machine. ....	104

Figure A.18 Select language. ....	105
Figure A.19 Preparing to install Ubuntu. ....	105
Figure A.20 Select the type of the installation. ....	106
Figure A.21 Accept format of the partition. ....	106
Figure A.22 Select the city. ....	107
Figure A.23 Select the language for the keyboard. ....	107
Figure A.24 User and password. ....	108
Figure A.25 Installation of Ubuntu. ....	108
Figure A.26 Restart to finish the installation. ....	109
Figure A.27 Login in the virtual machine. ....	109
Figure A.28 Full screen is not possible. ....	110
Figure A.29 Insert the ISO of Guest Additions. ....	110
Figure A.30 Run VBOXADDITONS_4.3.24_98716. ....	111
Figure A.31 Permit the installation by writing your password. ....	111
Figure A.32 Installing graphics libraries and desktop services. ....	112
Figure A.33 (a) and (b) shut down virtual machine. ....	113
Figure A.34 Turn on the virtual machine. ....	114
Figure A.35 Login in the system. ....	114
Figure A.36 Ubuntu in full screen mode. ....	115
Figure B.1 Login as user root. ....	117
Figure B.2 Configuration to access into root desktop. ....	117
Figure B.3 Login as root user. ....	118
Figure B.4 Installation of aptitude. ....	118
Figure B.5 Installation of necessary packages for installation of SUMO simulator. ....	119
Figure B.6 Unzip the file sumo-src-0.22.0.tar.gz. ....	119
Figure B.7 Building the SUMO binaries: ./configure. ....	119
Figure B.8 Building the SUMO binaries: make. ....	120
Figure B.9 Installing the SUMO binaries: make install. ....	120
Figure B.10 Environment variables a) edit bash b) update bash. ....	120
Figure B.11 Run SUMO. ....	121
Figure B.12 Unzip omnet-4.6. ....	121
Figure B.13 Environment variables for OMNeT++. ....	122
Figure B.14 Configuring the installation of OMNeT++. ....	122
Figure B.15 Building OMNeT++. ....	122
Figure B.16 Launch OMNeT++. ....	122
Figure B.17 Install INET framework. ....	123
Figure B.18. Extract the zip file sumo-winbin-0.22.0. ....	123
Figure B.19 Run sumo 0.22.0. ....	124
Figure B.20 Extract the zip file omnet-4.6-src-windows. ....	124
Figure B.21 Type the commands ./conFigure (a) and make (b). ....	125
Figure B.22 Start the OMNeT++ IDE. ....	125
Figure B.23 Install INET framework and import the OMNeT++ examples. ....	126
Figure B.24 Extract zip file veins-4a2. ....	126
Figure B.25 Unzip VEINS. ....	127
Figure B.26 Import a project in OMNeT++. ....	127

Figure B.27 Select the option existing projects into workspace. ....	128
Figure B.28 Select the folder omnet. ....	128
Figure B.29 Import VEINS into OMNeT++. ....	129
Figure B.30 Build VEINS in OMNeT++. ....	129
Figure B.31 Run SUMO in Linux System. ....	130
Figure B.32 Run SUMO in Windows System. ....	130
Figure B.33 Example scenario of SUMO running. ....	131
Figure B.34 Run python script in Linux System. ....	131
Figure B.35 Run python script in Windows System. ....	131
Figure B.36 Run Veins demo scenario. ....	132
Figure B.37 OMNeT++ and SUMO running in parallel to simulate VEINS demo scenario. ....	132
Figure C.1 OpenStreetMap web site. ....	134
Figure C.2 Select the area and export the map. ....	134
Figure C.3 Save the map in your computer. ....	135
Figure C.4 Generation of network through netconvert. ....	135
Figure C.5 Create an empty file in Linux System. ....	136
Figure C.6 Write the name of the new file as typemap.xml. ....	136
Figure C.7 Copy the content of website typemap. ....	137
Figure C.8 Save typemap.xml file. ....	137
Figure C.9 Command polyconvert. ....	138
Figure C.10 Generation of random routes in SUMO. ....	139
Figure C.11 Copy sumo files to veins folder ....	139
Figure C.12 Edit Erlangen.launchd.xml. ....	139
Figure C.13 Edit Erlangen.sumo.cfg. ....	140
Figure C.14 OMNeT scenario files. ....	140
Figure C.15 Create folders for dissemination methods files. ....	141
Figure C.16 Counter method files. ....	141
Figure C.17 Flooding method files. ....	141
Figure C.18 Probability method files. ....	141
Figure C.19 Create stats folder. ....	142
Figure C.20 Statistics files. ....	142
Figure C.21 Clean local variables. ....	142
Figure C.22 Clean project variables. ....	143
Figure C.23 Build project in OMNeT++. ....	143
Figure C.24 Run python script. ....	143
Figure C.25 Run omnetpp.ini. ....	144
Figure C.26 Select a dissemination method. ....	144
Figure C.27 Click on run to start simulation. ....	145
Figure C.28 Error during simulation. ....	145
Figure C.29 Configure playground parameters. ....	145
Figure C.30 Run simulation in SUMO. ....	146
Figure C.31 Run simulation in OMNeT++. ....	146
Figure C.32 Results of simulation in VEINS. ....	147

## List of tables

<i>Table 2.1 Working Definitions of a Smart City [3].</i>	20
<i>Table 2.2 The list of smart cities [7].</i>	21
<i>Table 3.1 Functions and users services of ITS.</i>	27
<i>Table 3.2 Classifying contactless mobile payments applications [11].</i>	28
<i>Table 4.1 Safety applications in VANET [2].</i>	38
<i>Table 7.1 Applications in the SUMO package [31].</i>	62
<i>Table 8.1. Simulation results of the classical Flooding technique in the Urban scenario.</i>	68
<i>Table 8.2 Simulation results of the classical Flooding technique in the Highway scenario.</i>	71
<i>Table 8.3 Simulation results of the Counter technique in the Urban scenario.</i>	75
<i>Table 8.4 Simulation results of the Counter scheme in highway scenario.</i>	78
<i>Table 8.5 Simulation results of the Probability technique in the Urban scenario.</i>	81
<i>Table 8.6 Simulation results of the Probability scheme in the Highway scenario.</i>	84

## Symbols and acronyms

APTS	Advanced Public Transportation Systems
ATIS	Advanced Traveler Information Systems
ATMS	Advanced Transportation Management Systems
AVCSS	Advanced Vehicle Control and Safety System
AVL	Automatic Vehicle Location
CanuMobiSim	CANU Mobility Simulation Environment
CD	Collision Detection
CICAS	Cooperative Intersection Collision Avoidance System
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CVO	Commercial Vehicle Operation
DCF	Distributed Coordination Function
DLR	German Aerospace Center
DM	Downtown Model
DSRC	Dedicated Short Range Communication
EDR	Event Data Recorder
ERP	Electronic Road Pricing
ETC	Electronic Toll Collection
FCC	Federal Communications Commission
GUI	Graphical User Interface
GPS	Global Positioning System
HAR	Highway Advisory Radio
HMI	Human Machine Interface
ICF	Intelligent Community Forum
IDM/IM	Intelligent Driving Model with Intersection Management
IDM/LC	Intelligent Driving Model with Lane Changing
ISA	Intelligent Speed Adaptation
IT	Information Technology
ITS	Intelligent Transportation Systems
IVC	Inter-vehicle Communications
JOSM	Java Open Street Map
MAC	Medium Access Control
MANET	Mobile Ad-Hoc Networks
MM	Manhattan Model

MOVE	Mobility model generator for Vehicular networks
NCTUns	National Chiao Tung University Network Simulator
OFDM	Orthogonal Frequency Division Multiplexing
OSM	Open Street Map
PDES	Parallel Distributed Simulation
PHY	Physical
PLCP	Physical Layer Convergence Procedure
RTS/CTS	Request to Send/Clear to Send
SINR	Signal to Interference plus Noise Ratio
SM	Simple Model
SNS	Staged Network Simulator
STRAW	Street Random Waypoint
SUMO	Simulation of Urban Mobility
TIGER	Topologically Integrated Geographic Encoding and Referencing
TMC	Traffic Management Centers
TOC	Traffic Operations Centers
TraCI	Traffic Control Interface
TraNS	Traffic and Network Simulation Environment
USB	Universal Serial Bus
V2V	Vehicle-to-Vehicle Communications
VANET	Vehicular Ad-Hoc Networks
VEINS	Vehicles in Network Simulation
VMT	Vehicle-Miles Traveled
VPS	Vehicle Positioning System
WAVE	Wireless Access in Vehicular Environments
WHO	World Health Organization
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network

# Chapter 1

## Introduction

### 1.1 Problem Statement

Urban and interurban traffic are among the major issues of contemporary society. According to the WHO (World Health Organization), road traffic injuries are the eighth leading cause of death globally, and the leading cause of death for young people aged 15-29. More than a million people die each year on the world's roads, and the cost of dealing with the consequences of these road traffic crashes runs to billions of dollars. Current trends suggest that by 2030 road traffic deaths will become the fifth leading cause of death [1]. Most accidents caused by drivers are usually accompanied by speeding, alcohol and drugs, and distraction mainly due by mobile phone use. On the other hand, traffic jams are a major drawback of large and medium size cities that cause high levels of pollution while raising the cost of travel. It may be noted that the quality of life in modern industrialized societies depends to some extent on these factors.

In order to increase the level of traffic safety, reducing the traffic pollution, and reducing the congestion, the ITSs (Intelligent Transport Systems) and specifically VANETs (Vehicular Ad-Hoc Networks) is a matter of major importance. So it is necessary a method of dissemination of emergence messages generated by safety applications. Basically, the purpose of emergency

message dissemination in vehicular networks is to inform vehicles about dynamic road conditions in order to achieve an efficient and safe transportation system. To achieve this goal there are many broadcast protocols specially designed for vehicular environments [2]. This work is focused on analyzing the performance of these dissemination protocols.

## 1.2 Objectives

In this section we present the main objective of this Master's Thesis. After that, we present the contents of the work.

### 1.2.1 General objective

- Simulation of Basic Multi-Hop Broadcast Techniques in Vehicular Ad-Hoc networks using the VEINS [32] simulator.

### 1.2.2 Specific objectives

- Do a brief description of the concept of smart cities, smart cars and intelligent transport systems (ITS).
- Study and disclose of vehicular networks and their applications.
- Study of dissemination protocols used in VANETs.
- Do an exhaustive study of the VEINS [32] simulator.
- Simulate and analyze the performance of some basic protocols of multi-hop broadcast over VANETs.
- Do a comparison of the protocols analyzed based on the simulation's results.
- Provide a comprehensive guide to install and use the VEINS [32] simulator.

## 1.3 Contents and organization of the Master's Thesis

This report is divided into nine chapters plus four annexes with the code developed by our research group that we have used in this Master's Thesis. Each chapter is structured as follows:

### Chapter 1. Introduction

The general framework and the problem on which this study is developed are presented. Also the objectives to be achieved with this work are set.

## **Chapter 2. Smart cities and vehicles**

Smart city concept and its applications are introduced. Also, smart vehicle concept and its general architecture is defined.

## **Chapter 3. Intelligent Transport Systems**

A brief description of Intelligent Transport Systems is made which includes its objectives and framework. Besides, the classification and various applications are shown.

## **Chapter 4. Vehicular Ad-Hoc Networks**

In this chapter the concept of vehicle networks (VANETs) is defined. Also, the main components and communication architecture such as In-vehicle Communication, Vehicle to Vehicle, Vehicle to road Infrastructure, and Vehicle to Broadband cloud are described. Besides, a brief review of the applications and the standardization (Dedicated Short Range Communication, IEEE 1609 Standards for wireless access in vehicular environments) of VANETs is made.

## **Chapter 5. Message dissemination in Vehicular Ad-Hoc Networks**

This chapter provides a review of the concept of broadcast on the link layer. In addition some methods of multi-hop broadcast such as *Flooding*, *Counter-based scheme*, *Distance-based scheme*, *Location-based scheme*, *Neighbor knowledge scheme*, *Cluster-based scheme*, and *Probability scheme* are briefly described. Also, other methods like *Weighted p-persistence scheme*, *Slotted 1-persistence scheme*, and *Slotted p-persistence scheme* which reduce the broadcast storm problem are shown.

## **Chapter 6. Simulation of Vehicular Ad-Hoc Networks**

This chapter describes the concepts of mobility generators, network simulators, and VANETs simulators. Also, a brief description of the characteristics of the most popular simulators is made.

## **Chapter 7. VEINS simulator**

This chapter shows a depth study about the VEINS framework: components and basic paradigms on which it is built. It emphasizes the advantage provided by this tool, which is bidirectional link between network simulation and simulation of traffic.

## **Chapter 8. Simulation of vehicular multi-hop – basic techniques using VEINS simulator**

Once the VEINS simulator platform is installed, simulations of two techniques of retransmissions of broadcast messages are carried out. Then, the results and some comments of the simulations are shown.

## **Chapter 9. Conclusions and future work**

Finally, a compilation of the conclusions reached during the course of the work is made. Also, some ideas as future work are presented.

# Chapter 2

## Smart cities and vehicles

### 2.1 Smart City

#### 2.1.1 Definition

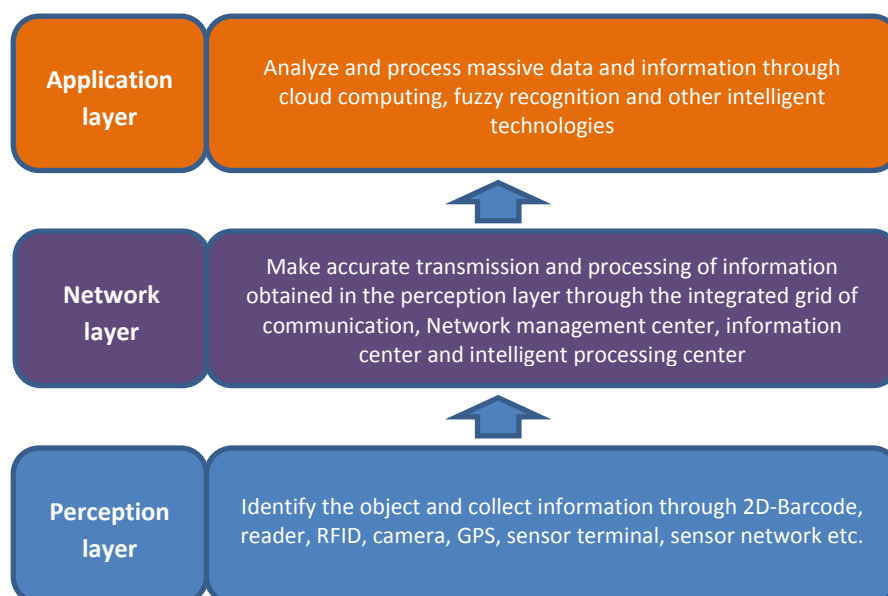
The definitions of smart city are various. The concept is used all over the world with different nomenclatures, context and meanings. A range of conceptual variants generated by replacing the word smart with adjectives such as digital or intelligent are readily used and reused [3]. Several working definitions have been put forward and adopted in both practical and academic use, as it is illustrated in Table 2.1.

Basically, the vision of Smart City is the urban center of the future, made safe, secure environmentally green, and efficient because all structures – whether for power, water, transportation, etc. are designed, constructed, and maintained making use of advanced, integrated materials, sensors, electronics, and networks which are interfaced with computerized systems comprised of database, tracking, and decision-making algorithms to stimulate sustainable economic growth and high quality of life for citizens [4, 5].

Working Definitions of a Smart City	
•	A city well performing in a forward-looking way in economy, people, governance, mobility, environment, and living, built on the smart combination of endowments and activities of self-decisive, independent and aware citizens.
•	A city that monitors and integrates conditions of all of its critical infrastructures, including roads, bridges, tunnels, rails, subways, airports, seaports, communications, water, power, even major buildings, can better optimize its resources, plan its preventive maintenance activities, and monitor security aspects while maximizing services to its citizens.
•	A city connecting the physical infrastructure, the IT infrastructure, the social infrastructure, and the business infrastructure to leverage the collective intelligence of the city.
•	A city striving to make itself smarter (more efficient, sustainable, equitable, and livable)
•	A city combining ICT and Web 2.0 technology with other organizational, design and planning efforts to dematerialize and speed up bureaucratic processes and help to identify new, innovative solutions to city management complexity, in order to improve sustainability and livability.
•	The use of Smart Computing technologies to make the critical infrastructure components and services of a city – which include city administration, education, healthcare, public safety, real estate, transportation, and utilities – more intelligent, interconnected, and efficient.

*Table 2.1 Working Definitions of a Smart City [3].*

The structure of smart city includes perception layer, network layer and application layer, which can make the future world increasingly appreciable and measurable, increasingly interconnection and interoperability and increasingly intelligent [6]. Figure 2.1 shows the technical architecture of smart city.



*Figure 2.1 Technical architecture diagram of smart city [6].*

The ICF (Intelligent Community Forum) annually announces cities awarded as Smart21 Communities, which earns high score in terms of five successful factors to be an intelligent community (i.e., broadband connectivity, knowledge workforce, digital inclusion, innovation, and marketing and advocacy). The Table 2.2 shows the list of the smart cities around the world.

Region	Cities
Africa	Nairobi County (Kenya); Cape Town, Nelson Mandela Bay (South Africa)
Asia Pacific	Ballarat, Coffs Harbour, Gold Coast City, Ipswich, Melbourne, Prospect, State of Victoria, Sunshine Coast, Whittlesea (Australia); Chongqing, Hong Kong, Jia Ding, Jiading New City, Shanghai, Tianjin, Tianjin Binhai New Area (China); Bangalore, Hyderabad, Jaipur (India); Ichikawa, MItaka, Yokosuka (Japan); Whanganui (New Zealand); Gangnam District, Hwa Seong Dong Tan, Seoul, Suwon (South Korea); Singapore (Singapore); Hsinchu City, Hwa Seong Dong Tan, New Taipei City, Seoul, Taichung City, Taipei, Taoyuan County (Taiwan)
Europe	Tirana (Albania); Tallinn (Estonia); Oulu (Finland); Besacon, Issy-les-Moulineaux (France); Frankfurt (Germany); Heraklion, Trikala (Greece); Sopron (Hungary); Reykjavik (Iceland); Isle of Man (Isle of Man); Malta (Malta); Eindhoven (Netherlands); Castelo de Vide, Evora (Portugal); Barcelona (Spain); Hammarby Sjostad, Vasteras, Kariskrona, Stockholm (Sweden); Birmingham, Dundee, Glasgow, London, Manchester, Sunderland (United Kingdom)
Middle East	Kabul (Afghanistan); Tel Aviv (Israel); Doha (Qatar); Dubai (United Arab Emirates)
North America	Burlington, Calgary, Edmonton, Fredericton, Kenora, Kingston, Moncton, Montreal Metro Area, Nunavut, Ottawa, Ottawa-Gatineau, Parkland County, Quebec City, Saint John, Stratford, Sudbury, Toronto, Vancouver, Waterloo, Western Valley, Windsor-Essex, Winnipeg (Canada); Durango, Tuxtla Gutierrez (Mexico); Adel, Albany, Arlington County, Ashland, Austin, Bettendorf, Bristol, Chattanooga, Cleveland, Columbus Region, Columbus, Corpus Christi, Dakota County, Danville, Dublin, Florida High Tech Corridor, LaGrange, Loma Linda, Mitchell, Monmouth, New York City, Northeast, Philadelphia, Riverside, San Francisco, Spanish Fork, Spokane, Walla Walla Valley, Westchester County, Windton-Salem (United States)
Middle / South America	Barceloneta (Puerto Rico), Curitiba, Pirai, Porto Alegre, Rio de Janeiro (Brazil)

*Table 2.2 The list of smart cities [7].*

## 2.1.2 Smart city applications

Generally, the construction of smart city can be divided into three levels, including the construction of public infrastructure, construction of public platform for smart city, the construction of application systems. Some typical applications are: Construction of Wireless City, Construction of Smart Home, Smart Public Services and Construction of Social Management, Construction of Smart

Transportation, Construction of Smart Medical Treatment, Construction of Smart Urban Management, Construction of Green City, Construction of Smart Tourism [6].

### 2.1.3 Construction of smart transportation

Every city can take good advantage of sensor network, the Internet of Things and other technical means according to their needs and traffic situation. So, every city can change the traditional transport system, and establish the smart traffic management system, including adaptive traffic signal control system, urban traffic control system and so on. The smart traffic management system can achieve the integration of urban planning, construction, management and operations, and provide comprehensive support for other subsystems of smart urban system [6].

## 2.2 Smart vehicles

### 2.2.1 Definition

With the introduction of WLAN (Wireless Local Area Network) technology such as the IEEE's 802.11 standard in recent years, many exciting applications are now becoming more and more feasible. One such application is the smart car or automated vehicle. In this application the user literally takes a back seat only specifying his or her destination. The vehicle then takes control and brings the user to their specified destination [8]. So, a vehicle is aware of its neighborhood including the presence and location of other vehicles that is an important evolution for the automotive industry.

Modern cars now possess a network of processors connected to a central computing platform that provides Ethernet, USB (Universal Serial Bus), Bluetooth, and IEEE 802.11 interfaces. Newer cars also have such features as:

- An EDR (Event Data Recorder), inspired by the black boxes found in airplanes. EDRs record all major data from the vehicle for crash reconstruction.
- A GPS (Global Positioning System) receiver, the accuracy of which can be improved by knowledge of road topology. GPS is currently used in many navigation systems.
- Front-End radar for detecting obstacles at distances far as 200 meters and short-distance radar or an ultrasound system, typically used for parking.

In conclusion, a smart car is a vehicle equipped with recording, processing, positioning, and location capabilities and it can run wireless security protocols. In Figure 2.2 it can be seen a smart vehicle's onboard instrumentation. The computing platform supervises protocol execution, including those

related to security. The communication facility supports wireless data exchange with other vehicles or fixed stations [9].

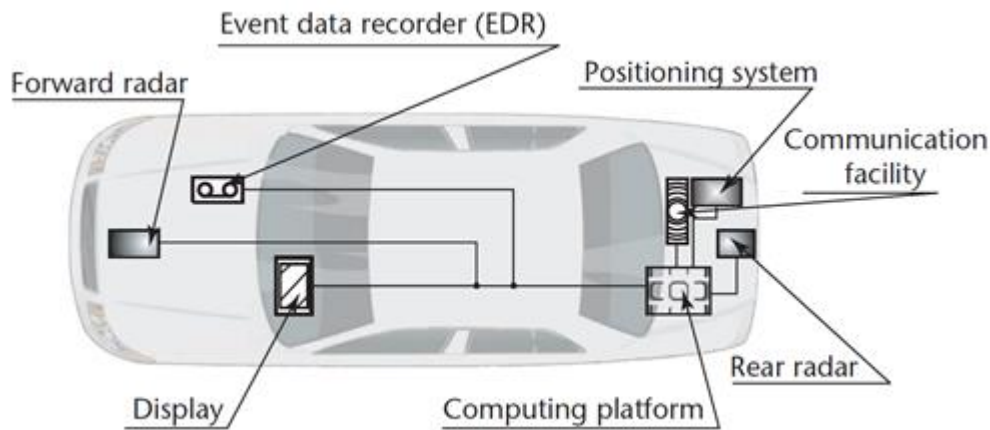


Figure 2.2 A smart car's onboard instrumentation [9].

### 2.2.2 General architecture

A smart car is a comprehensive integration of many different sensors, control modules, actuators, etc. A smart car can monitor the driving environment, assess the possible risks, and take appropriate actions to avoid or reduce the risk. In Figure 2.3 it can be seen a general architecture of a smart vehicle.

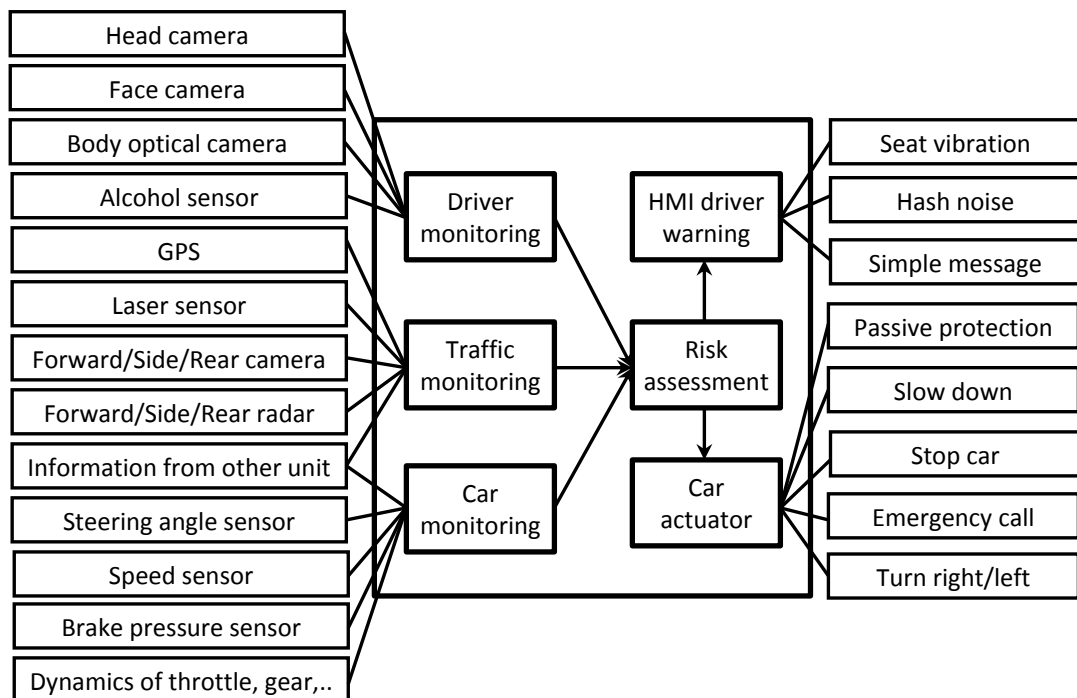


Figure 2.3 General architecture of a smart vehicle [10].

- **Traffic monitoring.** A variety of scanning technologies can be used to recognize the distance between the car and other road users. Active environments sensing in- and out-car will be a general capability in the near future. Lidar-radar or vision-based approaches can be used to provide the positioning information. The radar and lidar sensors provide information about the relative position and relative velocity of an object. Multiple cameras are able to eliminate blind spots, recognize obstacles, and record the surroundings. Besides, the car can get traffic information from the Internet or nearby cars.
- **Driver monitoring.** Drivers represent the highest safety risk. Smart cars present promising potentials to assist drivers in improving their situational awareness and reducing errors. With cameras monitoring the driver's gaze and activity, smart cars attempt to keep the driver's attention on the road ahead. Physiological sensors can detect whether the driver is in good condition.
- **Car monitoring.** The dynamics of a car can be read from the engine, the throttle and the brake. These data will be transferred by controller area networks to analyze whether the car functions normally.
- **Assessment module.** It determines the risk of the driving task according to the situation of the traffic, driver and car. Different levels of risk will lead to different responses, including notifying the driver through the HMI (Human Machine Interface) and taking emergency actions by car actuators.
- **HMI.** It warns the driver of the potential risks in non-emergent situations. For example, a tired driver would be awakened by an acoustic alarm or vibrating seat. Visual indications should be applied in a cautious way, since a complex graph or a long text sentence will seriously impair the driver's attention and possibly cause harm.
- **Actuators.** The actuators will execute specified control on the car without the driver's commands. The smart car will adopt active measures such as stopping the car in case that the driver is unable to act properly, or applying passive protection to reduce possible harm in abrupt accidents, for example, popping up airbags [10].

# Chapter 3

## Intelligent Transport Systems

### 3.1 Definition

IT (Information Technology) has transformed many industries, from education to health care to government, and is now in the early stages of transforming transportation systems. While many think improving country's transportation system solely means building new roads or repairing aging infrastructures, the future of transportation lies not only in concrete and steel, but also increasingly in using IT. IT enables elements within the transportation system, such as vehicles, roads, traffic lights, message signs, etc. to become intelligent by embedding them with microchips and sensors and empowering them to communicate with each other through wireless technologies [11].

ITS applies advanced technologies of electronics, communications, computers, control and sensing and detecting in all kinds of transportation system in order to improve safety, efficiency and service, and traffic situation through transmitting real-time information. In the leading nations in the world, ITS bring significant improvement in transportation system performance, including reduced congestion and increased safety and traveler convenience.

ITS include telematics and all types of communications in vehicles, between vehicles (e.g. car-to-car), and between vehicles and fixed locations (e.g. car-to-infrastructure). However, ITS are not

restricted to road transport, they also include the use of information and communication technologies for rail, water and air transport, including navigation systems, as it is shown in Figure 3.1. In general, the various types of ITS rely on radio services for communication and use specialized technologies [12].



*Figure 3.1 Intelligent transport systems [12].*

## 3.2 Objectives

Transport is in a foundation of every human civilization. Goods and people should be transferred from one place to another in safe, efficient and reliable way. The main objectives of ITS are:

- To improve traffic safety
- To relieve traffic congestion
- To improve transportation efficiency
- To reduce air pollution
- To increase the energy efficiency
- To promote the development of related industries

### 3.3 Framework

According to the concept framework of future ITS development planned by U.S. DOT and ITS-America, the relationship between ITS services was defined to ensure the compatibility and the interchangeability. 7 functions and 30 users services provided to drivers are defined, as it can be seen in Table 3.1.

Function	Users services
Travel and transportation management	<ul style="list-style-type: none"> <li>• Driving information during travel</li> <li>• Route guidance</li> <li>• Travel service information</li> <li>• Traffic control</li> <li>• Incident management</li> <li>• Emission monitoring and improvement</li> <li>• Rail road level crossing</li> </ul>
Travel demand management	<ul style="list-style-type: none"> <li>• Demand management and operation</li> <li>• Pre-trip information</li> <li>• Carpool matching and pre-booking</li> </ul>
Public transportation operation	<ul style="list-style-type: none"> <li>• Public transportation management</li> <li>• Public transportation information during travel</li> <li>• Personalized public transportation</li> <li>• The security of public transportation</li> </ul>
Electronic payment	<ul style="list-style-type: none"> <li>• Electronic payment service</li> </ul>
Commercial vehicle operation	<ul style="list-style-type: none"> <li>• The electronic customs clearance of commercial vehicle</li> <li>• Automatic security roadside inspection</li> <li>• Security monitoring in car</li> <li>• Commercial vehicle management program</li> <li>• The incident response of dangerous goods</li> <li>• Cargo flexibility</li> </ul>
Emergency management	<ul style="list-style-type: none"> <li>• Emergency notification and personal security</li> <li>• Emergency vehicle management</li> </ul>
Advanced vehicle control and safety system	<ul style="list-style-type: none"> <li>• Back-up collision prevention</li> <li>• Side collision prevention</li> <li>• Intersection collision prevention</li> <li>• The vision improvement of traffic accident prevention</li> <li>• Security preparation</li> <li>• Collision prevention before accident</li> <li>• Automatic highway system</li> </ul>

*Table 3.1 Functions and users services of ITS.*

### 3.4 Classification and applications of Intelligent Transport Systems

Given the wide range of ITS, it is useful to organize discussion of ITS applications through a taxonomy that arranges them by their primary functional intent. ITS applications can be grouped within five primary categories, as it can be seen in Table 3.2. These categories are: ATIS (Advanced

Traveler Information Systems), ATMS (Advanced Transportation Management Systems), Commercial Vehicle Operation (CVO), APTS (Advanced Public Transportation Systems), and Advanced Vehicle Control and Safety System (AVCSS) [11].

ITS Category	Specific ITS Applications
Advanced Traveler Information Systems (ATIS)	Real-time Traffic Information Provision
	Route Guidance/Navigation Systems
	Parking Information
	Roadside Weather Information Systems
Advanced Transportation Management Systems (ATMS)	Traffic Operations Centers (TOCs)
	Adaptive Traffic Signal Control
	Dynamic Message Signs (or “Variable” Message Signs)
	Ramp Metering
Commercial Vehicle Operation (CVO)	Electronic Toll Collection (ETC)
	Congestion Pricing/Electronic Road Pricing (ERP)
	Fee-Based Express (HOT) Lanes
	Vehicle-Miles Traveled (VMT) Usage Fees
	Variable Parking Fees
Advanced Public Transportation Systems (APTS)	Real-time Status Information for Public Transit System (e.g. Bus, Subway, Rail)
	Automatic Vehicle Location (AVL)
	Electronic Fare Payment (for example, Smart Cards)
Advanced Vehicle Control and Safety System (AVCSS)	Cooperative Intersection Collision Avoidance System (CICAS)
	Intelligent Speed Adaptation (ISA)

*Table 3.2 Classifying contactless mobile payments applications [11].*

### 3.4.1 Advanced Traveler Information Systems

ATIS, with advanced communication technology, makes road users can access real time information in the car, at home, in the office or outdoors as the reference of choosing transportation modes, travel trips and routes. The system mainly includes changeable message signs, HAR (Highway

Advisory Radio), GPS, the internet connection, telephone, fax, cable television, information kiosk and mobile etc.

### **3.4.2 Advanced Transportation Management System**

ATMS detects traffic situations, transmits them to control center via communication network, and then develops traffic control strategies by combining all kinds of traffic information. Furthermore, ATMS makes use of facilities to carry out traffic control and transmits the information to drivers and concerned departments, and implements traffic management measures, such as ramp metering, signal control, speed control, incident managements, electronic toll collection and high occupancy vehicle control and so on.

### **3.4.3 Commercial Vehicle Operation**

CVO applies the technology of ATMS, ATIS and AVCSS in commercial vehicle operation such as trucks, buses, taxis and ambulances in order to improve efficiency and safety. The system mainly includes automatic vehicle monitoring, fleet management, computer scheduling and electronic payment.

### **3.4.4 Advanced Public Transportation System**

APTS applies the technology of ATMS, ATIS and AVCSS in public transportation in order to improve the quality of service, and increase efficiency and the number of people who take public transportation. The system mainly includes automatic vehicle monitoring, VPS (Vehicle Positioning System), computer scheduling and E-tickets.

### **3.4.5 Advanced Vehicle Control and Safety System**

AVCSS applies advanced technologies in vehicles and roads, and helps drivers control vehicles in order to reduce accidents and improve traffic safety. The AVCSS mainly includes anti-collision warning and control, driving assistance, automatic lateral/longitudinal control, and the long-run plans of automatic driving and automatic highway system.

# Chapter 4

## Vehicular Ad-Hoc Networks

### 4.1 Definition

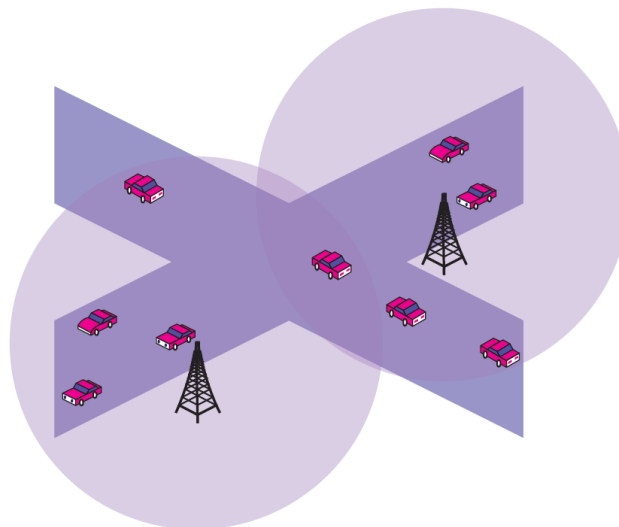
VANETs is part of MANETs (Mobile Ad-Hoc Networks), this means that every node can move freely within the network coverage and stay connected. In VANETs and MANETs, the nodes are mobile. However, the VANETs mobility is constrained to the roadside infrastructure, whereas MANETs movement is more random in nature. Nodes in VANETs are highly mobile and synchronize to fast topology changes having sufficient rechargeable battery power. VANETs is having facility of safety measure in vehicle, streaming communication between vehicles and telemetric device. It operates on Dedicated Short Range Communication (DSRC), with Wi-Fi, cellular, satellite and WiMAX (Worldwide Interoperability for Microwave Access) [13].

VANETs are emerging new technology to integrate the capabilities of new generation wireless networks to vehicles. The idea is to provide ubiquitous connectivity while on the road to mobile users, who are otherwise connected to the outside world through other networks at home or at the work place, and efficient vehicle-to-vehicle communications that enable the ITS. Therefore, VANETs are also called IVC (Inter-vehicle Communications) or V2V (Vehicle-to-Vehicle) communications [14].

ITS are the major applications of VANETs. Another important application for VANETs is providing Internet connectivity to vehicular nodes while on the move, so the users can download music, send emails, or play back-seat passenger games [14].

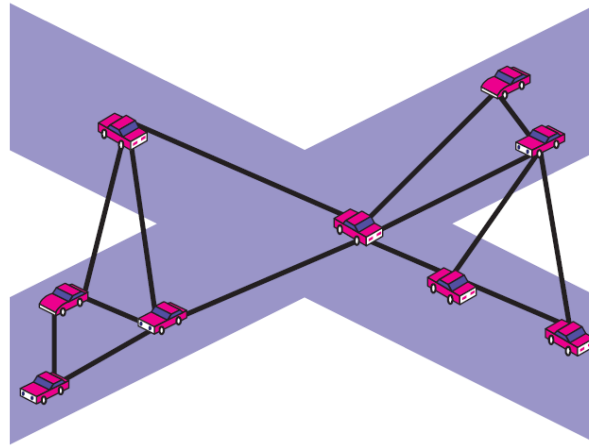
## 4.2 Architecture

MANETs generally do not rely on fixed infrastructure for communication and dissemination of information. VANETs follow the same principle and apply it to the highly dynamic environment of surface transportation. The architecture of VANETs falls within three categories: pure cellular/WLAN, pure ad hoc, and hybrid. VANETs may use fixed cellular gateways and WLAN access points at traffic intersections to connect to the Internet, gather traffic information or for routing purposes. The network architecture under this scenario is a pure cellular or WLAN structure, as it is shown in Figure 4.1. VANETs can combine both cellular network and WLAN to form the networks so that a WLAN is used where an access point is available and a 3G connection otherwise [14].



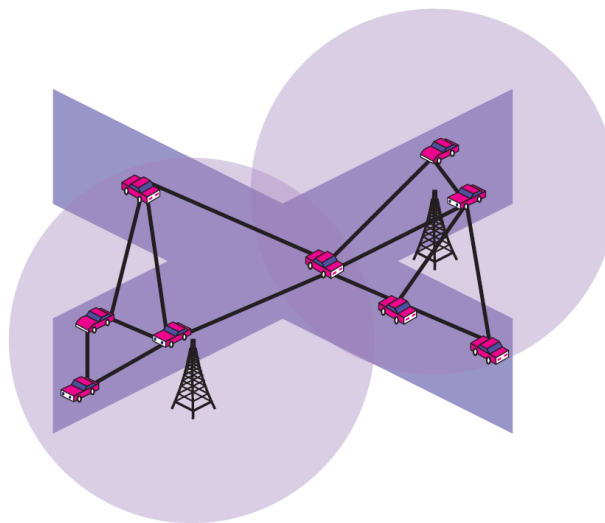
*Figure 4.1 WLAN/Cellular architecture [14].*

Stationary or fixed gateways around the sides of roads could provide connectivity to vehicles but are eventually unfeasible considering the infrastructure costs involved. In such a scenario, all vehicles and roadside wireless devices can form a mobile ad hoc network, as it is shown in Figure 4.2, to perform vehicle-to-vehicle communications and achieve certain goals, such as blind crossing (a crossing without light control) [14].



*Figure 4.2 Ad Hoc architecture [14].*

Hybrid architecture of combining cellular, WLAN and ad hoc networks together has also been a possible solution for VANETs, as it is shown in Figure 4.3. This architecture uses some vehicles with both WLAN and cellular capabilities as the gateways and mobile network routers so that vehicles with only WLAN capability can communicate with them through multi-hop links to remain connected to the world [14].



*Figure 4.3 Hybrid architecture [14].*

#### 4.2.1 Main Components

According to the IEEE 1471-2000 and ISO/IEC 42010 architecture standard guidelines, the VANETs system can be divided into three domains: the mobile domain, the infrastructure domain, and the generic domain.

As it can be seen in Figure 4.4, the mobile domain consists of two parts: the vehicle domain and the mobile device domain. The vehicle domain comprises all kinds of vehicles such as cars and buses. The mobile device domain comprises all kinds of portable devices like personal navigation devices and smartphones. Within the infrastructure domain, there are two domains: the roadside infrastructure domain and the central infrastructure domain. The roadside infrastructure domain contains roadside unit entities like traffic lights. The central infrastructure domain contains infrastructure management centers such as TMCs (Traffic Management Centers) and vehicle management centers [17].

However, the development of VANETs architecture varies from region to region. According to CAR-2-CAR Communication Consortium VANETs system architecture comprises three domains: in-vehicle, ad hoc, and infrastructure domain. As it is shown in Figure 4.5, the in-vehicle domain is composed of an on-board unit (OBU) and one or multiple application units (AUs). The connections between them are usually wired and sometimes wireless. However, the ad hoc domain is composed of vehicles equipped with OBUs and roadside units (RSUs). An OBU can be seen as a mobile node of an ad hoc network and RSU is a static node likewise. An RSU can be connected to the Internet via the gateway; RSUs can communicate with each other directly or via multi hop as well. There are two types of infrastructure domain access, RSUs and hot spots (HSs). OBUs may communicate with Internet via RSUs or HSs. In the absence of RSUs and HSs, OBUs can also communicate with each other by using cellular radio networks (GSM, GPRS, UMTS, WiMAX, and 4G) [17].

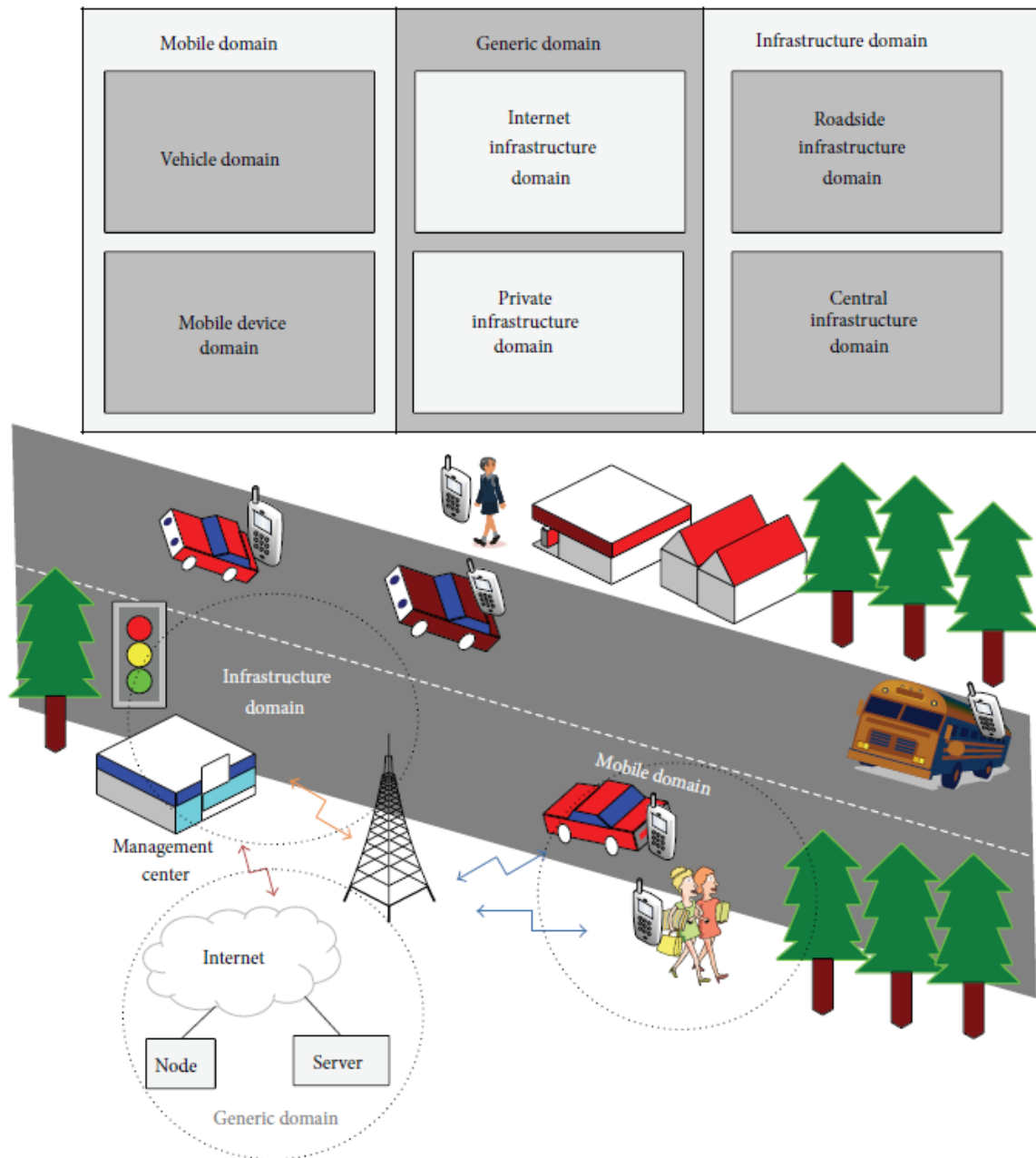


Figure 4.4 VANETs system domain [17].

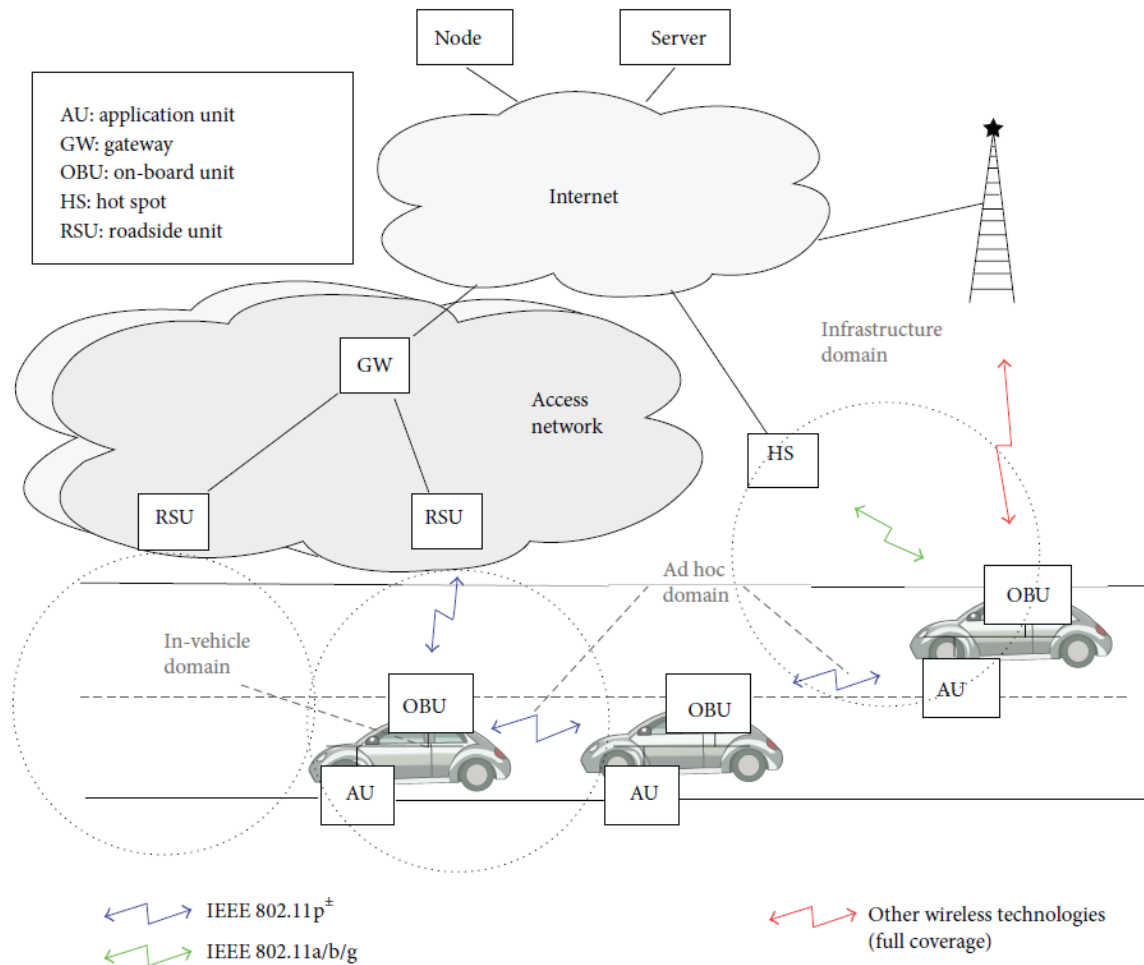


Figure 4.5 C2C-CC reference architecture [17].

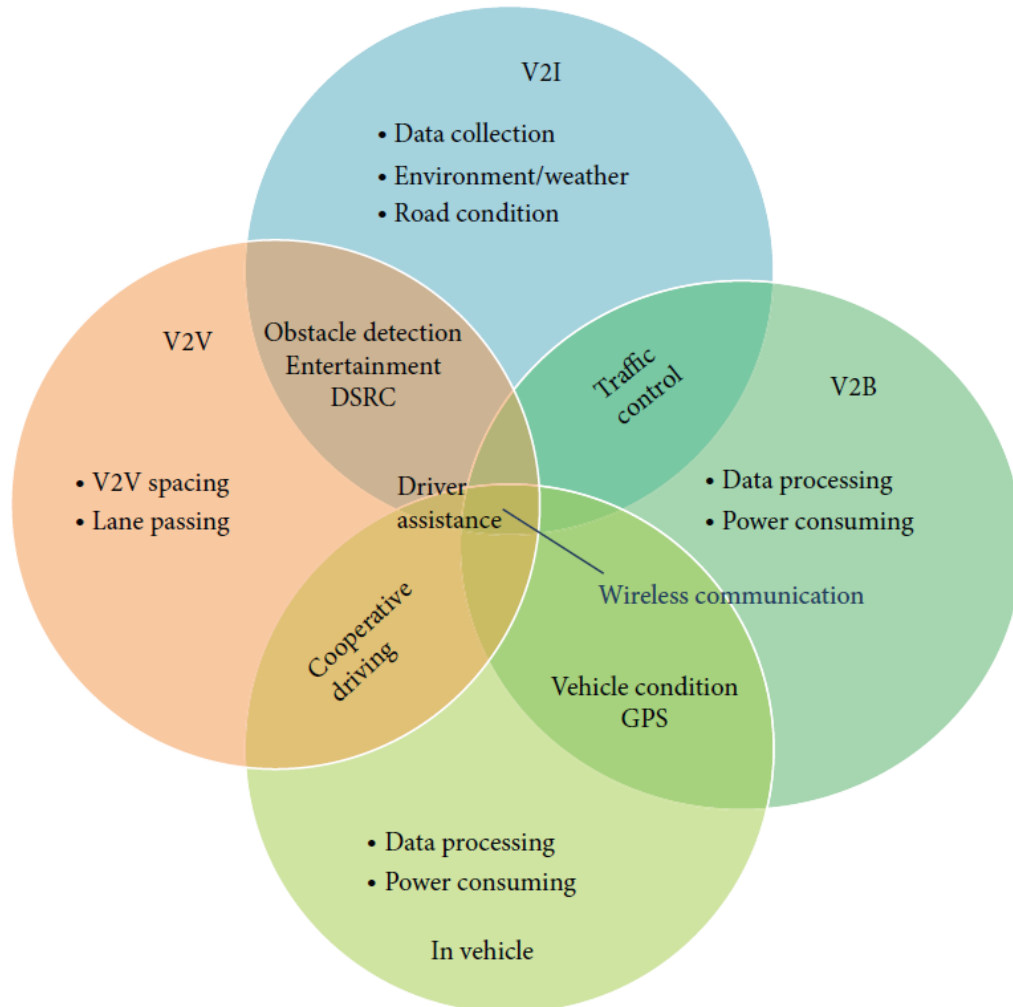
#### 4.2.2 Communication architecture

Communication types in VANETs can be categorized into four types [17]:

- **In-vehicle communication**, which is more and more necessary and important in VANETs research, refers to the in vehicle domain. In-vehicle communication system can detect a vehicle's performance and especially driver's fatigue and drowsiness, which is critical for driver and public safety.
- **Vehicle-to-vehicle (V2V)** communication can provide a data exchange platform for the drivers to share information and warning messages, so as to expand driver assistance.
- **Vehicle-to-road Infrastructure (V2I)** communication is another useful research field in VANETs. V2I communication enables real-time traffic/weather updates for drivers and provides environmental sensing and monitoring.
- **Vehicle-to-broadband cloud (V2B)** communication means that vehicles may communicate via wireless broadband mechanisms such as 3G/4G. As the broadband cloud may include

more traffic information and monitoring data as well as infotainment, this type of communication will be useful for active driver assistance and vehicle tracking.

