Politecnico Milano

Telecommunication Engineering

Master's Degree in Internet Engineering

Discrete Event Simulation in Vehicle-to-Vehicle Transmission Based on Geometrical Channel Models

---

Supervisor:
Chiar.mo Prof Maurizio Magarini

Co-Supervisors:
Dott. Francesco Linsalata
Dott. Eugenio Moro

Master thesis of:
Nicolò Calzi

Student number:
903430

Academic Year 2020/2021

Contents

1 Introduction **4**

1.1 Motivation 4

1.2 Research Problem 5

1.3 Research Methodology 5

2 Technologies for Vehicular Communications **6**

2.1 IEEE 802.11p 6

2.1.1 Technical Specifications 7

2.2 Intelligent Transport System 9

2.2.1 Communicating Messages 11

2.2.2 ETSI ITS-G5 13

2.3 Device-to-Device Technology 15

2.3.1 D2D and 5G Networks 16

2.3.2 D2D Applications 17

2.4 Summary Background Technologies 19

3 Vehicular Ad-hoc Networks **20**

3.1 Features and Applications 22

3.2 Routing Protocols 25

3.3 RSUs Deployment 26

3.4 Vehicle-to-Everything Communication Protocol 27

3.4.1 LTE and 5G - V2X Implementations 29

3.5 Internet of Vehicles 31
3.6 Summary VANET Technology 34

4 Geometry-based Efficient Propagation Model for Vehicle-to-Vehicle Communication 36

4.1 Geometry-based Models 36
4.2 Tool Features 37
4.3 Path Loss Type: Free Space and Obstructed View 38
4.4 Summary Gemv^2 Technology 40

5 Frameworks and Tools Supporting the Discrete Event Simulator 42

5.1 OMNeT++ 43
5.1.1 Model 43
5.1.2 Structure 44
5.1.3 Results 45
5.2 Artery 48
5.2.1 Features 49
5.2.2 Architecture 50
5.2.3 Middleware 51
5.2.4 Environment Model 51
5.2.5 Storyboard 52
5.2.6 Gemv^2 in Artery 52
5.3 Inet 53
5.3.1 Features 53
5.4 Vanetza 58
5.5 SUMO and TraCI 59
5.5.1 Gemv^2 in SUMO 60
5.6 R-Trees structure 60
5.7 Summary Frameworks and Tools Technologies 62

6 Simulation Environment **64**

6.1 Simulation flow . . . 65

6.2 INI file, NED files and SUMO files . . . 67

6.2.1 Modifications to the code . . . 77

6.3 Launching from the terminal . . . 77

6.4 LinkType choice . . . 78

6.4.1 Gemv^2 link type . . . 79

6.5 Graphical view . . . 79

6.5.1 Visualization in SUMO . . . 80

6.6 While running . . . 83

6.7 End of the simulation . . . 85

7 Numerical Results and Analysis **86**

7.1 Parameters and Files Involved . . . 86

7.1.1 Fixed Parameters . . . 86

7.1.2 Changing Parameters . . . 87

7.1.3 .ANF file . . . 88

7.2 Values Types . . . 89

7.2.1 Browse Data: Vectors values . . . 89

7.2.2 Browse Data: Scalars values . . . 90

7.2.3 Browse Data: Histograms values . . . 91

7.3 Recorded Data . . . 91

7.3.1 Data not influenced by parameters . . . 91

7.3.2 Data changing with parameters . . . 96

7.4 Analysis . . . 98

7.5 Graphs . . . 101

7.5.1 Path Loss Type . . . 101

7.5.2 Passed Up Packets count . . . 102

7.6 Hypothesis . . . 103

8 Conclusions and Future Works **106**

9 Ringraziamenti **110**

A Installation Guide **I**

A.1 Introduction II

A.1.1 Ubuntu 20.04 LTS II

A.1.2 Suggestions II

A.1.3 Synaptic II

A.1.4 Git III

A.2 OMNeT++ III

A.2.1 Installing the Prerequisite Packages III

A.2.2 Downloading and Unpacking IV

A.2.3 Environment Variables V

A.2.4 Configuring and Building V

A.2.5 Verify the Installation VI

A.2.6 Starting the IDE VI

A.3 SUMO VII

A.3.1 Installing the Prerequisite Packages VII

A.3.2 Installation VII

A.4 Artery VIII

A.4.1 Requirements VIII

A.4.2 Cloning the Artery repository VIII

A.4.3 Create build directory IX

A.5 Getting started IX

A.5.1 Import Artery project IX

List of Figures

2.1 Vehicle safety communication examples. 7

2.2 The channels available for IEEE 802.11p. 8

2.3 Standards and communication stack. 8

2.4 Schematic view about how ITS works. 11

2.5 System architecture for Cellular-VCS based on CAMs (BS broadcast) and DENMs (D2D broadcast). The base station controls the communication. CAMs are processed on a server in the core network. 12

2.6 C-ITS protocol stack with ETSI ITS-G5 (ETSI EN 302 665). BTP means basic transport protocol. 14

2.7 Cellular communication and D2D communication. Both single-hop and multi-hop (including D2D relay) networks formed by D2D links are shown. 15

2.8 D2D mechanism: the nearby communicating nodes open a direct link to each other instead of communicate with the base station. 17

2.9 Scheme 1: Evolution of the mentioned Background technologies. . . . 19

3.1 An example of a VANET network. 21

3.2 Applications in VANET. 22

3.3 Important parameters in a VANET network. 23

3.4 Classification of the VANET Applications [1]. 23

3.5 Implementation of RSUs in a VANET network. 26

3.6 Vehicle connections in the V2X paradigm. 27

3.7 Cellular-5G VANET model. 30

3.8 IoV illustration. 31

3.9 Mutual relationship between IoV and big data. 32

3.10 Scheme 2: VANET technology. 34

4.1 Link types and propagation effects captured by Gemv^2. White rectangles represent vehicles, gray rectangles represent buildings. 39