Figure 4.6 describes the key functions of each communication type.



*Figure 4.6 key functions of each communication type [17].*

Figure 4.7 shows how Vehicular networks in the forms of Intra-Vehicle (InV), Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), and Vehicle-to-Broadband-cloud (V2B) communications will enable a variety of applications for safety, traffic efficiency, driver assistance, as well as infotainment [18].

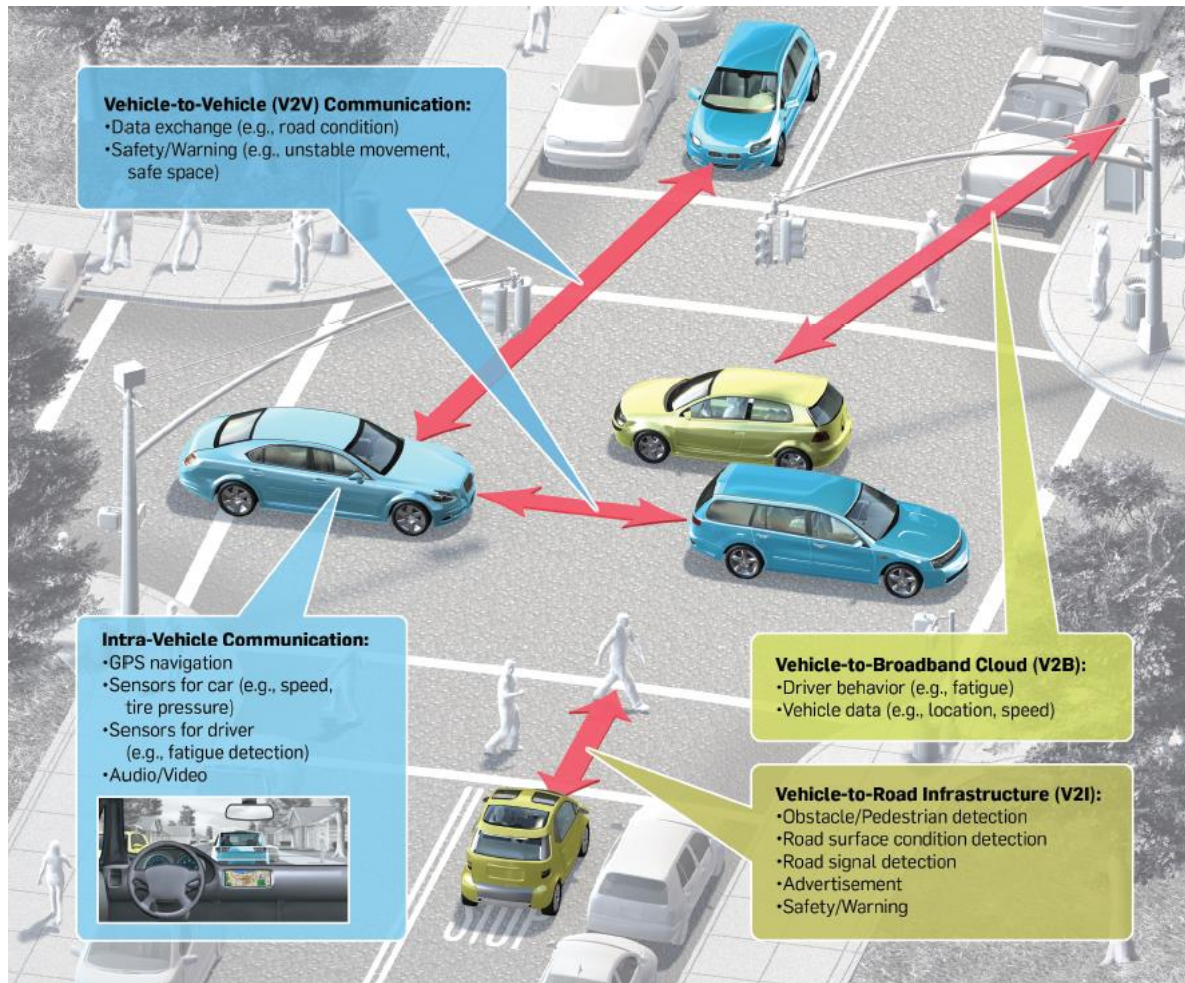


Figure 4.7 Communications in vehicle networks [18].

### 4.3 Vehicular network applications

Applications in vehicular environment usually can increase the road safety, improve traffic efficiency, and provide entertainment to passengers. In most cases, VANETs applications can be roughly organized into two major classes: Comfort applications and Safety applications [15]:

- **Comfort Applications:** This type of application improves passenger comfort and traffic efficiency and/or optimizes the route to a destination. Examples for this category are: traffic information system, weather information, gas station or restaurant location and price information, and interactive communication such as Internet access or music download.
- **Safety Applications:** Applications of this category increase the safety of passengers. The information is either presented to the driver or used to activate an actuator of an active safety system. Example applications of this class are: emergency warning system, lane-changing assistant, intersection coordination, traffic sign/signal violation warning, and road-

condition warning. Applications of this class usually demand direct vehicle-to-vehicle communication due to the stringent delay requirements.

A summary of the main applications of safety applications is presented in Table 4.1.

Safety Applications	Features			
	Communication	Messaging Type	Latency	Other requirements
Emergency Electronic Brake Lights	V2V	Event-triggered, time limited broadcast	100 ms	Range: 300m, high priority
Slow Vehicle Warning	V2V	Periodic permanent broadcast	100 ms	High priority
Pre-Crash Sensing	V2V	Periodic broadcast, unicast	50 ms	Range: 50 ms, high/mid priority for beaconing/unicast
Lane Change Warning	V2V	Periodic broadcast	100 ms	"Relative positioning accuracy: <2m range: 150 m"
Intersection Collision Warning	V2V-V2I	Periodic permanent broadcast	100 ms	Accurate positioning on a digital map, high priority
Hazardous Location Warning	V2V-V2I	Event-triggered time-limited Geo-Cast	100 ms	High priority

*Table 4.1 Safety applications in VANET [2].*

## 4.4 Standardization

Standards simplify product development, help reduce costs, and enable users to compare competing products. Only through the use of standards can the requirements of interconnectivity and interoperability be guaranteed and the emergence of new products can be verified to enable the rapid implementation of new technologies. There are many standards that relate to wireless access in vehicular environments. These standards range from protocols that apply to transponder equipment and communication protocols through to security specification, routing, addressing services, and interoperability protocols [16].

### 4.4.1 Dedicated Short Range Communication

DSRC is a short to medium range communications service that was developed to support vehicle-to-vehicle and vehicle-to-roadside communications. Such communications cover a wide range of applications, including vehicle-to-vehicle safety messages, traffic information, toll collection, drive-through payment, and several others. DSRC is aimed at providing high data transfers and low communication latency in small communication zones [16].

Wireless vehicular networks operating on the DSRC frequency bands are the main enabling technologies for the market of ITS. The FCC (Federal Communications Commission) allocated 75 MHz of spectrum at 5.9 GHz to be used by DSRC. It is a free but licensed spectrum. DSRC is free since the FCC does not charge for usage. The specifications defined by IEEE802.11p and IEEE1609 represent the most mature set of standards for DSRC/WAVE networks [2].

The DSRC spectrum is organized into 7 channels each of which is 10 MHz wide. One channel is restricted for safety communications only while two other channels are reserved for special purposes (such as critical safety of life and high power public safety). All the remaining channels are service channels which can be used for either safety or comfort applications. Safety applications are given higher priority over comfort applications to avoid their possible performance degradations and at the same time save lives by warning drivers of imminent dangers or events to enable timely corrective actions to be taken [16].

#### 4.4.2 IEEE 1609 Standards for wireless access in vehicular environments (WAVE)

The WAVE standards define architecture and a complementary, standardized set of protocols, services and interfaces that collectively enable secure vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) wireless communications. The primary goal was to develop public safety applications that can save lives and improve traffic flow, although other commercial services are also permitted [2].

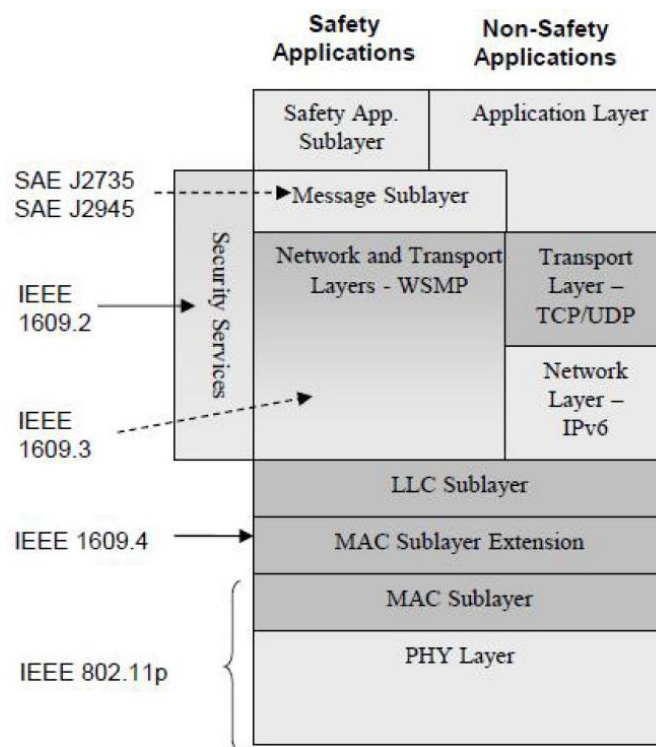


Figure 4.8 Overview of the WAVE stack and associated standards [2].

The layers of the WAVE protocol stack are shown in Figure 4.8. It is worth noting that IEEE 802.11p is limited by the scope of IEEE 802.11 which strictly works at the media access control and physical layers. The operational functions and complexity related to DSRC are handled by the upper layers of the IEEE 1609 standards. These standards define how applications that utilize WAVE will function in the WAVE environment, based on the management activities defined in IEEE P1609.1, the security protocols defined in IEEE P1609.2, and the network-layer protocol defined in IEEE P1609.3. The IEEE 1609.4 resides above 802.11p and this standard supports the operation of higher layers without the need to deal with the physical channel access parameters.

WAVE uses OFDM (Orthogonal Frequency Division Multiplexing) to split the signal into several narrowband channels to provide a data payload communication capability of 3, 4.5, 6, 9, 12, 18, 24 and 27 Mbps in 10 MHz channels [16].

## Chapter 5

# Message dissemination in Vehicular Ad-Hoc Networks

### 5.1 VANETs message dissemination

In VANETs safety applications, message dissemination typically refers to the process of spreading data over distributed wireless networks.

The dissemination of messages requires broadcast capabilities at the link layer, allowing a frame to be transmitted to all the vehicles in the radio scope. It also assumes implementation of network mechanisms to disseminate the message in the whole network. When the V2V communication is enabled, the message will be disseminated in a multi-hop technique and will be broadcasted by all the RSU when V2I communications are used. Also it is possible; RSUs broadcast the messages to some selected vehicles to forward the message to complete the dissemination [2].

### 5.2 Broadcast in the link layer

IEEE 802.11p/WAVE standard has a set of PHY (Physical) and MAC (Medium Access Control) layer specifications to enable communications in VANETs. The broadcast message is directly sent by the source vehicle to the vehicles in the radio range. In IEEE 802.11p, vehicles use a multichannel

concept for the delivery of safety-related and infotainment applications. Each vehicle periodically switch on a common control channel to monitor control and warning messages, and tune onto one of available service channels to exchange non safety-related data.

802.11p is known for not being able to manage the medium very efficiently, particularly in case of broadcast messages. Providing reliable delivery of broadcast messages in a VANETs introduces several key technical challenges such as: lack of acknowledgment, contention window size, hidden terminal problem, and multi-hop broadcasts.

So, MAC layer based on IEEE802.11p/WAVE plays a key role for data access control channels on the channel access mechanism. Channel access mechanism is directly related to congestion, channel access delay and reliability particularly in case of broadcast messages [2].

### 5.3 Vehicular multi-hop broadcast

This section describes the basic mechanisms used to efficiently disseminate messages to all the vehicles at several hops or in a certain geographic area. These mechanisms rely on the broadcast service offered by the IEEE 802.11 p, and must consequently compensate its lack of reliability. These solutions are the basic mechanisms used in more complex dissemination protocols [2].

#### 5.3.1 Blind flooding

A straight-forward approach to perform broadcast is by flooding. This scheme works as follows: the first time a vehicle receives a message broadcast, it retransmit it immediately in broadcast format, i.e. several vehicles retransmit the same messages with the same ID at the same time. Clearly, this costs  $n$  transmissions in a network of  $n$  vehicles [2].

#### 5.3.2 Counter-based scheme

When a host tries to rebroadcast a message, the rebroadcast message may be blocked by busy medium, back-off procedure, and other queued messages. There is a chance for the host to hear the same message again and again from other rebroadcasting hosts before the host actually starts transmitting the message [19]. Consequently, the vehicle senses the medium while it is waiting for the messages sent by its neighbors and counts the number of times it receives the same message. At the end of the waiting time, the vehicle rebroadcasts the message if it has received the message less than  $k$  times and discards it otherwise;  $k$  being a predefined threshold [2].

### 5.3.3 Distance-based scheme

This scheme uses the relative distance between hosts to make the decision. When a vehicle receives a message, it is able to measure the distance to the transmitter. It can be simply obtained from a GPS. The position of the transmitter is then included in the message and the distance computed as the difference between the receiver and the transmitter locations. In some cases, distance can also be evaluated from the radio signal strength at the receiver [2].

### 5.3.4 Location-based Scheme

This scheme uses the relative location of broadcasting vehicles to make the decision whether to drop a rebroadcast or not. Vehicles evaluate extra coverage area based on their location, if the additional area is greater than a threshold, vehicle will rebroadcast the message; otherwise it discards it. Such an approach may be supported by positioning devices such as GPS. In the context of VANETs, this scheme is very similar to the distance based scheme. However, the additional coverage is difficult to estimate in practice, since it depends on the radio environment (fading, shadowing, etc.) which is not known by the vehicles [2].

### 5.3.5 Neighbor knowledge scheme

This scheme is implemented via periodic hello messages to determine whether rebroadcast or not by that information. Most protocols require vehicles to share 1-hop or 2-hop neighborhood information with other vehicles [2].

### 5.3.6 Cluster-based scheme

This scheme is based on graph modeling [19]. The vehicles are divided in clusters and each one has a cluster head vehicle and cluster gateway vehicle. Once created the cluster, the algorithm for broadcasting will only allow the gateway or head using one of the following schemes: *Probability*, *Counter*, *Distance*, and *Location*, to retransmit emergency messages while the member will be inhibited from broadcasting, which minimizes broadcast [2].

### 5.3.7 Probability scheme

An intuitive way to reduce rebroadcasts is to use probabilistic rebroadcasting. Upon receiving a broadcast message for the first time, a host will rebroadcast it with probability  $P$  with  $0 < P \leq 100\%$ . Clearly, when  $P = 100\%$ , this scheme is equivalent to the *Flooding* [2]. Note that to respond to the contention and collision problems it is necessary to insert a small random delay (a number of slots) before rebroadcasting the message. So the timing of rebroadcasting can be differentiated [19].

## 5.4 The broadcast storm problem

In a CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) network, drawbacks of flooding include:

- **Redundant rebroadcasts.** When a mobile host decides to rebroadcast a broadcast message to its neighbors, all its neighbors already have the message.
- **Contention.** After a mobile host broadcasts a message, if many of its neighbors decide to rebroadcast the message, these transmissions (which are all from nearby hosts) may severely contend with each other.
- **Collision.** Because of the deficiency of backoff mechanism, the lack of RTS/CTS (Request to Send/Clear to Send) dialogue, and the absence of CD (Collision Detection), collisions are more likely to occur and cause more damage [19].

Excessive broadcast redundancy as a result of a broadcast storm leads to severe contention at the link layer, packet collisions, inefficient use of bandwidth and processing power, and, most important, service disruption due to high contention [20].

According [2] some authors proposed three distributed broadcast suppression methods: *Weighted p-persistence*, *Slotted 1-persistence*, and *Slotted p-persistence*.

### 5.4.1 Weighted p-persistence scheme

In the weighted p-persistence scheme, if vehicle  $V_j$  receives a packet from vehicle  $V_i$ , vehicle  $V_j$  first checks whether the packet has been received. If vehicle  $V_j$  receives this packet at the first time, then vehicle  $V_j$  has probability  $P_{ij}$  to re-broadcast the packet. Otherwise, vehicle  $V_j$  drops this packet.

$$P_{ij} = \frac{D_{ij}}{R}$$

Where:

$D_{ij}$  = Distance between vehicle  $V_i$  and  $V_j$

$R$  = Transmission range.

Neighbors of vehicle  $V_i$  change  $P_{ij}$  to 1 to ensure that the message must be broadcasted if they have not received the re-broadcast message after waiting a random time [21]. This scheme is illustrated in Figure 5.1.

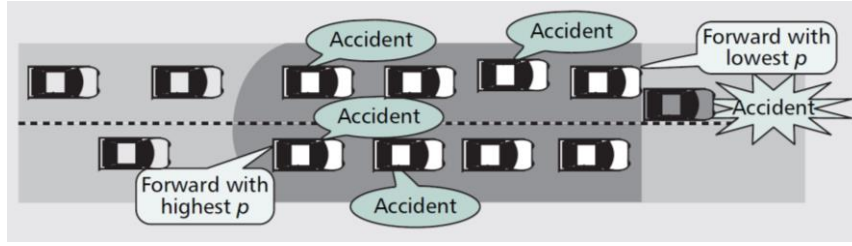


Figure 5.1 Weighted  $p$ -persistence [20].

#### 5.4.2 Slotted 1-persistence scheme

In the slotted 1-persistence scheme, If vehicle  $V_j$  firstly receives this packet from vehicle  $V_i$ , then vehicle  $V_j$  waits for  $T_{s_{ij}}$  time slots, vehicle  $V_j$ , has probability 1 to rebroadcast the packet [21].

$$T_{s_{ij}} = N_s \left( 1 - \left\lfloor \frac{\min(D_{ij}, R)}{R} \right\rfloor \right) * \tau$$

Where:

$N_s$  = Number of slots

$\tau$  = Propagation time for one hop transmission (medium access delay + propagation delay)

$\min(D_{ij}, R)$  = lowest value between  $D_{ij}$  and  $R$

The value of  $N$  should be carefully chosen according to the traffic density. For example it is recommended the value of 5 slots for traffic in rush hours and 3 slots for traffic in non-rush hours [2]. This scheme is illustrated in Figure 5.2.

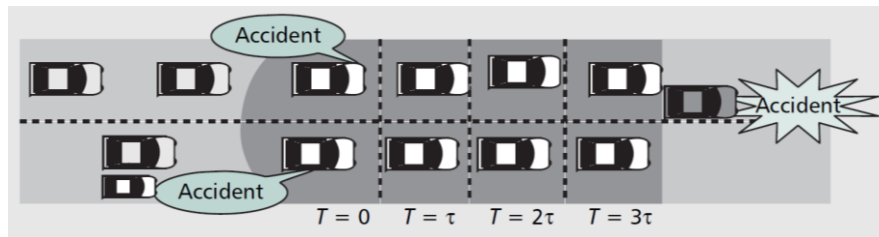


Figure 5.2 Slotted 1-persistence scheme [20].

#### 5.4.3 Slotted $p$ -persistence scheme

Slotted  $p$ -persistence scheme combines the weighted  $p$ -persistence and slotted 1-persistence schemes. If vehicle  $V_j$  firstly receives the packet from  $V_i$ , then vehicle  $V_j$  waits for  $T_{s_{ij}}$  time-slots. Vehicle  $V_j$ , has probability  $P_{ij}$  to rebroadcast the packet [21]. Figure 5.3 shows this scheme.

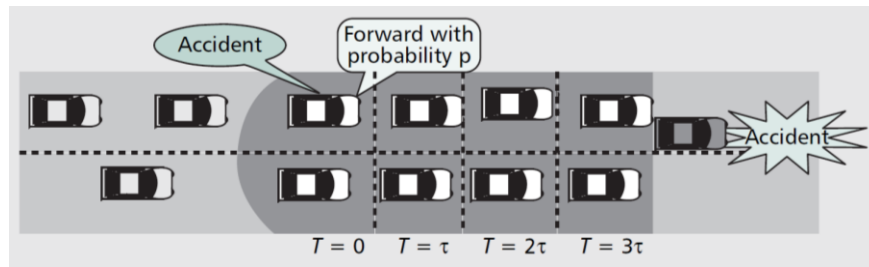


Figure 5.3 Slotted  $p$ -persistence scheme [20].

## 5.5 Farthest node scheme

By using GPS, a vehicle can know its location and that of the transmitter. A great deal of dissemination protocols uses this information to favor the farthest vehicles from the previous emitter as the next forwarder. It minimizes the number of redundant receptions and maximizes the coverage area [2].

## 5.6 Carry-store-forward mechanism

The approach to cope with disconnected networks is known as store- carry-forward. The main purpose of this mechanism is to assign selected vehicles the task of storing, carrying, and forwarding messages when new opportunities emerge. Many protocols for message dissemination complement its performance with store- carry-forward paradigm [2].

## Chapter 6

# Simulation of Vehicular Ad-Hoc Networks

### 6.1 Introduction

New protocols, scenarios and wireless technology schemes because of complexity and high expenses cannot be accomplished in large testbed. Due to this problem simulation of VANETs is essential to find capability of systems and new approaches. VANETs simulation requires two types of simulation components: Network and mobility. In most case network and mobility simulator are separate. There are several simulators available that can be used for VANETs simulation. VANETs simulation software can be classified into three different categories: (a) Mobility generators, (b) Network simulators, (c) Software which are integrated (a) and (b) or software can simulate both mobility and network (VANETs simulator). Figure 6.1 shows the classification of VANETs simulators [22].

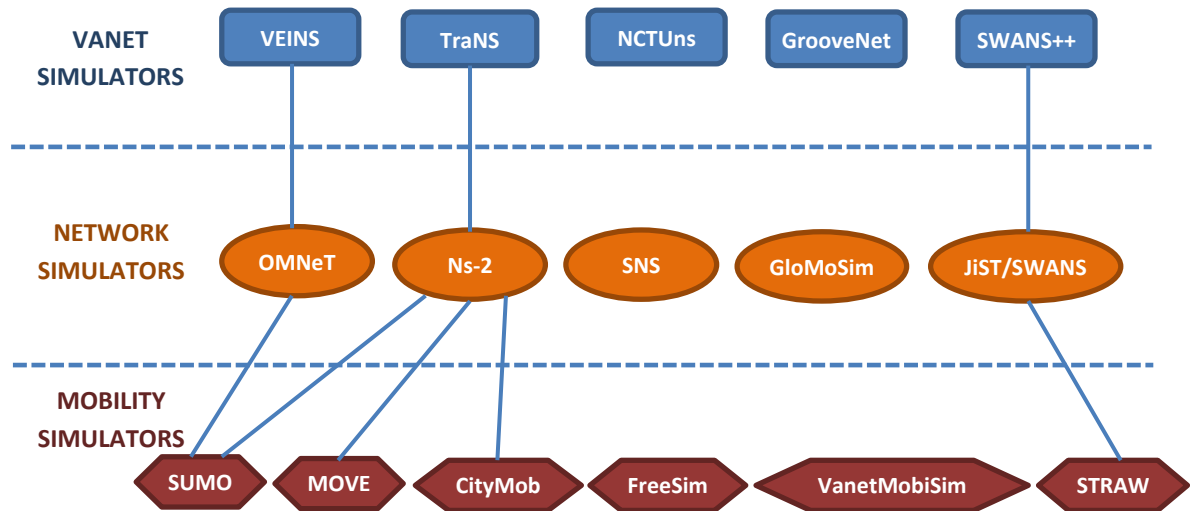


Figure 6.1 Classification of VANETs simulators [23].

## 6.2 Mobility generators

Traffic flow simulator generates required realistic vehicular mobility traces to be used in network simulator as an input.

Vehicular mobility simulations are usually categorized into two main types: microscopic and macroscopic. Macroscopic only focusing on mobility of flow of cars not each car individually. In Macroscopic simulation, the generations of vehicular traffic such as traffic density or traffic flows are defined. In the other types of the mobility models, microscopic approach, the movement of each individual vehicle and the vehicle behavior is important.

In microscopic model which is used in VANETs simulation, the required parameters for the mobility generator can be the roads map, scenario of cars traveling and some road and cars parameters like maximum cars speed, roads limitation, arrivals and departures times of each car, etc. Also, the output can be the coordinate of each vehicle at every time and their mobility parameters like speed, acceleration, etc.

### 6.2.1 SUMO

SUMO (Simulation of Urban Mobility) is an open source, highly portable, microscopic road traffic simulation package designed to handle large road networks. Its main features include collision free vehicle movement, different vehicle types, single-vehicle routing, multi-lane streets with lane changing, junction-based right-of-way rules, hierarchy of junction types, an open GUI (Graphical User Interface), and dynamic routing. SUMO can manage large environments, i.e., 10 000 streets, and it can import many network formats such as Visum, Vissim, ArcView, or XML-Descriptions. Thus,

by combining SUMO and openstreetmap.org, it is possible to simulate traffic in different locations of the globe. However, since SUMO is a pure traffic generator, its generated traces cannot be directly used by the available network simulators, which is a serious shortcoming [23].

### 6.2.2 FreeSim

It was created by the University of Southern California. FreeSim is a fully customizable macroscopic and microscopic free-flow traffic simulator that allows for multiple freeway systems to be easily represented and loaded into the simulator as a graph data structure with edge weights determined by the current speeds. Traffic and graph algorithms can be created and executed for the entire network or for individual vehicles or nodes, and the traffic data used by the simulator can be user generated or be converted from real-time data gathered by a transportation organization. Vehicles in FreeSim can communicate with the system monitoring the traffic on the freeways, which makes FreeSim ideal for ITS simulation. FreeSim is licensed under the GNU General Public License, and the source code is available freely for download [23].

### 6.2.3 STRAW

STRAW (Street Random Waypoint) provides accurate simulation results by using a vehicular mobility model on real US cities, based on the operation of real vehicular traffic. STRAW's current implementation is written for the JiST/SWANS discrete-event simulator, and its mobility traces cannot be directly used by other network simulators, such as ns-2. STRAW is part of the C3 (Car-to-Car Cooperation) project. A more realistic mobility model with the appropriate level of detail for vehicular networks is critical for accurate network simulation. The STRAW mobility model constrains node movement to streets defined by map data for real US cities and limits their mobility according to vehicular congestion and simplified traffic control mechanisms [23].

### 6.2.4 VanetMobiSim

This simulator was developed by the Institut Eurécom with the Politecnico di Torino. VanetMobiSim is an extension of the CanuMobiSim (CANU Mobility Simulation Environment) which focuses on vehicular mobility, and features realistic automotive motion models at both macroscopic and microscopic levels. At the macroscopic level, VanetMobiSim can import maps from the TIGER (Topologically Integrated Geographic Encoding and Referencing) database, or randomly generate them using Voronoi tessellation. VanetMobiSim adds support for multi-lane roads, separate directional flows, differentiated speed constraints and traffic signs at intersections. At the microscopic level, it supports mobility models such as IDM/IM (Intelligent Driving Model with Intersection Management), IDM/LC (Intelligent Driving Model with Lane Changing) and an

overtaking model (MOBIL), which interacts with IDM/IM to manage lane changes and vehicle accelerations and decelerations, providing realistic car-to-car and car-to-infrastructure interactions. VanetMobiSim is based on JAVA and can generate movement traces in different formats, supporting different simulation or emulation tools for mobile networks including NS-2, GloMoSim, and QualNet [23].

#### 6.2.5 MOVE

MOVE (Mobility model generator for Vehicular networks) rapidly generates realistic mobility models for VANETs simulations. MOVE is built on top of SUMO. The output of MOVE is a mobility trace file that contains information of realistic vehicle movements which can be immediately used by popular network simulation tools such as NS-2 or GloMoSim. In addition, MOVE provides a GUI that allows the user to quickly generate realistic simulation scenarios without the hassle of writing simulation scripts as well as learning about the internal details of the simulator [23].

#### 6.2.6 CityMob

CityMob is an NS-2 compatible mobility model generator proposed for use in VANETs. Citymob implements three different mobility models: (a) Simple Model (SM), (b) Manhattan Model (MM), and (c) realistic Downtown Model (DM). In DM model, streets are arranged in a Manhattan style grid, with a uniform block size across the simulation area. All streets are two-way, with lanes in both directions. Car movements are constrained by these lanes. Vehicles will move with a random speed, within an user-defined range of values. DM model also simulates semaphores at random positions (not only at crossings), and with different delays. DM adds traffic density in a way similar to a real town, where traffic is not uniformly distributed. Hence, there will be zones with a higher vehicle density. These zones are usually in the downtown, and vehicles must move more slowly than those in the outskirts [23].

### 6.3 Network simulator

Network simulator is usually used for simulation the computer networks. They are used for simulating the VANETs by evaluating the performance of network protocols for mobility of nodes and other required technique such as calculate and create the required components in a wireless network like detailed structure of all nodes (cars), sending and receiving packets roles, data traffic transmission, channels, etc.

Most currently used network simulators are developed for MANETs and hence require VANETs extensions (such as using the vehicular mobility generators) before they can simulate vehicular networks.

### 6.3.1 NS-2

NS-2 is a discrete event simulator developed by the VINT project research group at the University of California at Berkeley. The simulator was extended by the Monarch research group at Carnegie Mellon University to include: (a) node mobility, (b) a realistic physical layer with a radio propagation model, (c) radio network interfaces, and (d) the IEEE 802.11 MAC protocol using the Distributed Coordination Function (DCF).

However, the NS-2 distribution code had some significant shortcomings both in the overall architecture and the modeling details of the IEEE 802.11 MAC and PHY modules. The PHY is a full featured generic module capable of supporting any single channel frame based communications. The key features include cumulative Signal to Interference plus Noise Ratio (SINR) computation, preamble and Physical Layer Convergence Procedure (PLCP) header processing and capture, and frame body capture. The MAC now accurately models the basic IEEE 802.11 CSMA/CA mechanism, as required for credible simulation studies [23].

### 6.3.2 GloMoSim

GloMoSim is a scalable simulation environment for wireless and wired network. It has been designed using the parallel discrete-event simulation capability provided by Parsec. GloMoSim has been built using a layered approach similar to the OSI seven layer protocol model. Standard APIs are used between the different simulation layers. This allows the rapid integration of models developed at different layers by different people. The widely used QualNet simulator is a commercial version of GloMoSim [23].

### 6.3.3 SNS

Traditional wireless network simulators are limited in speed and scale because they perform many redundant computations both within a single simulation run, as well as across multiple invocations of the simulator. The staged simulation technique proposes to eliminate redundant computations through function caching and reuse. The central idea behind staging is to cache the results of expensive operations and reuse them whenever possible. SNS (Staged Network Simulator) is a staged simulator based on NS-2. On a commonly used ad hoc network simulation setup with 1500 nodes, SNS executes approximately 50 times faster than regular NS-2 and 30% of this improvement

is due to staging, and the rest to engineering. This level of performance enables SNS to simulate large networks. However, the current implementation is based on NS-2 version 2.1b9a, and it is not specifically designed to simulate VANETs scenarios [23].

#### 6.3.4 JiST/SWANS

JiST is a high performance discrete event simulation engine that runs over a standard Java virtual machine. It is a prototype of a new general purpose approach to building discrete event simulators that unifies the traditional systems and language-based simulator designs. It outperforms existing highly optimized simulation engines both in time and memory consumption. JiST converts an existing virtual machine into a simulation platform, by embedding simulation time semantics at the byte-code level. Thus, JiST simulations are written in Java, compiled using a regular Java compiler, and run over a standard, unmodified virtual machine.

SWANS is a scalable wireless network simulator built on top of the JiST platform. It was created primarily because existing network simulation tools are not sufficient for current research needs. SWANS contains independent software components that can be composed to form complete a wireless network or sensor network. Its capabilities are similar to NS-2 and GloMoSim, but SWANS is able of simulating much larger networks. SWANS leverages the JiST design to achieve higher simulation throughput, lower memory requirements, and run standard Java network applications over simulated networks. SWANS can simulate networks that are one or two orders of magnitude larger than what is possible with GloMoSim and NS-2, respectively, using the same amount of time and memory, and with a same level of detail [23].

#### 6.3.5 OMNeT++

OMNeT++ is an object-oriented modular discrete event network simulator. OMNeT++ has a component-based design, new features and protocols can be supported through modules. OMNeT++ supports network and mobility models through the independently developed Mobility Framework and INET Framework modules [24]. There are extensions for real-time simulation, network emulation, alternative programming languages (Java, C #), integration of databases, integration of SystemC, and other functions. OMNeT ++ is free for academic use and is already a platform widely used by the world scientific community.

### 6.4 VANET simulators

VANET simulators provide traffic and network simulation or can combine traffic and network simulator.

#### 6.4.1 GrooveNet

GrooveNet is an integrated simulator that supports multiple models that characterize communication, travel and traffic control to enable large scale simulations in street maps of any US city. The current limitations are that map database does not indicate one-way streets and the altitude of the street. GrooveNet is implemented in C++ and Qt graphics cross-platform library in Linux. GrooveNet is based on the TIGER database format and is able to dynamically load counties at run-time. On startup GrooveNet reads map database text files and converts the topology data into a binary encoded file with a graph structure [24].

#### 6.4.2 SWANS++

SWANS++ extends the network simulator SWANS by adding a GUI to visualize the scenario and a mobility model, STRAW for the vehicles movement in street scenarios. STRAW uses the simple RW mobility model, but it restricts the vehicles movement to real street boundaries, loaded from TIGER data files. The mobility model implemented in this simulator does not support lane changing and also it does not provide feedback between the mobility and networking modules [24].

#### 6.4.3 TraNS

TraNS (Traffic and Network Simulation Environment) is a GUI tool that integrates traffic and network simulators to generate realistic simulations of VANETs. TraNS allows the information exchanged in a VANETs to influence the vehicle behavior in the mobility model. There is an attempt to interface SUMO with NS-2 [15]. However, in our opinion, it is very expensive to understand the SUMO language and also unnecessary, because the communications engineer needs only a parsimonious model, easy to extend and/or modify [24].

#### 6.4.4 VEINS

VEINS (Vehicles in Network Simulation) is another simulator that couples a mobility simulator with a network simulator: SUMO is paired with OMNeT++ by extending SUMO to allow it to communicate with OMNeT++ through a TCP connection. In Veins, there is a manager module that is responsible for synchronizing the two simulators. This simulator has two separate events queues. At regular intervals, the manager module triggers the execution of one time step of the traffic simulation, receives the resulting mobility trace, and triggers position updates for all modules it had instantiated [24].

#### 6.4.5 NCTUns

NCTUns (National Chiao Tung University Network Simulator) was developed only as a network simulator, but the most recent version, integrates some traffic simulation capabilities, such as designing maps and controlling vehicles mobility. A large variety of maps can be designed using different types of supported road segments. The best feature available in NCTUns is that its network protocol stacks includes the Linux kernel protocol stack, including TCP/IP and UDP/IP, and the user level protocol stack and the MAC and PHY layer protocols. In this simulator, the code for the vehicles movement logic is integrated with the network simulation code, which makes it difficult to extend [24].

### 6.5 Selecting a convenient suit of simulators

When simulating a VANETs environment with large number of vehicles potentially transmitting several messages per second, the selection of a network communication simulator is crucial. Some important parameters to consider are user friendliness, scalability, and the interconnect ability of road traffic and network communication simulators.

VEINS was selected due to following features: Online re-configuration and re-routing of cars in reaction to network simulator, Fully-detailed models of IEEE 802.11p and IEEE 1609.4 DSRC/WAVE network layers, Supporting the realistic map and realistic traffic, User friendliness and interconnect ability. VEINS enables running of two simulators in parallel, connected via a TCP socket. VEINS framework developed based on MiXiM. MiXiM which is a framework for simulating wireless channels provides detailed models of wireless channels; connectivity, mobility and MAC layer protocols for OMNeT++. In Addition SUMO is an open source, microscopic and continuous road traffic simulation package designed to handle large road networks. Also SUMO accepts formats from different map providers. It is possible to choose popular OpenStreetMap (OSM) and JavaOpenStreetMap (JOSM) as interface and editor for OSM maps.

# Chapter 7

## VEINS Simulator

### 7.1 Introduction

VEINS is made up of two distinct simulators, OMNeT++ for network simulation and SUMO for road traffic simulation. To perform IVC evaluations, both simulators are running in parallel, connected via a TCP socket. The protocol for this communication has been standardized as the Traffic Control Interface (TraCI). This allows bidirectionally-coupled simulation of road traffic and network traffic. Movement of vehicles in the road traffic simulator SUMO is reflected in movement of nodes in an OMNeT++ simulation. Nodes can then interact with the running road traffic simulation, e.g. to simulate the influence of IVC on road traffic [32].

Aside from modules to model and to influence road traffic, Veins offers a comprehensive suite of IVC-specific models that can serve as a modular framework for simulating applications [32].

### 7.2 Network simulator: OMNeT ++

OMNeT++ is a C++ based discrete event simulator for modeling communication networks, multiprocessors and other distributed or parallel systems. The motivation of developing OMNeT++ was to produce a powerful open-source discrete event simulation tool that can be used by

academic, educational and research-oriented commercial institutions for the simulation of computer networks and distributed or parallel systems. This tool is available to the public since September 1997, it is a project born in the Department of Telecommunications of The University of Technology and Economics of Budapest [25].

### 7.2.1 Model structure

An OMNeT++ model consists of modules that communicate with message passing. The active modules are termed *simple modules*; they are written in C++, using the simulation class library. Simple modules can be grouped into *compound modules* and so forth; the number of hierarchy levels is not limited. Messages can be sent either via connections that span between modules or directly to their destination modules.

Both simple and compound modules are instances of *module types*. While describing the model, the user defines module types; instances of these module types serve as components for more complex module types. Finally, the user creates the system module as a network module which is a special compound module type without gates to the external world. When a module type is used as a building block, there is no distinction whether it is a simple or a compound module. This allows the user to transparently split a module into several simple modules within a compound module, or do the opposite, re-implement the functionality of a compound module in one simple module, without affecting existing users of the module type.

Modules communicate with messages which – in addition to usual attributes such as timestamp – may contain arbitrary data. Simple modules typically send messages via gates, but it is also possible to send them directly to their destination modules. Gates are the input and output interfaces of modules: messages are sent out through output gates and arrive through input gates. An input and an output gate can be linked with a connection. Connections are created within a single level of module hierarchy: within a compound module, corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module can be connected. Compound modules act as 'cardboard boxes' in the model, transparently relaying messages between their inside and the outside world. Properties such as propagation delay, data rate and bit error rate, can be assigned to connections.

Modules can have parameters. Parameters are mainly used to pass configuration data to simple modules, and to help define model topology. Parameters may take string, numeric or Boolean values [25]. Figure 7.1 shows the model structure in OMNeT++. Boxes represent simple modules

(thick border), and compound modules (thin border). Arrows connecting small boxes represent connections and gates.

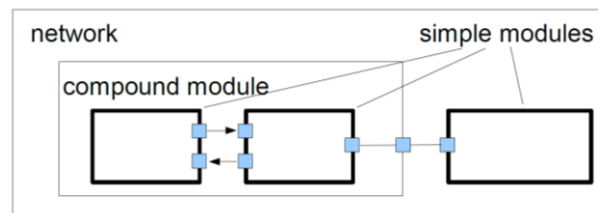


Figure 7.1 Model Structure in OMNeT++ [25].

### 7.2.2 Parallel Simulation Support

OMNeT++ also has support for parallel simulation execution. Very large simulations may benefit from the Parallel Distributed Simulation (PDES) feature, either by getting speedup, or by distributing memory requirements. If the simulation requires several Gigabytes of memory, distributing it over a cluster may be the only way to run it. For getting speedup (and not actually slowdown, which is also easily possible), the hardware or cluster should have low latency and the model should have inherent parallelism. Partitioning and other configuration can be configured in the INI file; the simulation model itself doesn't need to be changed (unless, of course, it contains global variables that prevent distributed execution in the first place.) The communication layer is MPI, but it's actually configurable, so if the user does not have MPI it is still possible to run some basic tests over named pipes. The Figure 7.2 explains the logical architecture of the parallel simulation kernel [25].

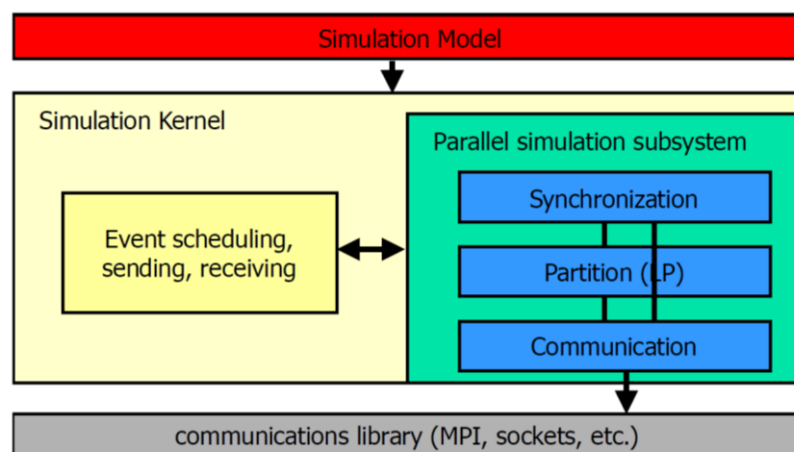


Figure 7.2 Logical Architecture of the OMNeT++ Parallel Simulation kernel [25].

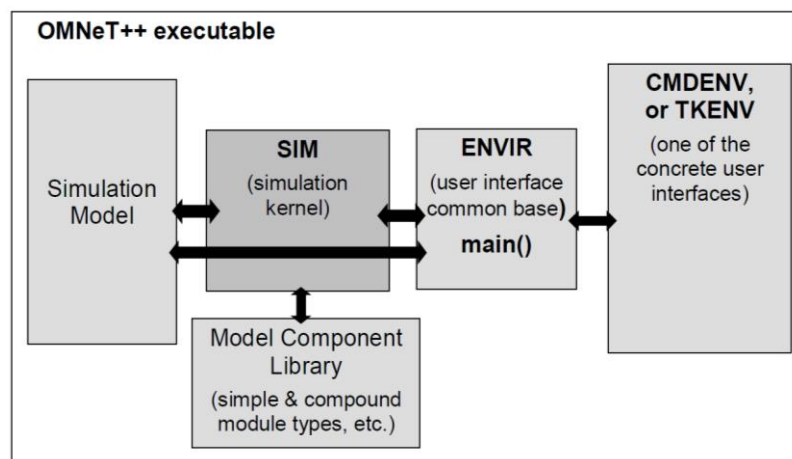
### 7.2.3 Simulation Model

OMNeT++ provides an extensive collection of libraries simulation, routines that serve to control these libraries, and a user interface that allows modeling, execution and debugging of simulations.

A simulation model in OMNeT++ consists of the following stages:

- Definition of topology and structure to be simulated, i.e. modules and interconnections through NED language.
- Definition of individual modules using C++, which are the active elements of the model.
- Compiling modules and linked with the library simulation.
- Specification of the particular parameters of the simulation.

Figure 7.3 shows the logical architecture of an OMNeT++ Simulation Program [25].



*Figure 7.3 Logical Architecture of an OMNeT++ Simulation Program [25].*

#### 7.2.4 Packet INET

INET framework is built on the platform OMNeT ++ and uses the same operating mode, that is, modules that communicate with each other by message passing.

INET Framework contains IPv4, IPv6, TCP, SCTP, UDP protocol implementations, and several application models. The framework also includes an MPLS model with RSVP-TE and LDP signaling. Link-layer models are PPP, Ethernet and 802.11. Static routing can be set up using network auto-configurators, or one can use routing protocol implementations.

The INET Framework supports wireless and mobile simulations as well. Support for mobility and wireless communication has been derived from the Mobility Framework [26].

#### 7.2.5 MiXiM

MiXiM is an OMNeT++ modeling framework created for mobile and fixed wireless networks (wireless sensor networks, body area networks, ad-hoc networks, vehicular networks, etc.). It offers

detailed models of radio wave propagation, interference estimation, radio transceiver power consumption and wireless MAC protocols (e.g. ZigBee) [27].

The model, which is used in the simulation VEINS framework, is based on real measurements using 802.11p/DSRC devices, with which it is possible to estimate the effect that buildings and other obstacles have on inter vehicular communication.

## 7.3 Traffic simulator: SUMO

The German Aerospace Center (DLR) started the development of the open source traffic simulation package SUMO back in 2001 [28].

### 7.3.1 Basic paradigms

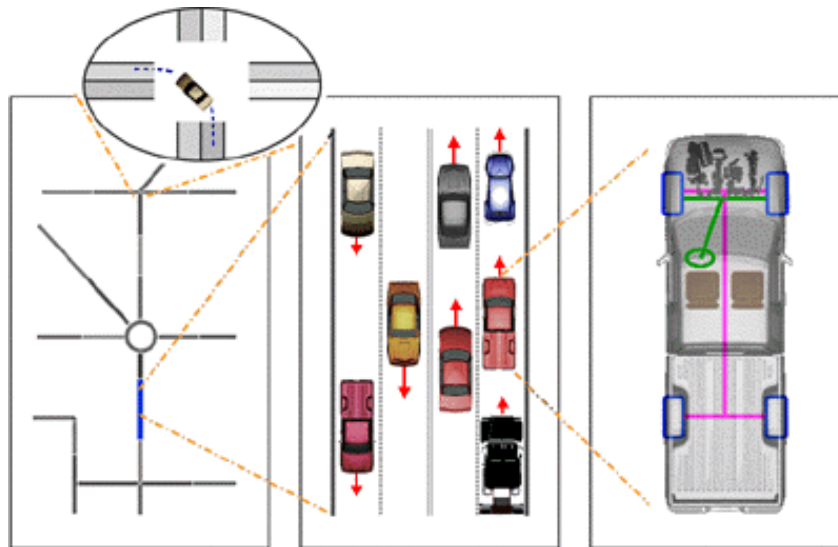
SUMO is conceived to simulate a traffic road network of the size of a city. As the simulation is multi-modal, which means that not only are car movements within the city modeled, but also public transport systems on the street network, including alternative train networks, the atomic part of the simulation is a single human being. This human being is described by a departure time and the route he/she takes which again is made up of sub-routes that describe a single traffic modality. Thus, a simulated person may take his/her car to the nearest public transportation system station and continue his travel by other means of transport. Apart from movements using motorized vehicles, a person may also walk. Walking is not simulated at all but is modeled estimating the time the person needs to reach the destination. Figure 7.4 displays a compound route.



Figure 7.4 Multimodality [29].

In traffic research, four classes of traffic flow models are distinguished according to the level of detail of the simulation. In *macroscopic* models traffic flow is the basic entity. *Microscopic* models simulate the movement of every single vehicle on the street, mostly assuming that the behavior of the vehicle depends on both, the vehicle's physical abilities to move and the driver's controlling behavior. Mesoscopic simulations are located at the boundary between microscopic and macroscopic simulations. Herein, vehicle movement is mostly simulated using queue approaches and single vehicles are moved between such queues. Sub-microscopic models regard single vehicles like microscopic, but extend them by dividing them into further substructures, which describe the

engine's rotation speed in relation to the vehicle's speed or the driver's preferred gear switching actions, for instance. This allows more detailed computations compared to simple microscopic simulations. However, sub-microscopic models require longer computation times. This restrains the size of the networks to be simulated. The Figure 7.5 shows the different simulation granularities; from left to right: macroscopic, microscopic, sub-microscopic, and within the circle: mesoscopic [30].



*Figure 7.5 The different traffic flow model [30].*

In the case of SUMO the traffic flow is simulated microscopically through the model developed by Stefan Krauss. This means, that every vehicle that moves within the simulated network is modeled individually and has a certain place and speed. In every time step which has duration of 1s, these values are updated in dependence to the vehicle ahead and the street network the vehicle is moving on.

The simulation of street vehicles in SUMO is time-discrete and space-continuous. Also, the car-driver model is continuous. When simulating traffic, the street attributes, such as maximum velocity and right of way rules are regarded [29].

### 7.3.2 Car-Driver Model

The model used currently within SUMO is the Gipps-model extension. It is capable of displaying main features of traffic like free and congested flow. In each time step the vehicle's speed is adapted to the speed of the leading vehicle in a way that yields to a collision-free system behavior. This velocity is called the safe velocity  $V_{safe}$ , and is computed using the following equation:

$$V_{safe}(t) = v_1(t) + \frac{g(t) - v_1(t) * \tau}{\frac{\bar{v}}{b(\bar{v})} + \tau}$$

Where:

$v_1(t)$  = Speed of the leading vehicle in time  $t$ .

$g(t)$  = Gap (distance) to the leading vehicle in time  $t$ .

$\tau$  = Driver's reaction time (usually 1s)

$b$  = Deceleration function.

To bind the acceleration to the vehicle's physical abilities, the resulting "wished" or "desired" speed is computed as the minimum of the vehicle's possible maximum velocity, the vehicle's speed plus the maximum acceleration with the safe velocity computed as shown above, therefore a vehicle will not "drive" or "accelerate" faster than is possible for it:

$$v_{des}(t) = \min[v_{safe}(t), v(t) + a, v_{max}]$$

Further, the driver is simulated by assuming he is making errors and so fails to perfectly adapt to the desired velocity. This is done by subtracting a random "human error" from the desired speed:

$$v(t) = \max[0, \text{rand}[v_{des}(t) - \varepsilon a, v_{des}(t)]]$$

As the vehicle must not drive backwards, once again – after the previous computations – the maximum of the computed speed and zero must be taken and will be the vehicle's final speed for the current time step [29].

### 7.3.3 Traffic Lights

Traffic lights play an important role within the traffic management as they improve traffic flow. Apart from simple right-of-way rules, each simulated junction may also be a junction with traffic lights [29].