4.2 Scheme 3: Gemv^2 technology. 40

5.1 OMNeT++ model. 44

5.2 Relations of the files in OMNeT++. The example comes directly from the files used in this work. 45

5.3 Artery architecture [2]. 50

5.4 Example of simulation running. Each vehicle has its range of transmission (visualized as a cone) and the field-of-view (visualized as dash lines). 52

5.5 INET-LayeredProtocolBase definition of signals. 55

5.6 INET-PhysicalLayer-Radio definition of signals and collected statistics. 55

5.7 Components, dependances and features of Vanetza's tools. 58

5.8 The example shows an exchange of TraCI messages. 59

5.9 [a] R-Tree structure. 61

5.10 [b] Relationships between rectangles in the structure. 61

5.11 Scheme 3: Tools and Frameworks. 62

6.1 Simulation flow. 66

6.2 omnetpp.ini simulation file 69

6.3 World.ned file: Design view. 70

6.4 World.ned file: Submodules. 70

6.5 World.ned file: Source view (C++ code). 71

6.6 Traci Manager. 72

6.7 Radio medium. 73

6.8 Physical Environment 73

6.9 Static Nodes and Nodes. 74

6.10 LinkType.sumo.cfg. Here the LOS type is shown. 74

6.11 LinkType.net.xml. Here the LOS type is shown. 75
6.12 LinkType.rou.xml. Here the LOS type is shown. 76
6.13 LinkType.poly.xml. Here the NLOSf type is shown. 76
6.14 Modifications. 77
6.15 Simulation schema. 77
6.16 Gemv^2 menù. 78
6.17 OMNeT++ graphical view. 79
6.18 LOS view: standard visualization. 80
6.19 LOS view: real-world visualization. 80
6.20 NLOSv view: real-world visualization. 81
6.21 NLOSb1 view: standard visualization. 81
6.22 NLOSb2: standard visualization. 82
6.23 NLOSf view: standard visualization. 82
6.24 RSU path. 83
6.25 Node path. 84
6.26 Target nodes. 84

7.1 Combinations of simulated parameters. 87
7.2 .anf file view. Here NLOSv type is shown. 88
7.3 ISO/OSI stack and the PassedUp/SentDown mechanism handling the packets. 90
7.4 Counts of LOS/NLOS in LOS type. 92
7.5 Counts of LOS/NLOS in NLOSv type. 92
7.6 Counts of LOS/NLOS in NLOSf type. 92
7.7 Transmission state. 93
7.8 Arrival computation count. 93
7.9 Radio frame send count. 94
7.10 Radio mode count. 94
7.11 Reception computation count. 95

7.12 Transmission count. 95
7.13 SentDown. 96
7.14 Reception state. 96
7.15 PassedUp. 97
7.16 packetErrorRate. 97
7.17 Arrival computation count → Reception computation count. 98
7.18 Radio frame send count == ∑(countLOS + countNLOS). 99
7.19 rcvdPkFromHL == sentDownPK. 99
7.20 Transmission count == ∑(rcvdPkFromHL) == ∑(sentDown). 100
7.21 Path Loss Type graph. 101
7.22 PassedUpPk:count graph. 102
7.23 Tabulated values. 103
7.24 Scalar values - sentDownPk:sum(packetBytes). 104
7.25 Scalar values - sentDownPk:count. 104
7.26 Vectors values - sentDownPk:vector(packetBytes). 104
7.27 Analysis of Transmission values in the NLOSv file. 105

Chapter 1

Introduction

1.1 Motivation

Digital innovation is here and now.

Nowadays, every technological company is running to be the leader in a specific market branch with products and services to be sold to other companies. To do so, a fast transition to the digitalisation of each and every of such items is needed and a bunch of new technologies is already expanding their role influencing the market.

Not only the concept of typical transmissions on a cable, wireless or optic network is considered, but the spread of mobility is leading to the connection of all the devices in order to create a completely connected world where everything can communicate an information.

In this work, starting from a general overview of the technologies involved (from the wireless project IEEE 802 to the Device-to-Device communication and its integration with LTE cellular network and 5G, from Intelligent Transport System to the Internet of Things) we will move to the Internet of Vehicles and its concepts of intelligent nodes (indeed cars, trucks and every other type of vehicles connected to an internet network) able to transmit whichever data to other nodes, to an infrastructure or to a pedestrian walking around.

We will see the concepts that aim to the creation of a specific network for this purpose, the so called Vehicular Ad-Hoc Network, its pros and cons and why there was the need for specific tools running different scenarios in order to study how a vehicle can store information to be communicated to others.

We will finally move to the tool used for this work, the geometrical model that adds features regarding the link type used in the simulation.
We will see the **Gemv²** tool analysing how a software specifically created for such simulation runs and produces results that can be reported to the concept of a real situation.

The main purpose of this work is to see what an already existing software specifically created to simulate a real-world scenario performs, analysing which results it can give to the end user. We will see how the vehicular communication behaves and what are the differences proposed when running on a free-space scenario or an high density scenario, being it due to high number of other vehicles or due to different obstructing objects.

1.2 Research Problem

When analysing a scenario there are fixed costs that are hardly handle if after all it finds out the proposed solution is not implementable. Plus, not all the conditions are easily replicable for all the possible scenarios we can find in the real world and not all traffic conditions can be found in the same site chosen for the analysis.

That is why the developers prefer to simulate different scenarios and traffic conditions on a simulator where, with a few changes in its code, can replicate what happens while physically running with a vehicle in such situations.

The used simulation scenarios gives an almost perfect account regarding what's there outside.

1.3 Research Methodology

The methodology involved for this work is the one attributable to what explained in "Motivation" section: from a general overview of the theoretical concepts, to the study of the Gemv^2 tool used.

We will see how Gemv^2 works, which files cooperate to the final result, the bunch of frameworks and other tools that are behind what the user sees on the screen; we will see how to launch the program and how to handle modules or modifications to the code.

At the end, we will analyse the results given and we will make some assumptions and hypothesis when using an already existing tool specifically created for this purpose.

Chapter 2

Technologies for Vehicular Communications

The main purpose of the first chapter is to give a general overview regarding the technical background and features attributable to vehicular communication. The specific aspects of any arguments not directly related with vehicular communication are not treated in this work but only mentioned when appropriate.

2.1 IEEE 802.11p

Starting from the very beginning, the "802" is the project coming from the Institute of Electrical and Electronics Engineers (IEEE) that aim to develop all the standards needed when working on a generic network, being it a LAN, a WPAN, a Token Ring net etc..

Our interest in this work is focused on the ".11p" implementation, the branch that gives support to all the technologies requested on a Wireless LAN (standard IEEE 802.11), and then, going deeper, on all the specifics with the aim of enhance the **Vehicular Communication**, a relatively new way of transmission while driving a vehicle (standard IEEE 802.11p). This additional document takes the name of Wireless Access in Vehicular Environment (WAVE), a standardization process originated from the allocation of the Dedicated Short Range Communications (DSRC) spectrum band in the United States and adapted to European standards.[3]

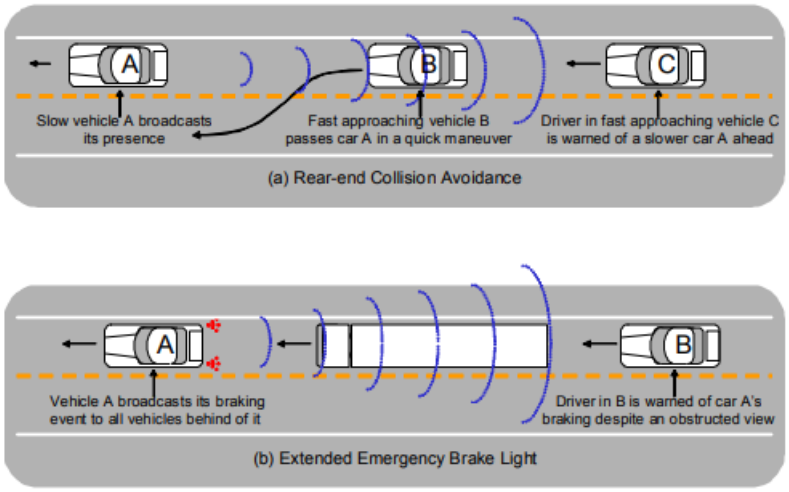


Figure 2.1: Vehicle safety communication examples.

IEEE 802.11p (also known as IEEE 820.11p WAVE) gives the basics required by the vehicular communication which includes not only vehicle-to-vehicle transmission but also vehicle-to-infrastructure and roadside units exchange of information in a high-speed an efficient way, due to the rapidly varying scenario. All these features will be explained later on in the dedicated sections.

2.1.1 Technical Specifications

Here, a brief introduction to the main characteristics of the IEEE 802.11p standard is given but without going into detail.

The IEEE 802.11p **physical layer** is based on the Orthogonal Frequency Division Multiplexing (OFDM) combined with a convolutional code. Since the vehicular channel varies rapidly with the environment surrounding the vehicles, some improvements have been made to the time-domain and frequency-domain parameters (the first have been doubles while the second have been halved).[4]

The IEEE 802.11p **MAC layer** skips some of the typical features of the stuck, such as authentication, association, and data confidentiality services which are not used. The layer uses the Enhanced Distributed Channel Access (EDCA) to implement a differentiation scheme in the Quality of Service (QoS) requested. It typically uses channels of 10 MHz bandwidth in the 5.9 GHz band, from 5.850 to 5.925 GHz (these essential features will lead us through all the work), basing the division on the Distributed Coordination Function (DCF) that, in turn, make uses of the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) algorithm. The spectrum of WAVE is allocated in the upper 5 GHz range (see Fig. 1) and the division sees one slot

for Accident Avoidance, two slots for Service Channels, one slot for the Controlling and one last slot for high power and long range transmission. The different channels cannot be used simultaneously and each station continuously alternates between the Control Channel and one of the Service Channels or the Safety Channels.[5]

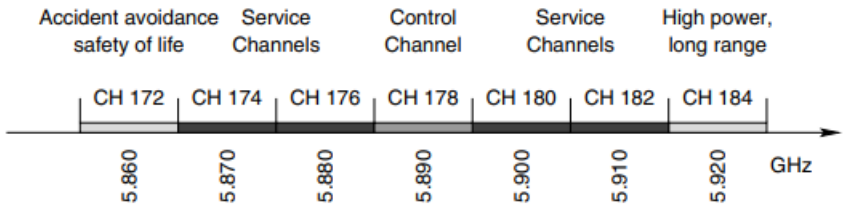


Figure 2.2: The channels available for IEEE 802.11p.

The listening period of the algorithm depends on the type of message that could be a Cooperative Awareness Message (CAM) or a Decentralized Event Notification Message (DENM), better explained later on.

The IEEE 802.11p standard is thus limited to MAC and PHY layers, adding its own features to the already existing levels of the stack; Pay attention, as we can see in the following picture, it does not substitute but indeed it is integrated with the other technologies because it is meant to work with them and not to substitute them.

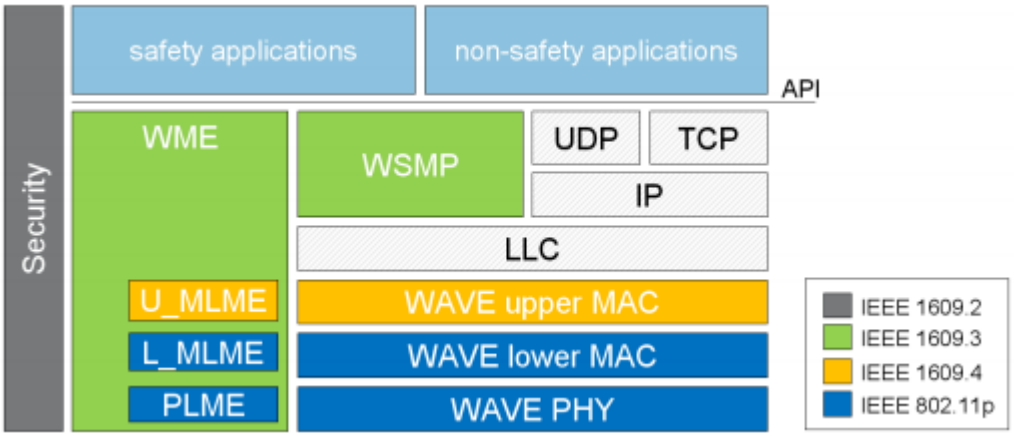


Figure 2.3: Standards and communication stack.