### 7.3.4 Features

The simulation platform SUMO offers many features:

- Microscopic simulation - vehicles, pedestrians and public transport are modeled explicitly
- Online interaction – control the simulation with TraCI
- Simulation of multimodal traffic, e.g., vehicles, public transport and pedestrians
- Time schedules of traffic lights can be imported or generated automatically by SUMO
- No artificial limitations in network size and number of simulated vehicles

- Supported import formats: OpenStreetMap, VISUM, VISSIM, NavTeq
- SUMO is implemented in C++ and uses only portable libraries [31]

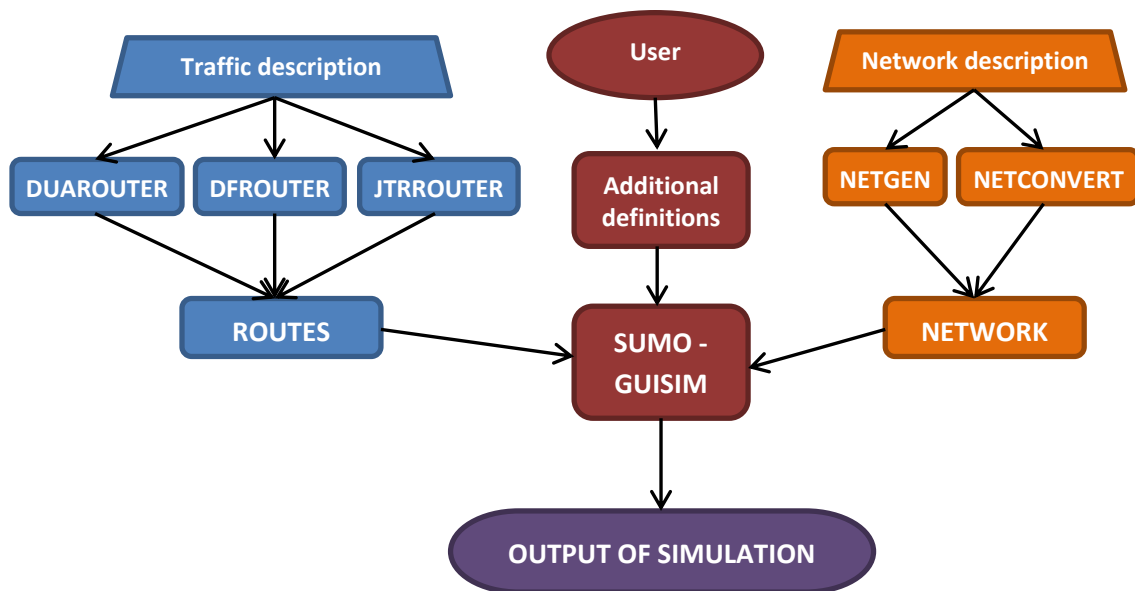
### 7.3.5 Components

Table 7.1 displays the applications which are contained in the SUMO package.

Application	Description
SUMO	Command line simulation
GUISIM	Simulation with a graphical user interface
NETCONVERT	Network importer
NETGEN	Abstract networks generator
OD2TRIPS	Converter from O/D matrices to trips
JTRROUTER	Routes generator based on turning ratios at intersections
DUAROUTER	Routes generator based on a dynamic user assignment
DFROUTER	Route generator with use of detector data
MAROUTER	Macroscopic user assignment based on capacity functions

*Table 7.1 Applications in the SUMO package [31].*

Figure 7.6 illustrates the relationships between these applications.



*Figure 7.6 Scheme of applications in SUMO.*

Depending on the application, configuration files have different extensions and it is important to consider. These extensions are:

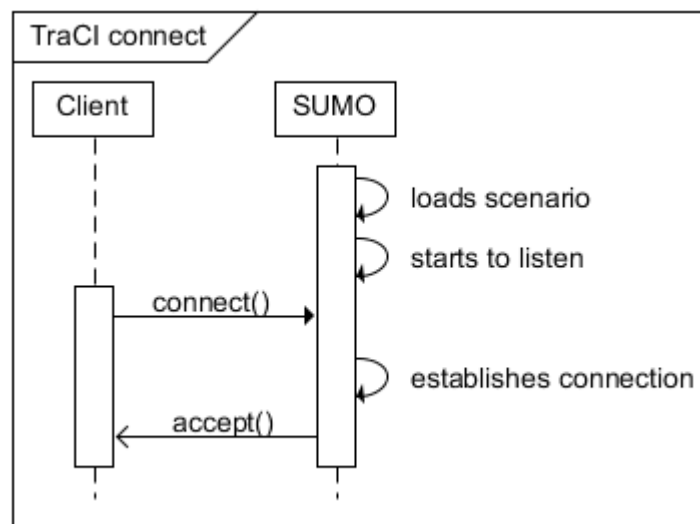
- \*.sumo.cfg: SUMO, GUISIM
- \*.netc.cfg: NETCONVERT

- \*.netg.cfg: NETGEN
- \*.rou.cfg: DUAROUTER
- \*.jtr.cfg: JTRROUTER
- \*.df.cfg: DFROUTER
- \*.od2t.cfg: OD2TRIPS

## 7.4 TraCI

TraCI gives the access to a road traffic simulator, in this case SUMO. TraCI allows to retrieve values of simulated objects and to manipulate their behavior on-line.

Based on architecture of type TCP client-server it is possible to access to TraCI, which acts as a server and the client is OMNeT++. After starting SUMO, a client connects to SUMO by setting up a TCP connection to the appointed SUMO port. Figure 7.7 shows the establishment of connection with SUMO [33].



*Figure 7.7 Establishing a connection to SUMO [33].*

The client application sends 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 to each command and additional results that depend on the given command.

The client has to trigger each simulation step in SUMO using the Command 0x02: Simulation Step. If any subscriptions have been done, the subscribed values are returned.

The client is responsible for shutting down the connection using the Command 0x7F: Close, as it is shown in Figure 7.8. The simulation will end, freeing all resources.

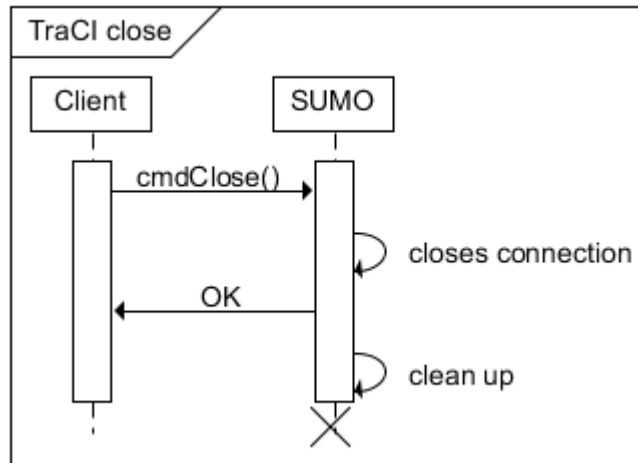


Figure 7.8 Closing a connection to SUMO.

#### 7.4.1 Sumo-launchd

Sumo-launchd is a proxy application that runs as a daemon. It accepts TCP connections from OMNeT++ and SUMO. The structure of communication between the simulators is illustrated in Figure 7.9 [2].

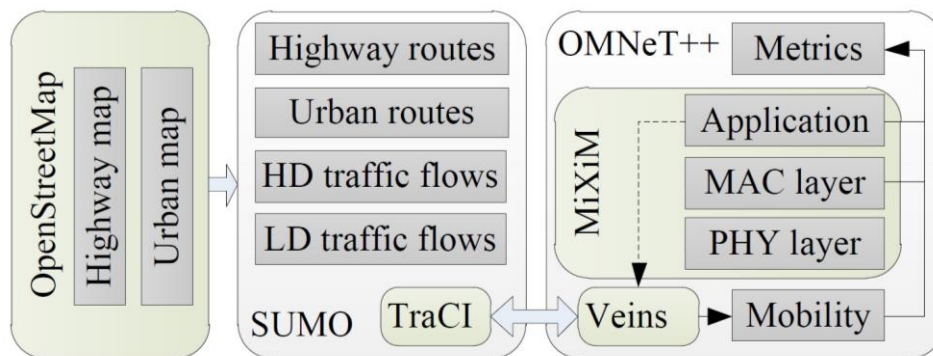
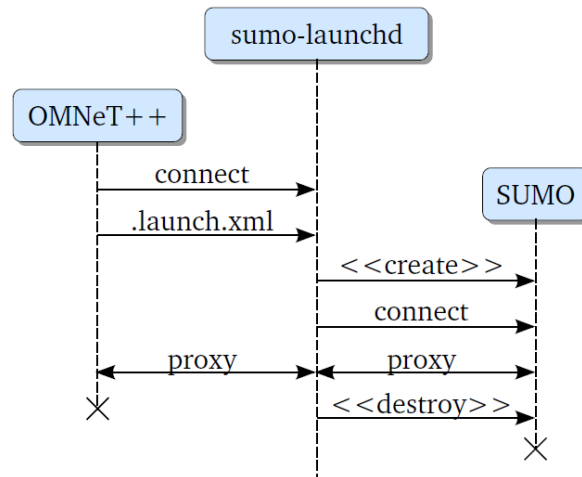


Figure 7.9 VEINS, simulation model [34].

OMNeT++ using the TCP protocol sends commands to control the state of the simulation SUMO thus influences the behavior of the vehicles. Next, SUMO executes the commands and responds with information mobility vehicles for simulating OMNeT++. Therefore, SUMO only runs commands when OMNeT++ has finished all simulation processes and at the end of the simulation the TCP session is closed. Figure 7.10 shows how sumo-launchd works in more details [2].



*Figure 7.10 Lifecycle management functionality of sumo-launchd [35].*

## Chapter 8

# Analysis of simulation results of vehicular multi-hop broadcast techniques using VEINS

### 8.1 Introduction

Both Urban and Highway scenarios have been considered in this Master's Thesis. In each scenario the number of nodes varies in the range of 10, 20, 50 and 100 with different traffic flows. Also, three broadcasting techniques are used in each scenario: *Flooding*, *Counter* and *Probability* with different configuration parameters. The implementations of these techniques were made by Cristhian Iza, who is a Ph.D. student at the UPC. The details of implementation of the code files are shown in the Annex D.

The performance metrics to be observed are:

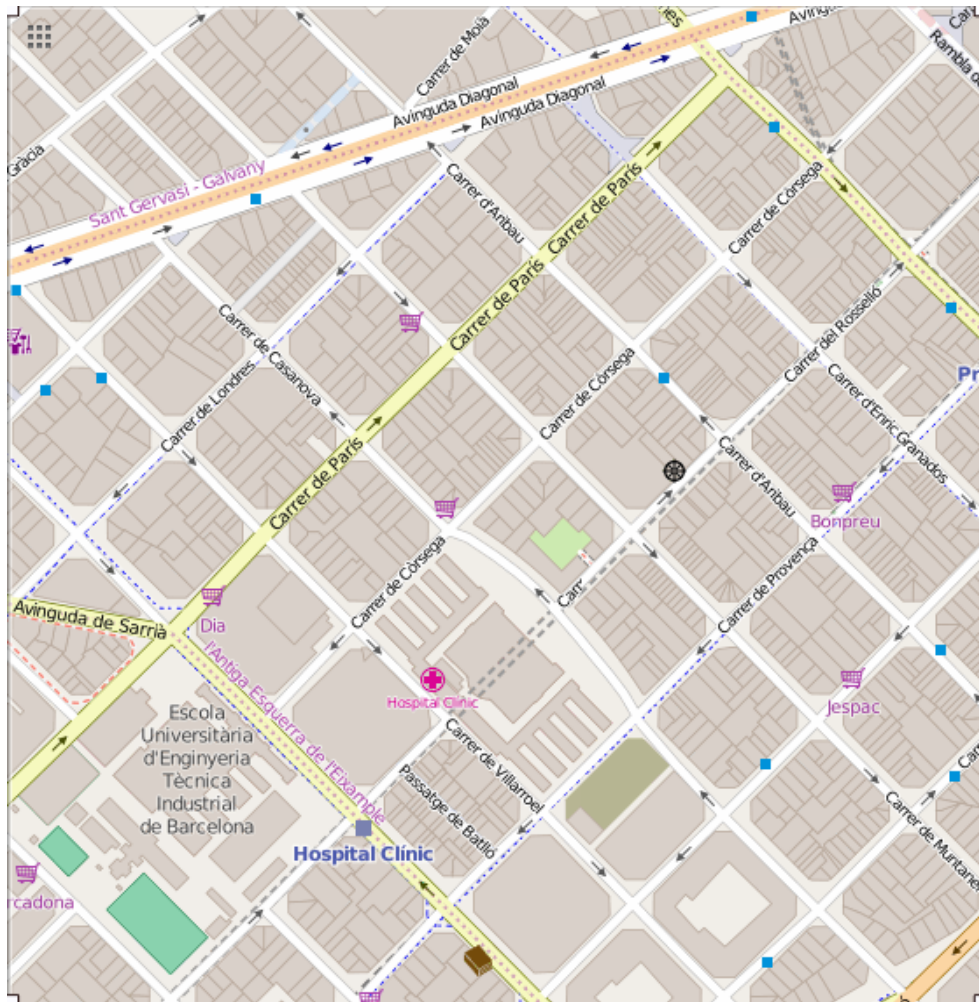
- *Number of retransmitting nodes*: It is the number of nodes which rebroadcast the message during of the message dissemination.
- *Percentage of reached nodes*: It is the percentage of nodes which receive the message.

- *Amount of sent packets*: It is the amount of packets which are sent during of the message dissemination.
- *Delay*: It is the total busy time of all nodes in the network.

Besides, all the figures are presented with confidence intervals (CI) of 90%, obtained from 10 simulations per each value of vehicle density using different movement traces per each simulation.

## 8.2 Urban scenario

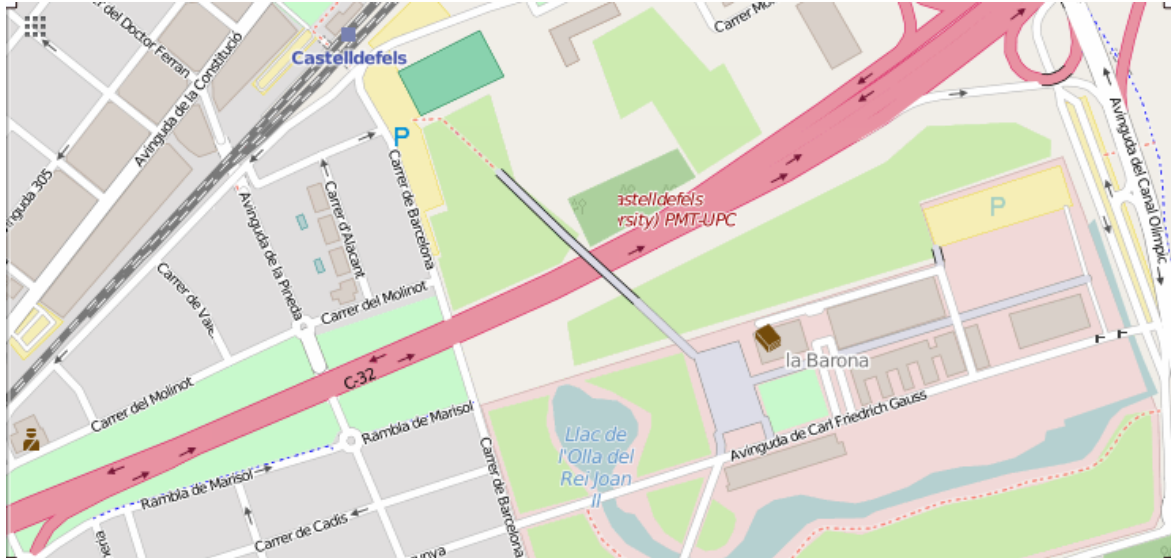
This scenario is a part of Barcelona city in Spain extracted from the OpenStreetMap [36]. Specifically, this scenario is a zone of the Eixample district. The dimensions are 1 km x 1 km. Figure 8.1 shows this piece of a generic Manhattan-style urban map.



*Figure 8.1 Urban scenario. Part of the Eixample district. Barcelona, Spain.*

### 8.3 Highway scenario

This scenario is a part of the C-32 Highway located in Castelldefels, Barcelona, Spain, which was extracted from the OpenStreetMap [36]. The dimension is 2 km of large. Figure 8.2 shows this map.



**Figure 8.2 Highway scenario. Part of the C-32 Highway. Barcelona, Spain.**

## 8.4 Flooding technique

This technique was described in section 5.3.1. Basically, each node that receives a packet rebroadcasts it immediately.

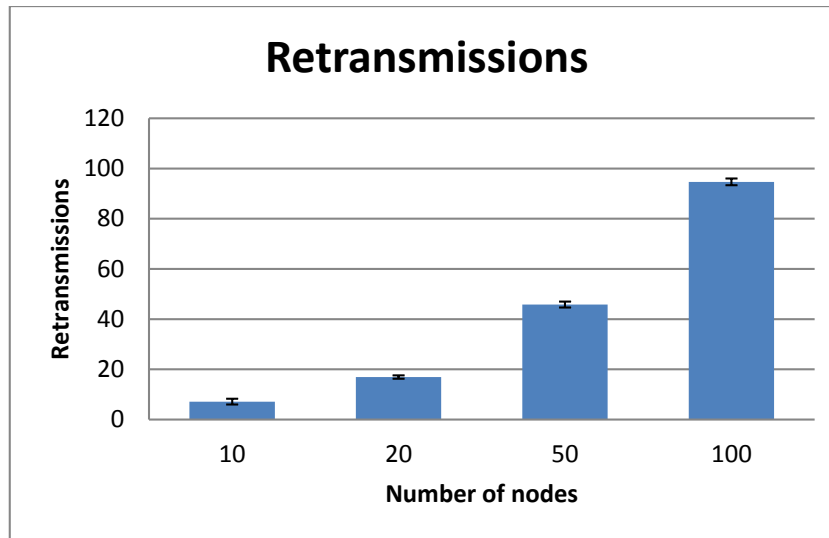
#### 8.4.1 Flooding technique in the Urban scenario

The simulation results for the *Flooding* technique in the Urban scenario are shown in Table 8.1.

	10 nodes	20 nodes	50 nodes	100 nodes
Retransmissions	7.1	16.9	45.8	94.7
Sent packets	30.2	158.7	909.8	2665.9
% Received packets	100	88.895	68.577	55.165
% Lost packets	0	11.105	31.423	44.835
Delay (ms)	0.448	2.439	2.09	2.453
% Reached nodes	85	93	97.8	99.8

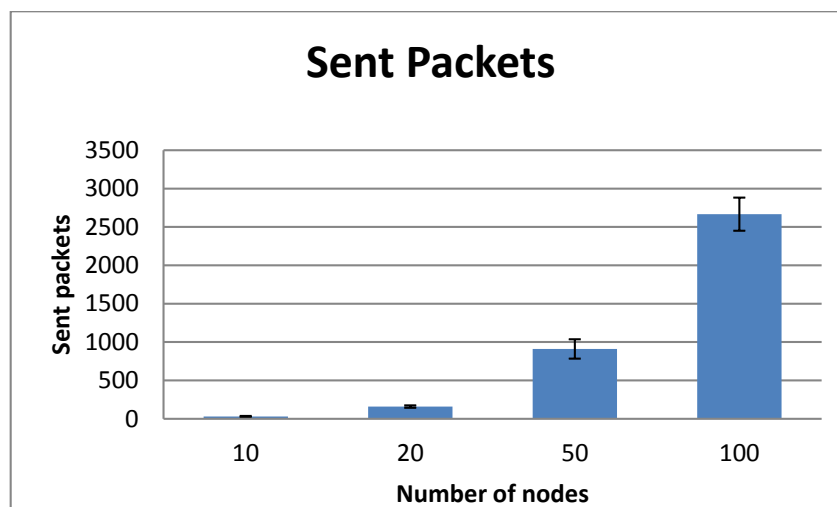
**Table 8.1. Simulation results of the classical Flooding technique in the Urban scenario.**

As it can be seen in Figure 8.3, the number of retransmitting nodes is very similar to the number of nodes in the scenario. Practically, between 71% and 95% of the nodes retransmit the packet as a broadcast message. This is what happens with the *Flooding* technique.



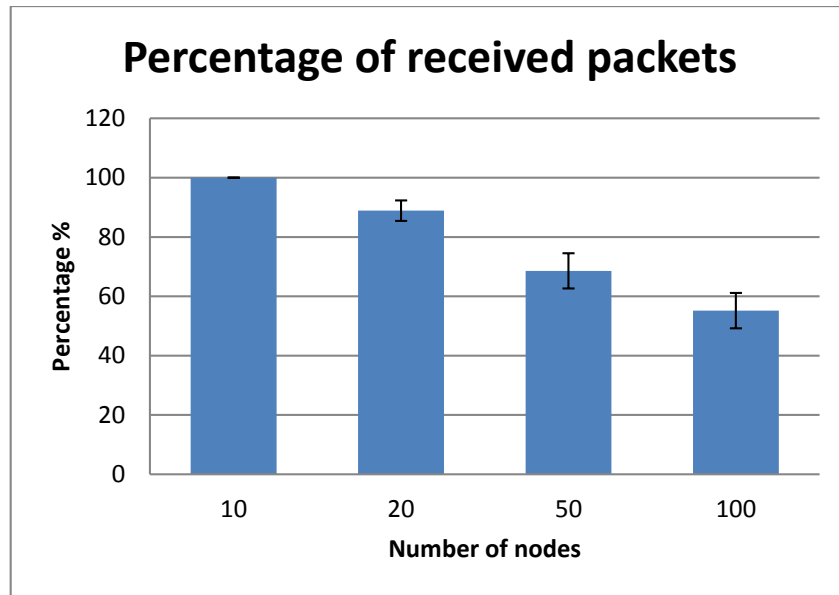
*Figure 8.3 The classical Flooding technique. Number of retransmitting nodes. CI 90%. Urban scenario.*

As a result of that, the number of sent packets increases too much. For example, the number of sent packets reaches 2666 for 100 nodes, which is so much in comparison with the 30 sent packets for 10 nodes as it is illustrated in Figure 8.4. As a conclusion, we can see that the *Flooding* technique scales very bad.

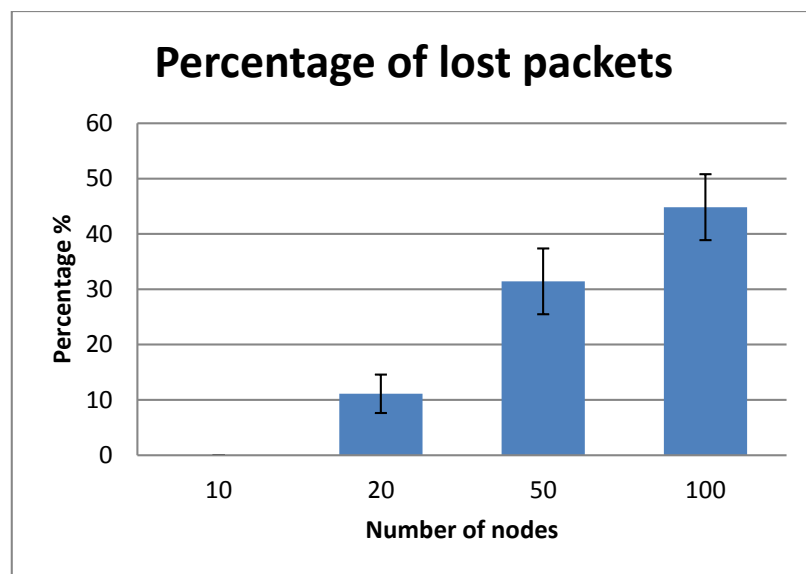


*Figure 8.4 The classical Flooding technique. Sent packets. CI 90%. Urban scenario.*

This situation produces that the probability of collisions increases. As it can be seen in Figures 8.5 and 8.6 as the number of nodes increases the percentage of received packets decreases. For example, the percentage of received packets for 50 nodes is only 68%, which is lower than 100% for 10 nodes. The devices invest certain resources to send packets, for example the battery, which are obviously wasted with the *Flooding* basic technique.



*Figure 8.5 The classical Flooding technique. Percentage of received packets. CI 90%. Urban scenario.*



*Figure 8.6 The classical Flooding technique. Percentage of lost packets. CI 90%. Urban scenario.*

Also, the delay for 20 or more nodes increases substantially around five times in comparison to the delay for 10 nodes, as it is shown in Figure 8.7. That is possible because from 20 nodes the number of sent packets increases so much.

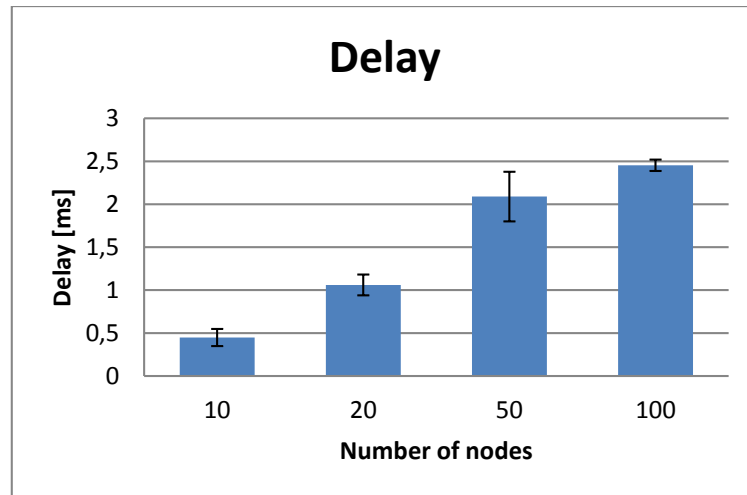


Figure 8.7 The classical Flooding technique. Delay. CI 90%. Urban scenario.

As it can be seen in Figure 8.8, as the number of nodes increases the percentage of reached nodes (i.e., nodes visited by a warning message being disseminated) increases too. That is possible because if the number of retransmitting nodes increases, the probability of receiving a packet also increases.

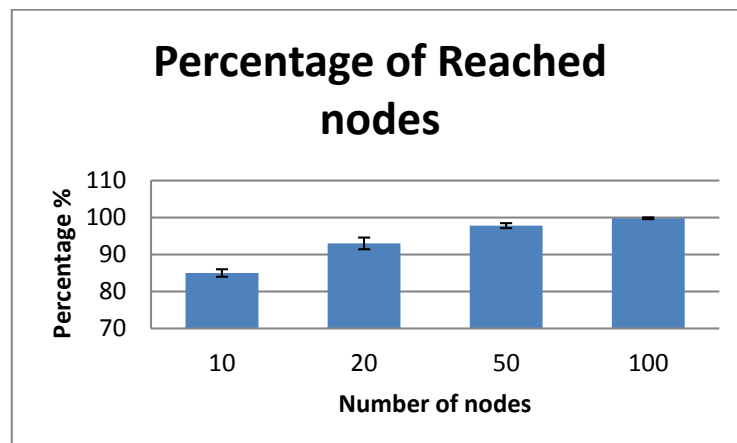


Figure 8.8 The classical Flooding technique. Percentage of reached nodes. CI 90%. Urban scenario.

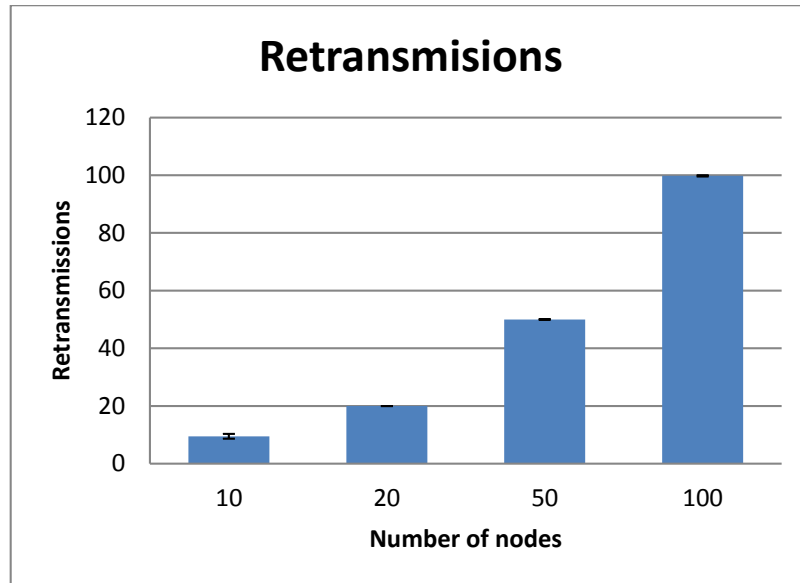
#### 8.4.2 Flooding technique in the Highway scenario

The simulation results for the *Flooding* technique in the Highway scenario are shown in Table 8.2.

	10 nodes	20 nodes	50 nodes	100 nodes
Retransmissions	9.5	20	50	99.8
Sent packets	76.2	329.6	1939.5	5664.8
% Received packets	100	80.445	37.107	22.295
% Lost packets	0	19.555	62.893	77.705
Delay (ms)	1.059	1.838	2.962	3.270
% Reached nodes	100	100	100	99.8

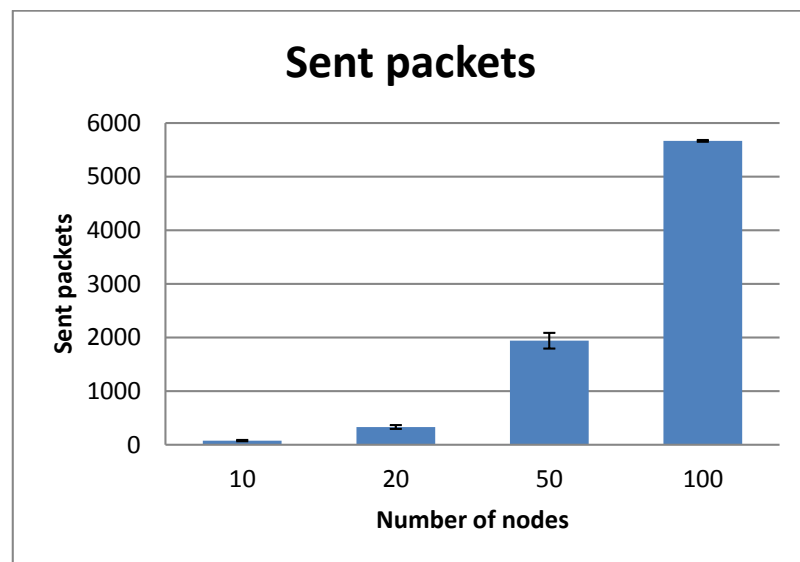
Table 8.2 Simulation results of the classical Flooding technique in the Highway scenario.

As it can be seen in Figure 8.9, the number of retransmitting nodes is close to the number of nodes in the scenario. Practically, each node rebroadcasts the considered packet.



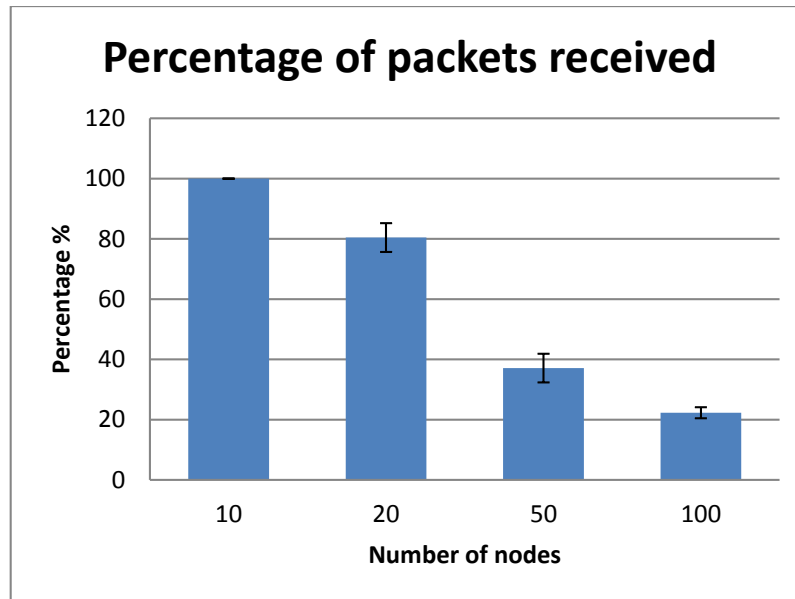
*Figure 8.9 The classical Flooding technique. Number of retransmitting nodes. CI 90%. Highway scenario.*

As a result, the number of sent packets increases too much. For example, the number of sent packets reaches 5665 for 100 nodes, which is so much in comparison with the 76 sent packets for 10 nodes, as it is illustrated in Figure 8.10.

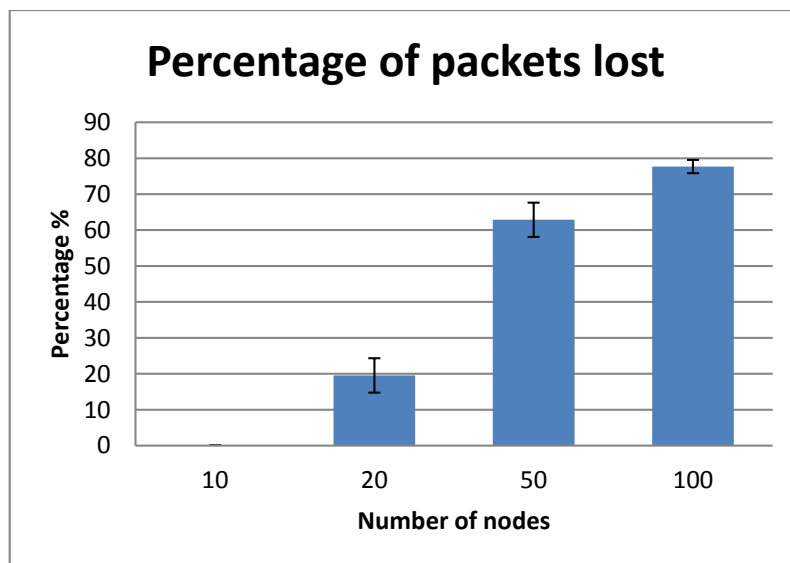


*Figure 8.10 The classical Flooding technique. Sent packets. CI 90%. Highway scenario.*

Because the number of retransmitting nodes and the number of sent packets increase, the probability of collisions also increases. Therefore, the percentage of received packets decreases drastically from 100% (10 nodes) to 22% (100 nodes), as it can be seen in Figures 8.11 and 8.12.

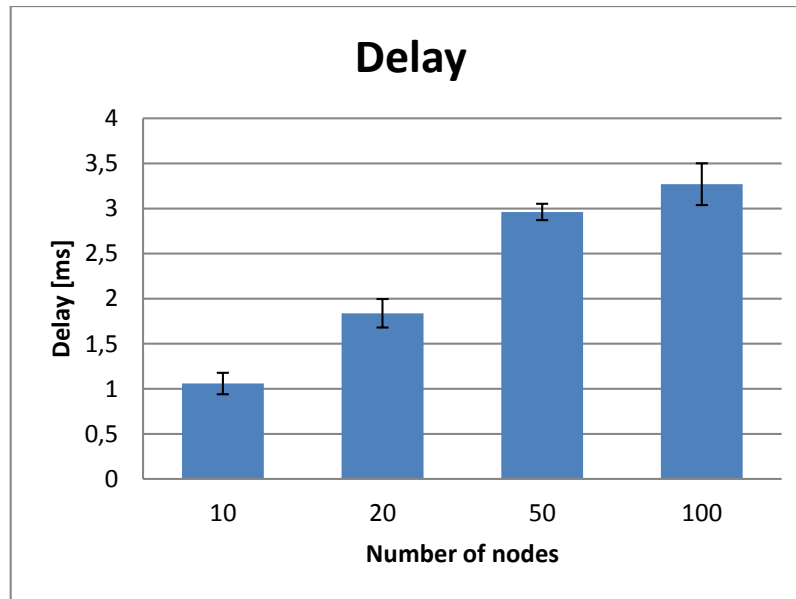


*Figure 8.11 The classical Flooding technique. Percentage of received packets. CI 90%. Highway scenario.*



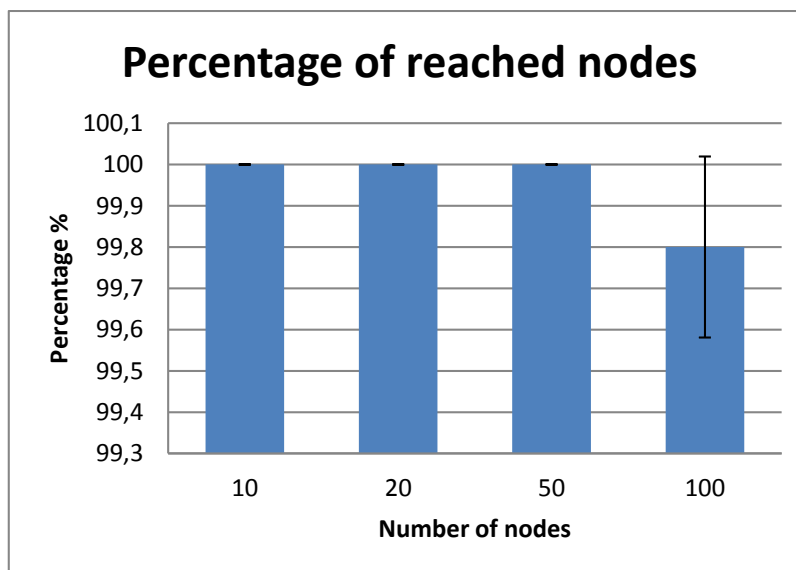
*Figure 8.12 The classical Flooding technique. Percentage of lost packets. CI 90%. Highway scenario.*

Because the percentage of received packets decreases, the delay also increases from 1 ms (10 nodes) to 3 ms (100 nodes), as it is shown in Figure 8.13.



*Figure 8.13 The classical Flooding technique. Delay. CI 90%. Highway scenario.*

As it can be seen in Figure 8.14, the percentage of reached nodes is nearly 100%. That is possible because the number of retransmitting nodes is almost 100%.



*Figure 8.14 The classical Flooding technique. Percentage of reached nodes. CI 90%. Highway scenario.*

### 8.4.3 Conclusions about the Flooding technique

According to the results shown in the two previous sections, it can be concluded that as the number of nodes increases, the network performance decreases drastically. Also, the *Flooding* technique works better in the Urban scenario than in the Highway scenario.

## 8.5 Counter technique

This technique was described in section 5.3.2. Specifically, each node initiates a counter  $c$  that will record the number of times a node receives the same packet. Such a counter is maintained by each node for each broadcast packet. Each node increments its  $c$  by one each time it receives the same packet. The node compares its  $c$  with a predefined counter threshold  $C$ . If  $c < C$ , the node rebroadcasts the packet. Otherwise the packet is dropped. In this simulation the threshold  $C$  take three values were 3, 4 and 5, respectively.

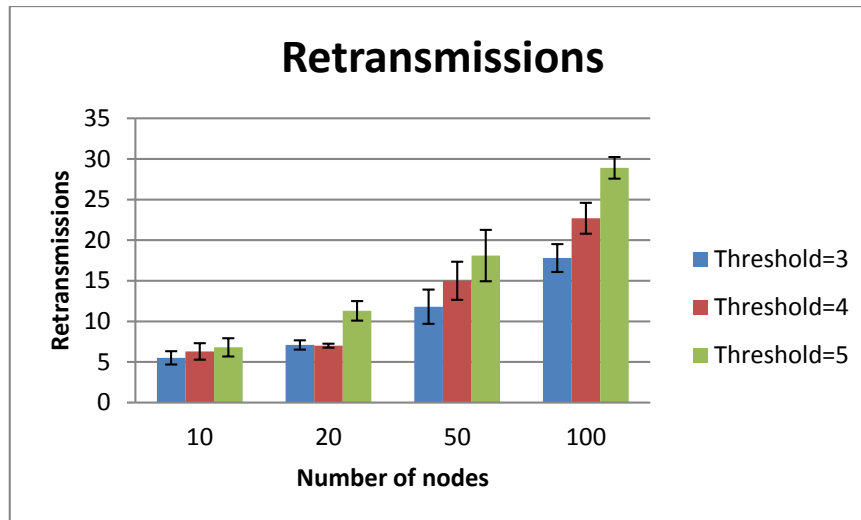
### 8.5.1 Counter technique in the Urban scenario

The simulation results for the *Counter* technique in the Urban scenario are shown in Table 8.3.

Nodes	Threshold	Retransmissions	Sent packets	Delay (ms)	% Reached nodes
10	3	5.5	21	0.308	92
	4	6.3	25.7	0.385	92
	5	6.8	28.4	0.423	92
20	3	7.1	61.6	0.412	91
	4	7	98.6	0.629	91
	5	11.3	97.7	0.646	91
50	3	11.8	193.1	0.492	94.6
	4	15	251.3	0.64	94.6
	5	18.1	302.7	0.875	94.6
100	3	17.8	429.7	0.537	96.2
	4	22.7	538.9	0.674	96.2
	5	28.9	660.8	0.828	96.2

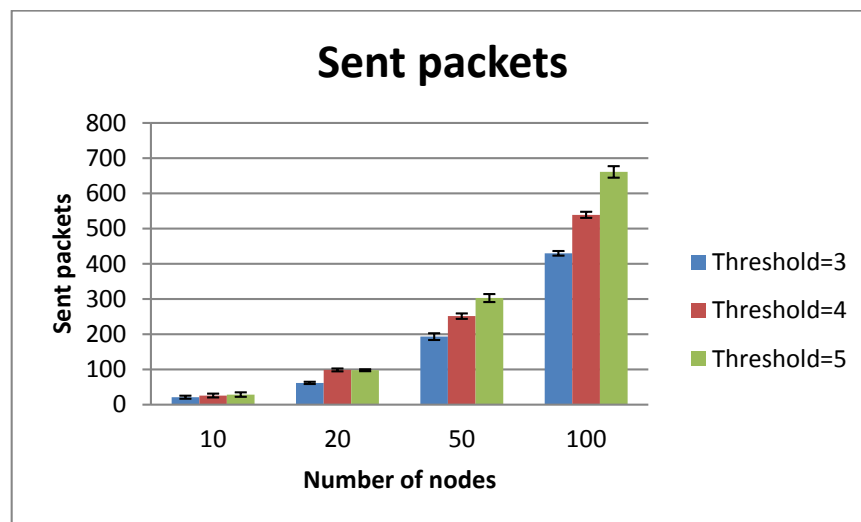
*Table 8.3 Simulation results of the Counter technique in the Urban scenario.*

As it can be seen in Figure 8.15, the number of retransmitting nodes when there are 10 and 20 nodes is nearly half of the number of nodes in the scenario. In case of 50 and 100 nodes the number of retransmitting nodes is 30% of the number of nodes in the scenario approximately. In addition, the number of retransmitting nodes depends also on the counter threshold value, as it is illustrated in Figure 8.15.



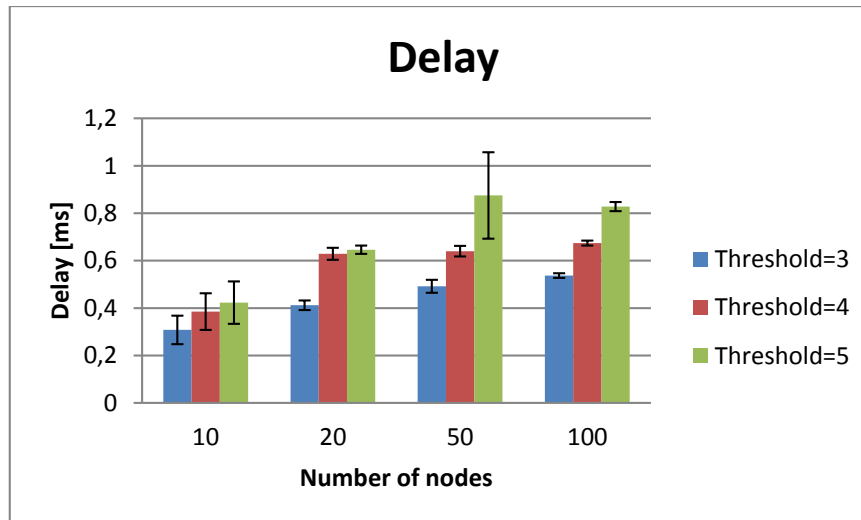
*Figure 8.15 The Counter technique. Number of retransmitting nodes. CI 90%. Urban scenario.*

As the number of nodes in the scenario and the counter threshold value increase the number of sent packets increases. For example, with a counter threshold value  $C = 5$ , the number of sent packets increases from 29 (10 nodes) to 661 (100 nodes). This situation is illustrated in Figure 8.16.



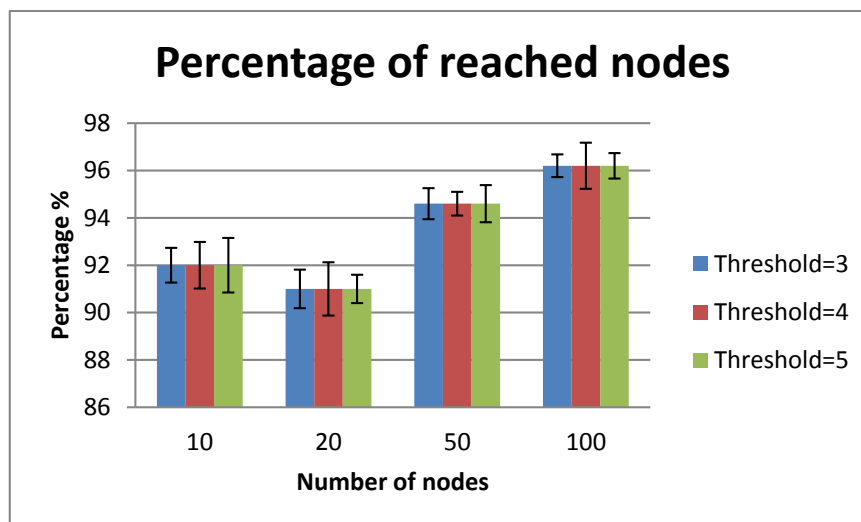
*Figure 8.16 The Counter technique. Sent packets. CI 90%. Urban scenario.*

As it can be seen in Figure 8.17, as the number of nodes in the scenario and the counter threshold value increase the delay increases as well. Also, with a counter threshold  $C = 4$  the delay for 20, 50 and 100 nodes is nearly equal.



*Figure 8.17 The Counter technique. Delay. CI 90%. Urban scenario*

The percentage of reached nodes depends only on the number of nodes, as it can be seen in Figure 8.18. This percentage is higher than 90% for whatever number of nodes in the scenario.



*Figure 8.18 The Counter technique. Percentage of reached nodes. CI 90%. Urban scenario*

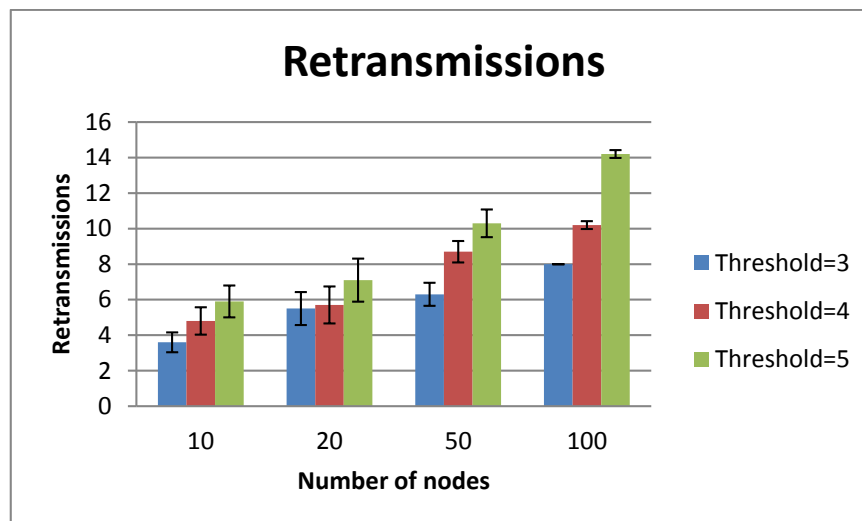
### 8.5.2 Counter technique in the Highway scenario

The simulation results for the *Counter* technique in the Highway scenario are shown in Table 8.4.

Nodes	Threshold	Retransmissions	Sent packets	Delay (ms)	% Reached nodes
10	3	3,6	27,3	0,388	95
	4	4,8	36,3	0,517	95
	5	5,9	44,9	0,641	95
20	3	5,5	82,3	0,527	100
	4	5,7	85,5	0,547	100
	5	7,1	107,2	0,682	100
50	3	6,3	219,8	0,543	100
	4	8,7	309,5	0,764	100
	5	10,3	365,5	0,902	100
100	3	8	416,2	0,509	99,8
	4	10,2	544,4	0,672	99,8
	5	14,2	748,6	0,915	99,8

*Table 8.4 Simulation results of the Counter scheme in highway scenario.*

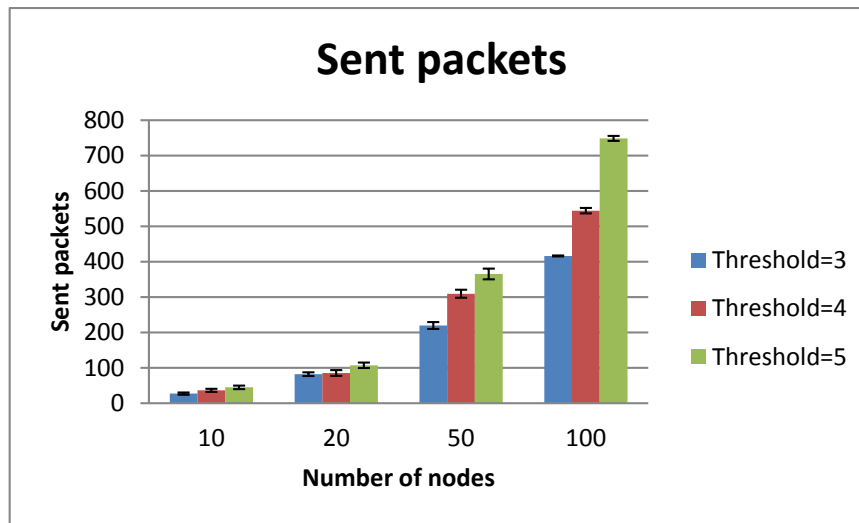
The number of retransmitting nodes depends on the counter threshold value and on the number of nodes in the scenario, as it is illustrated in Figure 8.19. Also, it can be concluded that the percentage of retransmitting nodes reduces as the number of nodes in the scenario increases. For example, for 10 nodes in the scenario the percentage of retransmitting nodes is about 50%, whereas for 100 nodes in the scenario this percentage is about 15%.



*Figure 8.19 The Counter technique. Number of retransmitting nodes. CI 90%. Highway scenario.*

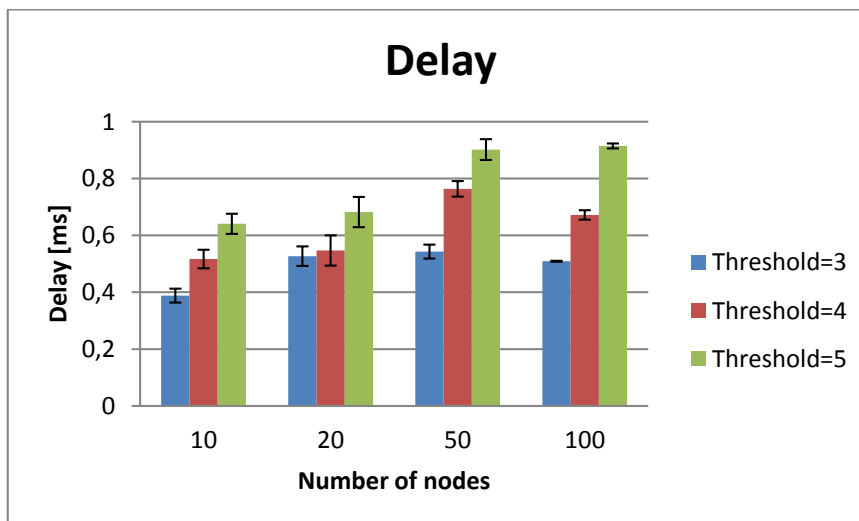
The number of sent packets depends on the number of nodes and on the counter threshold value. As the number of nodes and the counter threshold value increase the number of sent packets

increases as well. For example, in the case of a number of nodes  $N = 100$ , the number of sent packets is 416 with a counter threshold  $C = 3$ , while this value is nearly doubled with a counter threshold  $C = 5$ . This situation is depicted in Figure 8.20.



*Figure 8.20 The Counter technique. Sent packets. CI 90%. Highway scenario.*

As it can be seen in Figure 8.21, the delay is directly proportional to the number of nodes in the scenario and to the counter threshold value. For example, with 50 nodes in the scenario, the delay for counter threshold  $C = 3, 4$  and  $5$  are  $0.543$  ms,  $0.764$  ms and  $0.902$  ms, respectively.



*Figure 8.21 The Counter technique. Delay. CI 90%. Highway scenario.*

As it can be seen in Figure 8.22, the percentage of reached nodes depends only on the number of nodes. This percentage is higher than 95% independently of the number of nodes in the scenario.