2.2 Intelligent Transport System

The term **Intelligent Transport System (ITS)** refers to the integration of fields such as telecommunications, electronics and IT with everything related to the intelligent transports, from planning and designing to the managing of the physical transport itself. The aim is to improve the road safety (for vehicles, people and goods) and the quality of service given, with special attention on current issues such as the respect for natural resources and environment.

With the concept of "Smart City" transmuting cities into digital societies, making the life of its citizens easy in every aspects, Intelligent Transport System becomes the indispensable component among all. ITS's goal is to achieve traffic efficiency by minimizing traffic problems, spreading traffic information through the network in order to make people take the best choice based on the information received.[6]

The use is not just limited to traffic congestion control and information, but also to road safety and efficient infrastructure usage. It varies depending on the technologies applied, from basic management systems such as satellite navigation, traffic light control systems or speed detectors, up to advanced applications integrating real-time data from various external sources, such as meteorological information, bridge de-icing systems and so on.

In fact, some example of applications where it is likely to find the integration with ITS are [7]:

- **Traffic Mobility and Management Control:** vehicle's transit and stopover, traffic viability, electronic payments, emergency management, traffic lights monitoring and penalties;

- **Advanced Vehicle Control:** vision support, collision avoidance systems, intelligent cruise control systems, vehicle monitoring, automatic driving, autonomous and assisted navigation systems;

- **Advanced Information spreading System:** traffic and travel information systems, public transport information systems, remote assistance;

- **Public Transport Management:** localization, deposit management schemes, integrated payment systems, call service management;

- **Handling of the Freight Transport:** support to the logistics, resource management systems, Vehicle and cargo management systems, dangerous goods management.

ITS has now become a multidisciplinary conjunctive field of work and thus many organizations around the world have developed solutions to provide ITS applications to meet the needs. In fact, the application of ITS is widely accepted and used in many countries nowadays.

An interesting example about the integration of Intelligent Transport System with citizens life is coming from the city of Glasgow; here, ITS gives regular information to the daily commuters about public buses, timings, seat availability, current location of the bus, time taken to reach a particular destination, next location of the bus and the density of passengers inside the bus.

But how does ITS work? The Traffic Management Centre (TMC) is the vital unit of ITS, administered by the transportation authority. Here all data is collected and analyzed for further operations and control management of the traffic in real time. The process includes:

- **Data Collection:** The data here is collected via vary hardware devices that lay the base for further ITS functions. These devices are Automatic Vehicle Identifiers, GPS based automatic vehicle locators, sensors, camera etc. The hardware mainly records the data like traffic count, surveillance, travel speed and travel time, location, vehicle weight and delays.

- **Data Analysis:** The data that has been collected and received is processed in various steps and further altered and pooled for analysis in order to predict traffic scenario to be delivered appropriately to users.

- **Data Transmission:** Rapid and real-time info communication is the key to proficiency in ITS implementation. Traffic-related announcements are transmitted to travelers through internet, SMS or onboard units of vehicle.

- **Traveler Information:** Travel Advisory Systems (TAS) is used to inform about transportation updates to the traveling user. The system delivers real-time information like travel time, travel speed, delay, accidents on roads, change in route, diversions, work zone conditions etc. This information is delivered by a wide range of electronic devices like variable message signs, highway advisory radio, internet, SMS and automated cell.

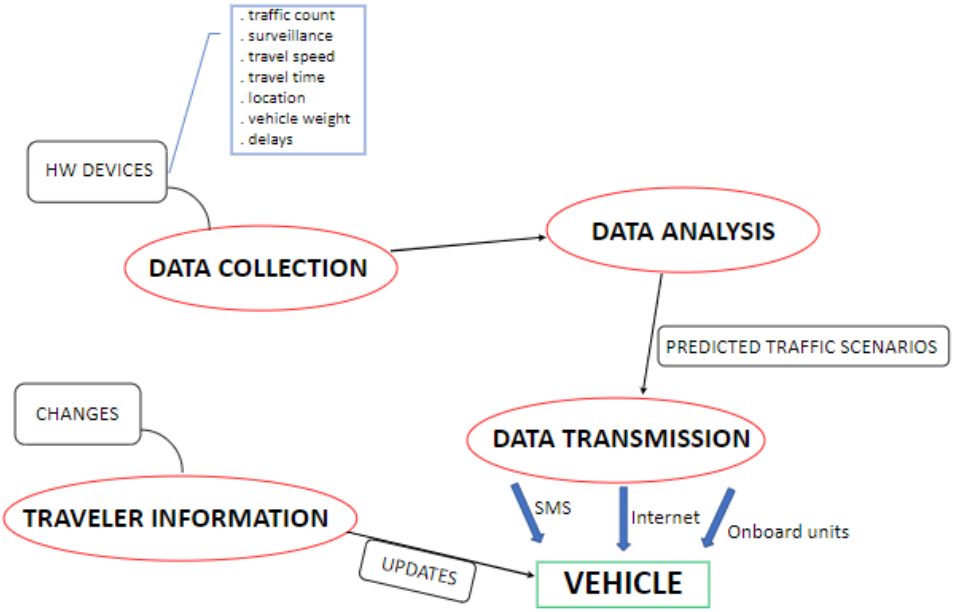


Figure 2.4: Schematic view about how ITS works.