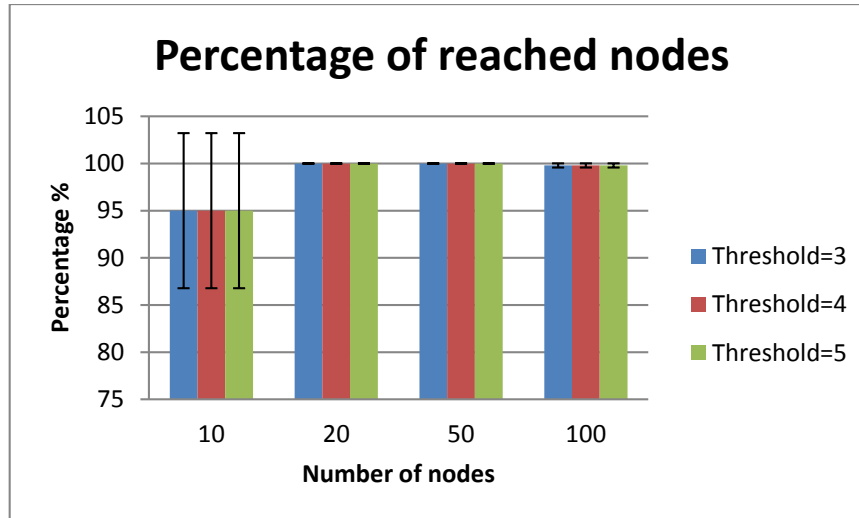


Figure 8.22 The Counter technique. Percentage of reached nodes. CI 90%. Highway scenario.

### 8.5.3 Conclusions about Counter technique

According to the results shown in the two previous sections, it can be concluded that the principal challenge in this technique is the selection of an appropriate counter threshold. So, using small counter threshold values would help in saving the amount of packets retransmitted by the vehicles. Nonetheless, this fact may affect reachability especially in sparse networks. Alternatively, larger counter threshold values are beneficial in sparse networks, but can unnecessary swamp a denser network with unneeded redundant packets, like it happens with the *Flooding* technique. Also, it can be concluded that the *Counter* technique works better in the Highway scenario than in the Urban scenario.

## 8.6 Probability technique

This technique was described in section 5.3.7. Basically, upon receiving a broadcast message for the first time, a node will rebroadcast it with a probability  $P$ . Clearly, when  $P = 1$ , this technique is equivalent to flooding. In this simulation the probability threshold values  $P$  that we have used were 0.1, 0.2, 0.5 and 0.8.

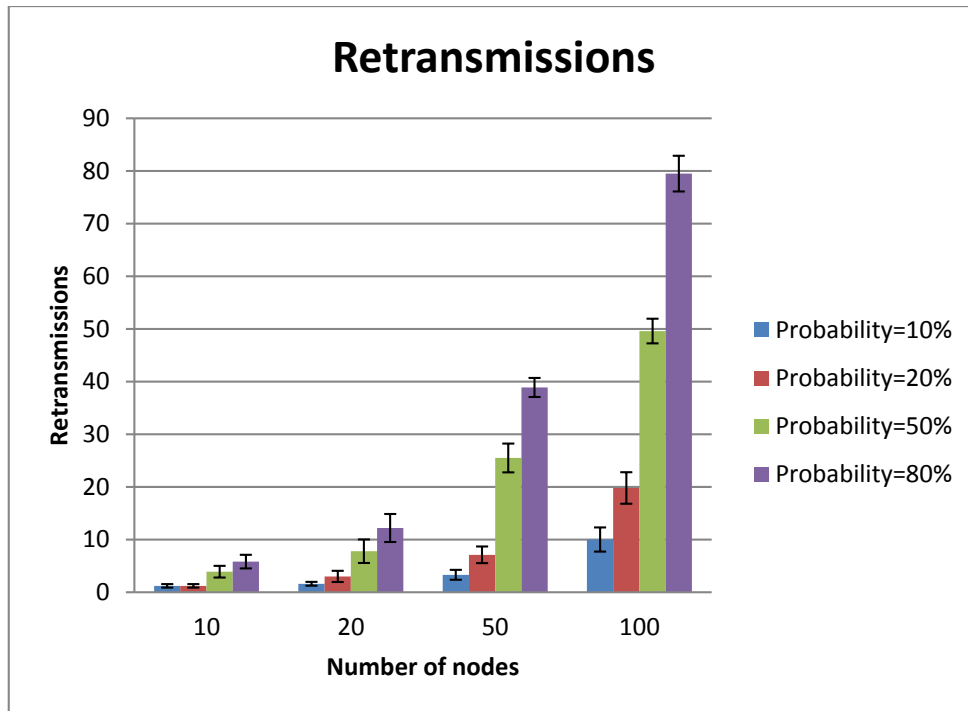
### 8.6.1 Probability technique in the Urban scenario

The simulation results of the *Probability* technique in the Urban scenario are shown in Table 8.5.

Nodes	Threshold	Retransmissions	Sent Packets	Delay (ms)	% Reached nodes
10	10	1,2	4,2	0,065	37
	20	1,2	4,2	0,065	37
	50	3,9	16,2	0,241	64
	80	5,8	22,9	0,345	67
20	10	1,6	11,6	0,079	43
	20	3	27,2	0,181	51,5
	50	7,8	72,8	0,443	61
	80	12,2	109,8	0,715	73
50	10	3,3	67	0,169	61,2
	20	7,1	158,9	0,398	66,8
	50	25,5	496,7	1,252	96
	80	38,9	724,2	1,831	96,4
100	10	10	307,6	0,381	72,7
	20	19,8	568,4	0,727	83,6
	50	49,6	1336,3	1,666	98,8
	80	79,5	2173,4	2,709	98,2

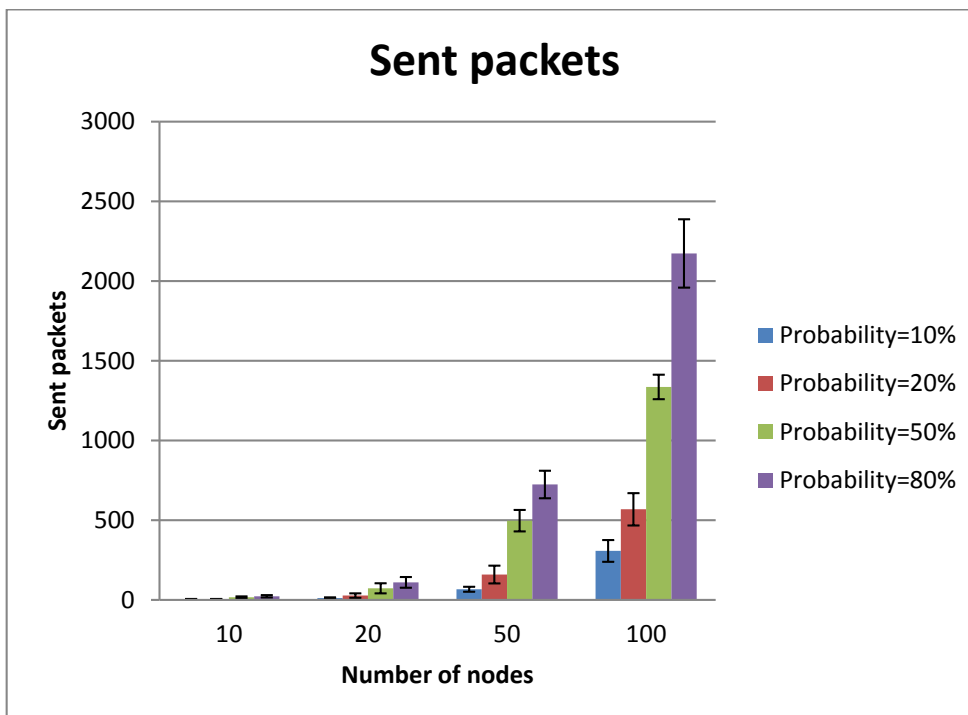
**Table 8.5 Simulation results of the *Probability* technique in the Urban scenario.**

As it can be seen in Figure 8.23, the number of retransmitting nodes depends on the probability threshold value and on the number of nodes in the scenario. Also, it can be seen that the percentage of the retransmitting nodes is very similar to the probability threshold value. For example, for 50 nodes in the scenario the percentage of retransmitting nodes is about 78% (0.78) with a probability threshold  $P = 0.8$ .



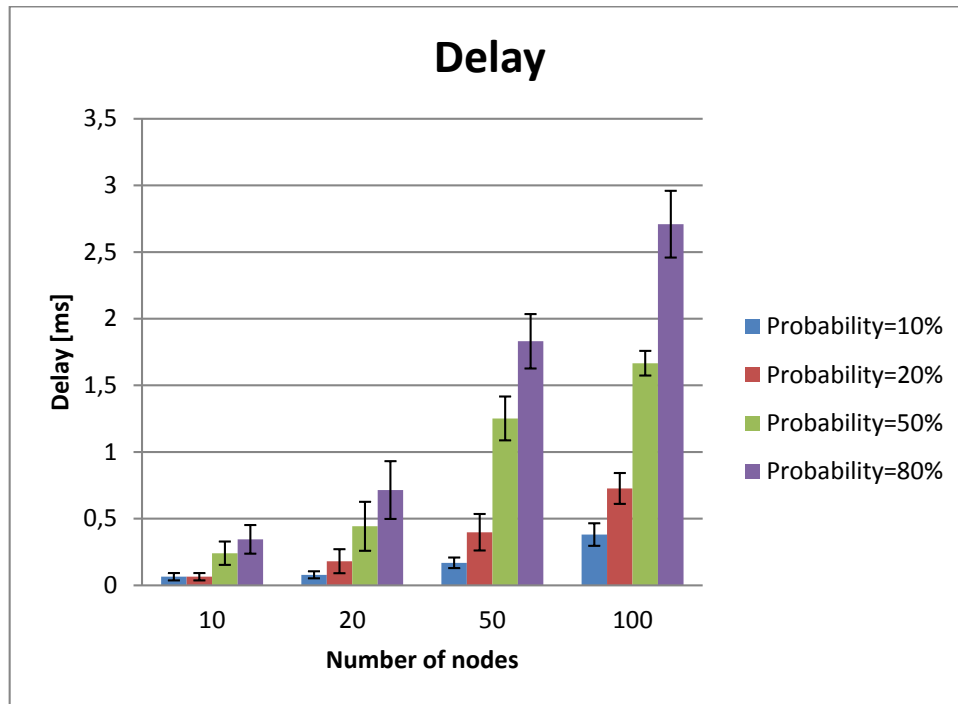
*Figure 8.23 The Probability technique. Number of retransmitting nodes. CI 90%. Urban scenario.*

As the probability threshold value increases the number of sent packets increases considerably. For example, in the case of 100 nodes in the scenario, the number of sent packets increase from 308 (probability threshold  $P = 0.1$ ) to 2173 (probability threshold  $P = 0.8$ ), which is an increment about 600%, as it is illustrated in Figure 8.24.



*Figure 8.24 The Probability technique. Sent packets. CI 90%. Urban scenario.*

As it can be seen in Figure 8.25, the delay is directly proportional to the number of nodes in the scenario and to the probability threshold value. For example, with 100 nodes in the scenario, the delay for probability threshold  $P = 0.1$ , 0.2, 0.5 and 0.8 are 0.381 ms, 0.727 ms, 1.666 ms and 2.709 ms, respectively. Thus, the delay with a probability threshold  $P = 0.8$  is practically 7 times the delay with a probability threshold  $P = 0.1$ .



*Figure 8.25 The Probability technique. Delay. CI 90%. Urban scenario.*

As it can be seen in Figure 8.26, the percentage of reached nodes depends on the number of nodes and on the probability threshold value. For a number of nodes lower than 20 the percentage of reached nodes is very little.

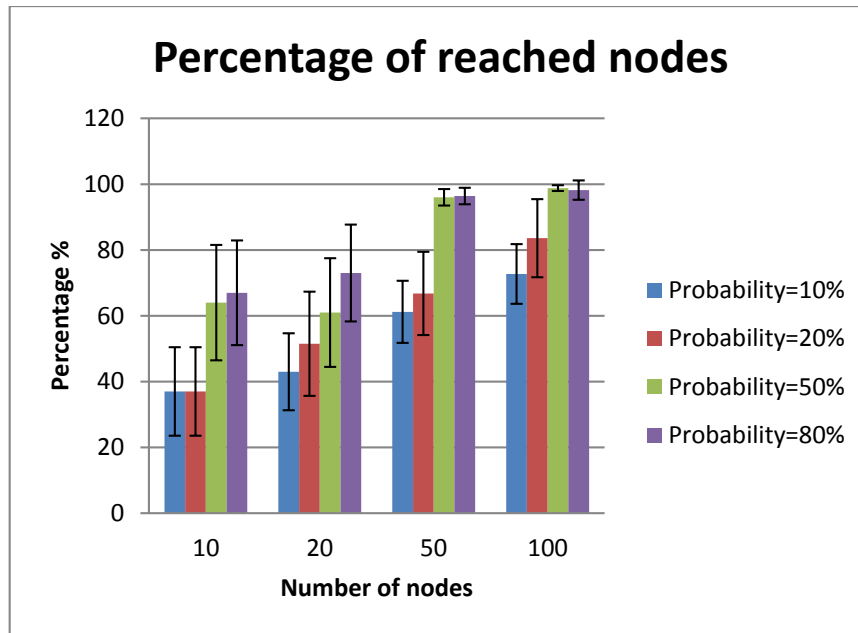


Figure 8.26 The Probability technique. Percentage of reached nodes. CI 90%. Urban scenario.

### 8.6.2 Probability technique in the Highway scenario

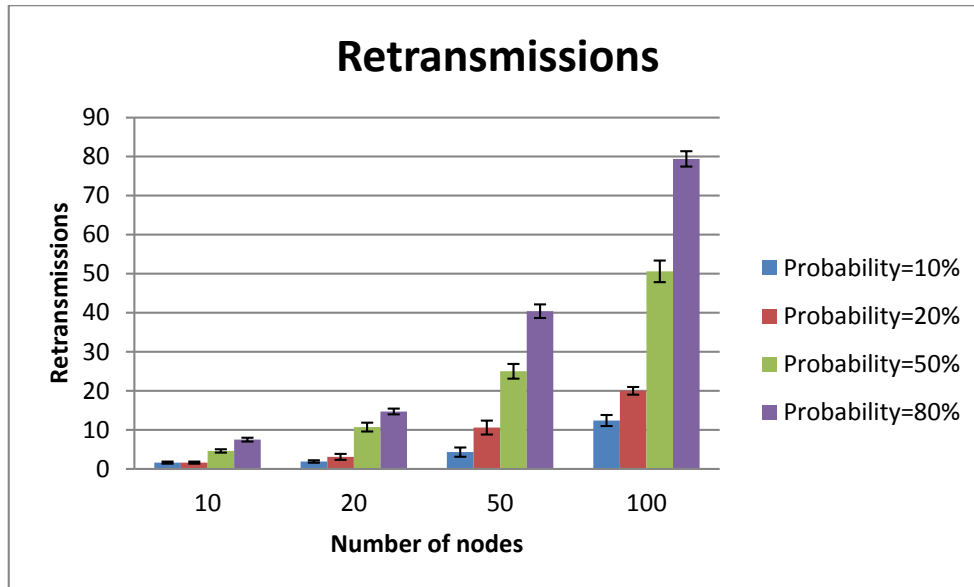
The simulation results of the *Probability* technique in the Highway scenario are shown in Table 8.6.

Nodes	Threshold	Retransmissions	Sent packets	Delay (ms)	% Reached nodes
10	10	1,6	4,2	0,175	82
	20	1,6	4,2	0,175	82
	50	4,6	16,2	0,504	93
	80	7,5	22,9	0,818	95
20	10	1,9	11,6	0,2	87,5
	20	3,1	27,2	0,318	94
	50	10,7	72,8	1,136	95,5
	80	14,7	109,8	1,519	100
50	10	4,3	67	0,399	93
	20	10,6	158,9	1,006	99,6
	50	25	496,7	2,378	100
	80	40,4	724,2	3,9	100
100	10	12,4	307,6	0,858	99,6
	20	20	568,4	1,361	99,8
	50	50,6	1336,3	3,464	99,8
	80	79,4	2173,4	5,24	99,8

Table 8.6 Simulation results of the Probability scheme in the Highway scenario.

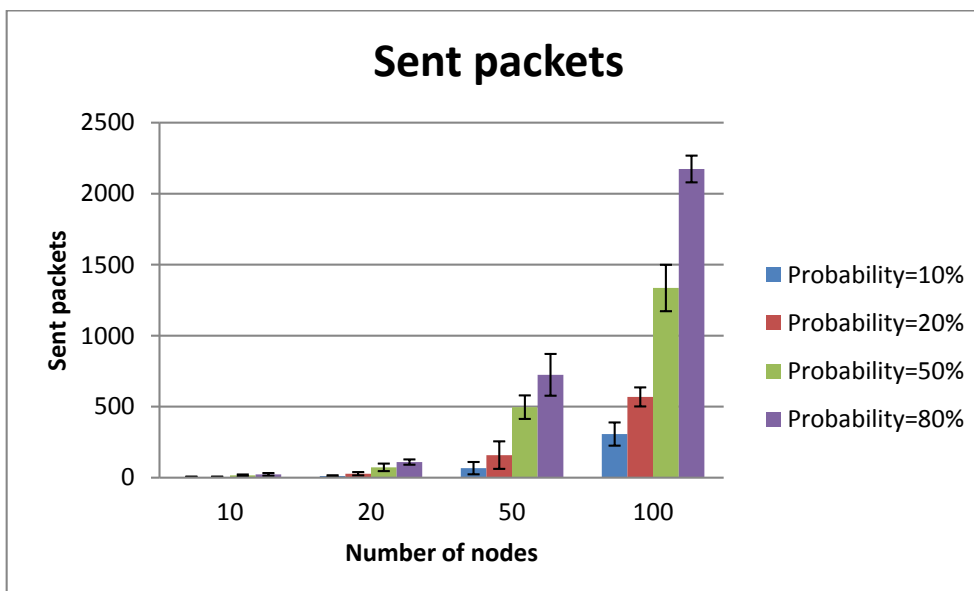
According to Figure 8.27, the number of retransmitting nodes is directly proportional to the probability threshold value. Also, it can be seen that the percentage of the retransmitting nodes is

very similar to the probability threshold value. For example, for 20 nodes in the scenario the percentage of retransmitting nodes is about 53% (0.53) with a probability threshold = 0.5.



*Figure 8.27 The Probability technique. Number of retransmitting nodes. CI 90%. Highway scenario.*

As the probability threshold value increases the number of sent packets increases drastically. For example, in the case of 50 nodes in the scenario, the number of sent packets increases from 67 (probability threshold  $P = 0.1$ ) to 724 (probability threshold  $P = 0.8$ ), which is an increment about 980%, as it is illustrated in Figure 8.28.



*Figure 8.28 The Probability technique. Sent packets. CI 90%. Highway scenario.*

As it can be seen in Figure 8.29, as the probability threshold value increases the delay increases as well. Also, the number of nodes influences the delay. For example, with 50 nodes in the scenario,

the delay for probability threshold  $P = 0.1, 0.2, 0.5$  and  $0.8$  are  $0.399$  ms,  $1.006$  ms,  $2.378$  ms and  $3.9$  ms, respectively.

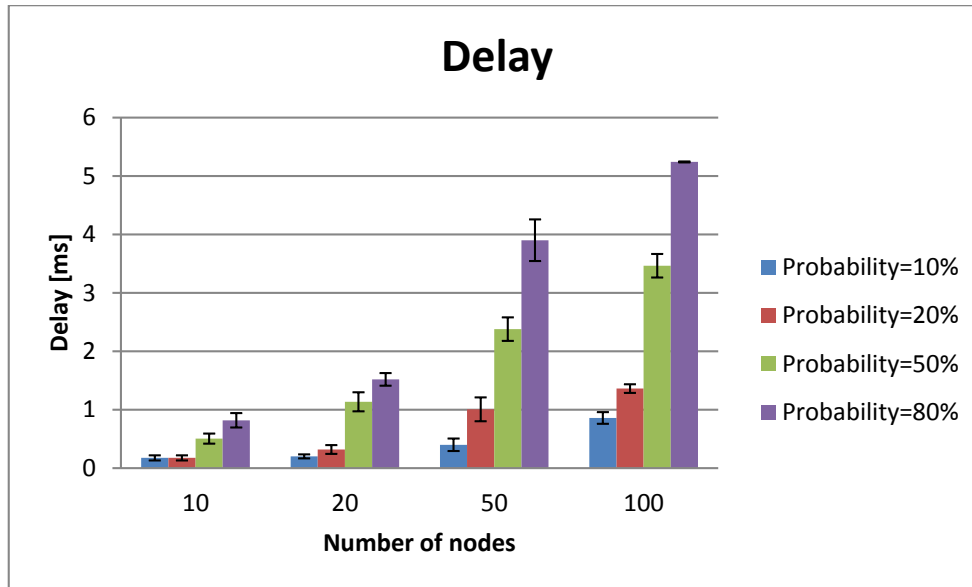


Figure 8.29 The Probability technique. Delay. CI 90%. Highway scenario.

Basically, the percentage of reached nodes depends on the number of nodes. The percentage of reached nodes is higher than 82% for any number of nodes and any probability threshold value, as it is shown in Figure 8.30.

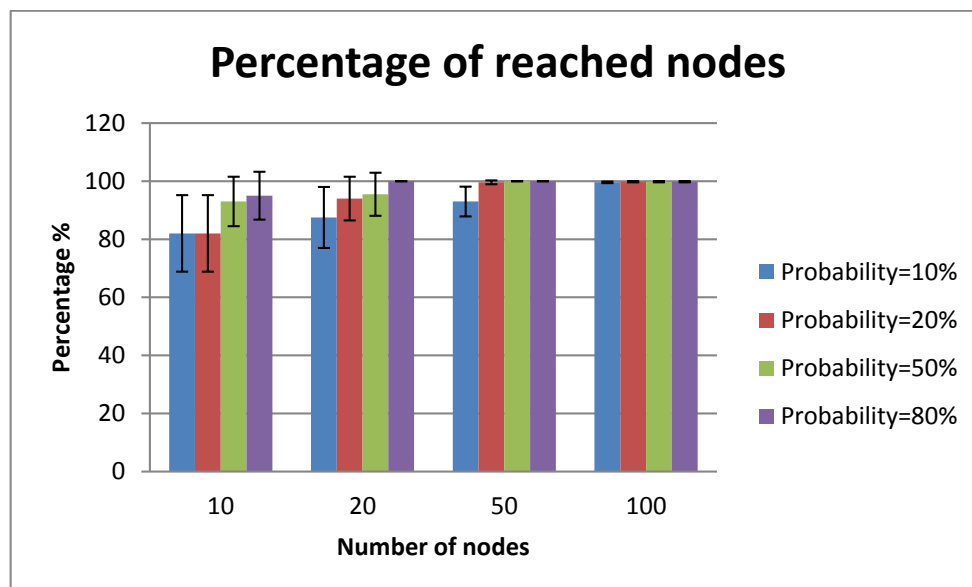


Figure 8.30 The Probability technique. Percentage of reached nodes. CI 90%. Urban scenario.

### 8.6.3 Conclusions about Probability technique

According to the results of the two previous sections, it can be concluded that the selection of the probability threshold value  $P$  is decisive in this technique. Also, in order to do the selection of the

probability threshold  $P$  it must be considered the characteristic of the scenario and the number of nodes. If this value is too small, the number of retransmitting nodes is small, which is good for the network, but the percentage of reached nodes can be too small.

On the other hand, if the probability threshold is too big, this technique works like the *Flooding* technique. So, the *Probability* technique will not solve the broadcast storm problem.

In addition, it can be concluded that the *Probability* technique works better in the Highway scenario than in the Urban scenario.

## 8.7 Comparison between Flooding, Counter and Probability techniques

The main configuration parameters of the three broadcasting techniques that we want to analyze are the number of retransmitting nodes, the percentage of reached nodes and the average packet delay. Also, the kind of scenario is important, so there is a comparison for both Urban and Highway scenarios.

### 8.7.1 Comparison in the Urban scenario

Figure 8.31 shows the number of retransmitting nodes obtained for each technique. In that figure it can be seen the broadcast storm problem using *Flooding* technique. Also, it can be seen how *Counter* and *Probability* techniques solve that problem. According to this parameter, it can be concluded that the *Probability* technique is the best option in the Urban scenario with a probability threshold value  $P = 0.1$  (10%).

Figure 8.32 shows the percentage of reached nodes for each technique. According to Figure 8.32, in the case of the *Probability* technique, the higher the probability threshold  $P$ , the higher the percentage of reached nodes. On the other hand, using the *Counter* technique the percentage of reached nodes is the same whatever the counter threshold  $C$  (3, 4 or 5). Also, based on the percentage of reached nodes *Counter* technique is the best option in the Urban scenario.

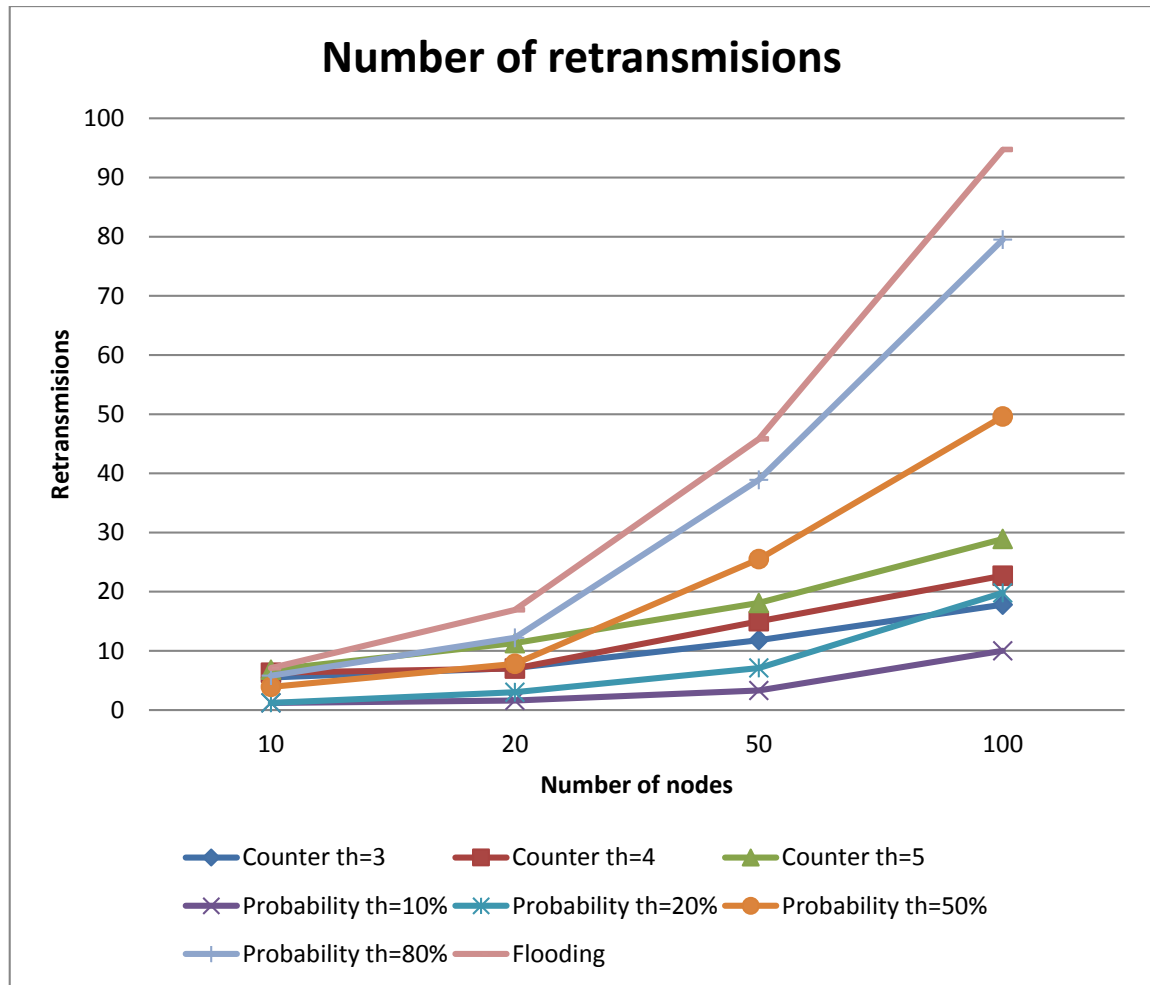


Figure 8.31 Number of retransmitting nodes for each technique in the Urban scenario.

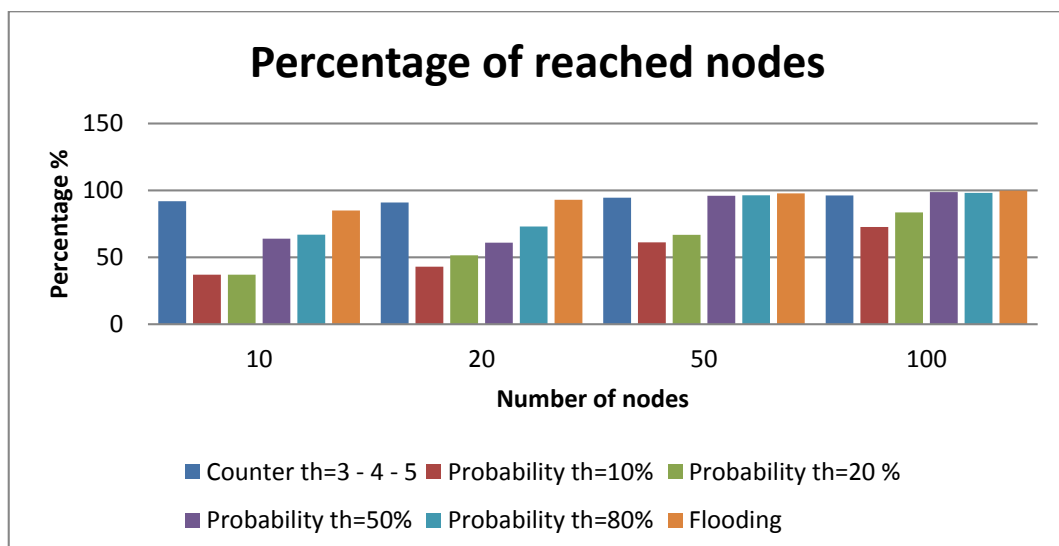


Figure 8.32 Percentage of reached nodes for each technique in the Urban scenario.

Figure 8.33 illustrates the delay for each technique in the Urban scenario. According to this parameter, the *Probability* technique is the best option with a probability threshold  $P = 0.1$  (10%). Also, the worst options are *Flooding* and *Probability* with a probability threshold  $P = 0.8$  (80%).

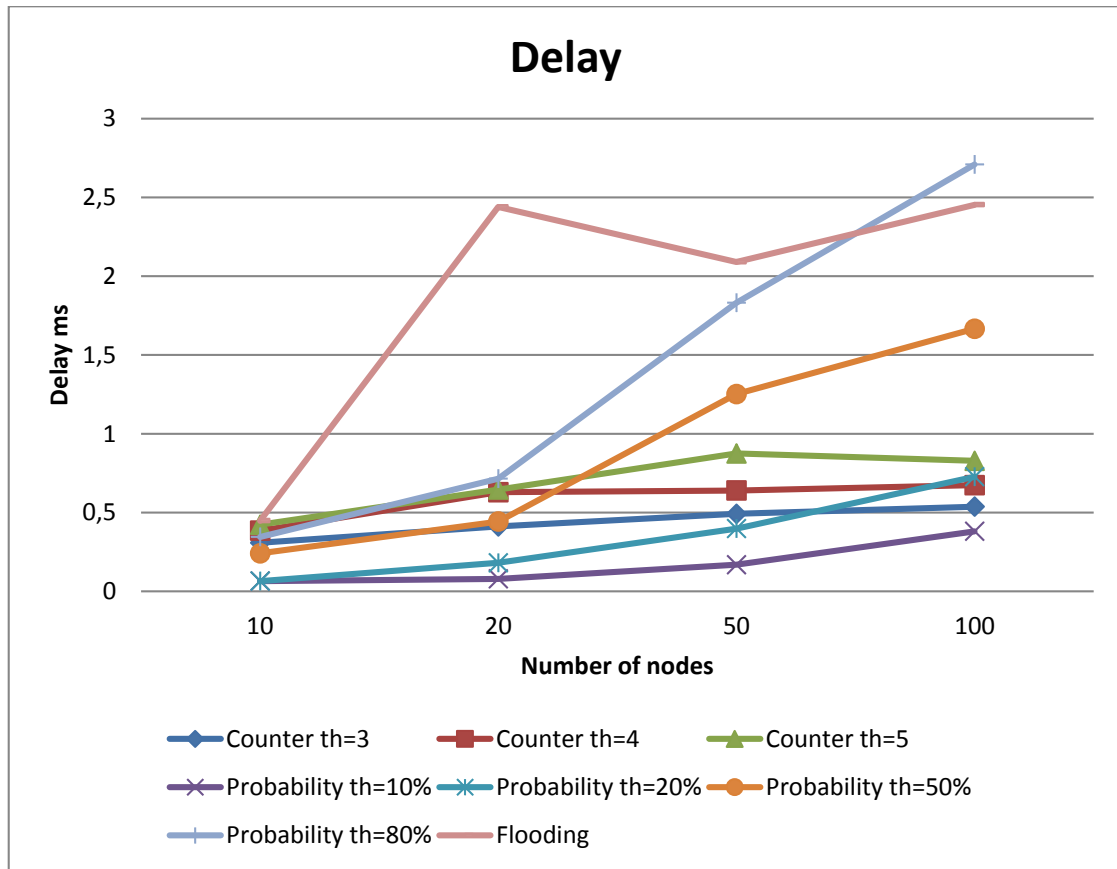


Figure 8.33 Delay for each technique in the Urban scenario.

Finally, according to the number of retransmitting nodes, the *Probability* technique with a probability threshold  $P = 10\%$  works better than the *Counter* technique with a counter threshold  $C = 3$ . However, the percentage of reached nodes with the *Probability* technique is too low in comparison with the *Counter* technique. In case of the delay, there is a little difference but it does not have much influence. In conclusion, the best technique in the Urban scenario is the *Counter* technique with a counter threshold  $C = 3$ .

### 8.7.2 Comparison in the Highway scenario

Figure 8.34 illustrates the number of retransmitting nodes for *Flooding*, *Counter* and *Probability* techniques. It can be seen the problem of broadcast storm using the *Flooding* technique. In case of *Probability* technique it can be seen that as the probability threshold value increases, the network performance decreases. According to this parameter, it can be concluded that until 50 nodes the

best technique is *Probability* with a probability threshold  $P = 0.1$  (10%). For a number of nodes above 50, the best technique is the *Counter* with a counter threshold  $C = 3$ .

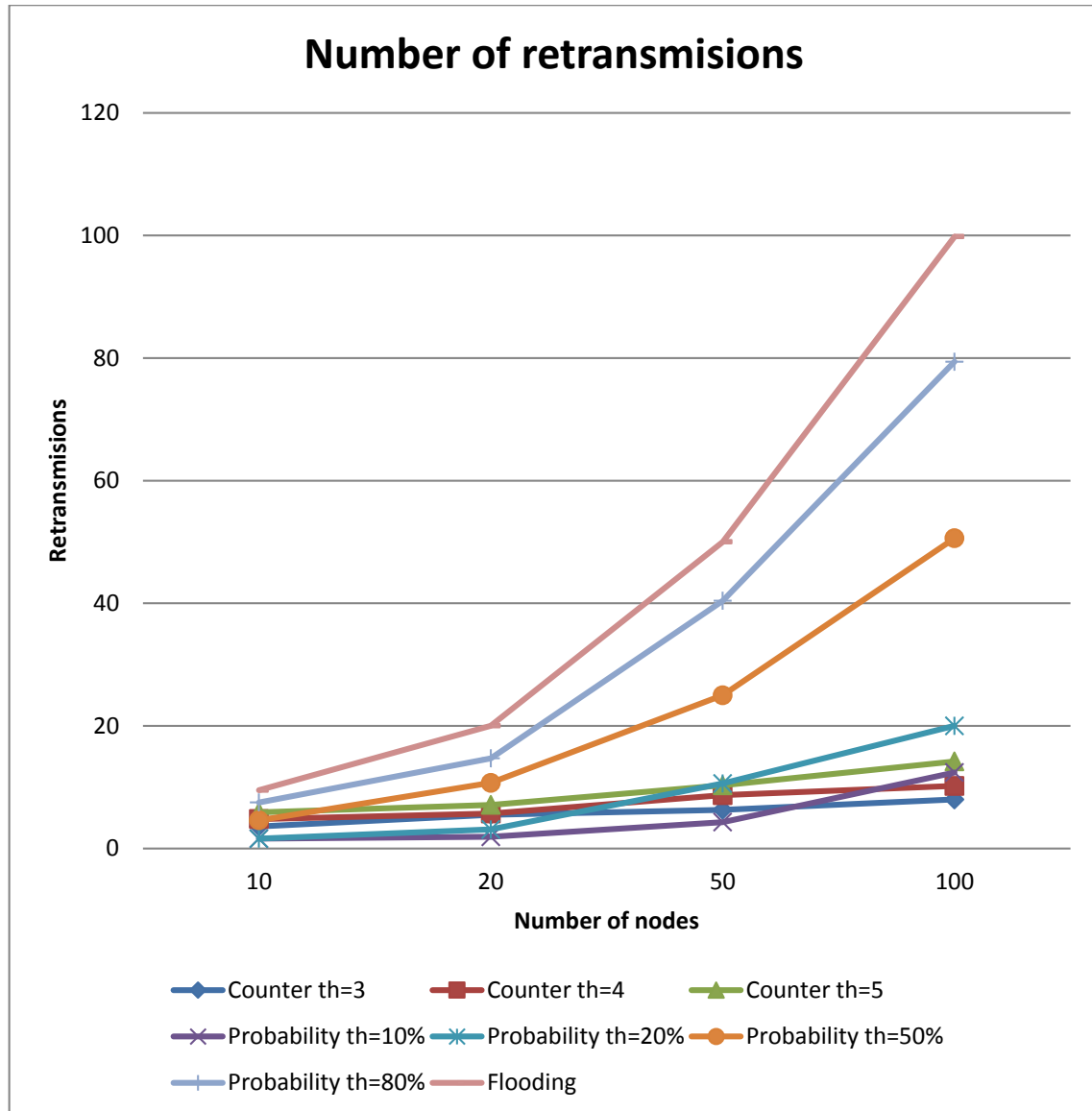


Figure 8.34 Number of retransmitting nodes for each technique in the Highway scenario.

Figure 8.35 shows the percentage of reached nodes for each technique. In case of the *Probability* technique, the percentage of reached nodes is very similar for every number of nodes. On the other hand, based on this parameter, the *Counter* technique has the best performance.

Figure 8.36 illustrates the delay for each scheme in the Highway scenario. Until 50 nodes the best technique is the *Probability* with a probability threshold  $P = 0.1$  (10%). For a number of nodes higher than 50 the best technique is the *Counter* with a counter threshold  $C = 3$ .

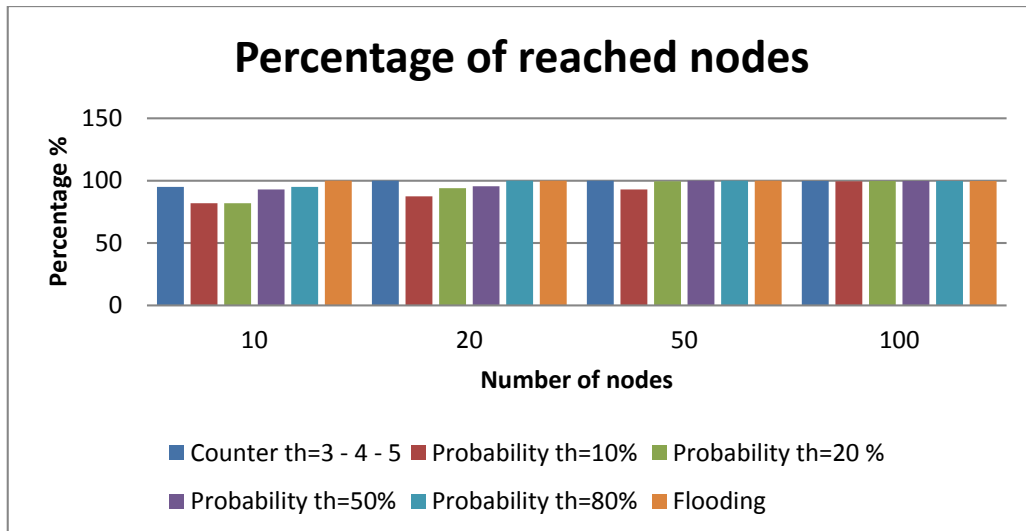


Figure 8.35 Percentage of reached nodes for each technique in the Highway scenario.

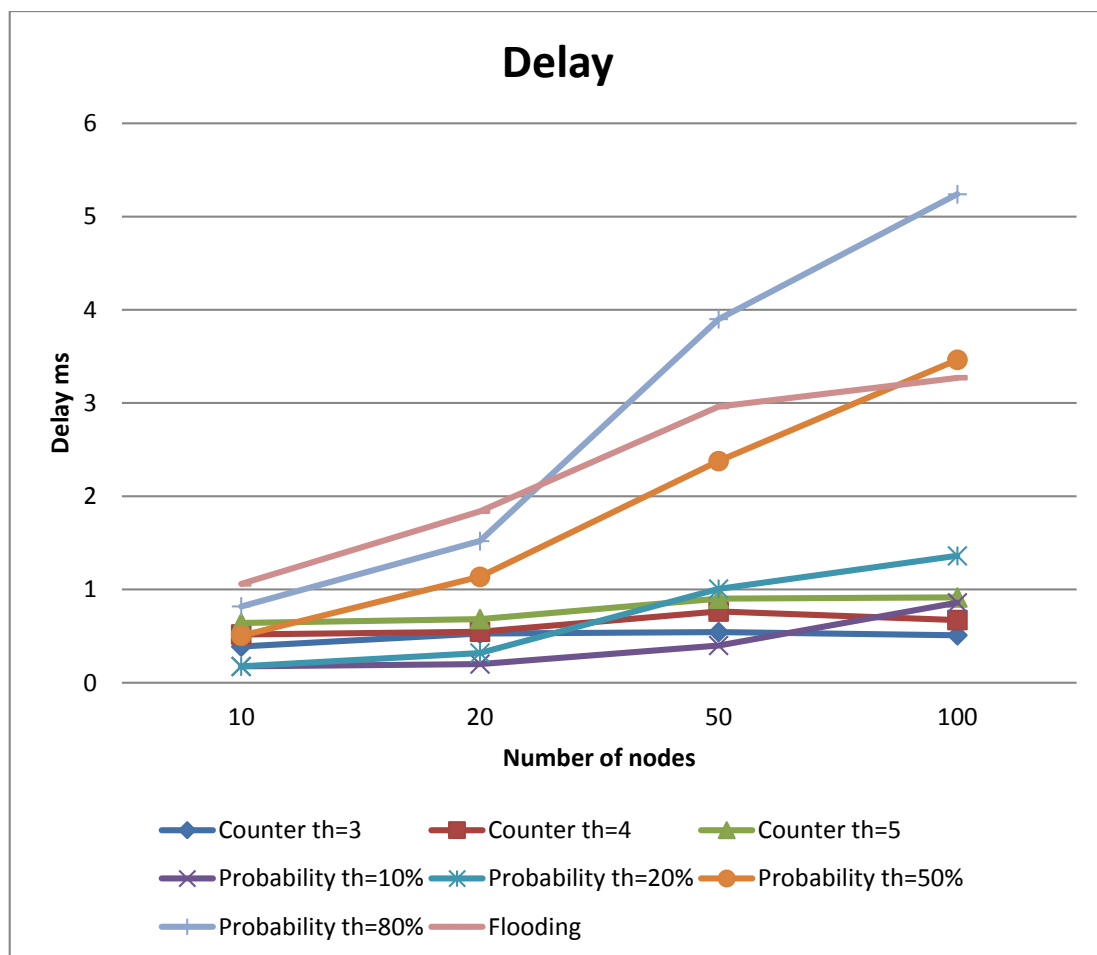


Figure 8.36 Delay for each technique in the Highway scenario.

In conclusion, with a number of nodes less than 50 the best technique is the *Probability* with a probability threshold  $P = 0.1$  (10%), and with a number of nodes higher than 50 the best technique is the *Counter* with a counter threshold  $C = 3$ .

## Chapter 9

# Conclusions and future work

### 9.1 Conclusions

During the previous eight chapters of this Master's Thesis, we have made an extensive study into the message dissemination field of vehicular networks. We have presented the concept of VANETs and its enormous potential, the wide range of applications and the advantages on road safety.

The development of VANETs depends on a series of factors such as technical requirements, standards for wireless communications, and simulations platforms. The simulation world is very important specially in large scale scenarios. Thus, there are many projects that have been developed in recent years to create realistic simulation frameworks designed for VANETs. To do that, necessarily it requires a module for modeling the network (inter-vehicular communication) and another for the simulation of mobility (traffic). In the state of the art one can find different solutions to combine both fields. However, the most successful model for performance is the hybrid simulator concept. In this context, VEINS [32] is a hybrid simulator which is based on OMNeT++ [25] (a network simulator) and SUMO [28] (a traffic simulator) and a real time communication system between both called TraCI.

After our deep analysis through simulations of several of the main broadcast techniques it can be concluded that the *Flooding* technique is the worst scheme because it produces too much delay, large number of retransmissions, and when the number of nodes increases the percentage of received packets decreases drastically. In the case of the so-called *Counter* scheme, it can be concluded that when its configurable threshold increases the performance decreases substantially, although it is the best scheme in comparison with the other. Finally, the *Probability* technique works better in highway scenarios than in urban scenarios. In general, in all the analyzed schemes the most important parameter, for both *Counter* and *Probability* technique, is the selection of a threshold according to the number of nodes and the features of the scenario.

## 9.2 Future work

Some approaches of future work are pointed out:

- A deep study of other VANET open source simulation solutions, such as TraNS [15] (SUMO and NS-2) which is a hybrid simulator with wide acceptance among the VANET research community.
- Work focusing on developing a unified system installation of VEINS. During the process of the installation of VEINS there were some problems especially with the compatibility of version between OMNeT++ and SUMO. So it would be useful to define a project to make a single and automatic installation of VEINS in both Windows and Linux systems.
- Implementation of novel techniques of message dissemination in OMNeT++ over the VEINS simulation tool, to see the benefits compared to other proposals available in the literature.

## References

- [1] World Health Organization. (2013). "Global status report on road safety 2013: supporting a decade of action". Available on:  
[http://www.who.int/violence\\_injury\\_prevention/road\\_traffic/en/](http://www.who.int/violence_injury_prevention/road_traffic/en/)
- [2] Cristhian Iza Paredes. (2014). "Design and performance evaluation of smart dissemination of emergence messages in Vehicular Ad-Hoc Networks". *Ph. D Thesis Proposal Universitat Politècnica de Catalunya*.
- [3] Hafedh Chourabi , Taewoo Nam , J. Ramon Gil-Garcia, Sehl Mellouli, Theresa A. Pardo, Hans Jochen Scholl, Shawn Walker, Karine Nahon. (2012). "Understanding Smart Cities: An Integrative Framework". *45<sup>th</sup> Hawaii International Conference on System Sciences*
- [4] Tuba Bakici, Esteve Almirall, Jonathan Wareham. (2012). "A Smart City: the Case of Barcelona". *Springer Science + Business Media, LLC 2012*
- [5] Robert E. Hall. (2000). "The Vision of A Smart City". *2<sup>nd</sup> International Life Extension Technology Workshop Paris, France*.
- [6] Kehua Su, Jie Li, Hongbo Fu. "Smart City and the Applications". *School of Computer – Wuhan University, Wuhan, Hubei, China*.
- [7] Intelligent Community Forum. website:  
[https://www.intelligentcommunity.org/index.php?src=gendocs&ref=Community\\_Alpha&link=Community\\_Alpha](https://www.intelligentcommunity.org/index.php?src=gendocs&ref=Community_Alpha&link=Community_Alpha).
- [8] Raymond Cunningham, Vinny Cahill. "System Support for Smart Cars: Requirements and Research Directions". *Distributed Systems Group, Department of Computer Science, Trinity College Dublin, Ireland*.
- [9] Jean-Pierre Hubaux, Srdjan Capkun, Jun Luo. (2004). "The Security and Privacy of Smart Vehicles". *EPFL – IEEE Security & Privacy*.
- [10] Jie Sun, Zhao-hui Wu, Gang Pan. (2009). "Context-aware smart car: from model to prototype". *Journal of Zhejiang University Science*
- [11] Stephen Ezell. (2010). "Explaining International IT Application Leadership: Intelligent Transportation Systems". *ITIF The Information Technology & Innovation Foundation*
- [12] ETSI. Web site: <http://www.etsi.org/technologies-clusters/technologies/intelligent-transport>
- [13] S. Swapna Kumar. (2014). "Vehicular Ad Hoc Network". *World Academy of Science, Engineering and Technology, International Journal of Computer, Control, Quantum and Information Engineering*.
- [14] Fan Li, Yu Wang. "Routing in Vehicular Ad Hoc Networks: A survey". *University of North Carolina at Charlotte*.
- [15] Saleh Yousefi, Mahmoud Siadat Mousavi, Mahmood Fathy. (2006). "Vehicular Ad Hoc Networks (VANETs): Challenges and Perspectives". *6<sup>th</sup> International Conference on ITS Telecommunications Proceedings*.
- [16] Sherali Zeadally, Ray Hunt, Yuh-Shyan Chen, Angela Irwin, Aamir Hassan. (2012). "Vehicular ad hoc networks (VANETS): status, results, and challenges". *Springer Science+Business Media, LLC 2010*.
- [17] Wenshuang Liang, Zhuorong Li, Hongyang Zhang, Shenling Wang, and Rongfang Bie. (2014). "Vehicular Ad Hoc Networks: Architectures, Research Issues, Methodologies, Challenges, and

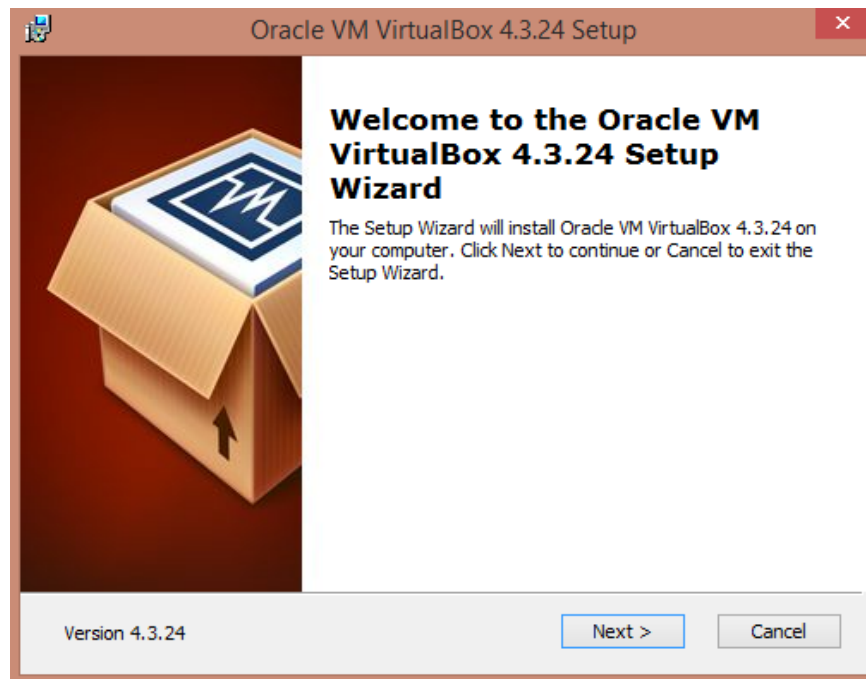
Trends". *Hindawi Publishing Corporation International Journal of Distributed Sensor Networks*.