2.2.1 Communicating Messages

When talking about ITS, there are two types of messages that an ITS station can forward to the communicating nodes in order to support vehicular safety and traffic efficiency applications, giving continuous status information about surrounding vehicles and asynchronous notification of events [8].

(1) **Cooperative Awareness Message (CAM)**

(2) **Decentralized Environmental Notification Message (DENM)**

The need of a standardization in the message mechanism comes from the issue arising in vehicular communication, that is a continuous and periodic status exchange between nodes regarding status of vehicles or of the roadside unit, locations, IDs, speed, etc.. and the need of asynchronous notifications from one node to another that may not be in range in the instant of the transmitted message.

The **CAM** is a periodic message that an ITS node forward in broadcast to all other nodes in range (single hop from the source node) when a security situation arises, such as when the vehicle is started or whenever there is a change in the location or speed status. In this way, the receiving station is aware of other vehicles in the proximity knowing its positions, movement and other relevant characteristics that may have been transmitted. Thus, the message will contain: (i) ID of the transmitting node, (ii) type

of station to which it is linked, meaning a mobile station or if it is a public/private authority or anything else, (iii) position of the node, from latitude and longitude to height and (iv) all sort of characteristics coming from the standard.

The **DENM** is generated every time the ITS station sees a relevant event to be notified in order to aware all the other nodes in the range of the event (e.g. a security issue situation or the transit of an emergency vehicle). This message handles the lifetime of the event: the ID generated by the source station could be re-used to eventually update the event or to inform new nodes entering the area about the event that is occurring; if the source node exits the area, a new station could take the ID of the event and handle it with new updates. The event usually contains the detection of the the awareness situation (e.g. vehicle breakdown, traffic jam, etc.), the position of the area, the detection time of the event and the expected expiring time of it. the area over which the DENM message should be disseminated and the frequency to which the message is forwarded [8].

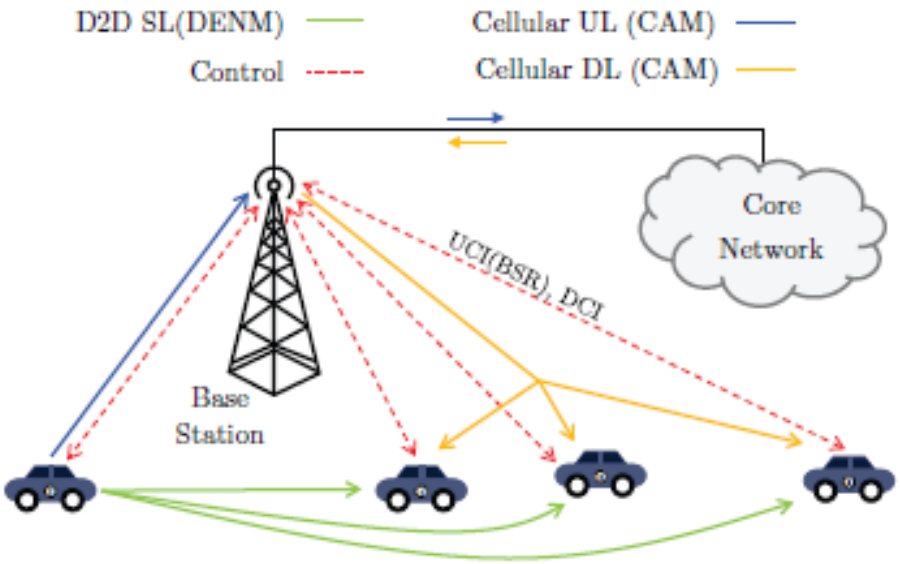


Figure 2.5: System architecture for Cellular-VCS based on CAMs (BS broadcast) and DENMs (D2D broadcast). The base station controls the communication. CAMs are processed on a server in the core network.

In the figure above we can better see the connection that occur from messages to classical communication mechanism such as the cellular and the D2D broadcast, in the specific the CAM message is associated with cellular network while the DENM message is associated with the D2D mechanism (this technology will be better explained in the dedicated section) [9].

The CAM message (which is a Base Station broadcast transmission) and the DENM

message (the D2D broadcast mechanism) are alternated in a Round Robin algorithm implementation; DENM messages have a higher scheduling priority w.r.t. CAM messages due to their safety application in the scenario.

2.2.2 ETSI ITS-G5

The ascending spread of Intelligent Transport System is possible thanks to the expansion of other related technologies with the same purpose of improving quality of citizens life through an increasingly whole network where every device in range can communicate with all the others, that is mobile device or vehicle or even infrastructure.

Various forms of wireless communication technologies have been proposed for Intelligent Transport Systems. In the USA, they developed a short-range communication mechanism (below 500mt) called Dedicated Short Range Communication (DSRC) while the European Telecommunication Standard Institure (ETSI) created its own vehicular communication paradigms as a certified standard common to all the European Nations, the ETSI ITS-G5 standard. Both the specifications are made from the Wi-Fi standard IEEE 802.11p protocol and, starting from it, other technologies can be implemented alongside to enable new features (e.g. extend the range of these protocols using ad hoc mobile networks or wireless mesh networks, using the Device-to-Device technology or the combination of Cellular 5G and LTE technologies).

We can say **ETSI ITS-G5** is an extension of the general WiFi standard 802.11p WAVE modified and optimized to operate in a dynamic automotive environment [10]; it makes use of the high mobility tools, it delivers safety messages in the 5.9 GHz channel specifically allocated for vehicle communication applications, it gives a bunch of new features to the already existing protocols introducing a sort of vehicular communication protocol stack, like for example the implementation of a specific set of messages that vehicles use to communicate each other.

Cooperative-ITS (C-ITS) can be reported as the technology that strengthens ETSI ITS-G5 with a series of tools in charge of supporting the connectivity between vehicles, vehicles and roadside infrastructure, traffic signals as well as with other road users; ETSI ITS-G5 is therefore a set of C-ITS tools that aim to achieve the inter-vehicle communications in a reliable and efficient way [11].