- [18] Miad Faezipour, Mehrdad Nourani, Adnan Saeed, and Sateesh Addepalli. (2012). "Progress and Challenges in Intelligent Vehicle Area Networks". *Communications of the ACM*.
- [19] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. "The Broadcast Storm Problem in a Mobile Ad Hoc Network". *Department of Computer Science and Information Engineering, National Central University, Chung-Li, 32054, Taiwan*.
- [20] N. Wisitpongphan, O.K. Tonguz, J.S. Parikh, P. Mudalige, F. Bai, V. Sadekar. (2007). "Broadcast Storm Mitigation Techniques in Vehicular Ad Hoc Network". *IEEE Wireless Communications*.
- [21] Yun-Wei Lin, Yuh-Shyan Chen, Sing-Ling Lee. (2010). "Routing Protocols in Vehicular Ad Hoc Networks: A survey and Future Perspectives". *Journal of Information Science and Engineering* 26,913-932
- [22] Hamed Noori. "Realistic Urban Traffic Simulation as Vehicular Ad-Hoc Network (VANET) via Veins Framework". *Proceeding of the 12<sup>th</sup> Conference of Fruct Association*.
- [23] Francisco J. Martínez, Chai Keong Toh, Juan Carlos Cano, Carlos T. Calafate, Pietro Manzoni. (2009). "A survey and comparative study of simulators for vehicular ad hoc networks (VANETs)". *John Wiley & Sons, Ltd*.
- [24] Evjola Spaho, Leonard Barolli, Gjergji Mino, Fatos Xhafa, Vladi Kolici. (2011). "VANET Simulators: A Survey on Mobility and Routing Protocols". *IEEE, 2011 International Conference on Broadband and Wireless Computing, Communication and Applications*.
- [25] András Varga, Rudolf Hornig. (2008). "An Overview of The OMNeT++ Simulation Environment". *SIMUTools, Marseille, France*.
- [26] "INET Framework for OMNeT++ Manual". (2012). Available on: <http://omnetpp.org/doc/inet/api-current/neddoc/index.html>
- [27] <http://mixim.sourceforge.net/>
- [28] Michael Behrisch, Laura Bieker, Jakob Erdmann, Daniel Krajzewicz. (2011). "SUMO – Simulation of Urban Mobility". *SIMUL 2011: The Third International Conference on Advances in System Simulation*.
- [29] Daniel Krajzewicz, Georg Hertkorn, Peter Wagner, Christian Rössel. "SUMO (Simulation of Urban Mobility) An open-source traffic simulation". *German Aerospace Centre, Institute for Transportation Research, Centre for Applied Informatics Cologne, Germany*.
- [30] SUMO User Documentation. Available on: [http://sumo.dlr.de/wiki/SUMO\\_User\\_Documentation](http://sumo.dlr.de/wiki/SUMO_User_Documentation)
- [31] Institute of Transportation Systems web site: [http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931\\_read-41000/](http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/)
- [32] Veins documentation, available on: <http://veins.car2x.org/documentation/>
- [33] TraCI documentation, available on: <http://sumo.dlr.de/wiki/TraCI/Protocol>
- [34] MJ (Thinus) Booyen. (2012). "Tutorial: Simulating VANET and ITS (using OMNeT++ and SUMO)". *University Stellenbosch, Department of Electrical and Electronic Engineering*.
- [35] Christoph Sommer. (2011). "Car-to-X Communication in Heterogeneous Environments". *Universität Erlangen – Nürnberg*.
- [36] Frederik Ramm. (2014). "OpenStreetMap Data in Layered GIS Format".

## ANNEXS

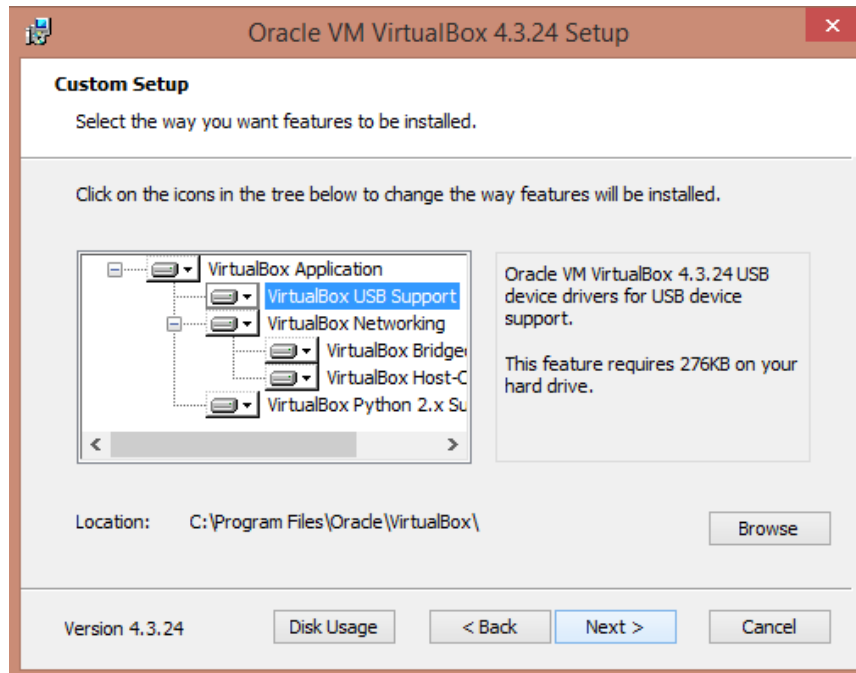
### A. How install Virtual Box and create a virtual machine with Linux Ubuntu System

First of all, make a folder **Omnet Ubuntu** in your Desktop. Then, download there the Virtual Box from the website: <https://www.virtualbox.org/wiki/Downloads>, to date version is 4.3.24. Figure A.1 shows the beginning of the installation by double clicking on the **VirtualBox-4.3.24-98716-Win** which you have downloaded. Press **Next**.



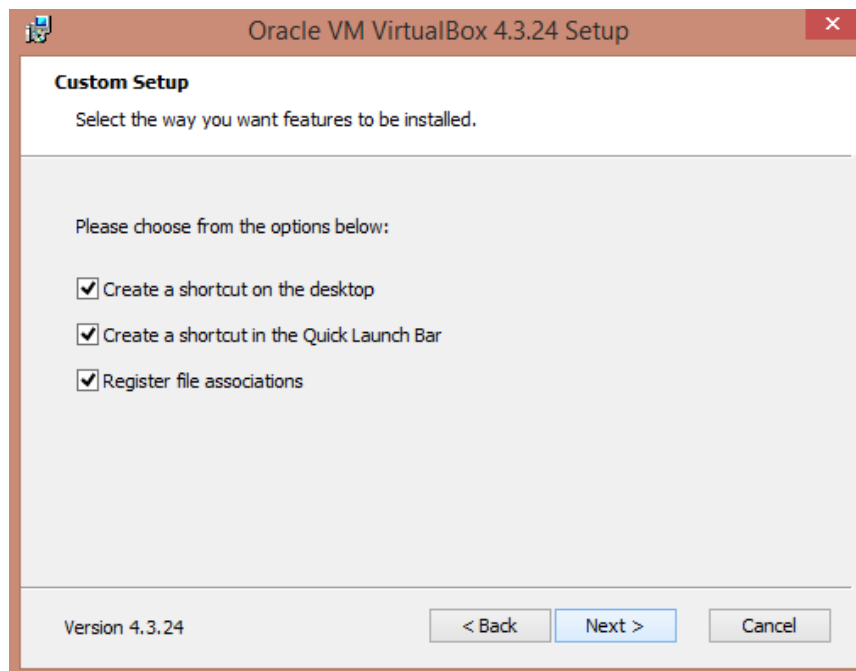
*Figure A.1 Beginning of the installation of the Virtual Box.*

Then, in the next window press **Next** in order to maintain the default destination folder, as it is illustrated in Figure A.2. If you want to change this folder you can press Browse.



*Figure A.2 Select the destination folder.*

Figure A.3 shows the window in which you can choose to create or not shortcuts press **Next**.



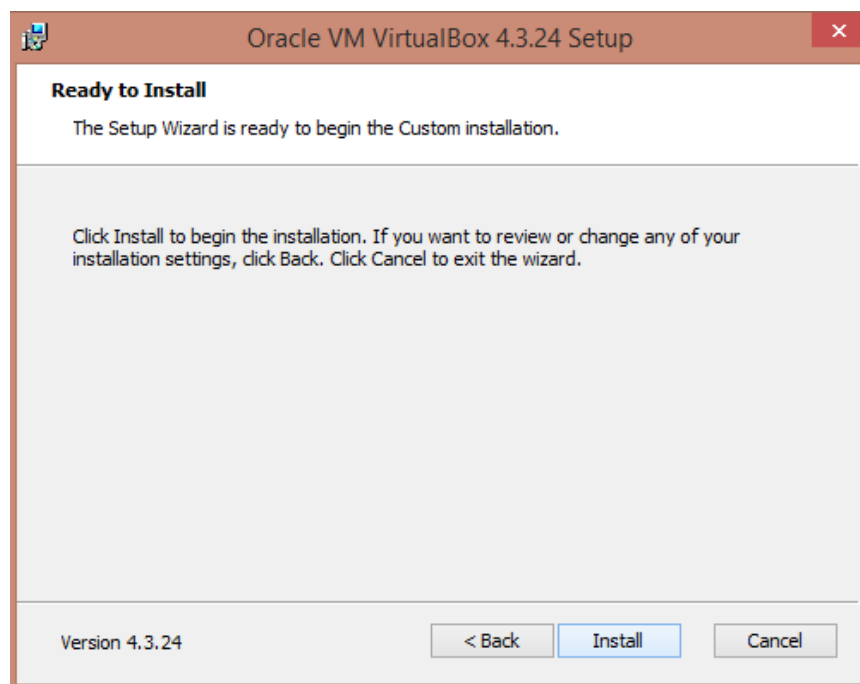
*Figure A.3 Create shortcuts.*

Then, in the next window press **Yes** in order to continue with the installation, as it is shown in Figure A.4.



*Figure A.4 Warning about network interfaces.*

Then, click on **Install** to begin the installation, as it is illustrated in Figure A.5.



*Figure A.5 Install the Virtual Box.*

The installation begins. It will take a while. Finally, the VirtualBox-4.3.24-98716-Win was installed successfully. Click on **Finish** button, as it is shown in Figure A.6.



Figure A.6 Installation is completed successfully.

## A.1 How to create a virtual machine in Virtual Box

In this section you can see how to create a virtual machine of Linux Ubuntu 14.04 in Virtual Box 4.3.24. Download the ISO of Ubuntu 14.04 from this website:

<http://www.ubuntu.com/download/desktop> and save in the **Omnet Ubuntu** folder which you have created in the previous section.

Then, run the Virtual Box by double clicking on the **Oracle VM VirtualBox** icon created in your desktop and click on **Máquina > Nueva**, as it is shown in Figure A.7.

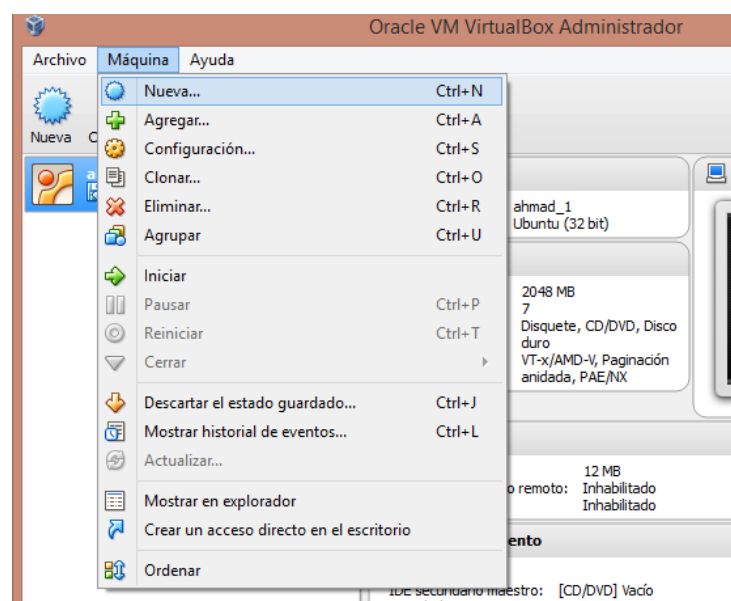
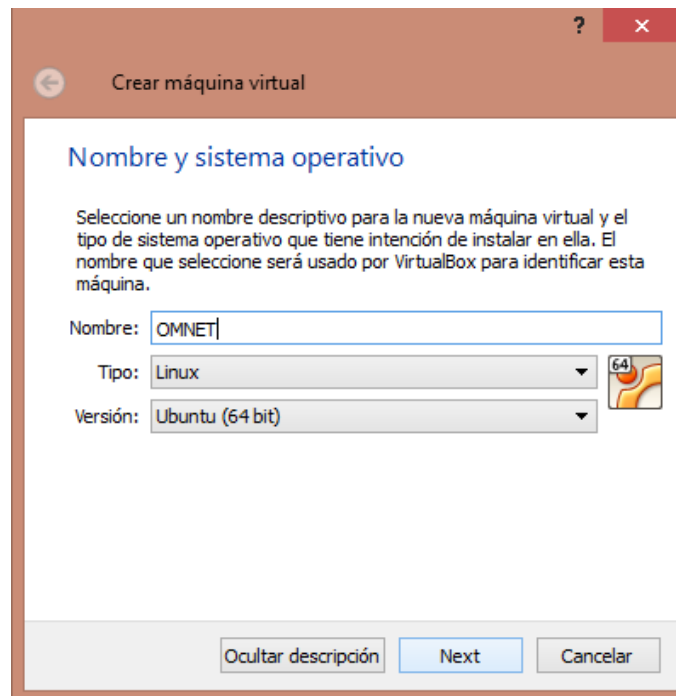


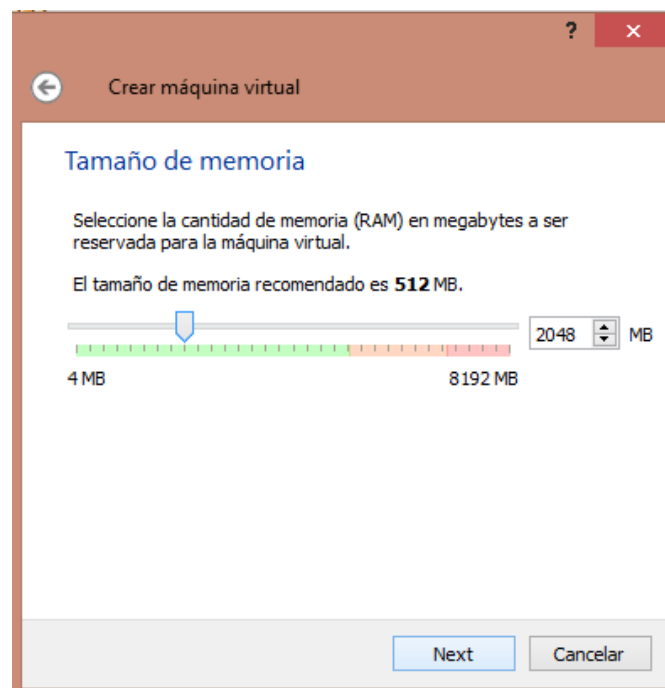
Figure A.7 Create a new virtual machine.

Then, write a name, for example **OMNET**, select **Tipo: Linux**; **Versión: Ubuntu (64bit)** and press **Next**, as it is illustrated in Figure A.8.



*Figure A.8 Select the Operating System of the virtual machine.*

In the next window select the amount of RAM memory to reserve for the virtual machine, for example 2048 MB, and press **Next**, as it is shown in Figure A.9.



*Figure A.9 Select the amount of RAM memory.*

Then, select the option **Crear un disco duro virtual ahora** and click on **Crear** button, as it is illustrated in Figure A.10.

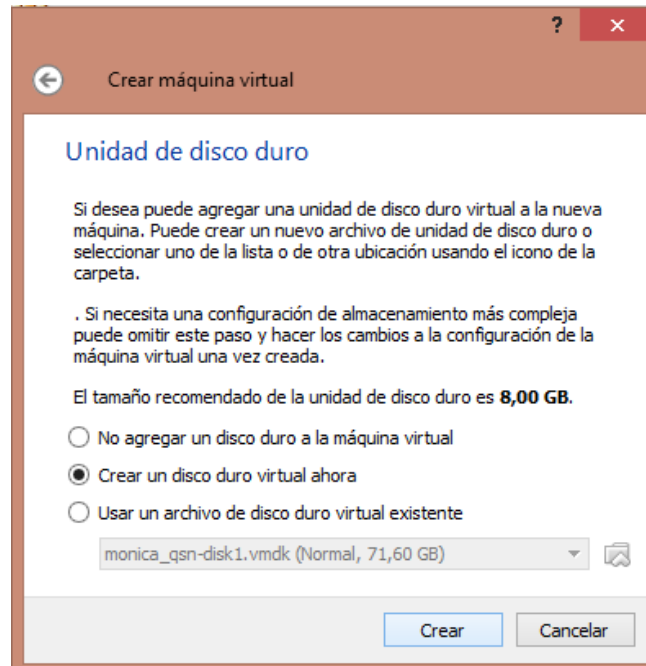


Figure A.10 Create a virtual hard drive.

In the next window select the type of the virtual hard drive, for example the option **VDI (VirtualBox Disk Image)** and press **Next**, as it is shown in Figure A.11.

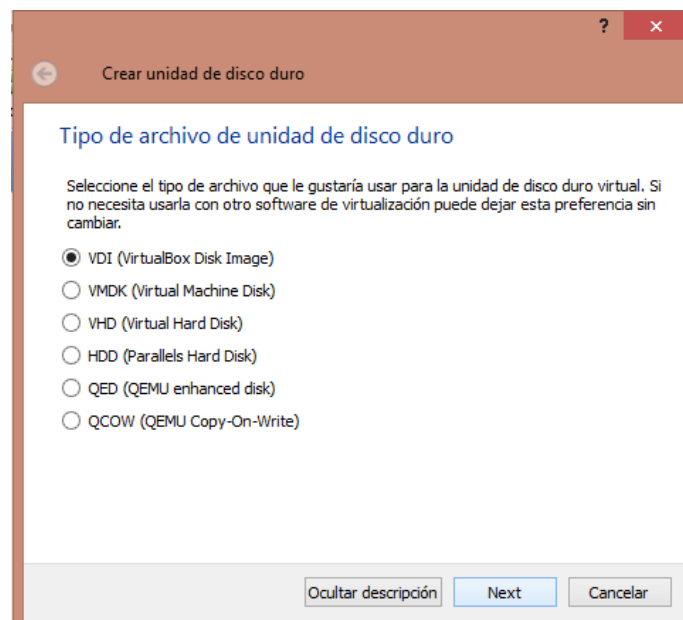


Figure A.11 Select the type of the virtual hard drive.

Then, select the type of storage in the virtual hard drive, for this manual the option **Tamaño fijo** and click on **Next** button, as it is illustrated in Figure A.12.

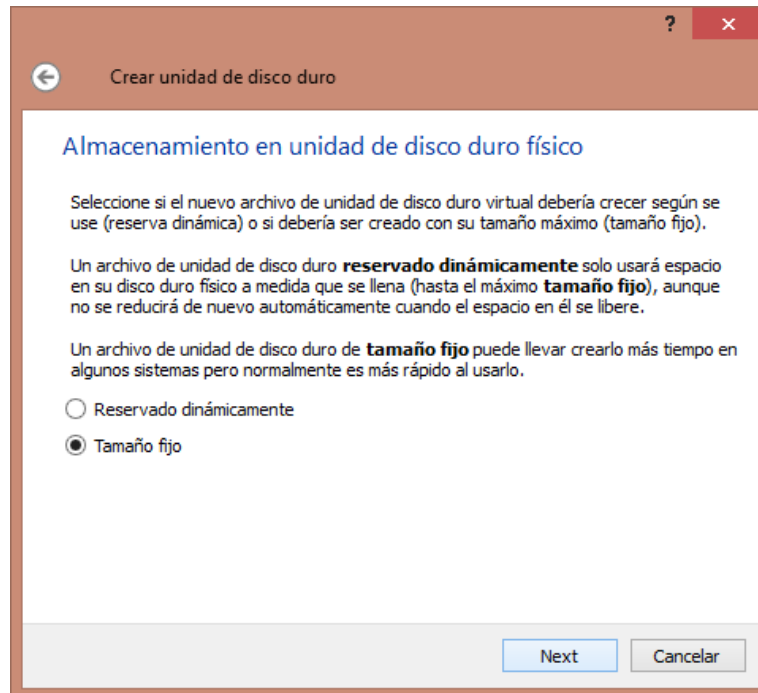


Figure A.12 Select the type of storage in the virtual hard drive.

In the next window write a name and select the maximum capacity for the virtual hard drive. For example **OMNET** and **20,00 GB**, as it is shown in Figure A.13. Then, press **Crear**. It will take a while.

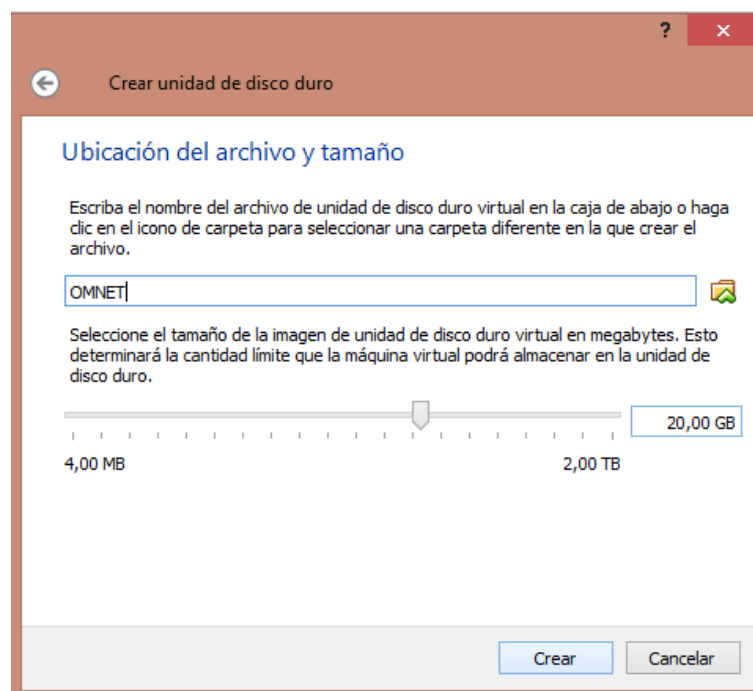


Figure A.13 Location and size of the virtual hard drive.

Then, select the virtual machine and click on **Iniciar**, as it is illustrated in Figure A.14.

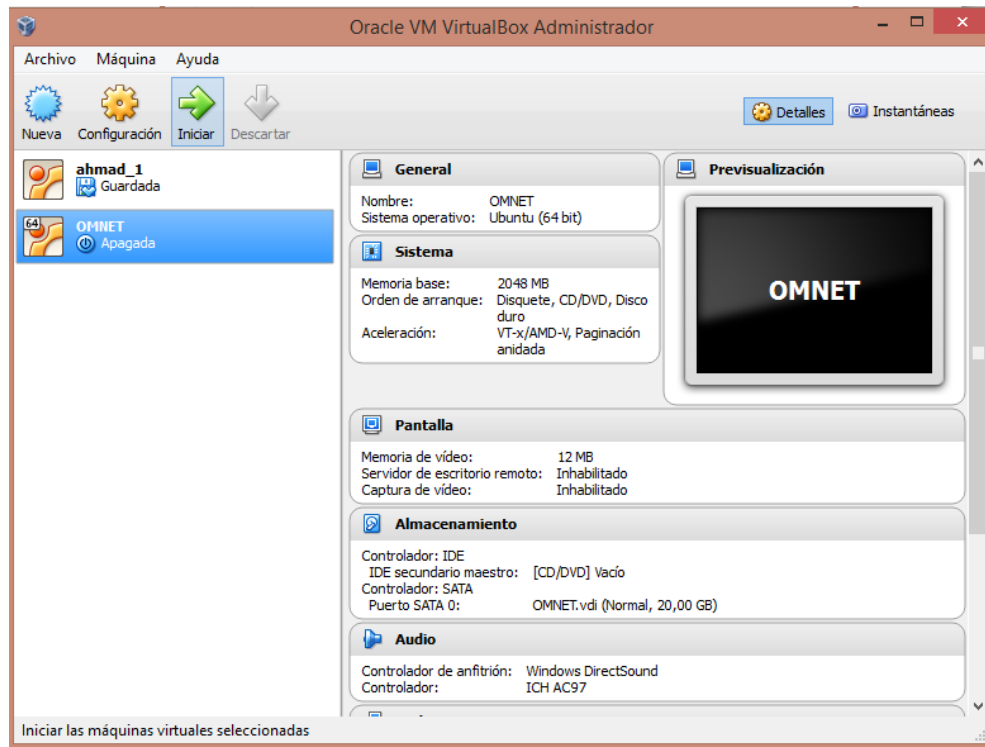


Figure A.14 Run the virtual machine.

Then, click on that icon in the right side of the window in order to select the ISO of Ubuntu which you have downloaded, as it is show in Figure A.15.

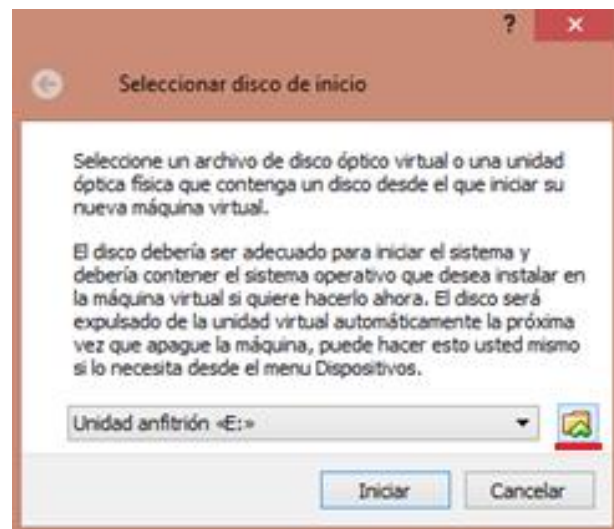
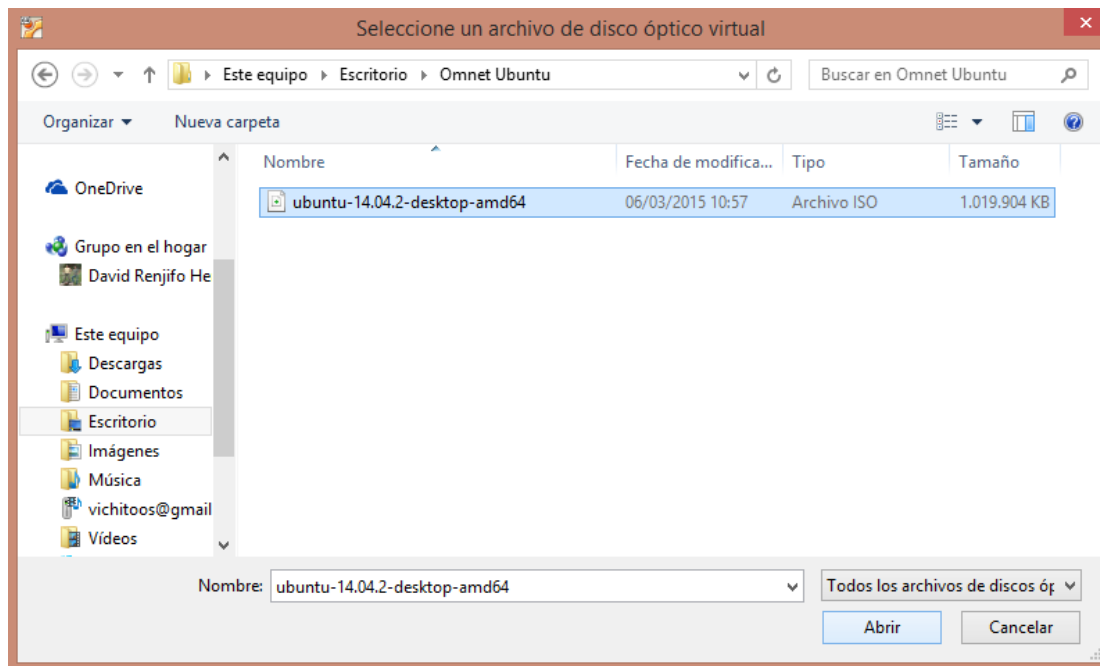


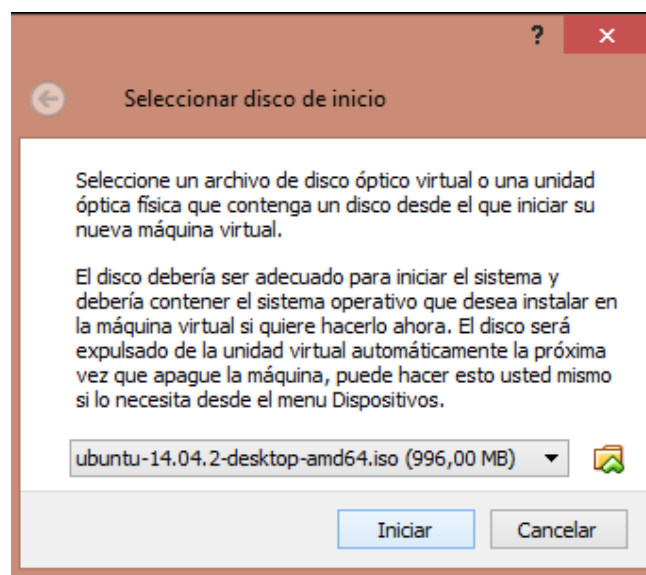
Figure A.15 Select the startup disk.

In the next window choose the directory of the ISO of Ubuntu which is found inside the **Omnet Ubuntu** folder. Then, click on **Abrir**, as it is illustrated in Figure A.16.



*Figure A.16 Choose the folder with the ISO of Ubuntu.*

Then, click on **Iniciar** button, as it is shown in Figure A.17.



*Figure A.17 Run the ISO of Ubuntu in the virtual machine.*

Then, select the language and click on **Instalar Ubuntu** button, as it is illustrated in Figure A.18.

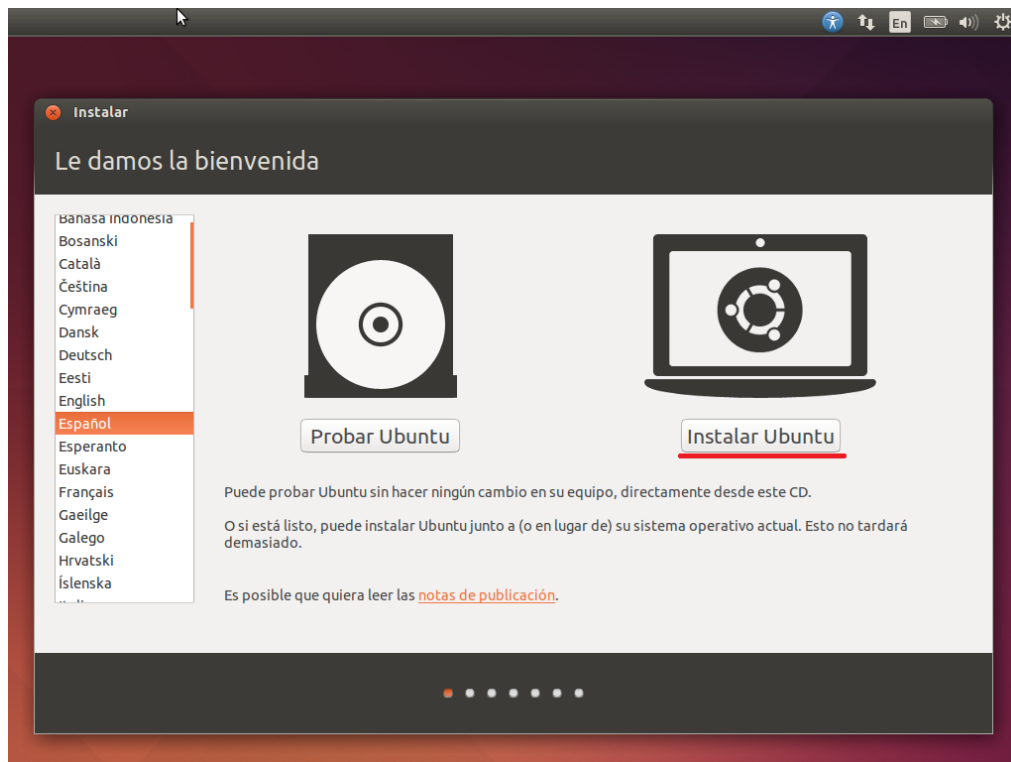


Figure A.18 Select language.

In the next window click on **Continuar**. This window shows some recommendations before the installation of Ubuntu, as it is illustrated in Figure A.19.

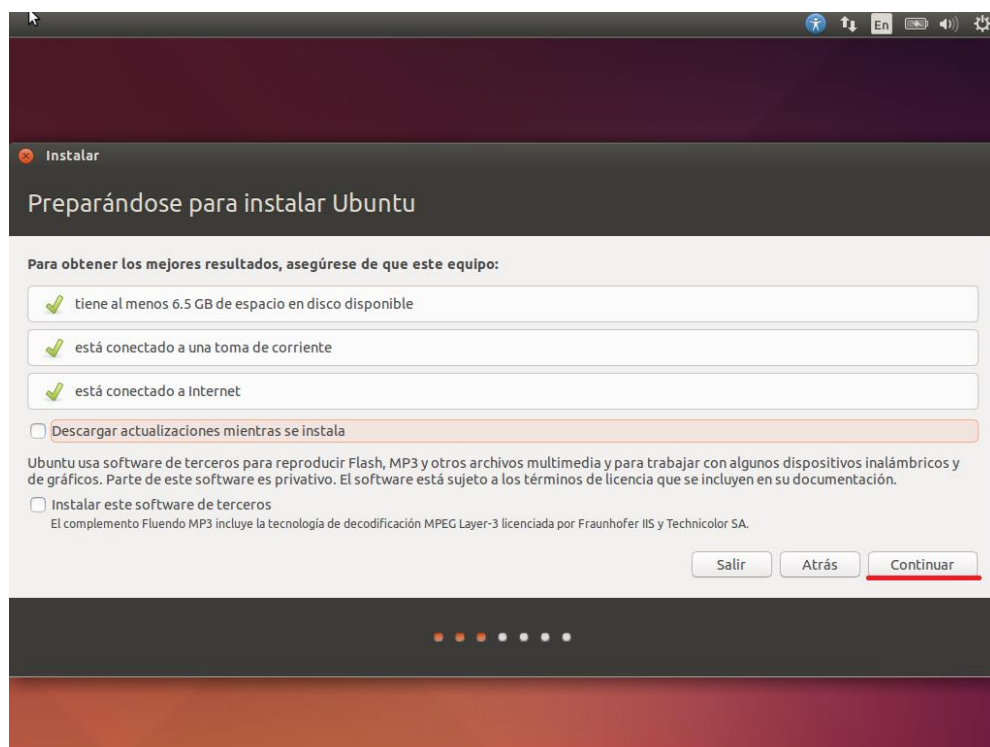


Figure A.19 Preparing to install Ubuntu.

Then, select the option **Borrar disco e instalar Ubuntu** and press **Instalar ahora**, as it is illustrated in Figure A.20.

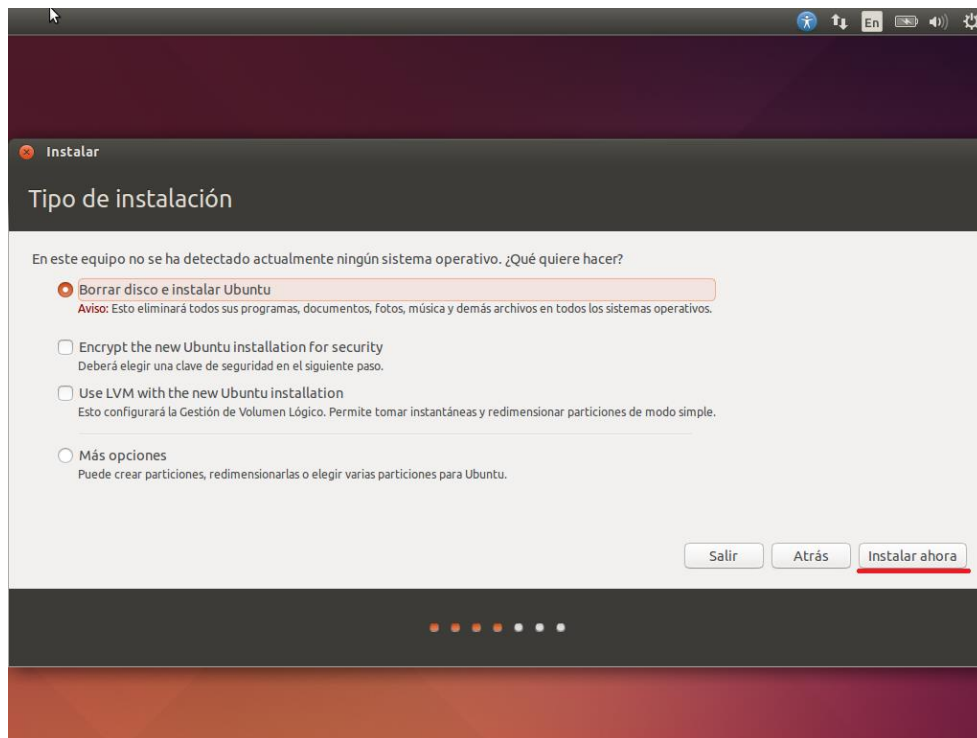


Figure A.20 Select the type of the installation.

In the next window press **Continuar** in order to continue with the installation, as it is shown in Figure A.21.

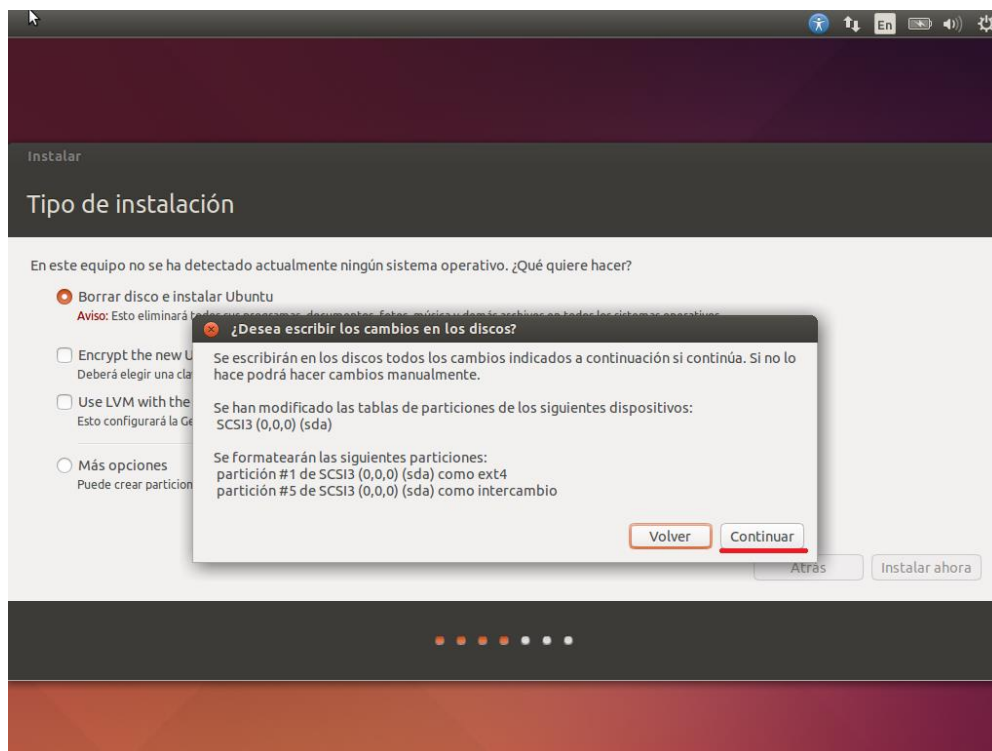


Figure A.21 Accept format of the partition.

Then, write the city where you live and press **Continuar**, as it is illustrated in Figure A.22.

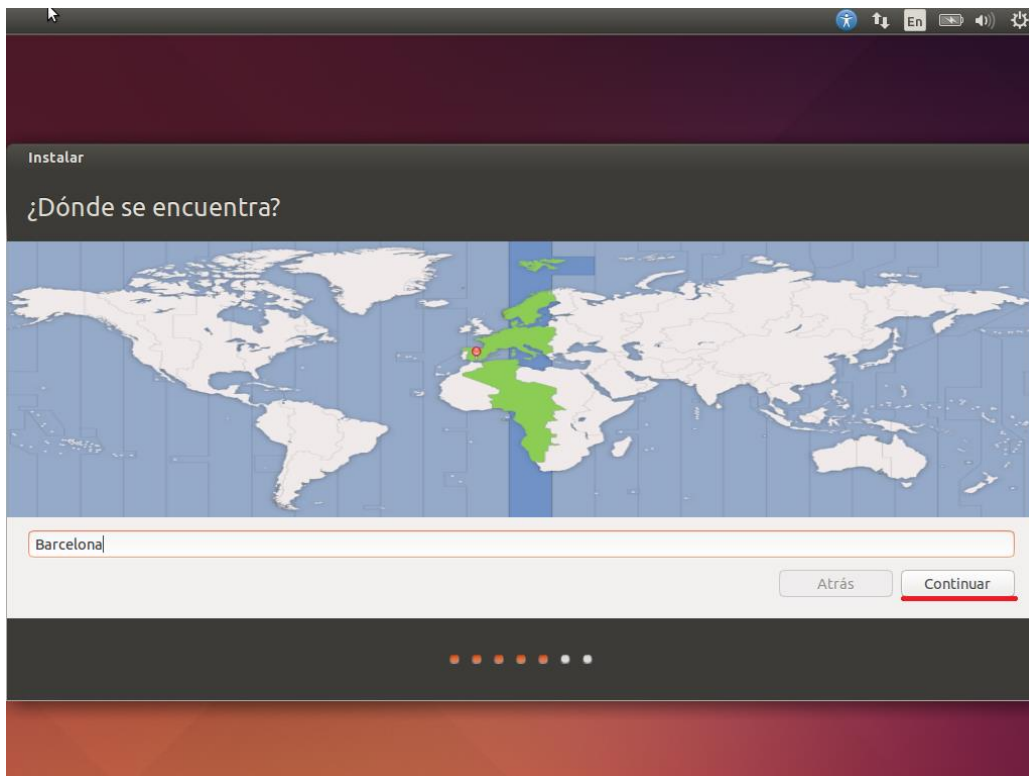


Figure A.22 Select the city.

In the next window select the language for the keyboard and press **Continuar**, as it is shown in Figure A.23.

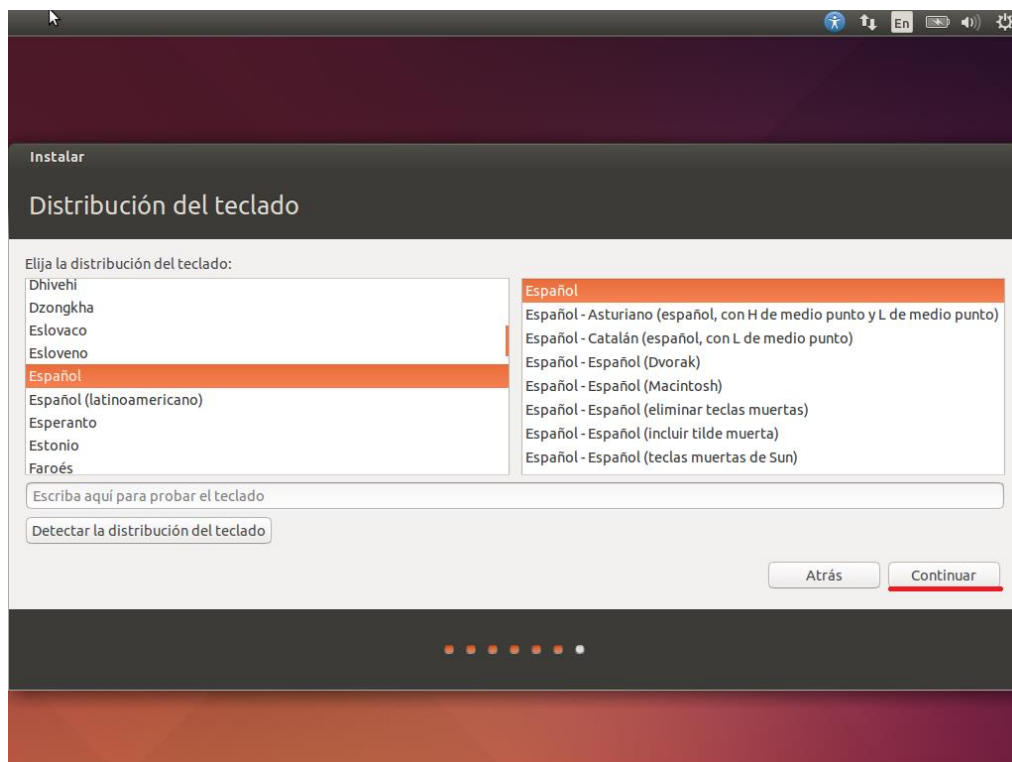
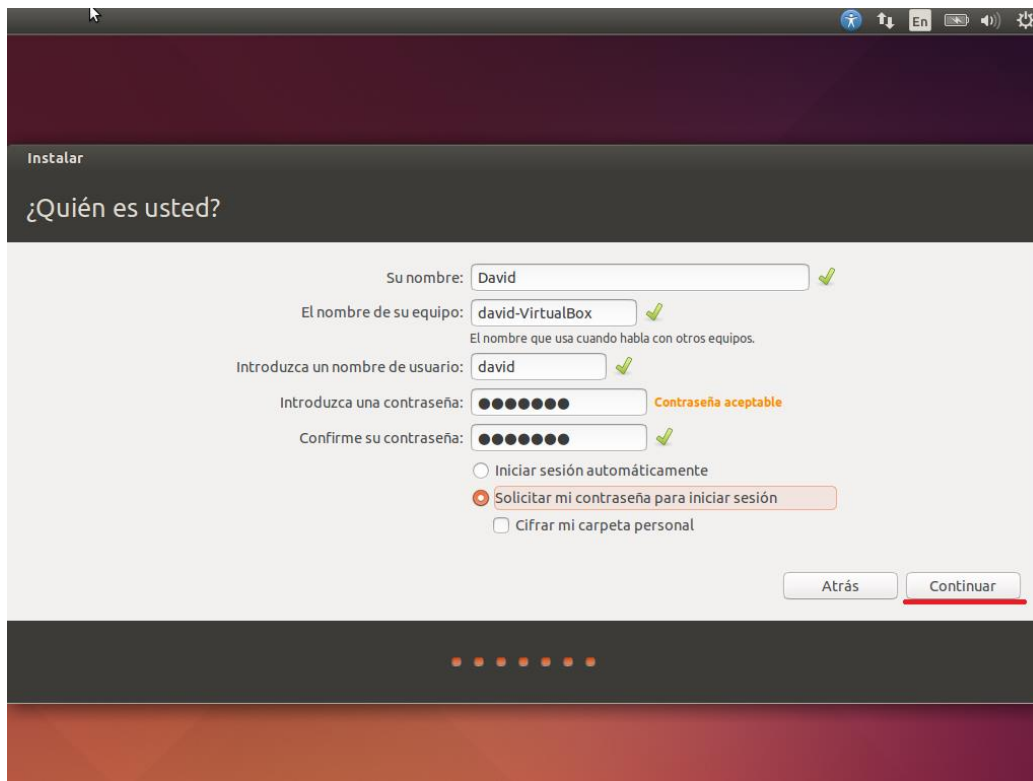


Figure A.23 Select the language for the keyboard.

Then, write a name of user and password and press **Continuar**, as it is illustrated in Figure A.24.



Instalar

¿Quién es usted?

Su nombre: David ✓

El nombre de su equipo: david-VirtualBox ✓  
El nombre que usa cuando habla con otros equipos.

Introduzca un nombre de usuario: david ✓

Introduzca una contraseña: ●●●●●● Contraseña aceptable

Confirme su contraseña: ●●●●●● ✓

☐ Iniciar sesión automáticamente

☒ Solicitar mi contraseña para iniciar sesión

☐ Cifrar mi carpeta personal

Atrás Continuar

Figure A.24 User and password.

The installation of Ubuntu will begin. It will take a while, as it is shown in Figure A.25.

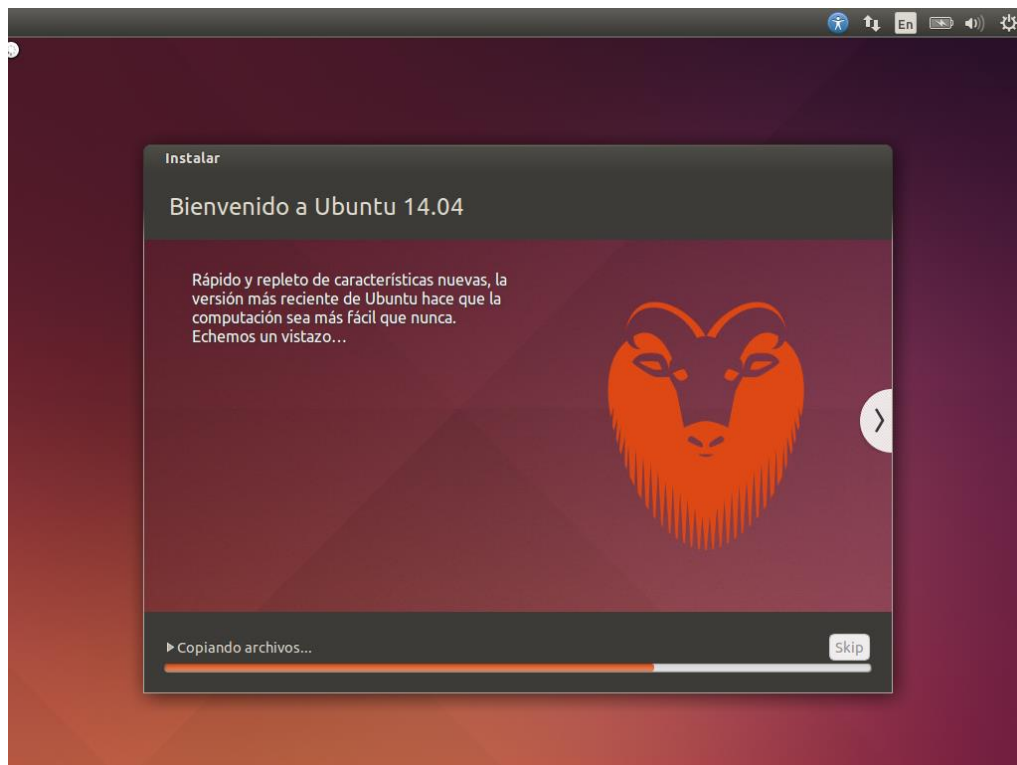
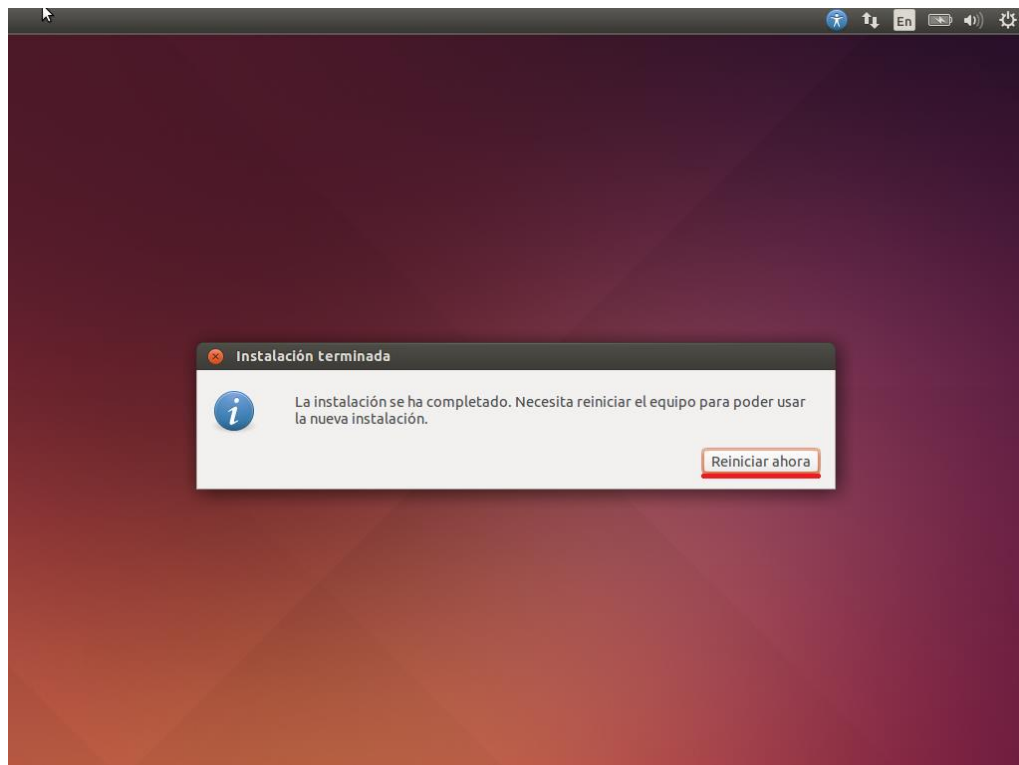


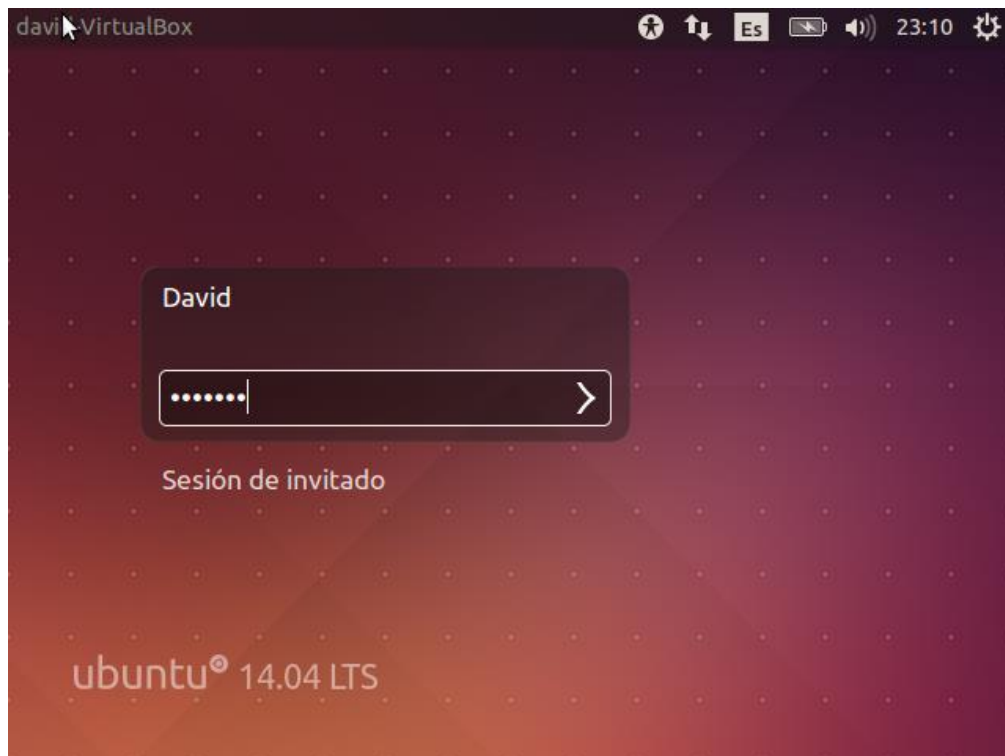
Figure A.25 Installation of Ubuntu.

Then, restart the virtual machine in order to finish the installation by pressing **Reiniciar ahora**, as it is illustrated in Figure A.26.



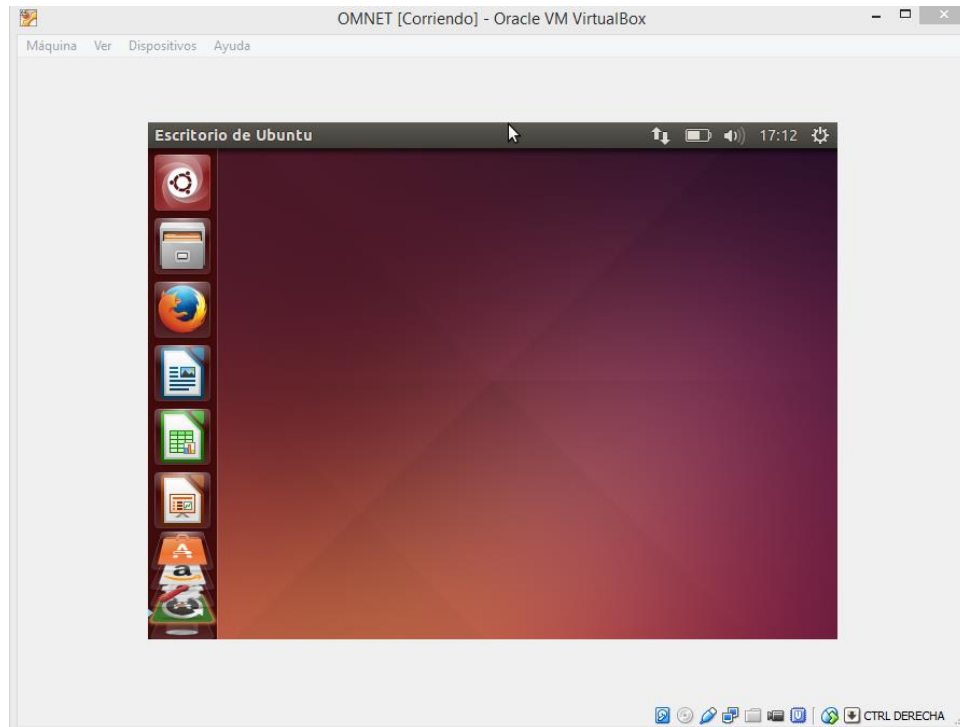
*Figure A.26 Restart to finish the installation.*

Then, login with your password and press **Enter**, as it is shown in Figure A.27.



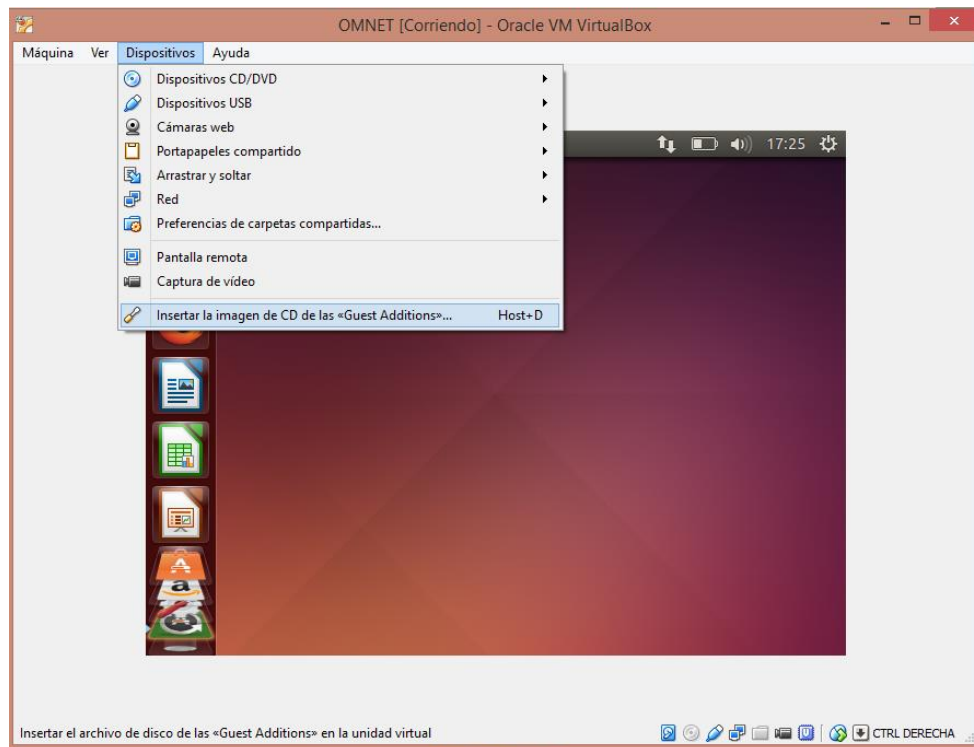
*Figure A.27 Login in the virtual machine.*

The Ubuntu was installed successfully but you cannot see in full screen, as it is illustrated in Figure A.28.



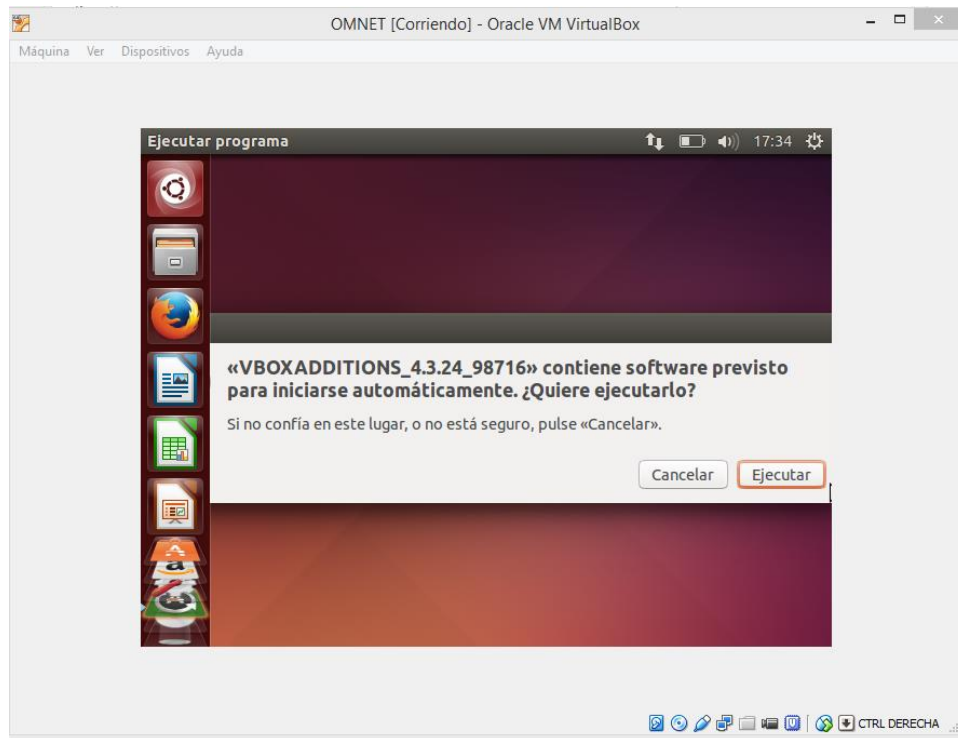
*Figure A.28 Full screen is not possible.*

In order to fix that, in the virtual box window click **on Dispositivos > Insertar la imagen de CD de las Guest Additions**, as it is illustrated in Figure A.29.



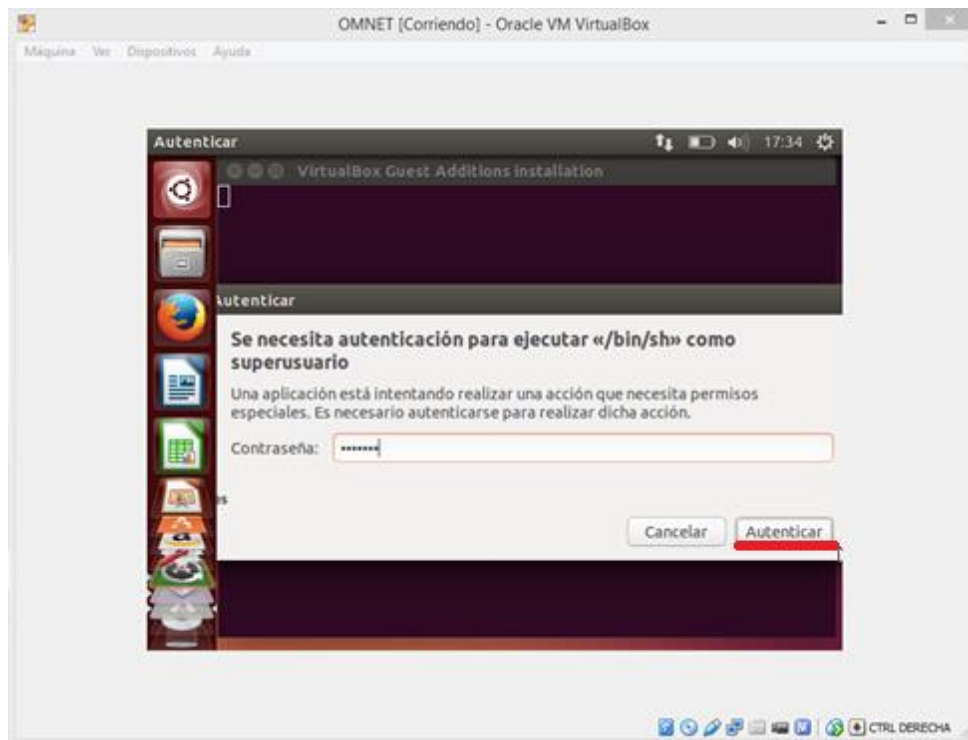
*Figure A.29 Insert the ISO of Guest Additions.*

Then, run VBOXADDITIONS\_4.3.24\_98716 by clicking on **Ejecutar** in the message box, as it is shown in Figure A.30.



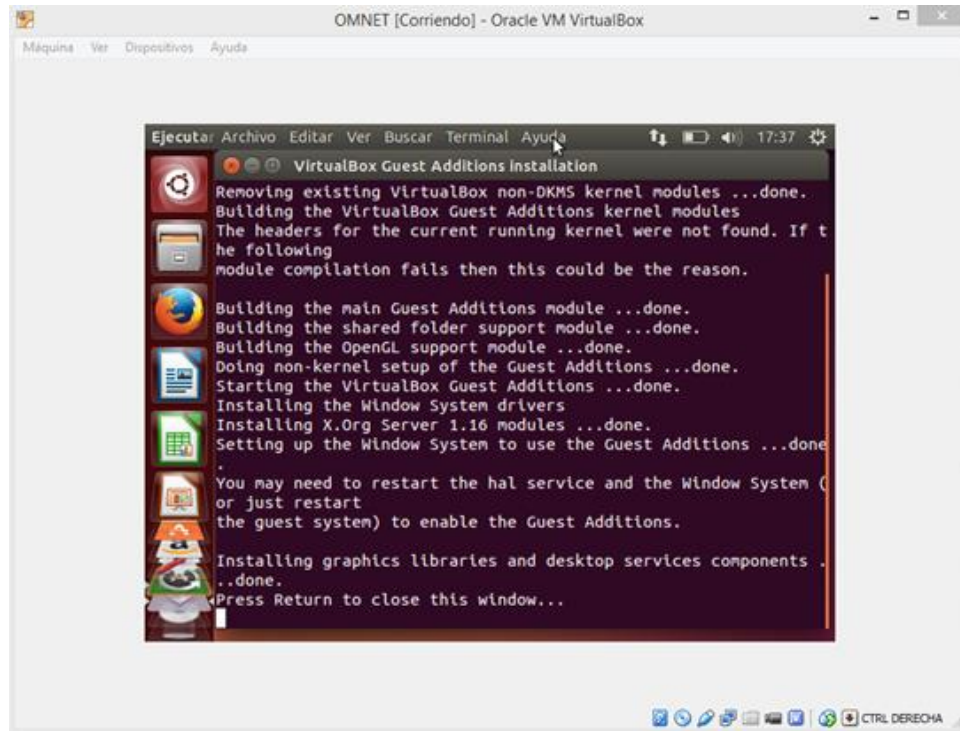
*Figure A.30 Run VBOXADDITIONS\_4.3.24\_98716.*

Then, write your password in order to permit the installation, as it is illustrated in Figure A.31 and press **Autenticar**.



*Figure A.31 Permit the installation by writing your password.*

Then, the graphics libraries and desktop services components will be installed press **Enter** when it is completed, as it is shown in Figure A.32.

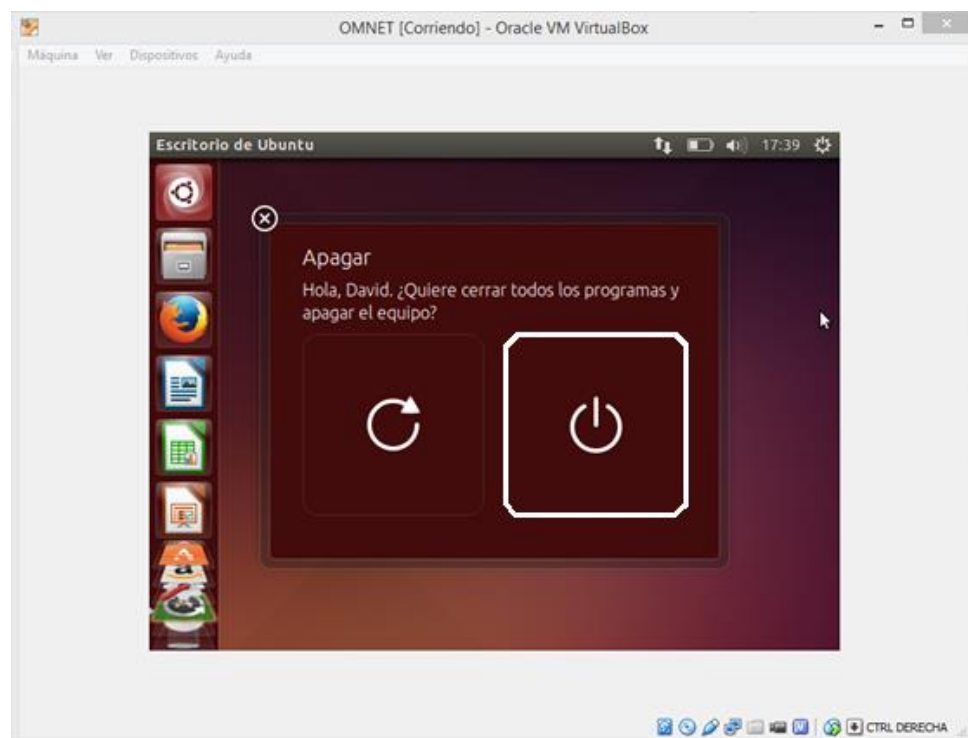


*Figure A.32 Installing graphics libraries and desktop services.*

Then, you must shut down the virtual machine in order to the changes takes effect, as it is illustrated in Figure A.33, so click on that icon in the upper right side and click on Apagar (a), and then click on the icon in the right side (b).



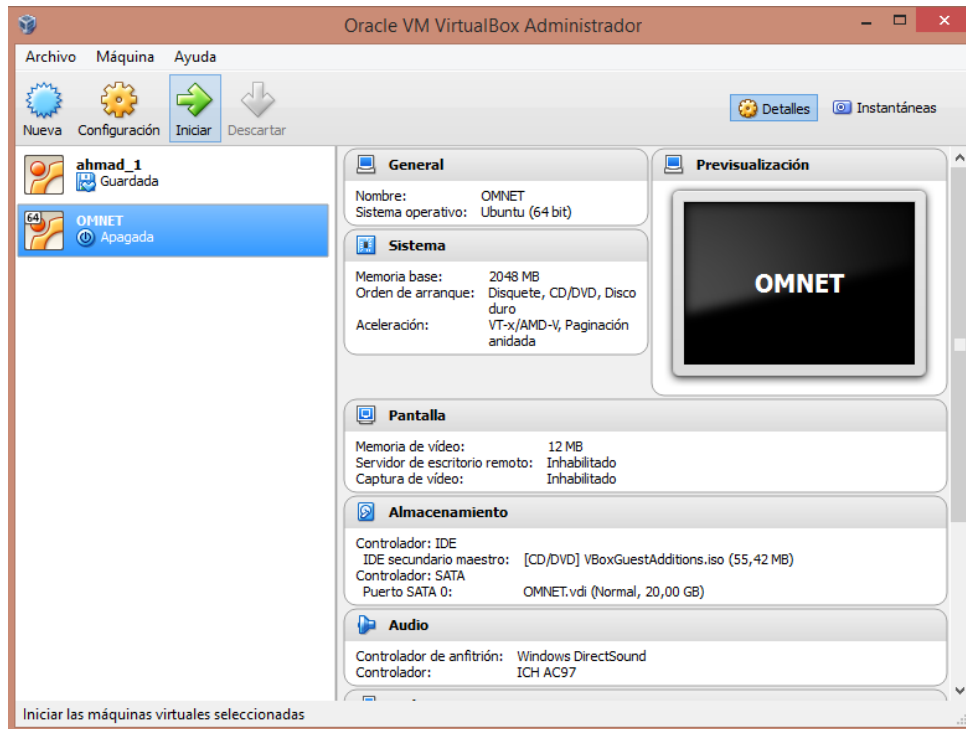
(a)



(b)

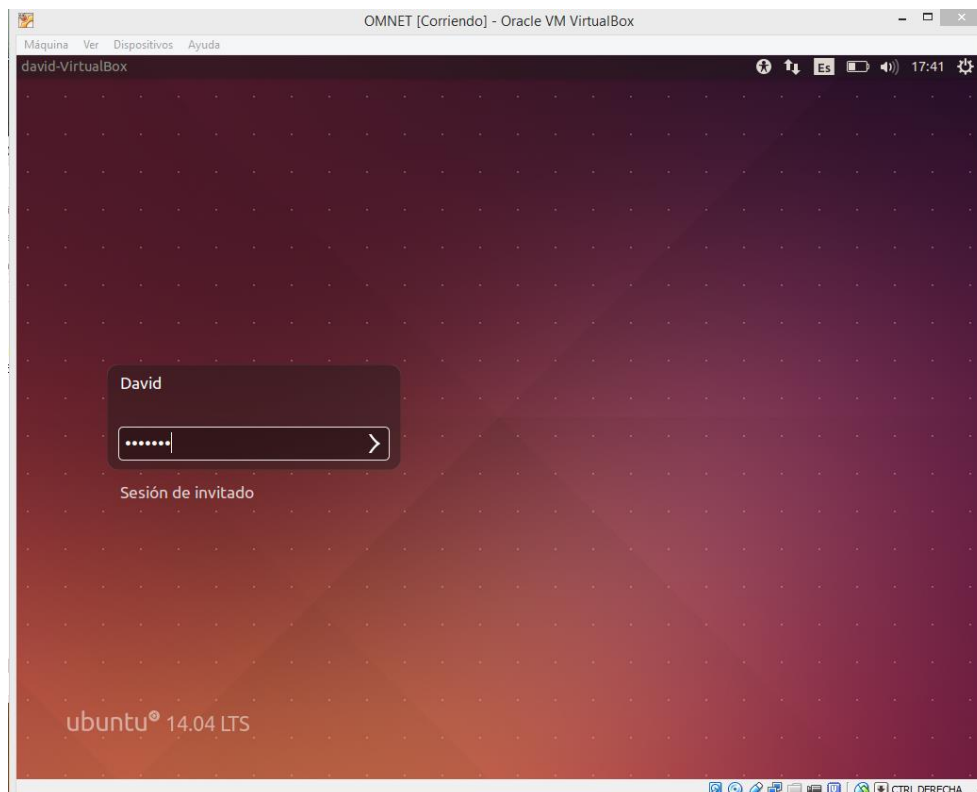
*Figure A.33 (a) and (b) shut down virtual machine.*

Then, turn on again the virtual machine, so select the virtual machine that you have created and click on **Iniciar**, as it is shown in Figure A.34.



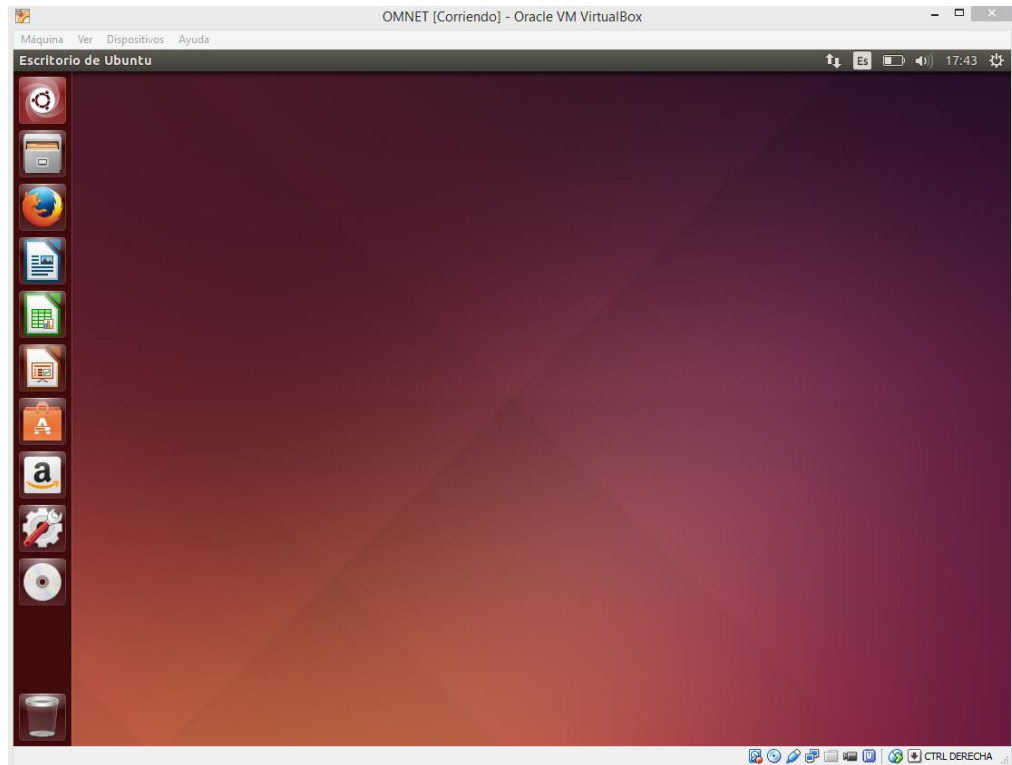
*Figure A.34 Turn on the virtual machine.*

Finally, login in the system, as it is shown in Figure A.35.



*Figure A.35 Login in the system.*

As you can see in Figure A.36, the Ubuntu is run in full screen mode.



*Figure A.36 Ubuntu in full screen mode.*

The installation of Ubuntu was completed successfully.

## B. Installation of SUMO, OMNeT++ and Veins simulator

This is a guide for installation of SUMO, OMNeT++ and VEINS simulators in both Windows and Linux Systems.

Since many different teams and individual developers work to implement improvements and new features, this type of open source tools evolves relatively quickly. So the markings on this project concerning the versions of the modules may lose value in the near future. To date, it is recommended to use version 0.22.0 of SUMO, version 4.6 of OMNeT++, and version 4a2 of VEINS.

The installation in Microsoft Windows does not require any previous configuration. In case of Linux Environment a new installation of operating system is required.

### B.1 Get the source code for the programs

#### B.1.1 Get the source code for SUMO

For both Windows and Linux Systems download *sumo-src-0.22.0.tar.gz* from this website <http://sourceforge.net/projects/sumo/files/sumo/>

#### B.1.2 Get the source code for OMNeT++

For both Windows and Linux Systems go to the following website and download *omnetpp-4.6-src*: <http://www.omnetpp.org/omnetpp>

#### B.1.3 Get the source code for VEINS

For both Windows and Linux users download *veins-4a2.zip* from the website: <http://veins.car2x.org/download/>

For Windows users go to the section B.4, for Linux users continue with the next section.

## B.2 Installation in Linux

### B.2.1 Installation of packages in Linux Ubuntu

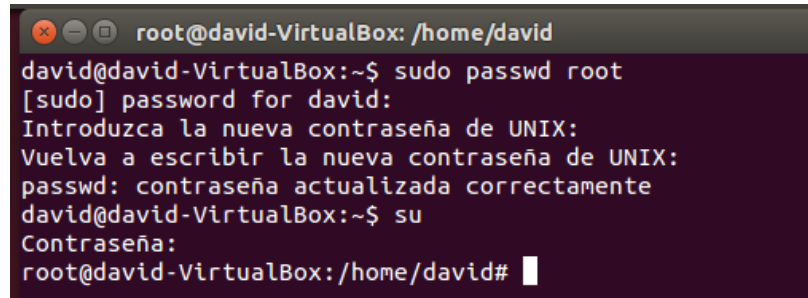
SUMO and OMNeT++ require several packages to be installed on the virtual machine. First of all, you need super-user permissions, open a new terminal and type the command:

```
sudo passwd root
```

Write your password, in this case the password for david user. Then, write twice the password for the user root and type the command:

`su`

Then, write the password for the user root, as it is illustrated in Figure B.1.

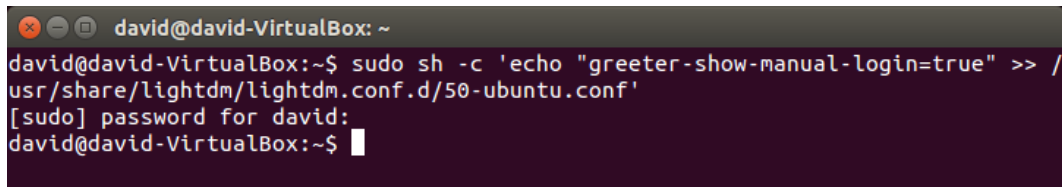


```
root@david-VirtualBox: /home/david
david@david-VirtualBox:~$ sudo passwd root
[sudo] password for david:
Introduzca la nueva contraseña de UNIX:
Vuelva a escribir la nueva contraseña de UNIX:
passwd: contraseña actualizada correctamente
david@david-VirtualBox:~$ su
Contraseña:
root@david-VirtualBox:/home/david#
```

*Figure B.1 Login as user root.*

Then, you must be logged as a root user, in order to access in root desktop type the following command, as it is shown in Figure B.2.

```
sudo sh -c 'echo "greeter-show-manual-login=true" >> /usr/share/lightdm/lightdm.conf.d/50-ubuntu.conf'
```



```
david@david-VirtualBox: ~
david@david-VirtualBox:~$ sudo sh -c 'echo "greeter-show-manual-login=true" >> /usr/share/lightdm/lightdm.conf.d/50-ubuntu.conf'
[sudo] password for david:
david@david-VirtualBox:~$
```

*Figure B.2 Configuration to access into root desktop.*

Then, you must restart the system. As a result, you can login as root user and access into desktop, as it is illustrated in Figure B.3.

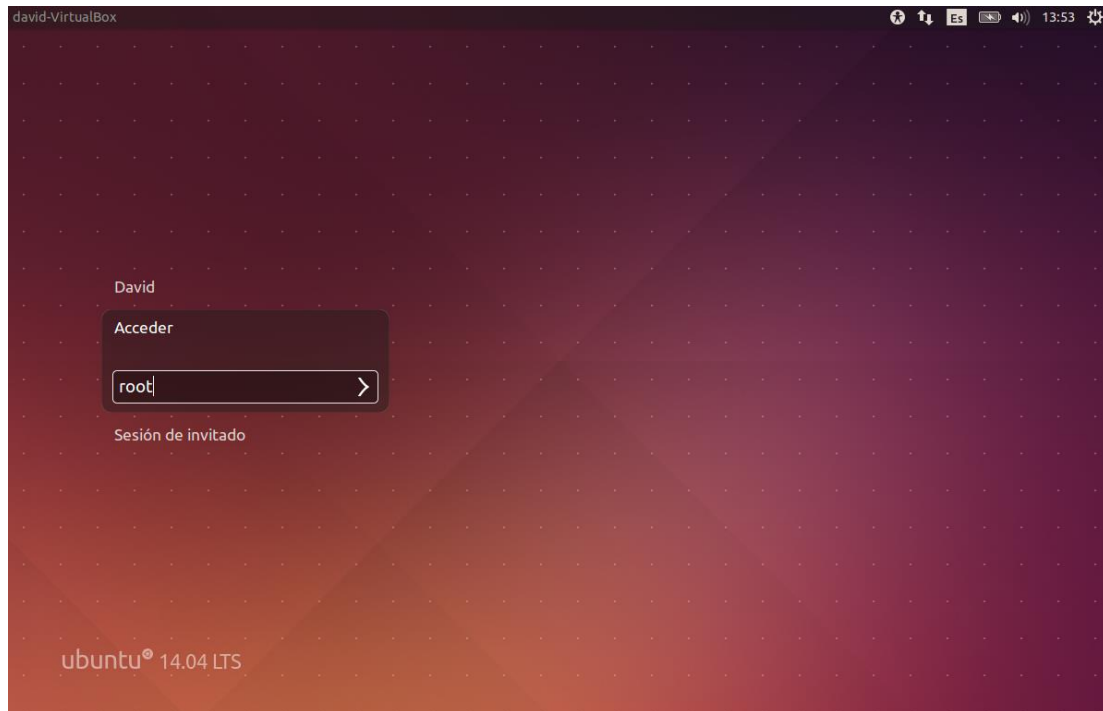


Figure B.3 Login as root user.

In order to install the packages that you need, open a terminal and write the next command, as it is shown in Figure B.4.

*apt-get install aptitude*

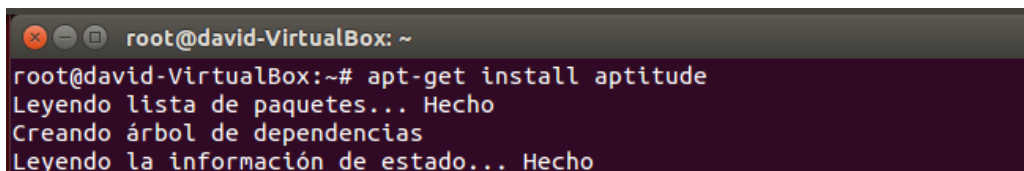


Figure B.4 Installation of aptitude.

At the confirmation questions (*Do you want to continue? [Y/N]*), answer Y. Then, type the command:

*aptitude install build-essential gcc g++ bison flex perl tcl-dev tk-dev blt libxml2-dev zlib1g-dev default-jre doxygen graphviz libwebkitgtk-1.0-0 openmpi-bin libopenmpi-dev libpcap-dev autoconf automake libtool libproj0 libgdal1-dev libfox-1.6-dev libgdal-dev libxerces-c-dev*

In the confirmation questions (*Do you accept this solution? [y/n/q]*), answer Y, as it is illustrated in Figure B.5.

```
root@david-VirtualBox: /home/david
Seleccionando el paquete aptitude previamente no seleccionado.
Preparing to unpack .../aptitude_0.6.8.2-1ubuntu4_amd64.deb ...
Unpacking aptitude (0.6.8.2-1ubuntu4) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Configurando libboost-iostreams1.54.0:amd64 (1.54.0-4ubuntu3.1) ...
Configurando libept1.4.12:amd64 (1.0.12) ...
Configurando aptitude-common (0.6.8.2-1ubuntu4) ...
Configurando libcwidget3 (0.5.16-3.5ubuntu1) ...
Configurando aptitude (0.6.8.2-1ubuntu4) ...
update-alternatives: utilizando /usr/bin/aptitude-curses para proveer /usr/bin/aptitude (aptitude) en m
odo automático
Processing triggers for libc-bin (2.19-0ubuntu6.5) ...
root@david-VirtualBox:/home/david# aptitude install build-essential gcc g++ bison flex perl tcl-dev tk-
dev blt libxml2-dev zlib1g-dev default-jre doxygen graphviz libwebkitgtk-1.0-0 openmpi-bin libopenmpi-d
ev libpcap-dev autoconf automake libtool libproj0 libgdal1-dev libfox-1.6-dev libgdal-dev libxerces-c-d
ev
```

*Figure B.5 Installation of necessary packages for installation of SUMO simulator.*

## B.2.2 Installation of SUMO

Open a new terminal and make a folder **omnet** inside the *home* folder. Then, unzip the file *sumo-src-0.22.0.tar.gz* which you have downloaded in section B.1.1 by typing the next command, as it is shown in Figure B.6.

```
tar xzf sumo-src-0.22.0.tar.gz
```

```
root@david-VirtualBox: /home/omnet
root@david-VirtualBox:~# cp /root/Descargas/sumo-src-0.22.0.tar.gz /home/omnet/
root@david-VirtualBox:~# cd /home/omnet/
root@david-VirtualBox:/home/omnet# tar xzf sumo-src-0.22.0.tar.gz
root@david-VirtualBox:/home/omnet#
```

*Figure B.6 Unzip the file sumo-src-0.22.0.tar.gz.*

Type the following commands in order to build and install the SUMO binaries, as it is illustrated in Figure B.7, B.8 and B.9.

```
cd sumo-0.22.0/
./configure --prefix=/home/omnet/sumo-0.22.0 --enable-debug
make
make install
```

```
root@david-VirtualBox:/home/omnet# cd sumo-0.22.0/
root@david-VirtualBox:/home/omnet/sumo-0.22.0# ./configure --prefix=/home/omnet/sumo-0.22.0 --enable-debug
```

*Figure B.7 Building the SUMO binaries: ./configure.*

```
Optional features summary
-----
Enabled: Debug InternalLanes DoublePrecision Subsecond TRACI GDAL GUI
Disabled: Profiling MemoryChecks PROJ UnitTests Python
root@david-VirtualBox:/home/omnet/sumo-0.22.0# make
```

*Figure B.8 Building the SUMO binaries: make.*

```
make[1]: se ingresa al directorio «/home/omnet/sumo-0.22.0»
make[1]: No se hace nada para «all-am».
make[1]: se sale del directorio «/home/omnet/sumo-0.22.0»
root@david-VirtualBox:/home/omnet/sumo-0.22.0# make install
```

*Figure B.9 Installing the SUMO binaries: make install.*

Then, set the environment variables by editing `.bashrc` in your home directory. Type the command:

```
gedit ~/.bashrc
```

Add the following line at the end of the file, then save it:

```
export PATH=$PATH:/home/omnet/sumo-0.22.0/bin/
```

Then, type the following command for the changes to take effect, as it is shown in Figure B.10 a) and b).

```
source ~/.bashrc
```

```
root@david-VirtualBox: /home
root@david-VirtualBox:/home# gedit ~/.bashrc
(gedit:29291): dconf-WARNING **: failed to commit cha
```

a)

```
(gedit:29291): dconf-WARNING **: failed to commit changes to dconf: La conexi3n est1 cerrada
root@david-VirtualBox:/home# source ~/.bashrc
root@david-VirtualBox:/home#
```

b)

*Figure B.10 Environment variables a) edit bash b) update bash.*

Finally, you can launch SUMO typing the command **sumo-gui** in the terminal, as it is show in Figure B.11.

```
sumo-gui
```

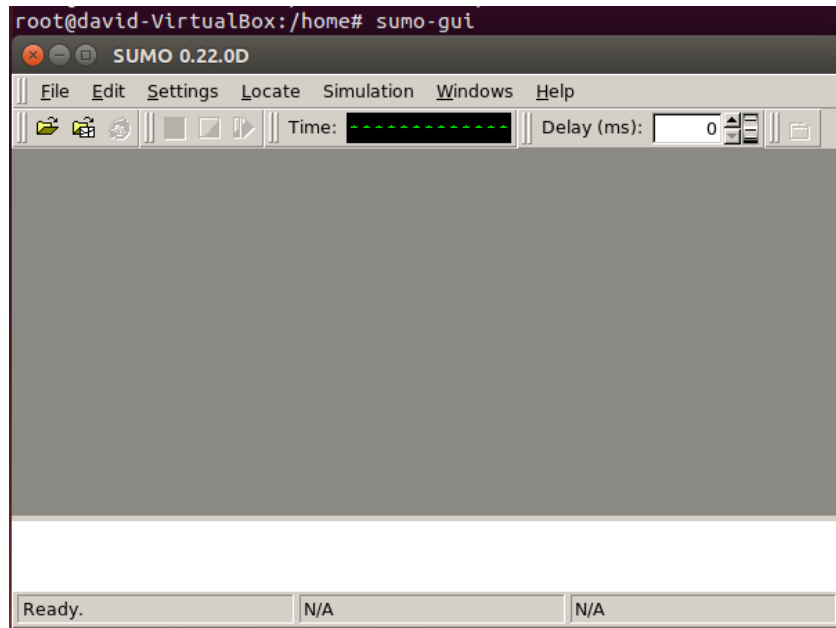


Figure B.11 Run SUMO.

The installation of SUMO was successfully.

### B.2.3 Installation of OMNeT++

Go to the **omnet** folder that you have created in the section B.2.2. Then, unzip the file `omnetpp-4.6-src.tgz` which you downloaded in section B.1.1 by typing the next command, as it is show in Figure B.12.

```
tar xzf omnetpp-4.6-src.tgz
```

```
root@david-VirtualBox:/home/omnet# cp /root/Descargas/omnetpp-4.6-src.tgz /home/omnet/
root@david-VirtualBox:/home/omnet# tar xzf omnetpp-4.6-src.tgz
```

Figure B.12 Unzip omnet-4.6.

Set the environment variables by editing the file `bash`. Type the command:

```
gedit ~/.bashrc
```

Add the following line at the end of the file and save it:

```
export PATH=$PATH:/home/omnet/omnetpp-4.6/bin/
```

Then, type the next command for the changes to take effect, as it is shown in Figure B-13 a) and b).

```
source ~/.bashrc
```

```

root@david-VirtualBox: /home
root@david-VirtualBox:/home/omnet# cd ..
root@david-VirtualBox:/home# gedit ~/.bashrc

(gedit:2318): dconf-WARNING **: failed to commit changes to dconf: La conexi3n e
st3 cerrada

** (gedit:2318): CRITICAL **: log.vala:104: Unable to connect to Zeitgeist: La c
onexi3n est3 cerrada

```

a)

```

(gedit:2318): dconf-WARNING **: failed to commit changes to dconf: La conexi3n e
st3 cerrada
root@david-VirtualBox:/home# source ~/.bashrc
root@david-VirtualBox:/home#

```

b)

Figure B.13 Environment variables for OMNeT++.

After that, in the top-level OMNeT++ directory type the following commands in order to configure, compile and launch OMNeT++, as it is illustrated in Figures B.14, B.15 and B.16.

```
./configure
make
omnetpp
```

```

root@david-VirtualBox:/home# source ~/.bashrc
root@david-VirtualBox:/home# cd omnet/omnetpp-4.6
root@david-VirtualBox:/home/omnet/omnetpp-4.6# ./configure

```

Figure B.14 Configuring the installation of OMNeT++.

```

Scroll up to see the warning messages (use shift+PgUp), and search config.log
for more details. While you can use OMNeT++ in the current configuration,
be aware that some functionality may be unavailable or incomplete.

Your PATH contains /home/omnet/omnetpp-4.6/bin. Good!
root@david-VirtualBox:/home/omnet/omnetpp-4.6# make

```

Figure B.15 Building OMNeT++.

```

Now you can type "omnetpp" to start the IDE
root@david-VirtualBox:/home/omnet/omnetpp-4.6# omnetpp
Starting the OMNeT++ IDE...
root@david-VirtualBox:/home/omnet/omnetpp-4.6#

```

Figure B.16 Launch OMNeT++.

Then, a window will appear that will allow us to install the INET framework and import the OMNeT++ examples. So, press **Ok** and the program will download and install, as it is shown in Figure B.17.

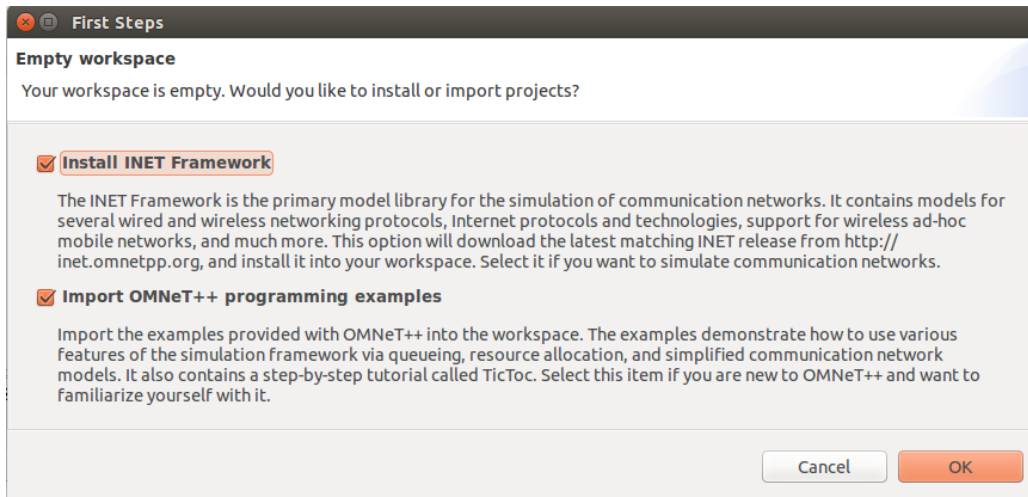


Figure B.17 Install INET framework.

The OMNeT++ was installed successfully. In order to continue with the installation of VEINS simulator go to the section B.4.

## B.3 Installation in Windows

### B.3.1 Installation of SUMO

Make a folder **omnet** in the partition **C:\** and extract there the zip file *sumo-winbin-0.22.0* which you downloaded in the section B.1.1. So, right-click on the zip file in Windows Explorer, and select *Extract All* from the menu. You can also use external programs like 7zip, as it is shown in Figure B.18.

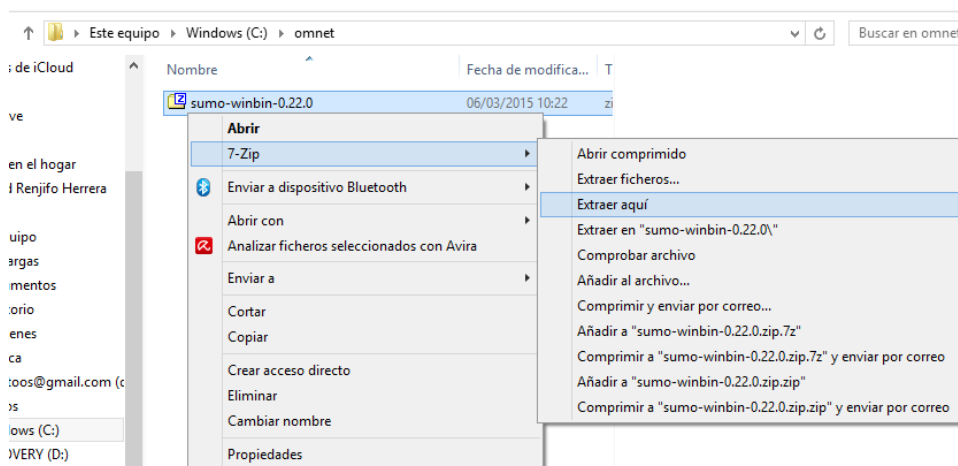


Figure B.18. Extract the zip file *sumo-winbin-0.22.0*.

Then, go to folder **C:/omnet/sumo-0.22.0/bin** and double click on **sumo-gui** to verify that the program runs correctly, as it is shown in Figure B.19.

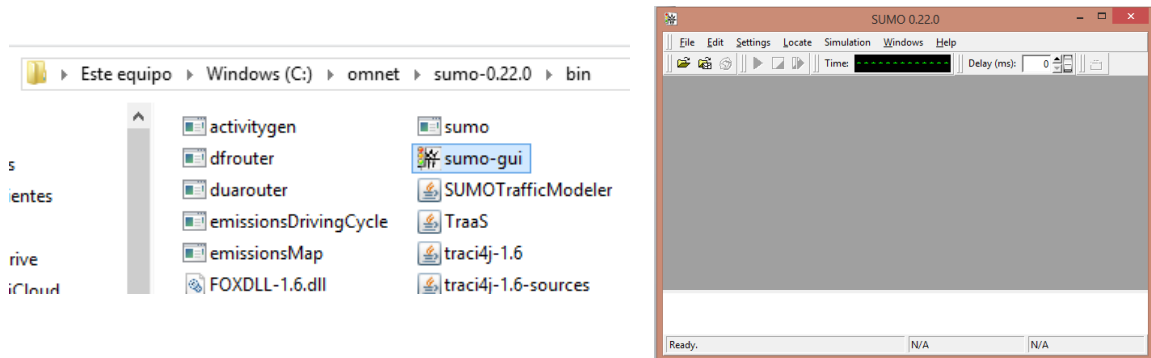


Figure B.19 Run sumo 0.22.0.

The sumo 0.22.0 was installed successfully.

### B.3.2 Installation of OMNeT

Extract the zip file *omnet-4.6-src-windows* which you have downloaded in section B.1.2 in the **omnet** folder that you have created before in section B.3.1, as it is shown in Figure B.20.

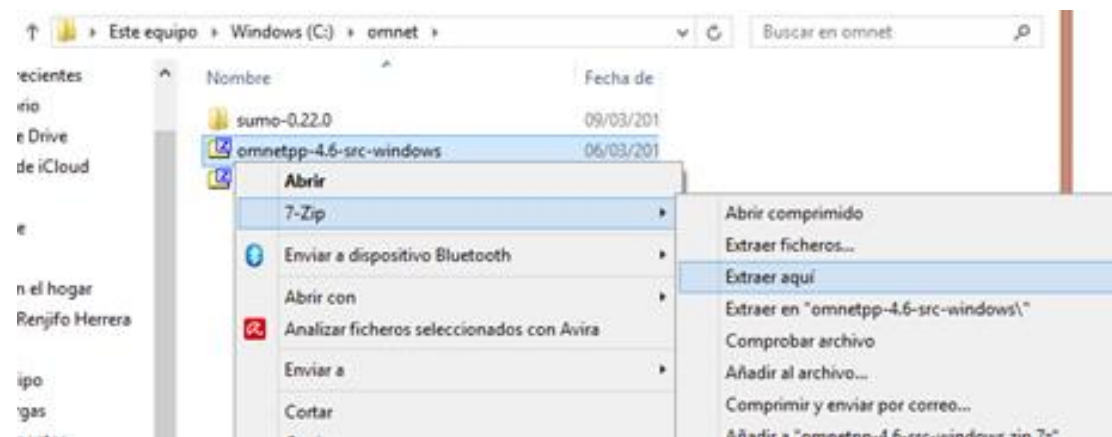


Figure B.20 Extract the zip file *omnet-4.6-src-windows*.

After that, go to the folder **C:/omnet/omnetpp-4.6** and start *mingwenv.cmd* by double-clicking it. It will bring up a console with the MSYS bash shell. After that, build the simulation libraries by entering the following commands:

```
$. ./configure
$. make
```

This process is shown in Figure B.21 (a) and (b).

" data-bbox="214 82 799 182"/>

a)

" data-bbox="217 220 796 334"/>

b)

*Figure B.21 Type the commands `./configure` (a) and `make` (b).*

Now, you can launch the IDE by typing the next command in the console *mingwenv.cmd*, as it is shown in Figure B.22.

omnetpp

" data-bbox="147 503 869 746"/>

*Figure B.22 Start the OMNeT++ IDE.*

Then, a window will appear that will allow you to install the INET framework and import the OMNeT++ examples. So, press **Ok** and the program will download and install, as it is shown in Figure B.23. The OMNeT++ was installed successfully.