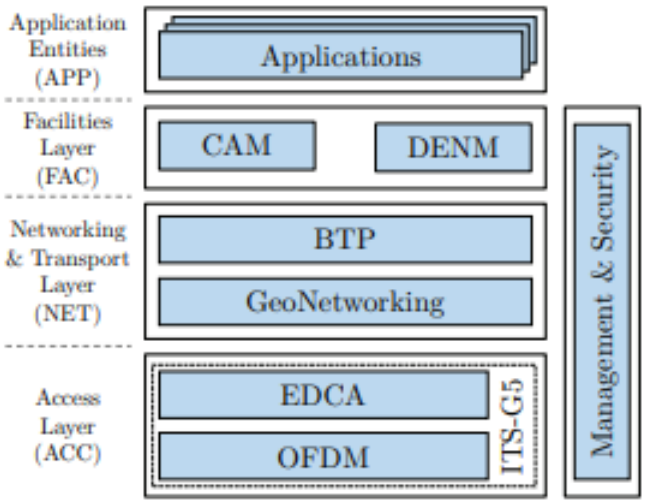


Figure 2.6: C-ITS protocol stack with ETSI ITS-G5 (ETSI EN 302 665). BTP means basic transport protocol.

Other specifications can be strongly found in the Control Channel and Service Channel inside the device containing an ITS-G5 technology, the so called ITS-G5 STA; the device shall operate outside the context of a Businness Support System (BSS) typical of the classical devices [12].

2.3 Device-to-Device Technology

Device-to-device (D2D) communication can be seen as the technology behind every transmission that occurs without involving a network infrastructure, such as an access point or a base station, and thus allowing user equipments (UEs) in close proximity to communicate using a direct link. The main purpose is to reduce the time taken by the conventional transmission enabling ultra-low latency communication period and boosted data rate, resulting in an improvement of throughput, energy efficiency, delay and fairness [13].

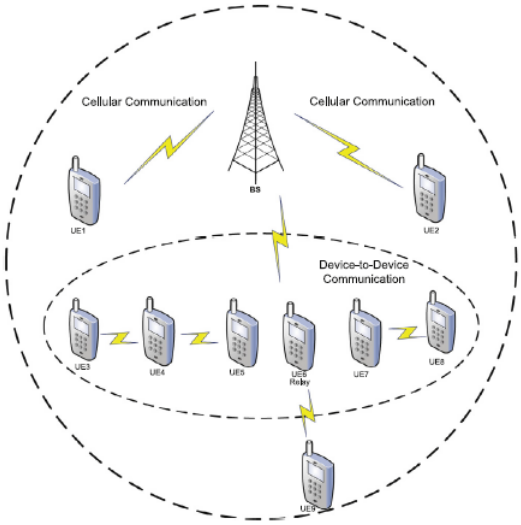


Figure 2.7: Cellular communication and D2D communication. Both single-hop and multi-hop (including D2D relay) networks formed by D2D links are shown.

We can say Device-to-Device refers to the communication between closed devices which can be cell phones or even vehicles, with the spreading advancement of technologies where everything, a vehicle indeed, is connected to a network [14]. Various implementations deriving from the fifth generation (5G) wireless network communication, such as short-range wireless technologies (like Bluetooth, WiFi Direct and LTE Direct) and vehicle-to-vehicle paradigm, can be used to enable D2D communication; alternatively, D2D can be used to bring advanced features to such technologies.

An interesting example is the combination of D2D with the continuing growing technology of the Internet of Things (IoT) resulting in the so called Internet of Vehicles (IoV): here, a vehicle connected to a wireless network can advertise nearby vehicles before it changes street, lane on a highway or when it accelerates or slows down. This is the already mentioned vehicle-to-vehicle paradigm that it will be dealt later on.

2.3.1 D2D and 5G Networks

The upcoming 5G network is expected to support aggregate data rates 1000 times faster than the current 4G network. The spectral efficiency and energy efficiency should be 10 times higher than those of 4G network, with an end-to-end latency around 1 ms [14].

5G network will bring a large amount of new technologies, including Device-to-Device communication; D2D is fundamental to 5G thanks to its dedicated short-range link where each link can operate concurrently with the others, improving network capacity and avoiding interference. Moreover, D2D can enable other technologies strictly related to 5G such as heterogeneous network and massive-MIMO, implementations that will not be explained here in the provided but which serve to improve spectral efficiency and data rates.

D2D is now working with a dedicated channel named Dedicated Short-Range Communications (DSRC), which can be referred as an implementation of standard Wi-Fi specification 802.11p, the traditional wireless access in automotive industry. The limitations are greatly growing since first year of cooperation, that is why new paradigms have been created taking an important role in the advancement of such technology.

Cellular-Vehicle-to-Everything (C-V2X) is the new area involving the strong characteristics of D2D and its agile way of acting with 5G together with the recent paradigm of vehicle communications. C-V2X is often used to reference classical cellular communication, taking the name of LTE-V2X [**?**].

The implementation of D2D in cellular networks not only improves performance but also solves some of the problems of such networks:

(a) **Synchronization**, helping a UE selecting the time slot and frequency slot that best fits its area in a more efficient way (talking about energy-efficient) in the communication with other peers, both directly in range or not;

(b) **Peer discovery**, implementing an efficient method to discover other nearby UEs quickly and with low power consumption;

(c) **Mode selection**, choosing between cellular and D2D among the set of candidates UE that could implement D2D communication taking into account best performance objective like spectral efficiency, low latency, transmitting power and minimum QoS at receiver;

(d) **Resource (radio) allocation**, to maintain the direct link in the D2D pairs (here D2D provides the algorithms to be used);

(e) **Interference management**, with a precise scheduling to lower down the interference occurring in cellular and D2D when sharing frequencies. The D2D proposed algorithms involve optimization problems to balance transmitting power and required QoS;

(f) **D2D mobility**, working with dynamic users and not only static ones;

(g) **Pricing**, to maintain a stable direct link charging, at the same time, services to users;

(h) **Security**, guaranteeing privacy and anonymity of users.