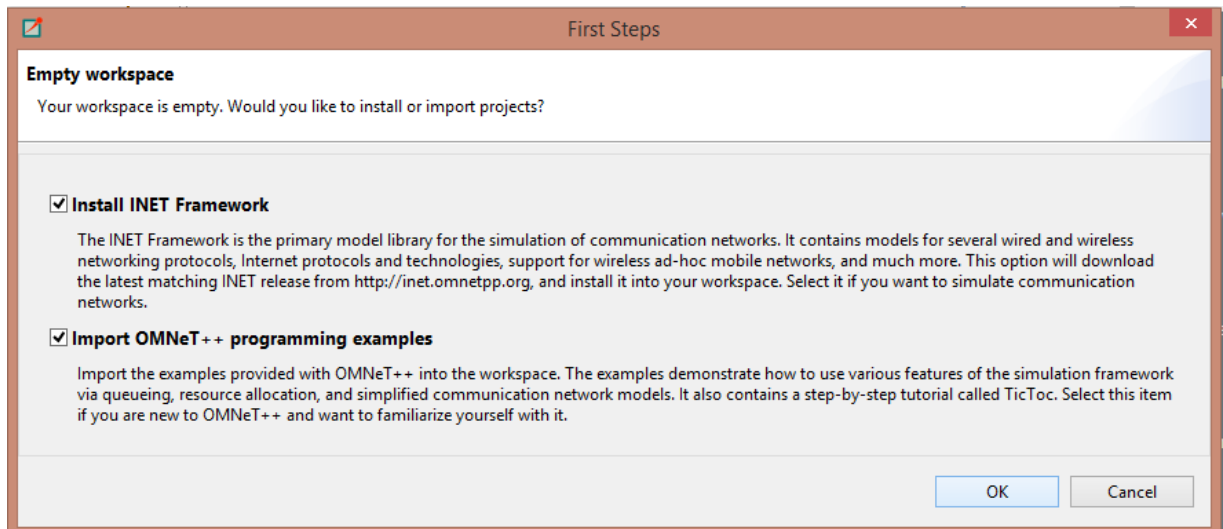


Figure B.23 Install INET framework and import the OMNeT++ examples.

## B.4 Installation of Veins

This section shows the steps to follow in order to install and launch Veins for both Windows and Linux environments.

### B.4.1 Import VEINS in OMNeT++ IDE

First step you must unzip the zip file that you have download in the section B.1.3.

- **Windows Users**

Extract the zip file **veins-4a2** in the **omnet** folder that you created before in section B.3.1, as it is shown in Figure B.24.

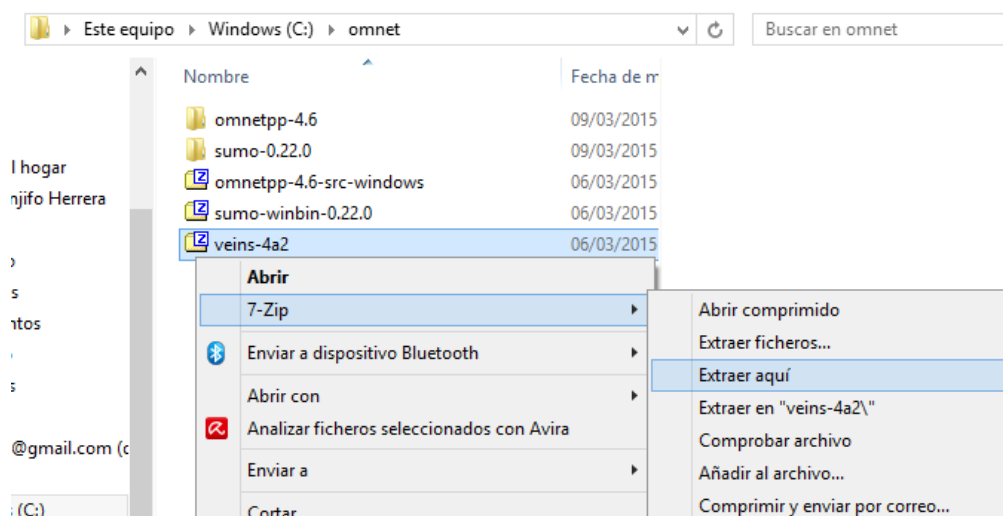


Figure B.24 Extract zip file veins-4a2.

- **Linux Users**

Unzip the file *veins-4a2.zip* inside the **omnet** folder that you created in section B.2.2 by typing the next command in the terminal, as it is shown in Figure B.25.

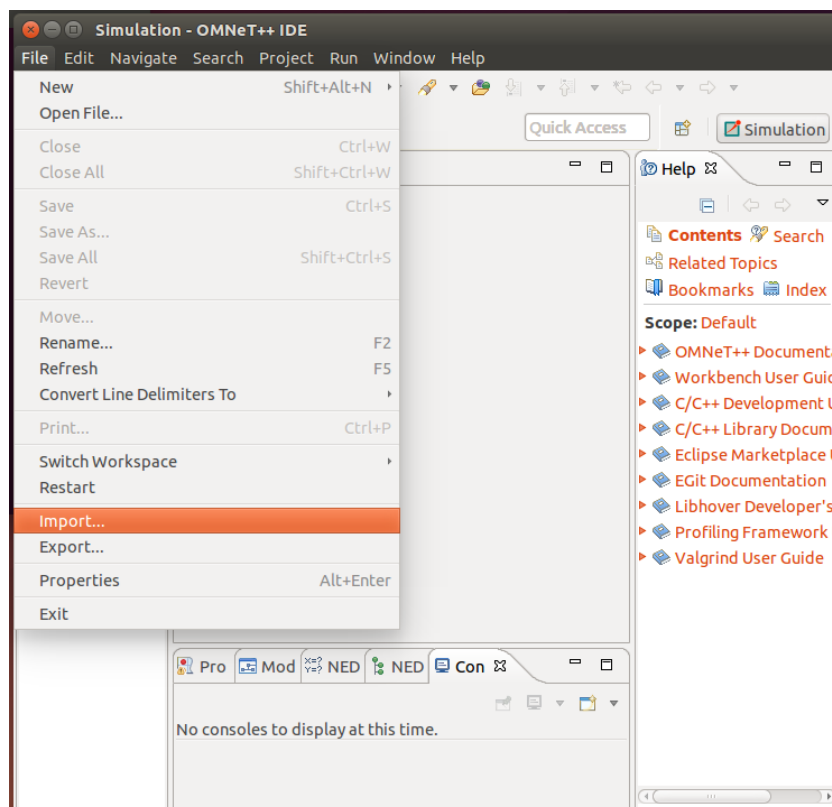
```
unzip veins-4a2
```

```
root@david-VirtualBox:/home/omnet# cp /root/Descargas/veins-4a2.zip /home/omnet/
root@david-VirtualBox:/home/omnet# unzip veins-4a2.zip
Archive:  veins-4a2.zip
2d65fc5264b012c8115d5c2948bbbddfef1baf88
  creating:  veins-veins-4a2/
  inflating:  veins-veins-4a2/.cproject
```

*Figure B.25 Unzip VEINS.*

Then, launch OMNeT++ IDE by typing the command **omnetpp** in the terminal as you have done before during installation of OMNeT++.

For both Windows and Linux users import the project into your OMNeT++ IDE workspace by clicking: **File > Import**, as it is shown in Figure B.26.



*Figure B.26 Import a project in OMNeT++.*

A window will appear and you must select **General: Existing Projects into Workspace** and press **Next >**, as it is illustrated in Figure B.27.

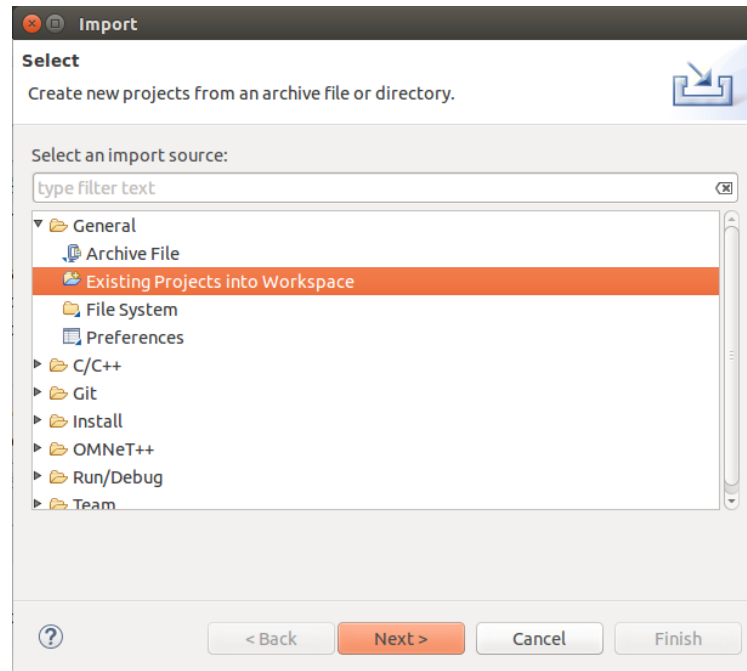


Figure B.27 Select the option existing projects into workspace.

In the new window press **Browse...**, search and select the folder **omnet/veins-veins-4a2** that you have created before during extraction the file *veins-4a2.zip* and press **Aceptar**, as it is illustrated in Figure B.28.

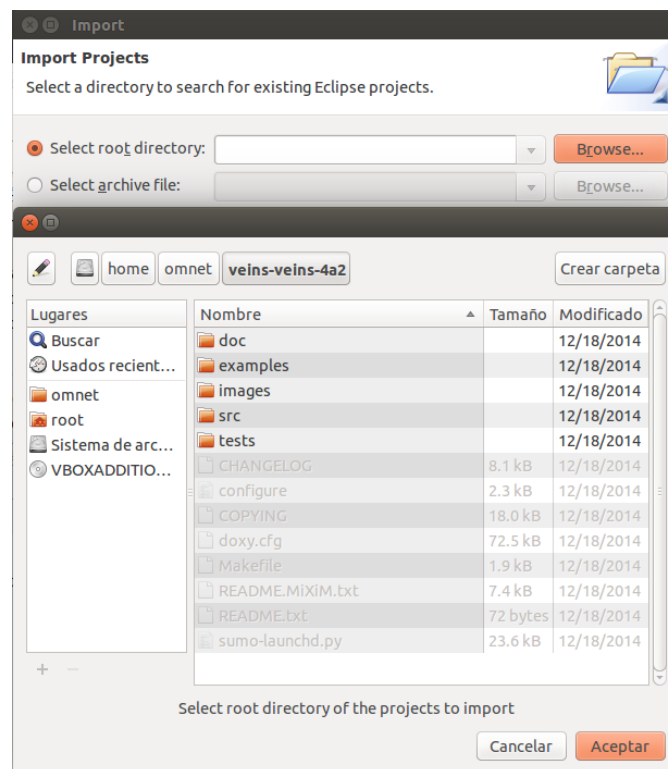
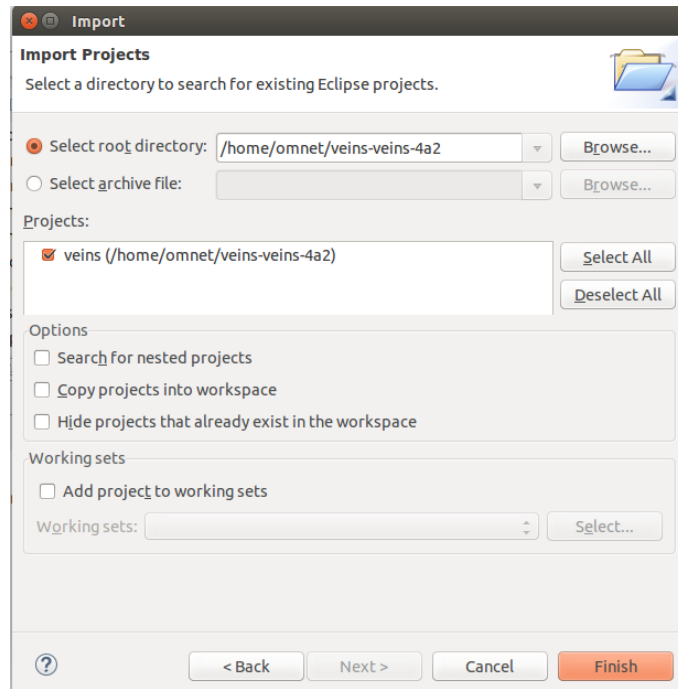


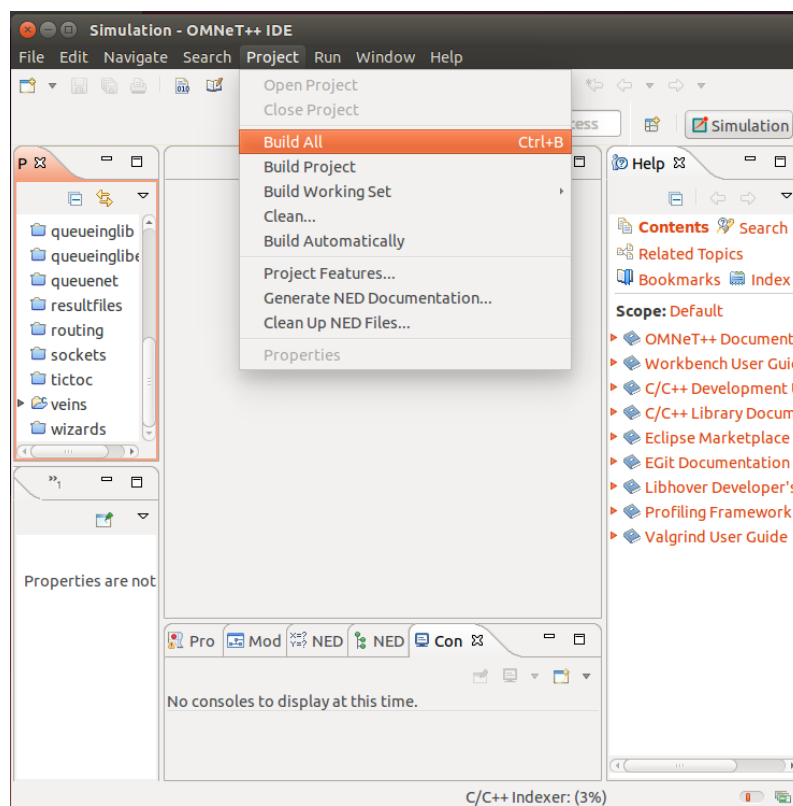
Figure B.28 Select the folder omnet.

Then, press **Finish** and VEINS will be imported successfully, as it is shown in Figure B.29.



*Figure B.29 Import VEINS into OMNeT++.*

Finally, you must build the project by clicking **Project > Build All** in OMNeT++, as it is illustrated in Figure B.30.



*Figure B.30 Build VEINS in OMNeT++.*

## B.4.2 Run the Veins demo scenario

First, you should be sure that SUMO is working correctly. So, in the terminal go to the path `/home/omnet/veins-veins-4a2/examples/veins` and run SUMO using the next commands, as it is shown in Figure B.31.

```
cd /home/omnet/veins-veins-4a2/examples/veins/  
/home/omnet/sumo-0.22.0/bin/sumo-gui -c erlangen.sumo.cfg
```

For Windows users at the moment of typing the commands use ***mingwenv.cmd*** terminal and change the directory `/home/` for `/c/`, as it is shown in Figure B.32. You should see a line saying "Loading configuration... done."

```
root@david-VirtualBox:/home/omnet# omnetpp  
Starting the OMNeT++ IDE...  
root@david-VirtualBox:/home/omnet# cd /home/omnet/veins-veins-4a2/examples/veins/  
root@david-VirtualBox:/home/omnet/veins-veins-4a2/examples/veins# /home/omnet/sumo-0.22.0/bin/sumo-gui  
-c erlangen.sumo.cfg  
libGL error: pci id for fd 4: 80ee:beef, driver (null)  
libGL error: core dri or dri2 extension not found  
libGL error: failed to load driver: vboxvideo  
Loading configuration... done.
```

Figure B.31 Run SUMO in Linux System.

```
M /c/omnet/veins-veins-4a2/examples/veins  
Welcome to OMNeT++ 4.6!  
/c/omnet/omnetpp-4.6$ omnetpp  
Starting the OMNeT++ IDE...  
/c/omnet/omnetpp-4.6$ omnetpp  
Starting the OMNeT++ IDE...  
/c/omnet/omnetpp-4.6$ cd /c/omnet/veins-veins-4a2/examples/veins  
/c/omnet/veins-veins-4a2/examples/veins$ /c/omnet/sumo-0.22.0/bin/sumo-gui.exe -c erlangen.sumo.cfg  
Loading configuration... done.
```

Figure B.32 Run SUMO in Windows System.

Then, you get an impression of example scenario looks like, as it is illustrated in Figure B.33. As a result, you can conclude that SUMO is working correctly.

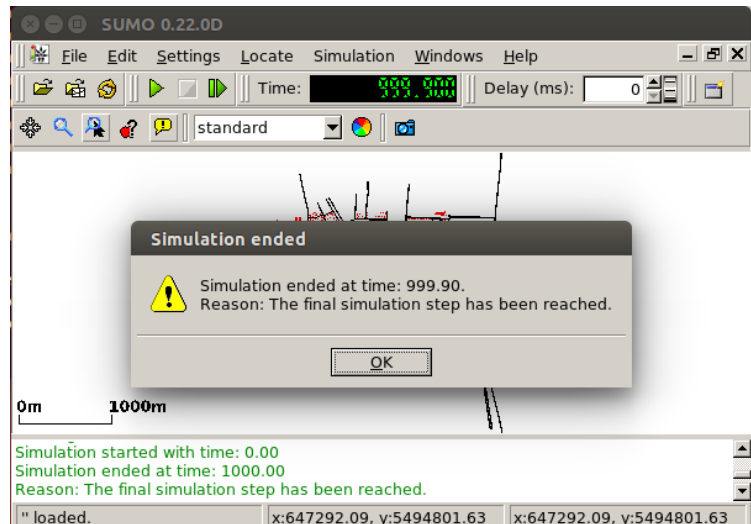


Figure B.33 Example scenario of SUMO running.

The next step is to run SUMO and OMNeT simultaneously. VEINS comes with a small python script wills proxy TCP connections between OMNeT++ and SUMO. To do that this script starts a new copy of the SUMO simulation for every OMNeT++ simulation connecting. So, in the terminal type the next command to start it.

```
/home/omnet/veins-veins-4a2/sumo-launchd.py -vv -c  
/home/omnet/sumo-0.22.0/bin /sumo-gui
```

The script will print Listening on port 9999, as it is illustrated in Figure B.34, and wait for the simulation to start. Leave this window open and switch back to the OMNeT++ IDE.

For Windows users change the directory /home/ for /c/, as it is illustrated in Figure B.35.

```
root@david-VirtualBox:/home/omnet/veins-veins-4a2/examples/veins# /home/omnet/veins-veins-4a2/sumo-launchd.py -vv -c /home/omnet/sumo-0.22.0/bin/sumo-gui
Logging to /tmp/sumo-launchd.log
Listening on port 9999
```

Figure B.34 Run python script in Linux System.

```
/c/omnet/veins-veins-4a2/examples/veins$ /c/omnet/veins-veins-4a2/sumo-launchd.py -vv -c /c/omnet/sumo-0.22.0/bin/sumo-gui.exe
Logging to c:\users\davicho\appdata\local\temp\sumo-launchd.log
Listening on port 9999
```

Figure B.35 Run python script in Windows System.

Then, you can simulate *the Veins demo scenario* in the OMNeT++ IDE, for both Windows and Linux users, right-click on **veins/examples/veins/omnetpp.ini** and choose **Run As > 1 OMNeT++ simulation**, as it is shown in Figure B.36.

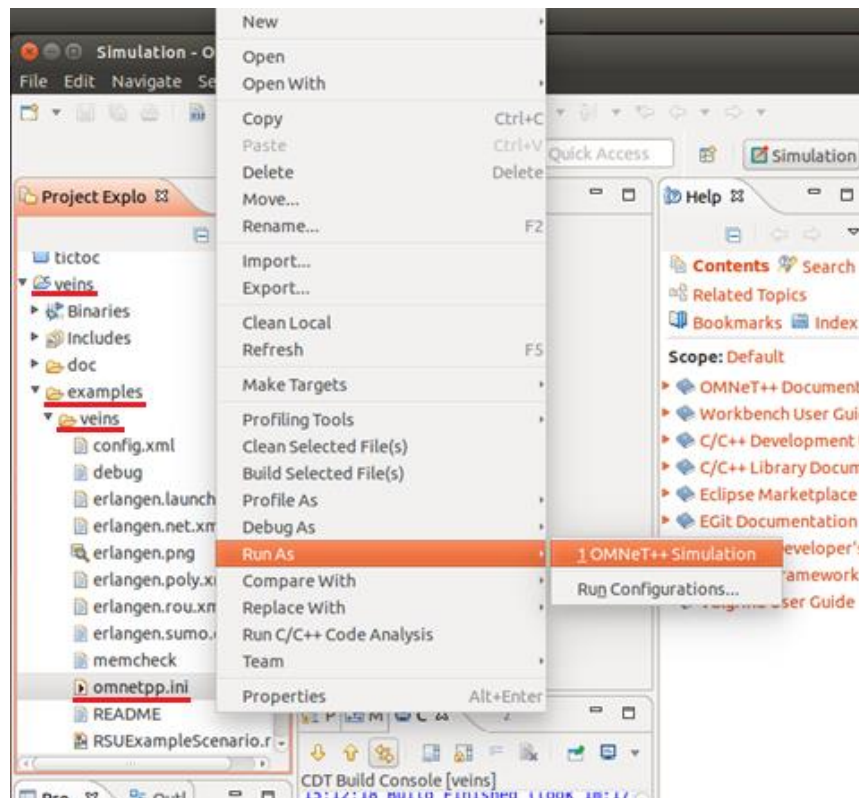


Figure B.36 Run Veins demo scenario.

If everything worked as intended this will give you a working simulation scenario using OMNeT++ and SUMO running in parallel to simulate a stream of vehicles that gets interrupted by an accident, as it is illustrated in Figure B.37.



Figure B.37 OMNeT++ and SUMO running in parallel to simulate VEINS demo scenario.

## C. Example of simulation in VEINS

### C.1 Importing networks and generation of routes in SUMO

SUMO offers the possibility to import real network topologies for simulation, which is an important advantage, since it is a very interesting possibility. The user can use real and concrete scenarios in order to study the behavior of a given IVC.

The SUMO simulator is able to import networks from several sources; however, it will use OpenStreetMap.

#### C.1.1 OpenStreetMap

The OpenStreetMap (OSM) project ([www.openstreetmap.org](http://www.openstreetmap.org)) has collected an enormous amount of free spatial data and the database is growing every day. Many people want to use this data for their own GIS projects but have been hindered by the use of a non-standard data format in the OSM project.

The mapping from OSM data to other formats is not an exact science. OSM rules on how to map certain features are often not well defined and there is no mandatory quality control. This openness allows a lot of flexibility and is part of the reason why OSM has been able to collect so much data in such a short time frame, but it makes using the data more difficult. When using or exporting the data, many decisions have to be made on how to extract the different features into something usable for the task at hand [36].

#### C.1.2 Get a map from OpenStreetMap

Open your web browser and go to the web site: <https://www.openstreetmap.org> and search a city that you want, for example Barcelona, Spain, as it is shown in Figure C.1.

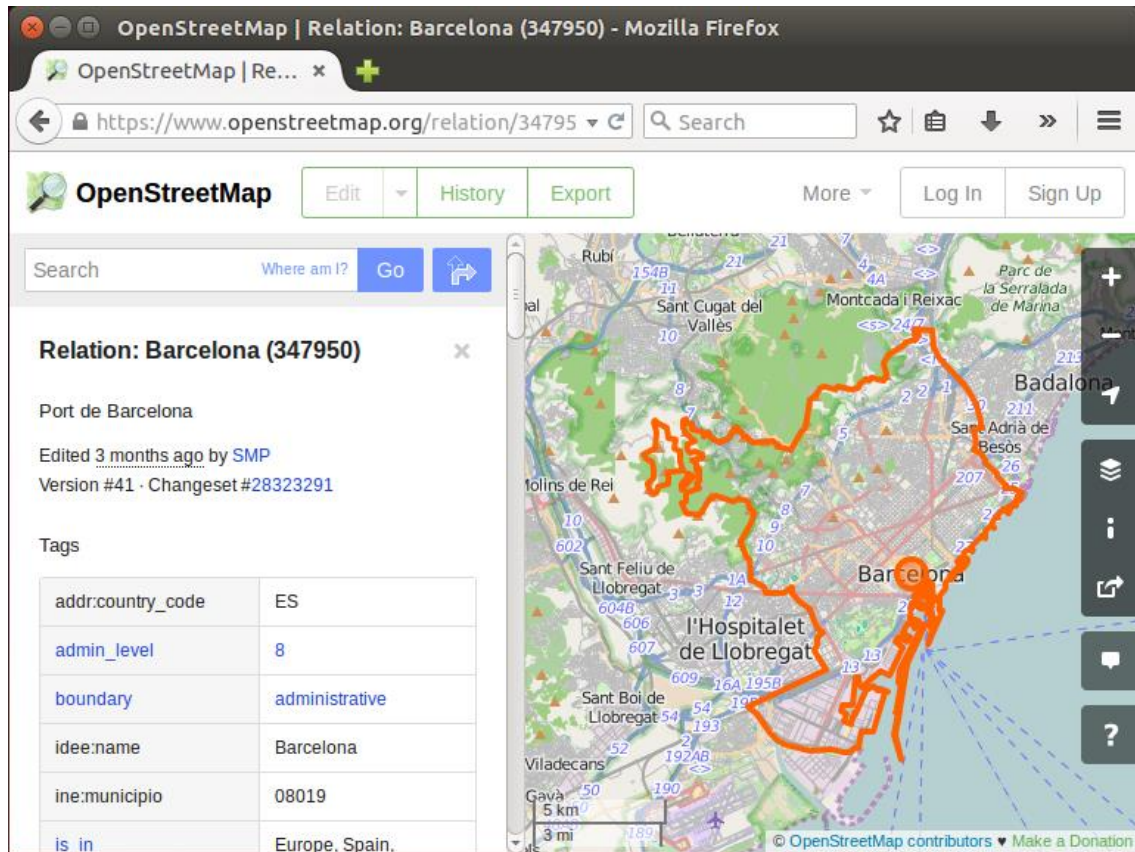


Figure C.1 OpenStreetMap web site.

Then, click on **Export** button which is located in top and after that click on **Manually select a different area**. Choose the area which you want to export and click on **Export** button which is located in the left as illustrated in Figure C.2. A window appears asked if you want to open or save the map, you must click on **save** for download the file *map.osm*, as it is shown in Figure C.3.

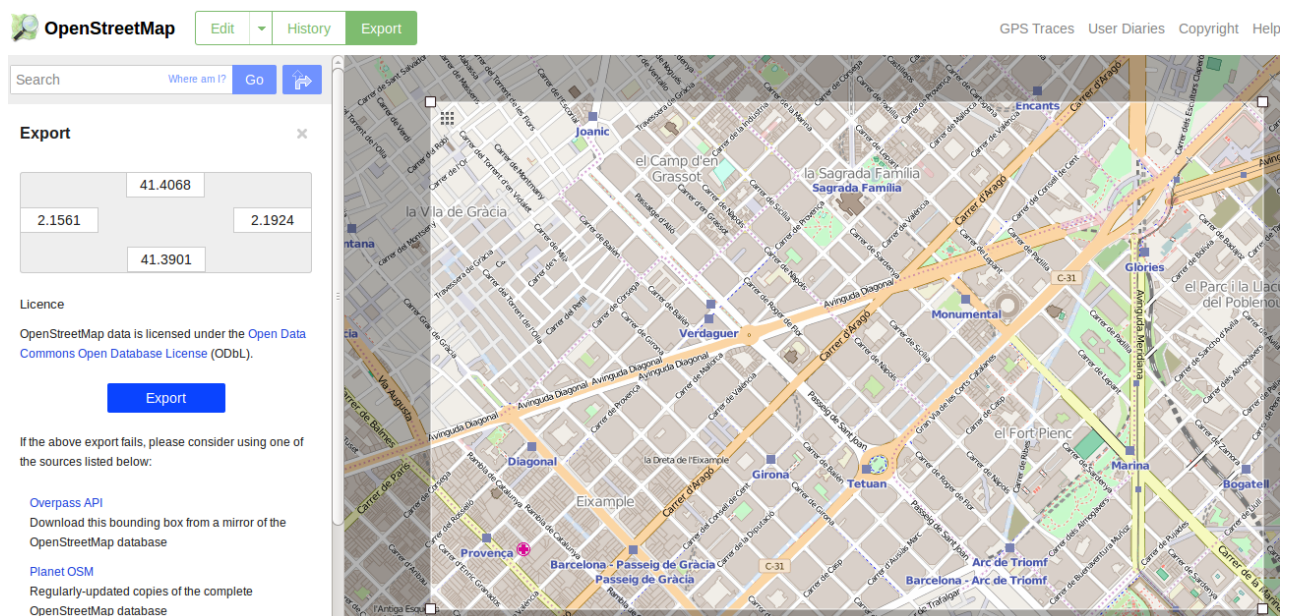


Figure C.2 Select the area and export the map.

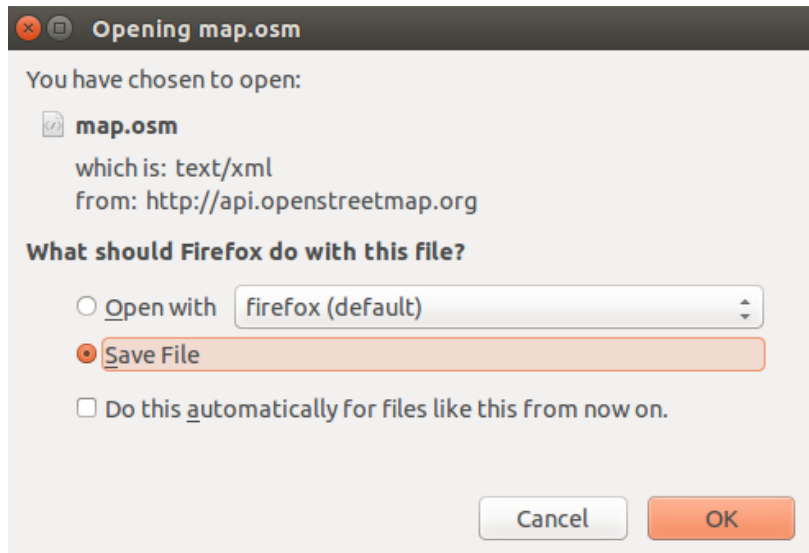


Figure C.3 Save the map in your computer.

### C.1.3 Preparation the map for use in SUMO

Copy the file *map.osm* which you downloaded in previous section in *sumo/bin* directory, after open a terminal and execute the following command in order to convert the map into a road network that is understood by SUMO, as it is illustrated in Figure C.4.

```
netconvert --osm-files map.osm -o barcelona.net.xml
```

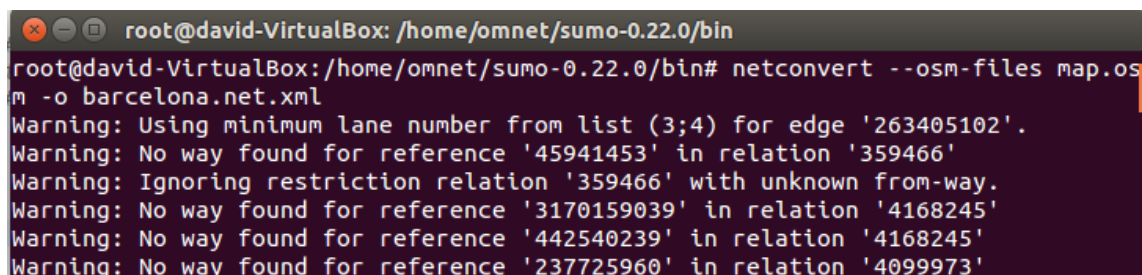
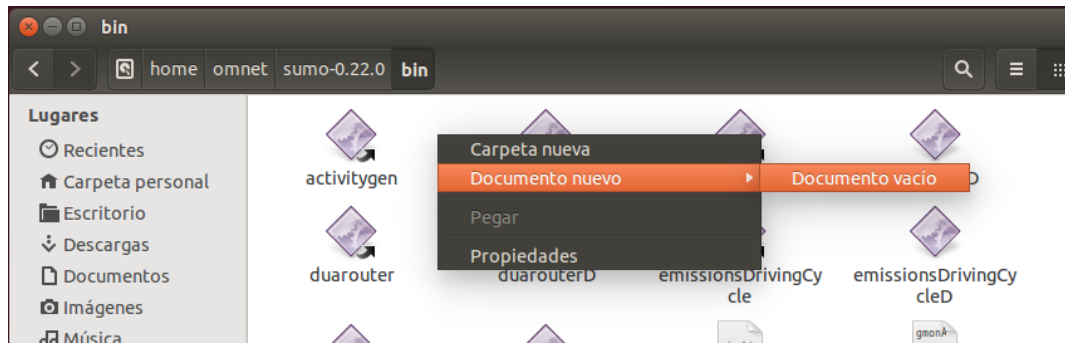
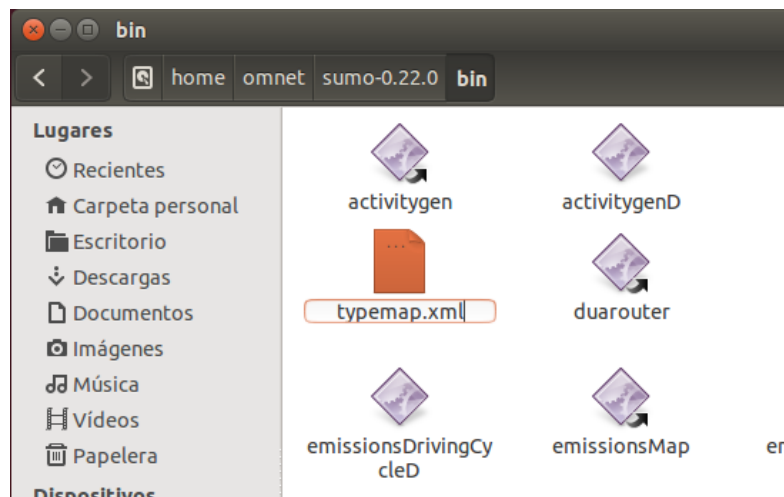


Figure C.4 Generation of network through netconvert.

Now, it is necessary to create the *typemap.xml* file. There are many ways to do that, one option is to use the GUI in Linux environment. So, go to the *sumo/bin* directory and right click on the folder and select **Documento Nuevo > Documento vacío** and name the file as **typemap.xml**, as it is shown in Figures C.5 and C.6.



*Figure C.5 Create an empty file in Linux System.*



*Figure C.6 Write the name of the new file as typemap.xml.*

Then, open your web browser and go to the following web site:

<https://github.com/bluemix/SUMO/blob/master/typemap.xml> and copy its contents into the typemap.xml file which you created before and save it, as it is shown in Figures C.7 and C.8.

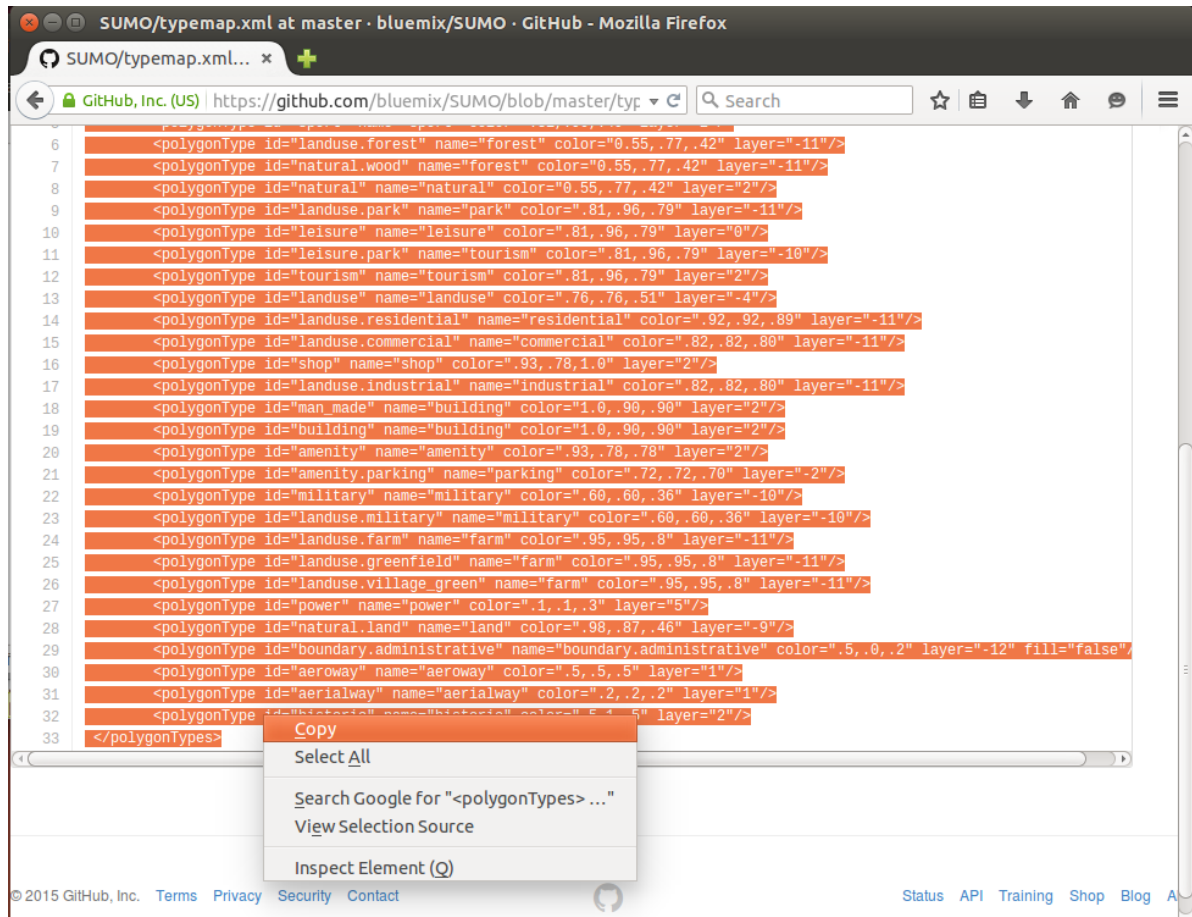


Figure C.7 Copy the content of website typemap.

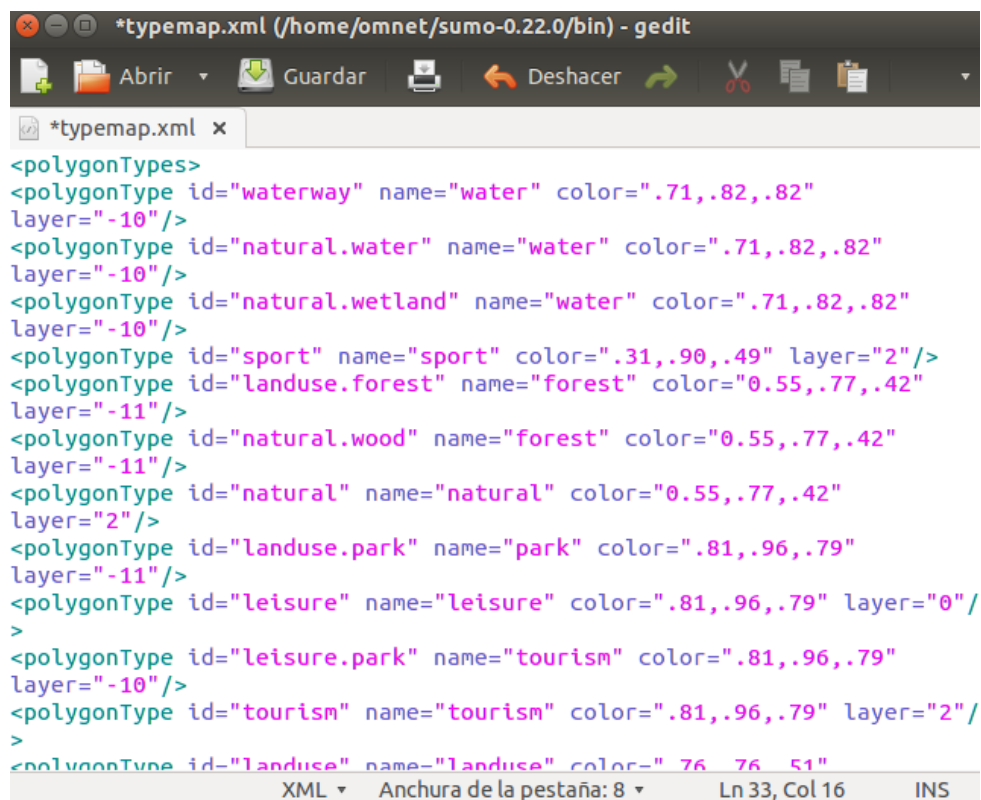
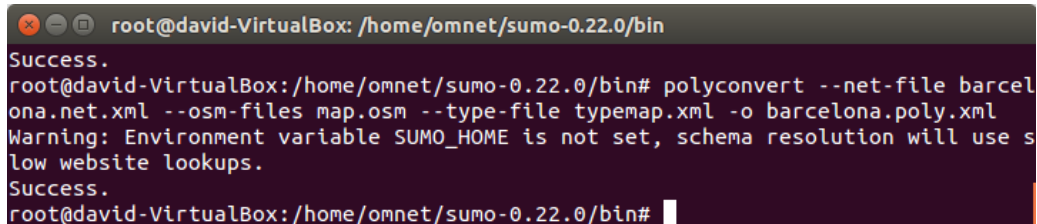


Figure C.8 Save typemap.xml file.

Then, go to the terminal and execute the next command in order to show the map correctly in SUMO. This step is illustrated in Figure C.9.

```
polyconvert --net-file barcelona.net.xml --osm-files map.osm --type-file  
typemap.xml -o barcelona.poly.xml
```



```
root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin
Success.
root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin# polyconvert --net-file barcelona.net.xml --osm-files map.osm --type-file typemap.xml -o barcelona.poly.xml
Warning: Environment variable SUMO_HOME is not set, schema resolution will use slow website lookups.
Success.
root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin#
```

*Figure C.9 Command polyconvert.*

So, the map is ready for use with SUMO.

#### C.1.4 Generation of routes in SUMO

Having defined the network topology, it only remains to generate the so-called traffic demand, that is, the description of the routes that follow the vehicles.

There are several methods to generate traffic demand in SUMO:

- Using Route definitions.
- Using definitions travel.
- Using definitions of flows (similar to above but uniting vehicles with similar travel in groups).
- Using definitions of flows at intersections rotation rate (the link target is not specified, and instead the probability of making turns at intersections shown).
- Using random routes.

In this case it will use random routes. There is a Python script developed with the aim of producing random routes, his name is *randomTrips.py*. Currently, it is the most recommended method to achieve this functionality. However, note that the results are not always entirely realistic.

So, open a terminal, go to the sumo/bin folder, and type the next command, as it can be seen in Figure C.10.

```
python /home/omnet/sumo-0.22.0/tools/trip/randomTrips.py --net-file  
barcelona.net.xml --route-file barcelona.rou.xml --begin 0 --end 100 --length
```

```

root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin
root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin# python /home/omnet/sumo-0.22.0/tools/trip/randomTrips.py --net-file barcelona.net.xml --route-file barcelona.rou.xml --begin 0 --end 100 --length
calling /home/omnet/sumo-0.22.0/bin/duarouter -n barcelona.net.xml -t trips.trips.xml -o barcelona.rou.xml --ignore-errors --begin 0.0 --end 100.0 --no-step-loading
Warning: No connection between '38963514#1' and '313379196#9' found.

```

*Figure C.10 Generation of random routes in SUMO.*

## C.2 Prepare files before simulating

In this step you must copy the files that you have generated in sections C.1.3 and C.1.4 from **sumo** folder to **veins** folder, as it is shown in Figure C.11. In order to do that type the following commands:

```

cp barcelona.net.xml /home/omnet/veins-veins-4a2/examples/veins/
cp barcelona.poly.xml /home/omnet/veins-veins-4a2/examples/veins/
cp barcelona.rou.xml /home/omnet/veins-veins-4a2/examples/veins/

```

```

root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin
root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin# cp barcelona.net.xml /home/omnet/veins-veins-4a2/examples/veins/
root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin# cp barcelona.poly.xml /home/omnet/veins-veins-4a2/examples/veins/
root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin# cp barcelona.rou.xml /home/omnet/veins-veins-4a2/examples/veins/
root@david-VirtualBox: /home/omnet/sumo-0.22.0/bin#

```

*Figure C.11 Copy sumo files to veins folder*

After that, you have to edit the configuration files of VEINS. So, go to the path `/home/omnet/veins-veins-4a2/examples/veins/`. Then, open with **gedit** the files `erlangen.launchd.xml` and `erlangen.sumo.cfg` and write the name of files which you copied before i.e. `barcelona.net.xml`, `barcelona.rou.xml`, `barcelona.poly.xml`, as it is illustrated in Figures C.12 and C.13.

```

*erlangen.launchd.xml (/home/omnet/veins-veins-4a2/examples/veins) - gedit
Abrir Guardar Deshacer
*erlangen.launchd.xml x erlangen.sumo.cfg x
<?xml version="1.0"?>
<!-- debug config -->
<launch>
  <copy file="barcelona.net.xml" />
  <copy file="barcelona.rou.xml" />
  <copy file="barcelona.poly.xml" />
  <copy file="erlangen.sumo.cfg" type="config" />
</launch>

```

*Figure C.12 Edit Erlangen.launchd.xml.*



```
*erlangen.sumo.cfg (/home/omnet/veins-veins-4a2/examples/veins) - gedit
Abrir Guardar Deshacer
*erlangen.launchd.xml x *erlangen.sumo.cfg x
<?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="barcelona.net.xml"/>
    <route-files value="barcelona.rou.xml"/>
    <additional-files value="barcelona.poly.xml"/>
  </input>
```

Figure C.13 Edit Erlangen.sumo.cfg.

Then, you need to create the scenario in OMNeT++, in this manual will be used **FranciscoScenario.ned** and **omnetpp.ini** which were created by Cristhian Iza, who is a Ph.D. student at the UPC. The codes of these files are available on Annex D. Create these files and put them into `/home/omnet/veins-veins-4a2/examples/veins` folder, as it is shown in Figure C.14.

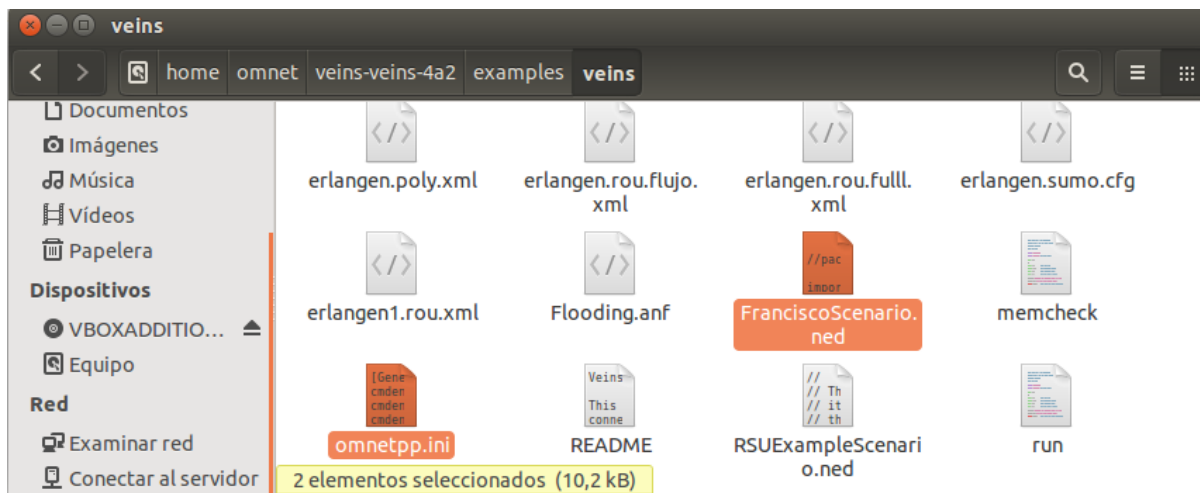


Figure C.14 OMNeT scenario files.

You need to create files which are dissemination methods of messages. So, create **counter**, **flooding** and **probability** folders inside `/home/omnet/veins-veins-4a2/src/veins/modules/application/traci`, as it is shown in Figure C.15. After, create the files to each technique, as it is shown in Figures C.16, C.17, and C.18.

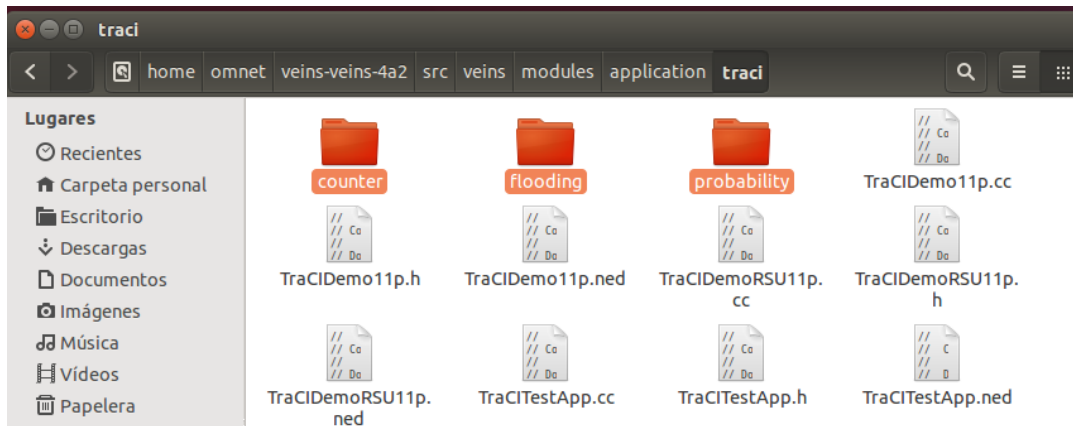


Figure C.15 Create folders for dissemination methods files.

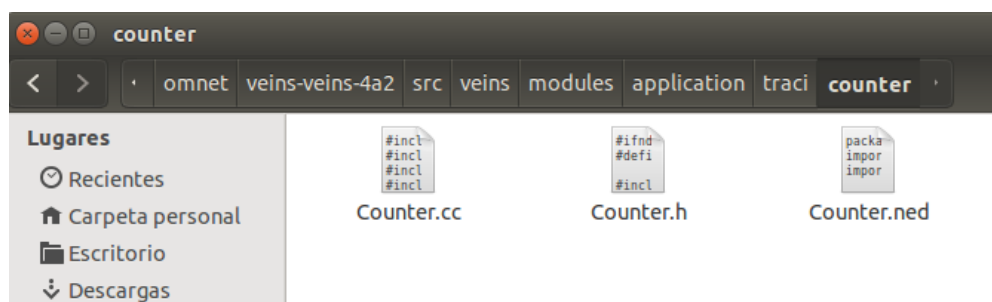


Figure C.16 Counter method files.

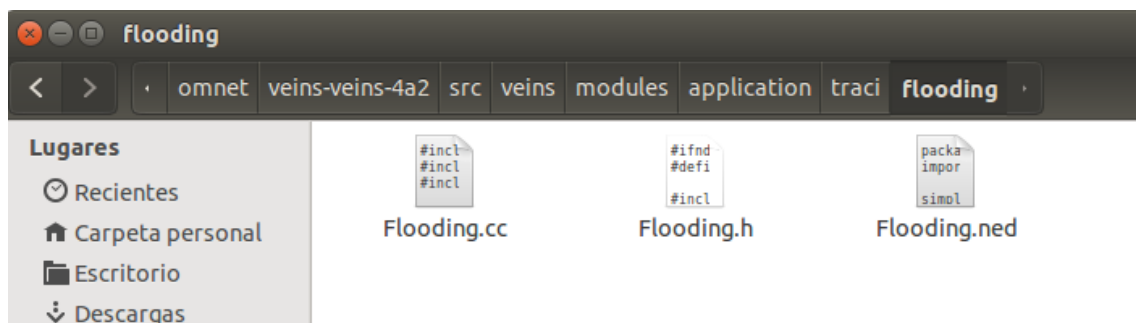


Figure C.17 Flooding method files.