In conclusion, D2D is another enabling technology for 5G networks but, in contrast to others, it works in cohesion with and not only as part of 5G. An intelligent implementation that lightens the network at the cost of more sophisticated algorithms [15].

When two devices are connected to the same cell, they release from the network and connect to each other on a direct link.

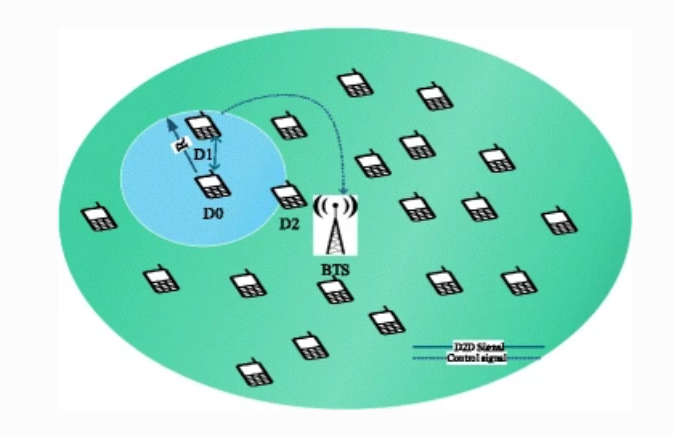


Figure 2.8: D2D mechanism: the nearby communicating nodes open a direct link to each other instead of communicate with the base station.

This technology will impact considerably the vehicular communication providing information about traffic condition, accidents, roads conditions, thus enhancing quality of driving for users.

2.3.2 D2D Applications

The applications where nowadays we can see a D2D communication are numerous and include those systems where a large amount of data needs to be transferred quickly with a short range connection. In addition to the above-mentioned applications to support 5G, here are some other examples specifics of D2D [14]:

- **Local data services:** D2D supports local data services through unicast, groupcast and broadcast transmissions. Examples include:

  (a) Information sharing, through D2D links to transfer files, audios and videos with higher data rates and lower energy than those in conventional cellular channels.

  (b) Data and computation offloading, where a device with a good Internet connectivity can act as an hotspot to which data is offloaded/cached from the BS and from which other devices may download data using D2D links.

- **Coverage extension:** when a UE encounters poor signal quality while connecting to the Base Station, it can find a better link to the BS connecting with another or others UEs in a closer position (or better conditions when considering mountainous environment) to the BS; the UEs acting as relays compose a multi-hop communication or a parallel-paths communication of collaborative devices.

  (a) Emergency communications can be considered special case of coverage extensions. In case of natural disasters (hurricanes, earthquakes, etc..) the traditional communication network may not work due to the damage caused; a network can be established via D2D multi-hop communication to the closer working BS.

- **Machine-to-machine (M2M) communication:** it is an enabling technology for Internet-of-Things (IoT) since it affords ultra low latency time and, thus, real-time responses. A particular application is vehicle-to-vehicle (V2V) communication where D2D links can be utilized to share information between neighboring vehicles quickly and offload traffic efficiently, as well as vehicle-to-infrastructure (V2I) and vehicle-to-pedestrian (V2P) communication.

2.4 Summary Background Technologies

In the scheme below are shown all the technologies mentioned in chapter 2:

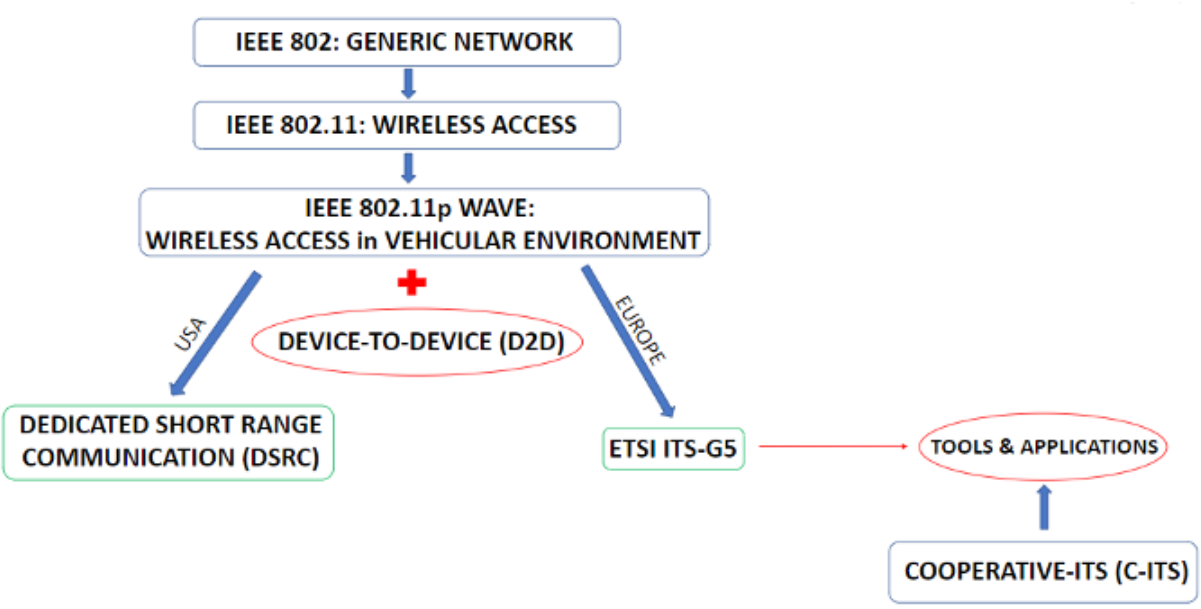


Figure 2.9: Scheme 1: Evolution of the mentioned Background technologies.

Chapter 3

Vehicular Ad-hoc Networks

Classical networks are considerably far away from their first developments; from cable to optical to wireless networks, they experienced an enormous growth in their deployments and applications in everyday life.