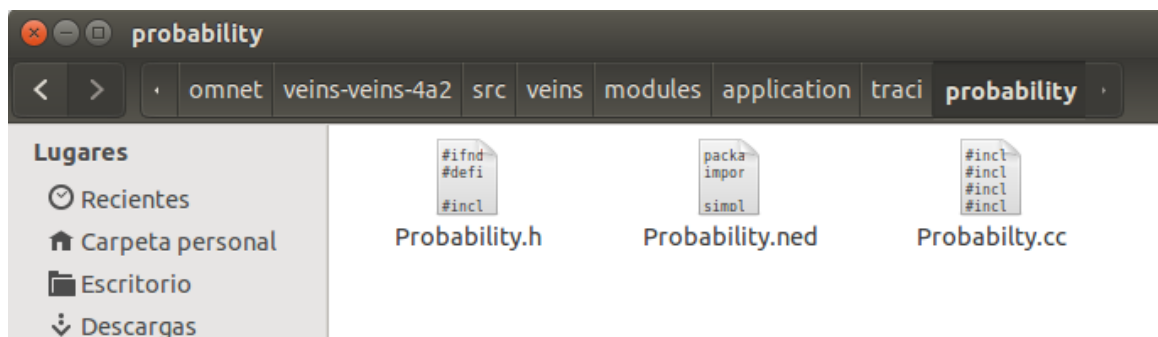


Figure C.18 Probability method files.

After that, create a folder named stats into `/home/omnet/veins-veins-4a2/src/veins/modules` and created inside the files **FranciscoStatistics.\***, as it is shown in Figures C.19 and C.20.

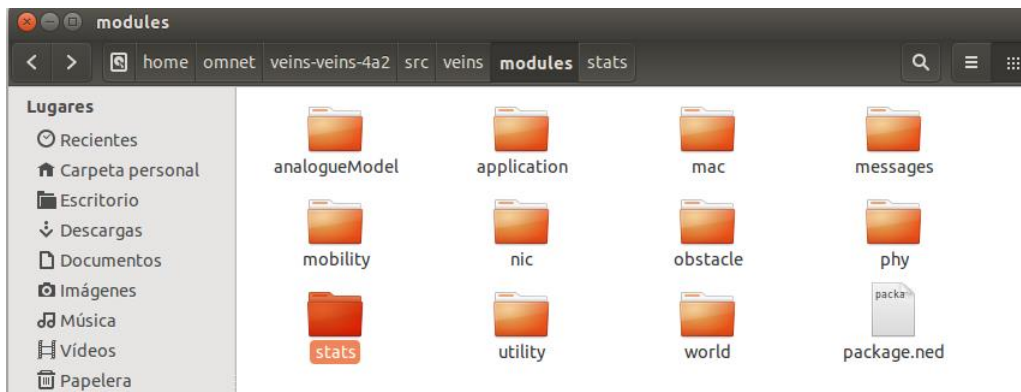


Figure C.19 Create stats folder.

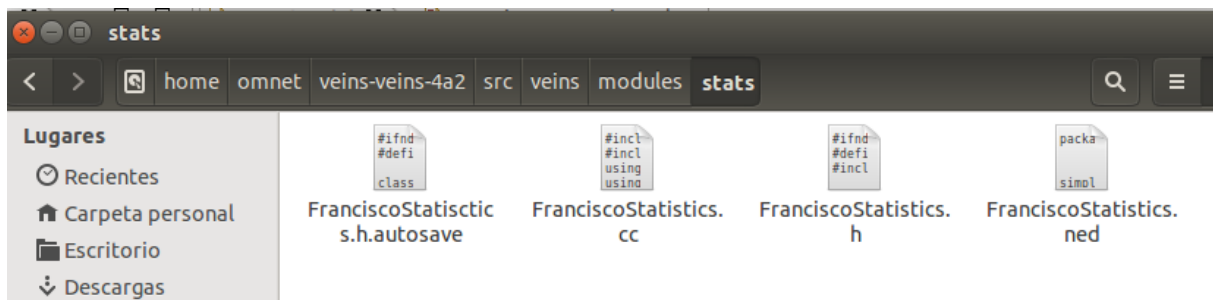


Figure C.20 Statistics files.

Now, you are ready for run the simulation.

### C.3 Running the simulation

Before running the simulation, it is recommendable clean the local variables of Veins. So, in OMNeT++ workspace right-click on *veins* and click on **clean local**, **clean project** and **build project**, as it is illustrated in Figures C.21, C.22 and C.23.

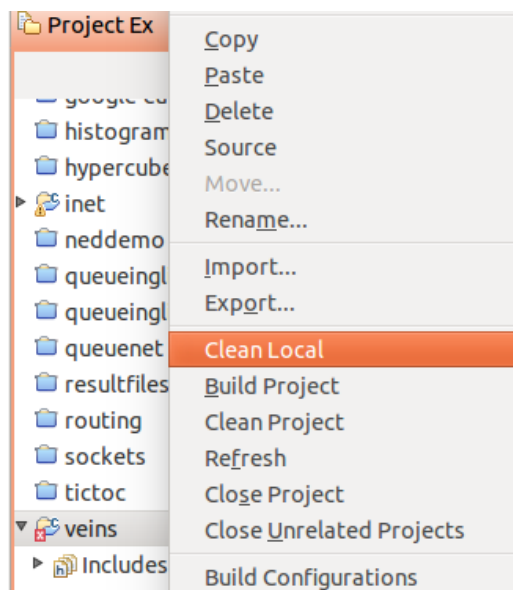
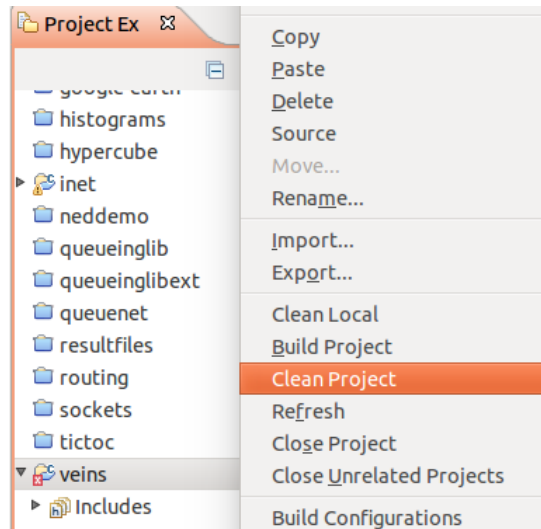
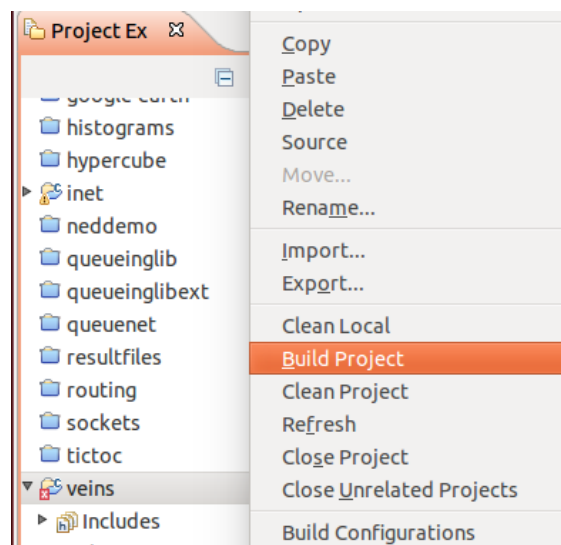


Figure C.21 Clean local variables.



*Figure C.22 Clean project variables.*



*Figure C.23 Build project in OMNeT++.*

Then, type the following command, as it is shown in Figure C.24.

```
/home/omnet/veins-veins-4a2/sumo-launchd.py -vv -c  
/home/omnet/sumo-0.22.0/bin/sumo-gui
```

```

root@david-VirtualBox: /home/omnet/veins-veins-4a2/examples/veins
root@david-VirtualBox: /home/omnet/veins-veins-4a2/examples/veins# /home/omnet/ve
ins-veins-4a2/sumo-launchd.py -vv -c /home/omnet/sumo-0.22.0/bin/sumo-gui
Logging to /tmp/sumo-launchd.log
Listening on port 9999
    
```

*Figure C.24 Run python script.*

Then, right-click on **omnetpp.ini** and select **Runs As – 1 OMNeT++ Simulation**, as it is illustrated in Figure C.25.

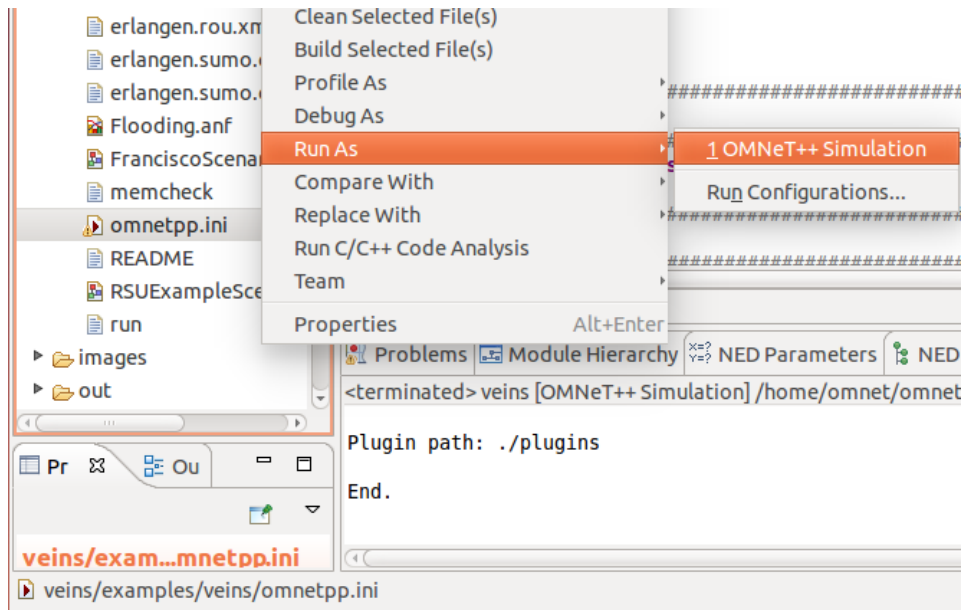


Figure C.25 Run omnetpp.ini.

After that, select one dissemination method of messages, in this case flooding, and click on **OK**, as it is shown in Figure C.26.

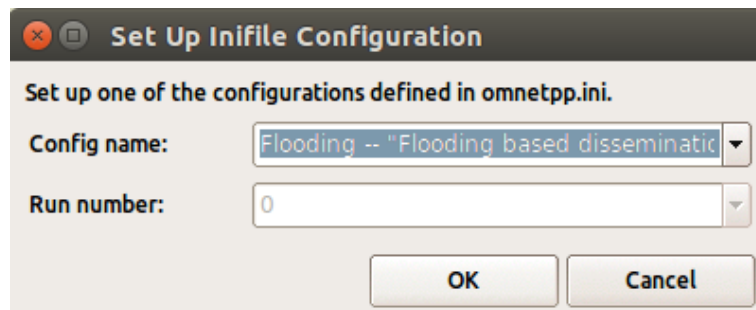


Figure C.26 Select a dissemination method.

Then, a window will appear and click on **RUN** to start the simulation, as it is illustrated in Figure C.27.

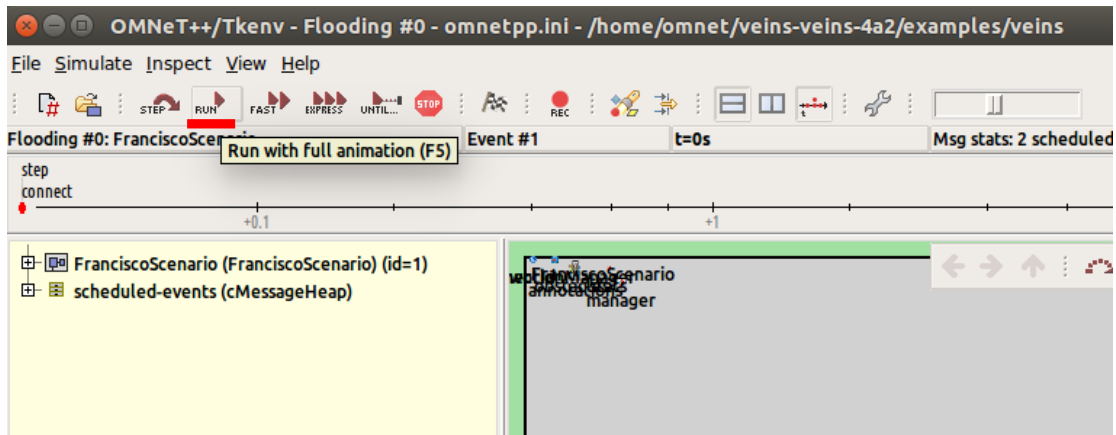


Figure C.27 Click on run to start simulation.

In some cases an error may occur, as illustrated in Figure C.28, it is because the map is bigger than the playground parameter. So, you need to edit this parameter in *omnetpp.ini* file, as it is shown in Figure C.29.

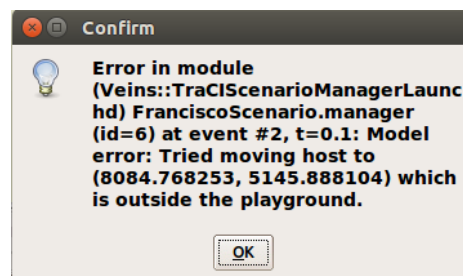


Figure C.28 Error during simulation.

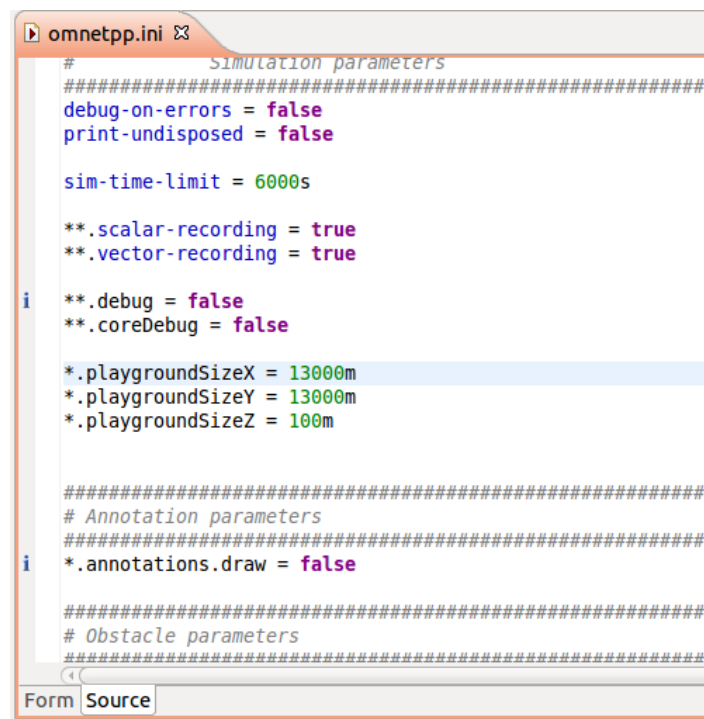


Figure C.29 Configure playground parameters.

Then, you need to repeat the previous steps in order to restart the simulation, if the playground parameter is ok the simulation starts, as it is shown in Figures C.30 and C.31.

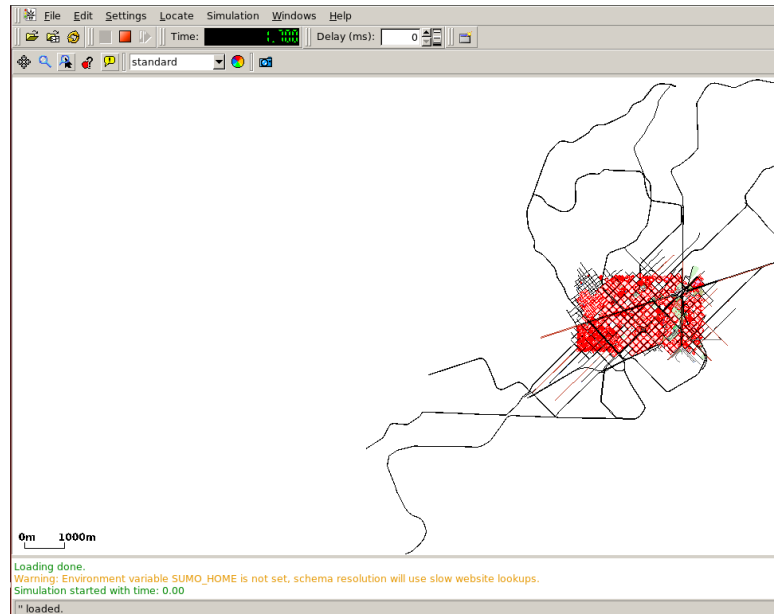


Figure C.30 Run simulation in SUMO.

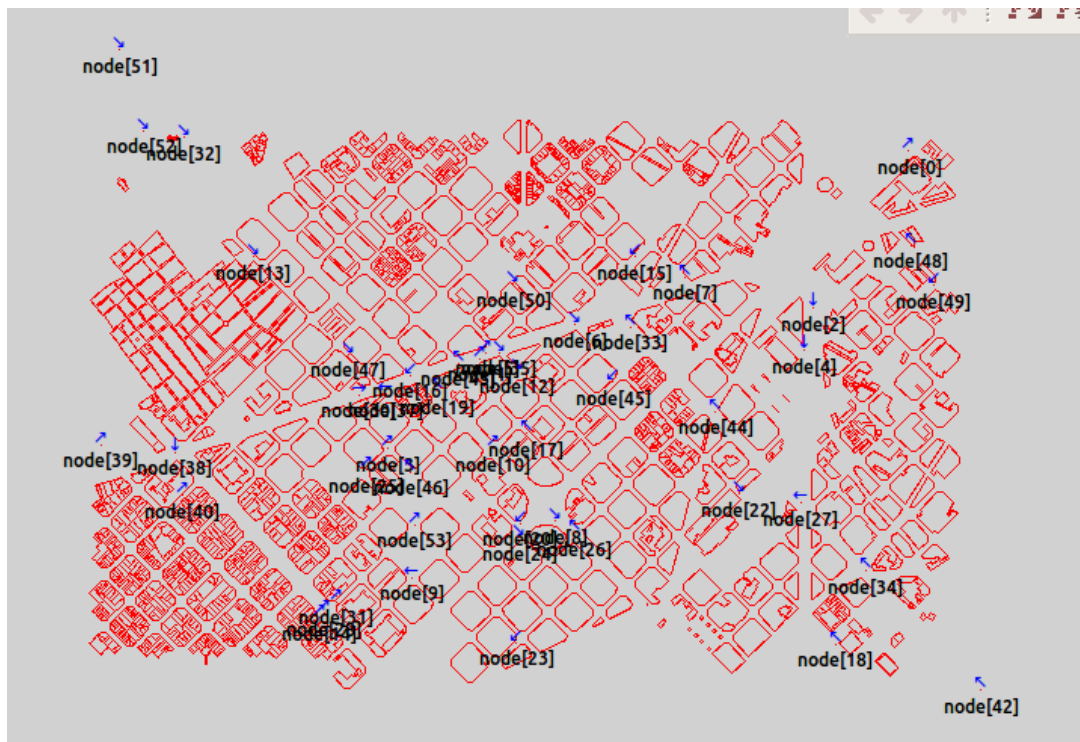


Figure C.31 Run simulation in OMNeT++.

Finally, you can see the results in result folder and click on *flooding.anf*, as it is illustrated in Figure C.32.

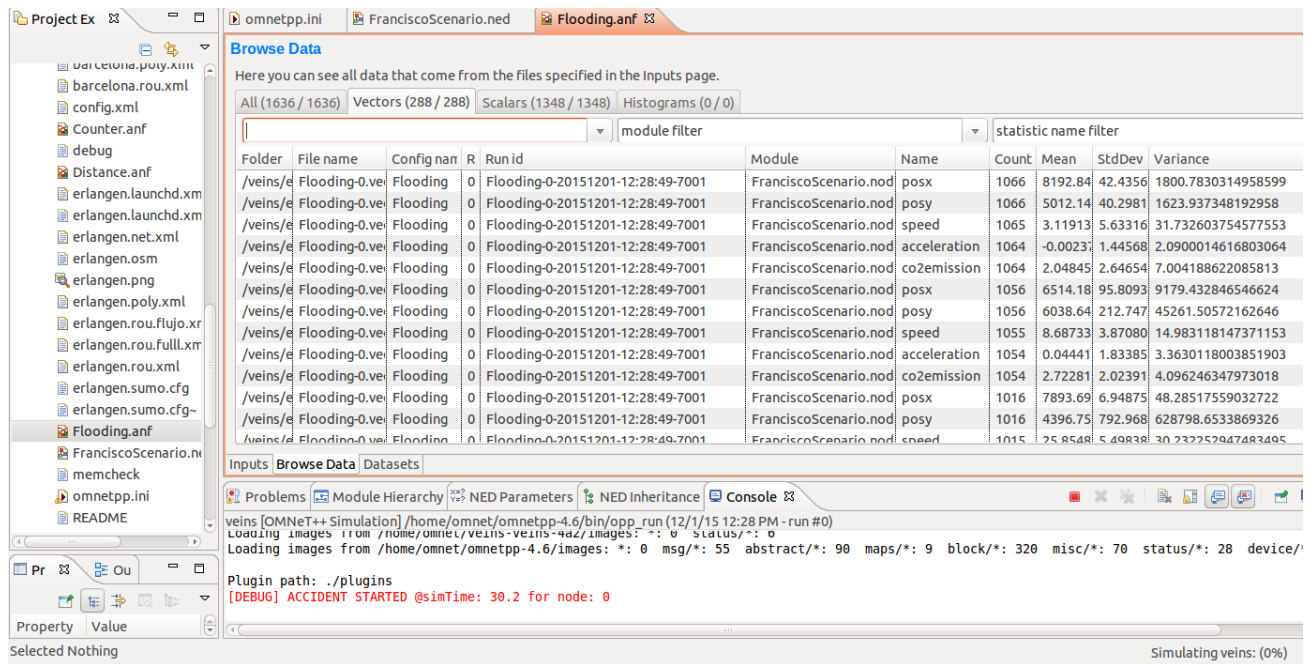


Figure C.32 Results of simulation in VEINS.

## D. Source code of files used in the simulations

### D.1 Implementation of omnetpp

#### *omnetpp.ini*

```
[General]
cmdenv-express-mode = true
cmdenv-autoflush = true
cmdenv-status-frequency = 10000000s

tkenv-image-path = bitmaps
ned-path = .

network = FranciscoScenario

#####
#      Simulation parameters          #
#####
debug-on-errors = true
print-undisposed = false

sim-time-limit = 6000s

**.scalar-recording = true
**.vector-recording = true

**.debug = false
**.coreDebug = false

*.playgroundSizeX = 13000m
*.playgroundSizeY = 13000m
*.playgroundSizeZ = 50m

#####
# Annotation parameters              #
#####
*.annotations.draw = false

#####
# Obstacle parameters                #
#####
*.obstacles.debug = false

#####
#      WorldUtility parameters        #
#####
*.world.useTorus = false
*.world.use2D = false
```

```
#####
#      TraCIScenarioManager parameters      #
#####
*.manager.updateInterval = 0.1s
*.manager.host = "localhost"
*.manager.port = 9999
*.manager.moduleType = "org.car2x.veins.nodes.Car"
*.manager.moduleName = "node"
*.manager.moduleDisplayString = ""
*.manager.autoShutdown = true
*.manager.margin = 25
*.manager.launchConfig = xmldoc("erlangen.launchd.xml")

#####
#      11p specific parameters              #
#                                           #
#      NIC-Settings                        #
#####
*.connectionManager.pMax = 20mW
*.connectionManager.sat = -89dBm
*.connectionManager.alpha = 2.0
*.connectionManager.carrierFrequency = 5.890e9 Hz
*.connectionManager.sendDirect = true

*.**.nic.mac1609_4.useServiceChannel = false

*.**.nic.mac1609_4.txPower = 20mW
*.**.nic.mac1609_4.bitrate = 18Mbps

*.**.nic.phy80211p.sensitivity = -89dBm
*.**.nic.phy80211p.maxTXPower = 10mW
*.**.nic.phy80211p.useThermalNoise = true
*.**.nic.phy80211p.thermalNoise = -110dBm
*.**.nic.phy80211p.decider = xmldoc("config.xml")
*.**.nic.phy80211p.analogueModels = xmldoc("config.xml")
*.**.nic.phy80211p.usePropagationDelay = true

#####
#      WaveAppLayer                        #
#####
*.node[*].appl.debug = false
*.node[*].appl.headerLength = 256 bit
*.node[*].appl.sendBeacons = false
*.node[*].appl.dataOnSch = false
*.node[*].appl.sendData = true
*.node[*].appl.beaconInterval = 1s
*.node[*].appl.beaconPriority = 3
*.node[*].appl.dataPriority = 2
*.node[*].appl.maxOffset = 0.005s
```

```
#####
#           Mobility           #
#####
*.node[*].veinsmobilityType = "org.car2x.veins.modules.mobility.traci.TraCIMobility"
*.node[*].mobilityType = "TraCIMobility"
*.node[*].mobilityType.debug = true
*.node[*].veinsmobilityType.debug = true
*.node[*].veinsmobility.x = 0
*.node[*].veinsmobility.y = 0
*.node[*].veinsmobility.z = 1.895
*.node[0].veinsmobility.accidentCount = 1

#[Config Stats]
*.node[*].veinsmobility.posx.result-recording-modes = -    # try removing superflous stats

*.manager.actualNumVehiclesSignal.scalar-recording = true
*.manager.actualNumVehicleSignal.vector-recording = true

*.node[*].appl.warningReceivedSignal.scalar-recording = true
*.node[*].appl.warningReceivedSignal.vector-recording = true

*.node[*].appl.beaconReceivedSignal.scalar-recording = true
*.node[*].appl.beaconReceivedSignal.vector-recording = true

*.node[*].appl.messageReceivedSignal.scalar-recording = true
*.node[*].appl.messageReceivedSignal.vector-recording = true

*.node[*].appl.newWarningReceivedSignal.scalar-recording = true
*.node[*].appl.newWarningReceivedSignal.vector-recording = true

*.stats.allBeaconsReceivedSignal.scalar-recording = true
*.stats.allBeaconsReceivedSignal.vector-recording = true

*.stats.allNewWarningsReceivedSignal.scalar-recording = true
*.stats.allNewWarningsReceivedSignal.vector-recording = true

*.stats.allWarningsReceivedSignal.scalar-recording = true
*.stats.allWarningsReceivedSignal.vector-recording = true

*.stats.allMessagesReceivedSignal.scalar-recording = true
*.stats.allMessagesReceivedSignal.vector-recording = true

*.stats.numAccidentsSignal.scalar-recording = true
*.stats.numAccidentsSignal.vector-recording = true

[Config Flooding]
description = "Flooding based dissemination"
**.debug = false
**.coreDebug = false
*.annotations.draw = true
*.node[*].applType = "Flooding"
*.node[*].appl.sendBeacons = false
```

```
*.node[*].appl.indexOfAccidentNode = 0
*.node[*].appl.indexOfAccidentNode = 0
*.node[0].veinsmobility.accidentCount = 1
*.node[0].veinsmobility.accidentStart = 40s
*.node[0].veinsmobility.accidentDuration = 20s
```

#### [Config Counter]

```
description = "Counter based dissemination"
*.node[*].applType = "Counter"
*.node[*].appl.counterThreshold = 5
**.debug = false
**.coreDebug = false
*.annotations.draw = true
*.node[*].appl.indexOfAccidentNode = 0
*.node[0].veinsmobility.accidentCount = 1
*.node[0].veinsmobility.accidentStart = 90s
*.node[0].veinsmobility.accidentDuration = 30s
```

#### [Config Probability]

```
description = "Probability based dissemination"
*.node[*].applType = "Probability"
*.node[*].appl.ProbabilityThreshold = 10
**.debug = false
**.coreDebug = false
*.annotations.draw = true
*.node[*].appl.indexOfAccidentNode = 0
```

## D.2 Implementation of FranciscoScenario

### FranciscoScenario.ned

```
import org.car2x.veins.modules.stats.FranciscoStatistics;

////////////////////////////////////
import org.car2x.veins.nodes.RSU;
import org.car2x.veins.nodes.Scenario;
////////////////////////////////////

import org.car2x.veins.base.connectionManager.ConnectionManager;
import org.car2x.veins.base.modules.BaseWorldUtility;
import org.car2x.veins.modules.mobility.traci.TraCIScenarioManagerLaunchd;
import org.car2x.veins.modules.obstacle.ObstacleControl;
import org.car2x.veins.modules.world.annotations.AnnotationManager;

network FranciscoScenario
{
    parameters:
        double playgroundSizeX @unit(m); // x size of the area the nodes are in (in meters)
        double playgroundSizeY @unit(m); // y size of the area the nodes are in (in meters)
        double playgroundSizeZ @unit(m); // z size of the area the nodes are in (in meters)
```

```
@display("bgb=$playgroundSizeX,$playgroundSizeY");
```

submodules:

```
obstacles: ObstacleControl {
  parameters:
    @display("p=240,50");
}
annotations: AnnotationManager {
  parameters:
    @display("p=260,50");
}
connectionManager: ConnectionManager {
  parameters:
    @display("p=150,0;i=abstract/multicast");
}
world: BaseWorldUtility {
  parameters:
    playgroundSizeX = playgroundSizeX;
    playgroundSizeY = playgroundSizeY;
    playgroundSizeZ = playgroundSizeZ;
    @display("p=30,0;i=misc/globe");
}
manager: TraCIScenarioManagerLaunchd {
  parameters:
    @display("p=512,128");
}
stats: FranciscoStatistics {
  parameters:
    @display("p=440,50");
}
```

connections allowunconnected:

```
}
```

## D.3 Implementation of Counter technique

### D.3.1 Counter.cc

```
#include "Counter.h"
#include "veins/modules/messages/WaveShortMessage_m.h"
#include <iostream>
#include <cstdio>
#include <cstring>
```

```
using Veins::TraCIMobility;
using Veins::TraCIMobilityAccess;
using std::sprintf;
using std::strcmp;
```

```
Define_Module(Counter)
```

```
void Counter::initialize(int stage)
{
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {

        // configurable variables in omnetpp.ini
        counterThreshold = par("counterThreshold").longValue();
        indexOfAccidentNode = par("indexOfAccidentNode").longValue();
        randomRebroadcastDelay = par("randomRebroadcastDelay").doubleValue();
        //

        traci = TraCIMobilityAccess().get(getParentModule());
        stats = FranciscoStatisticsAccess().getIfExists();
        ASSERT(stats);
        beaconReceivedSignal = registerSignal("beaconReceivedSignal");
        warningReceivedSignal = registerSignal("warningReceivedSignal");
        messageReceivedSignal = registerSignal("messageReceivedSignal");
        newWarningReceivedSignal = registerSignal("newWarningReceivedSignal");

        lastDroveAt = simTime();
        sentMessage = false;
    }
}

void Counter::receiveSignal(cComponent *source, simsignal_t signalID, cComponent::cObject
*obj)
{
    Enter_Method_Silent();
    if (signalID == mobilityStateChangedSignal) {
        handlePositionUpdate(obj);
    }
}

void Counter::onBeacon(WaveShortMessage *wsm)
{
    // Beacons are not used in this algorithm
}

void Counter::onData(WaveShortMessage *wsm)
{
    // statistics recording
    emit(warningReceivedSignal, 1);
    emit(messageReceivedSignal, 1);
    stats->updateAllWarningsReceived();
    stats->updateAllMessagesReceived();

    // prevent originating disseminator from participating in further dissemination attempts
    if (sentMessage)
        return;
}
```

```
// add the new message to storage
receivedMessages[wsm->getTreeId()].push_back(wsm->dup());

// is it a new warning message?
if (receivedMessages[wsm->getTreeId()].size() == 1) {
    // statistics recording
    stats->updateNewWarningsReceived();
    emit(newWarningReceivedSignal, 1);

    // add a random waiting period before proceeding. Please see:
    // * onSelfMsg for continuation.
    // * .randomBroadcastDelay configuration in omnetpp.ini
    char buf[64];
    sprintf(buf, "%ld", wsm->getTreeId());
    // scheduleAt sends message to self (see handleSelfMsg() below and
    randomRebroadcastDelay in omnetpp.ini
    scheduleAt(simTime() + SimTime(randomRebroadcastDelay, SIMTIME_MS), new
    cMessage(buf));
}
}

void Counter::handlePositionUpdate(cComponent::cObject *obj)
{
    // stopped for for at least 10s?
    if (traci->getSpeed() < 1) {
        if ((simTime() - lastDroveAt >= 10)
            && (!sentMessage)
            && (indexOfAccidentNode == getParentModule()->getIndex())) {

            std::cerr << "[INFO] ACCIDENT STARTED @simTime: " << simTime().str() << " for node: " <<
            getParentModule()->getIndex() << endl;

            findHost()->getDisplayString().updateWith("r=16,red");
            if (!sentMessage)
                sendMessage(traci->getRoadId());
        }
    }
    else {
        lastDroveAt = simTime();
    }
}

void Counter::sendMessage(std::string blockedRoadId)
{
    t_channel channel = dataOnSch ? type_SCH : type_CCH;
    WaveShortMessage* wsm = prepareWSM("data", dataLengthBits, channel, dataPriority, -1,2);
    wsm->setWsmData(blockedRoadId.c_str());
    sendWSM(wsm);

    sentMessage = true;
}
```

```
void Counter::handleSelfMsg(cMessage *msg)
{
    // for "data" and "beacon" self messages
    if ((!strcmp(msg->getName(), "data")) || (!strcmp(msg->getName(), "beacon"))) {
        BaseWaveApplLayer::handleSelfMsg(msg);
        return;
    }
    else { // for "rebroadcast" self messages
        // if the number of times a warning message is received exceeds the counterThreshold
        // configuration variable, do not rebroadcast.
        if (receivedMessages[atoi(msg->getName())].size() >= (unsigned)counterThreshold)
            return;
        // if greater than threshold.. rebroadcast.
        sendWSM(receivedMessages[atoi(msg->getName())][0]->dup());
    }
}
```

### D.3.2 Counter.h

```
#ifndef Counter_H
#define Counter_H

#include "veins/modules/application/ieee80211p/BaseWaveApplLayer.h"
#include "veins/modules/mobility/traci/TraCIMobility.h"
#include "veins/modules/stats/FranciscoStatistics.h"
#include <vector>
#include <map>

using Veins::TraCIMobility;
using Veins::AnnotationManager;
using std::vector;
using std::map;

typedef std::vector<WaveShortMessage*> WaveShortMessages;

class Counter : public BaseWaveApplLayer
{
public:
    virtual void initialize(int stage);
    virtual void receiveSignal(cComponent *source, simsignal_t signalID, cObject *obj);

protected:
    TraCIMobility* traci;
    FranciscoStatistics* stats;
    vector<WaveShortMessage*> warningMessages;
    simsignal_t beaconReceivedSignal;
    simsignal_t warningReceivedSignal;
    simsignal_t newWarningReceivedSignal;
    simsignal_t messageReceivedSignal;
    simtime_t lastDroveAt;
    bool sentMessage;
```

```

long counterThreshold;
long indexOfAccidentNode;
double randomRebroadcastDelay;
map<long, WaveShortMessages> receivedMessages; // treeld, WSM vector

```

```

protected:
    virtual void onBeacon(WaveShortMessage *wsm);
    virtual void onData(WaveShortMessage *wsm);
    virtual void handlePositionUpdate(cObject *obj);
    virtual void sendMessage(std::string blockedRoadId);
    virtual void handleSelfMsg(cMessage *msg);
};

```

```

#endif // Counter_H

```

### D.3.3 Counter.ned

```

package org.car2x.veins.modules.application.traci.counter;
import org.car2x.veins.modules.application.ieee80211p.BaseWaveApplLayer;
import org.car2x.veins.modules.application.ieee80211p.BaseWaveApplLayer;

simple Counter extends BaseWaveApplLayer
{
    @class(Counter);
    @display("i=block/app2");

    int counterThreshold = default(3);
    volatile double randomRebroadcastDelay = default(uniform(0,500));
    int indexOfAccidentNode = default(0);

    @signal[warningReceivedSignal](type=long);
    @statistic[warningReceivedSignal](record=count,vector; description="Warning Message Received");

    @signal[beaconReceivedSignal](type=long);
    @statistic[beaconReceivedSignal](record=count,vector; description="Beacon Message Received");

    @signal[messageReceivedSignal](type=long);
    @statistic[messageReceivedSignal](record=count,vector; description="Message Received");

    @signal[newWarningReceivedSignal](type=long);
    @statistic[newWarningReceivedSignal](record=count,vector; description="New Warning Message Received");
}

```

## D.4 Implementation of Flooding technique

### D.4.1 Flooding.cc

```
#include "Flooding.h"
#include "veins/modules/messages/WaveShortMessage_m.h"
#include <iostream>

using Veins::TraCIMobility;
using Veins::TraCIMobilityAccess;

Define_Module(Flooding)

void Flooding::initialize(int stage)
{
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {

        traci = TraCIMobilityAccess().get(getParentModule());
        stats = FranciscoStatisticsAccess().getIfExists();
        ASSERT(stats);
        beaconReceivedSignal = registerSignal("beaconReceivedSignal");
        warningReceivedSignal = registerSignal("warningReceivedSignal");
        messageReceivedSignal = registerSignal("messageReceivedSignal");
        newWarningReceivedSignal = registerSignal("newWarningReceivedSignal");

        indexOfAccidentNode = par("indexOfAccidentNode").longValue();

        lastDroveAt = simTime();
        sentMessage = false;
    }
}

void Flooding::receiveSignal(cComponent *source, simsignal_t signalID, cComponent::cObject
*obj)
{
    Enter_Method_Silent();
    if (signalID == mobilityStateChangedSignal) {
        handlePositionUpdate(obj);
    }
}

void Flooding::onBeacon(WaveShortMessage *wsm)
{
    // not used for this algorithm
}

void Flooding::onData(WaveShortMessage *wsm)
{
    // statistics recording
```

```

    emit(warningReceivedSignal, 1);
    emit(messageReceivedSignal, 1);
    stats->updateAllWarningsReceived();
    stats->updateAllMessagesReceived();

    // prevent originating disseminator from participating in further dissemination attempts
    if (sentMessage)
        return;

    bool messagesRepeat = false;

    // is this a new warning message?
    size_t i;
    for ( i = 0; i < warningMessages.size(); ++i) {
        WaveShortMessage* warningMessage = warningMessages[i];
        if (wsm->getTreelId() == warningMessage->getTreelId()) {
            messagesRepeat = true;
        }
    }

    if (traci->getRoadId()[0] != ':')
    //   traci->commandChangeRoute(wsm->getWsmData(), 9999);

    // rebroadcast only if new message
    if (!messagesRepeat) {
        sendWSM(wsm->dup());

        stats->updateNewWarningsReceived();
        emit(newWarningReceivedSignal, 1);

        warningMessages.push_back(wsm->dup());
    }
}

void Flooding::handlePositionUpdate(cComponent::cObject *obj)
{
    // stopped for for at least 10s?
    if (traci->getSpeed() < 1) {
        if ((simTime() - lastDroveAt >= 10)
            && (!sentMessage)
            && (indexOfAccidentNode == getParentModule()->getIndex())) {

            std::cerr << "[DEBUG] ACCIDENT STARTED @simTime: " << simTime().str() << " for node: "
            << getParentModule()->getIndex() << endl;

            findHost()->getDisplayString().updateWith("r=16,red");
            if (!sentMessage) sendMessage(traci->getRoadId());
        }
    }
    else {
        lastDroveAt = simTime();
    }
}

```

```

}

void Flooding::sendMessage(std::string blockedRoadId)
{
    sentMessage = true;

    t_channel channel = dataOnSch ? type_SCH : type_CCH;
    WaveShortMessage* wsm = prepareWSM("data", dataLengthBits, channel, dataPriority, -1,2);
    wsm->setWsmData(blockedRoadId.c_str());
    sendWSM(wsm);
}

```

#### D.4.2 Flooding.h

```

#ifndef FLOODING_H
#define FLOODING_H

#include "veins/modules/application/ieee80211p/BaseWaveApplLayer.h"
#include "veins/modules/mobility/traci/TraCIMobility.h"
#include "veins/modules/stats/FranciscoStatistics.h"
#include <vector>

using Veins::TraCIMobility;
using Veins::AnnotationManager;
using std::vector;

class Flooding : public BaseWaveApplLayer
{
public:
    virtual void initialize(int stage);
    virtual void receiveSignal(cComponent *source, simsignal_t signalID, cObject *obj);

protected:
    TraCIMobility* traci;
    FranciscoStatistics* stats;
    vector<WaveShortMessage*> warningMessages;
    simsignal_t beaconReceivedSignal;
    simsignal_t warningReceivedSignal;
    simsignal_t newWarningReceivedSignal;
    simsignal_t messageReceivedSignal;
    simtime_t lastDroveAt;
    bool sentMessage;
    long indexOfAccidentNode;

protected:
    virtual void onBeacon(WaveShortMessage *wsm);
    virtual void onData(WaveShortMessage *wsm);
    virtual void handlePositionUpdate(cObject *obj);
    virtual void sendMessage(std::string blockedRoadId);
};

```

```
#endif // FLOODING_H
```

### D.4.3 Flooding.ned

```
package org.car2x.veins.modules.application.traci.flooding;
import org.car2x.veins.modules.application.ieee80211p.BaseWaveApplLayer;

simple Flooding extends BaseWaveApplLayer
{
    @class(Flooding);
    @display("i=block/app2");

    int indexOfAccidentNode = default(0);

    @signal[warningReceivedSignal](type=long);
    @statistic[warningReceivedSignal](record=count,vector; description="Warning Message
    Received");

    @signal[beaconReceivedSignal](type=long);
    @statistic[beaconReceivedSignal](record=count,vector; description="Beacon Message
    Received");

    @signal[messageReceivedSignal](type=long);
    @statistic[messageReceivedSignal](record=count,vector; description="Message Received");

    @signal[newWarningReceivedSignal](type=long);
    @statistic[newWarningReceivedSignal](record=count,vector; description="New Warning Message
    Received");
}
```

## D.5 Implementation of Probability technique

### D.5.1 Probability.cc

```
#include "Probability.h"
#include "veins/modules/messages/WaveShortMessage_m.h"
#include <iostream>
#include <cstdio>
#include <cstring>

using Veins::TraCIMobility;
using Veins::TraCIMobilityAccess;
using std::sprintf;

Define_Module(Probability)

void Probability::initialize(int stage)
{
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {
```

```
// std::cerr << "In Probability::initialize()" << endl;

ProbabilityThreshold = par("ProbabilityThreshold").doubleValue();
indexOfAccidentNode = par("indexOfAccidentNode").longValue();
randomRebroadcastDelay = par("randomRebroadcastDelay").doubleValue();

traci = TraCIMobilityAccess().get(getParentModule());
stats = FranciscoStatisticsAccess().getIfExists();
ASSERT(stats);
beaconReceivedSignal = registerSignal("beaconReceivedSignal");
warningReceivedSignal = registerSignal("warningReceivedSignal");
messageReceivedSignal = registerSignal("messageReceivedSignal");
newWarningReceivedSignal = registerSignal("newWarningReceivedSignal");

lastDroveAt = simTime();
sentMessage = false;
}
}

void Probability::receiveSignal(cComponent *source, simsignal_t signalID, cComponent::cObject
*obj)
{
    Enter_Method_Silent();
    if (signalID == mobilityStateChangedSignal) {
        handlePositionUpdate(obj);
    }
}

void Probability::onBeacon(WaveShortMessage *wsm)
{
// std::cerr << "In Probability::onBeacon()" << endl;
}

void Probability::onData(WaveShortMessage *wsm)
{
// std::cerr << "In Probability::onData()" << std::endl;
emit(warningReceivedSignal, 1);
emit(messageReceivedSignal, 1);
stats->updateAllWarningsReceived();
stats->updateAllMessagesReceived();

// prevent originating disseminator from participating in further dissemination attempts
if (sentMessage)
    return;

receivedMessages[wsm->getTreeId()].push_back(wsm->dup());

// is it a new warning message?
if (receivedMessages[wsm->getTreeId()].size() == 1) {
    stats->updateNewWarningsReceived();
}
```

```

    emit(newWarningReceivedSignal, 1);

    char buf[64];
    sprintf(buf, "%ld", wsm->getTreeId());
    scheduleAt(simTime() + SimTime(randomRebroadcastDelay, SIMTIME_MS), new
cMessage(buf));
    }
}

void Probability::handlePositionUpdate(cComponent::cObject *obj)
{
    // stopped for for at least 10s?
    if (traci->getSpeed() < 1) {
        if ((simTime() - lastDroveAt >= 10)
            && (!sentMessage)
            && (indexOfAccidentNode == getParentModule()->getIndex())) {

            std::cerr << "[DEBUG] ACCIDENT STARTED @simTime: " << simTime().str() << " for node: "
<< getParentModule()->getIndex() << endl;

            findHost()->getDisplayString().updateWith("r=16,red");
            if (!sentMessage)
                sendMessage(traci->getRoadId());
        }
    }
    else {
        lastDroveAt = simTime();
    }
}

void Probability::sendMessage(std::string blockedRoadId)
{
    t_channel channel = dataOnSch ? type_SCH : type_CCH;
    WaveShortMessage* wsm = prepareWSM("data", dataLengthBits, channel, dataPriority, -1,2);
    wsm->setWsmData(blockedRoadId.c_str());
    sendWSM(wsm);

    sentMessage = true;
}

void Probability::handleSelfMsg(cMessage *msg)
{
    if ((!strcmp(msg->getName(), "data")) || (!strcmp(msg->getName(), "beacon"))) {
        BaseWaveApplLayer::handleSelfMsg(msg);
        return;
    }
    else {
        // IS A REBROADCAST
        if (uniform(0,1)< ProbabilityThreshold)
            return;
        sendWSM(receivedMessages[atoi(msg->getName())][0]->dup());
    }
}

```

## D.5.2 Probability.h

```
#ifndef Probability_H
#define Probability_H

#include "veins/modules/application/ieee80211p/BaseWaveApplLayer.h"
#include "veins/modules/mobility/traci/TraCIMobility.h"
#include "veins/modules/stats/FranciscoStatistics.h"
#include <vector>
#include <map>

using Veins::TraCIMobility;
using Veins::AnnotationManager;
using std::vector;
using std::map;

typedef std::vector<WaveShortMessage*> WaveShortMessages;

class Probability : public BaseWaveApplLayer
{
public:
    virtual void initialize(int stage);
    virtual void receiveSignal(cComponent *source, simsignal_t signalID, cObject *obj);

protected:
    TraCIMobility* traci;
    FranciscoStatistics* stats;
    vector<WaveShortMessage*> warningMessages;
    simsignal_t beaconReceivedSignal;
    simsignal_t warningReceivedSignal;
    simsignal_t newWarningReceivedSignal;
    simsignal_t messageReceivedSignal;
    simtime_t lastDroveAt;
    bool sentMessage;
    double ProbabilityThreshold;
    long indexOfAccidentNode;
    double randomRebroadcastDelay;
    map<long, WaveShortMessages> receivedMessages; // treeld, WSM vector

protected:
    virtual void onBeacon(WaveShortMessage *wsm);
    virtual void onData(WaveShortMessage *wsm);
    virtual void handlePositionUpdate(cObject *obj);
    virtual void sendMessage(std::string blockedRoadId);
    virtual void handleSelfMsg(cMessage *msg);
};

#endif // Probability_H
```

### D.5.3 Probability.ned

```
package org.car2x.veins.modules.application.traci.probability;
import org.car2x.veins.modules.application.ieee80211p.BaseWaveApplLayer;

simple Probability extends BaseWaveApplLayer
{
    @class(Probability);
    @display("i=block/app2");

    double ProbabilityThreshold = default(0.5);
    volatile double randomRebroadcastDelay = default(uniform(0,500));
    int indexOfAccidentNode = default(0);

    @signal[warningReceivedSignal](type=long);
    @statistic[warningReceivedSignal](record=count,vector; description="Warning Message
Received");

    @signal[beaconReceivedSignal](type=long);
    @statistic[beaconReceivedSignal](record=count,vector; description="Beacon Message
Received");

    @signal[messageReceivedSignal](type=long);
    @statistic[messageReceivedSignal](record=count,vector; description="Message Received");

    @signal[newWarningReceivedSignal](type=long);
    @statistic[newWarningReceivedSignal](record=count,vector; description="New Warning
Message Received");
}
```

## D.6 Implementation of FranciscoStatistics

### D.6.1 FranciscoStatistics.cc

```
#include "FranciscoStatistics.h"
#include <iostream>
using std::cerr;
using std::endl;

Define_Module(FranciscoStatistics)

void FranciscoStatistics::initialize(int stage)
{
    if (stage == 0) {
        allBeaconsReceivedSignal = registerSignal("allBeaconsReceivedSignal");
        allNewWarningsReceivedSignal = registerSignal("allNewWarningsReceivedSignal");
        allWarningsReceivedSignal = registerSignal("allWarningsReceivedSignal");
        allMessagesReceivedSignal = registerSignal("allMessagesReceivedSignal");
        allBeaconsReceived = allWarningsReceived = newWarningsReceived = allMessagesReceived =
0;
        numAccidentsOccurred = 0;
    }
}
```

```

    }
}

void FranciscoStatistics::finish()
{
}

void FranciscoStatistics::updateAllBeaconsReceived()
{
    ++allBeaconsReceived;
    emit(allBeaconsReceivedSignal, allBeaconsReceived);
}

void FranciscoStatistics::updateNewWarningsReceived()
{
    ++newWarningsReceived;
    emit(allNewWarningsReceivedSignal, newWarningsReceived);
    // cerr << "num warnings: " << newWarningsReceived << simTime().str() << endl;
}

void FranciscoStatistics::updateAllWarningsReceived()
{
    emit(allWarningsReceivedSignal, ++allWarningsReceived);
}

void FranciscoStatistics::updateAllMessagesReceived()
{
    emit(allMessagesReceivedSignal, ++allMessagesReceived);
}

void FranciscoStatistics::incrementAccidentOccurred()
{
    emit(numAccidentsSignal, ++numAccidentsOccurred);
}

```

#### D.6.2 FranciscoStatistics.h

```

#ifndef FRANCISCOSTATISTICS_H
#define FRANCISCOSTATISTICS_H
#include <csimplemodule.h>

class FranciscoStatistics : public cSimpleModule
{
public:
    void updateAllBeaconsReceived();
    void updateNewWarningsReceived();
    void updateAllWarningsReceived();
    void updateAllMessagesReceived();

    int getNumberOfAccidentsOccurred() { return numAccidentsOccurred; }

```

```

void incrementAccidentOccurred();

protected:
    int allBeaconsReceived;
    int newWarningsReceived;
    int allWarningsReceived;
    int allMessagesReceived;
    int numAccidentsOccurred;

    simsignal_t allBeaconsReceivedSignal;
    simsignal_t allNewWarningsReceivedSignal;
    simsignal_t allWarningsReceivedSignal;
    simsignal_t allMessagesReceivedSignal;
    simsignal_t numAccidentsSignal;

protected:
    virtual void initialize(int stage);
    virtual void finish();

};

class FranciscoStatisticsAccess
{
public:
    FranciscoStatisticsAccess() {

    }

    FranciscoStatistics* getIfExists() {
        return dynamic_cast<FranciscoStatistics*>(simulation.getModuleByPath("stats"));
    }
};

#endif // FRANCISCOSTATISTICS_H

```

### D.6.3 FranciscoStatistics.ned

```

package org.car2x.veins.modules.stats;

simple FranciscoStatistics
{
    parameters:
        @class(FranciscoStatistics);
        @display("i=block/table2_vl.png");

        @signal[allBeaconsReceivedSignal](type=long);
        @statistic[allBeaconsReceivedSignal](record=count,vector; description="All Beacons Received");

        @signal[allNewWarningsReceivedSignal](type=long);
        @statistic[allNewWarningsReceivedSignal](record=count,vector; description="All New Warnings Received");
}

```

```
@signal[allWarningsReceivedSignal](type=long);
@statistic[allWarningsReceivedSignal](record=count,vector; description="All Warnings
Received");

@signal[allMessagesReceivedSignal](type=long);
@statistic[allMessagesReceivedSignal](record=count,vector; description="All Messages
Received");

//@signal[numAccidentsSignal](type=long);
//@statistic[numAccidentsSignal](record=count,vector; description="Number Of Occurred
Accidents");
}
```