One of the biggest achievement is, without any doubt, the displacement from wired/ wireless and centralized networks to wireless **decentralized** networks and then to **mobile** decentralized networks, where "decentralized" means that there is no longer need for an infrastructure to run correctly (fixed routers and/or access points to be linked to the network) but every node in range is in charge to forward information to other nodes in a dynamic way (one time, the selected node can be part of the transmission, another time the same node could be the final receiver of the transmission). The new technology can be referred to a Wireless Ad-hoc NETwork (WANET) or a Mobile Ad-hoc NETwork (MANET).

Focusing on MANET, the key role is the independent movement of each device which compose the network and thus the continuously changing link from device to device; from this, there are some challenges to go through: each device must continuously maintain the information update about real-time routing status of the network and about the status of each node in range, each sharing limited communicating bandwidth to do not overlap the signals, all of that considering a highly dynamic changing topology.

With the progresses of new technologies where everything is connected to a network, even vehicles are becoming increasingly important in the spread of the information to the point that a specific branch of MANET has been specialized in vehicular communication; that is how **Vehicular Ad-hoc NETwork (VANET)** is born.

VANET is a particular case of wireless multihop network, that has the constraint of fast topology changes due to the high node mobility, where nodes are indeed vehicles

on a street [16].

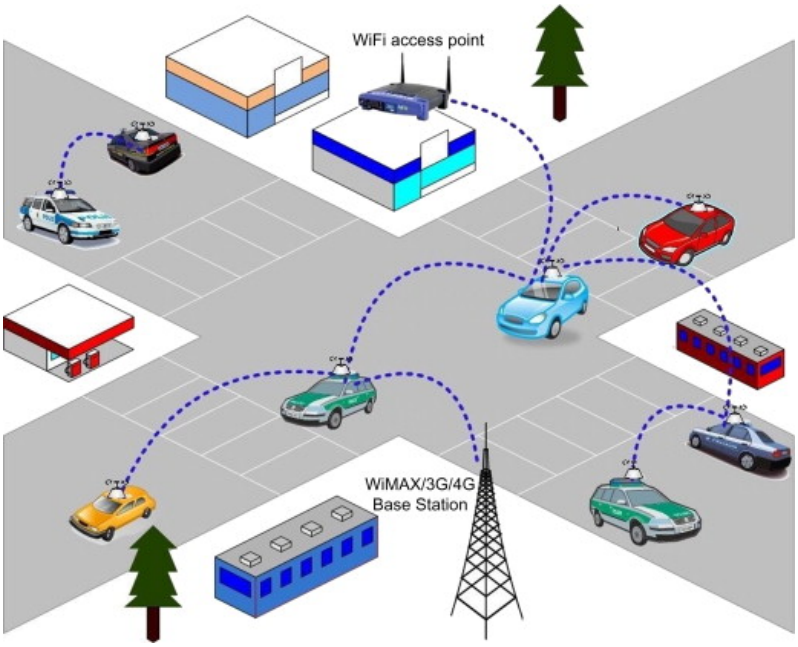


Figure 3.1: An example of a VANET network.

VANET enables a wide range of applications (see fig. 3.2), such as prevention of collisions, safety, blind crossing, dynamic route scheduling, real-time traffic condition monitoring, etc. Moreover, it provides Internet connectivity to vehicular nodes. The key is to install sensors on the vehicles and make them communicate with sensors on other vehicles, sensors on the infrastructure (the so called Roadside units) and sensors on the devices of people; in this way, the driver is always updated regarding the surrounding environment while he is driving.

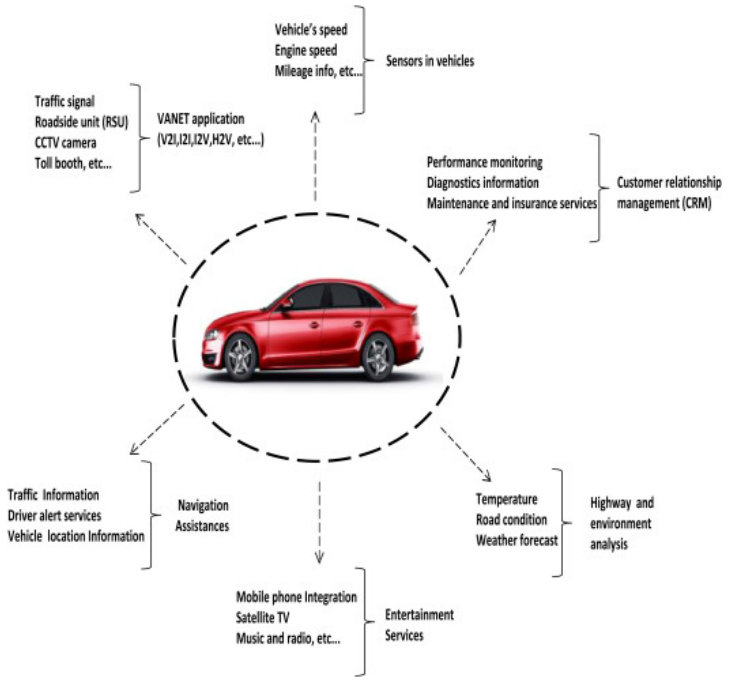


Figure 3.2: Applications in VANET.

Going a step back in the previously mentioned technologies, VANET is the network giving support to the Intelligent Transport System (ITS). While ITS has the role to improve road safety and traffic efficiency offering services [17], VANET gives the physical tools (sensors on vehicles) to apply ITS's purpose. The common goal is the full concept of smart cities and that is where ITS + VANET together with the Internet of Things (IoT) want to go in the early future.

3.1 Features and Applications

The high dynamic varying scenario in which a vehicle is running, implies that information cannot be stored for a long time because it would take up space for more relevant information in the new portion of scenario where the vehicle has come. The contents of the information produced and consumed by vehicles has local and limited relevance in terms of time (explicit lifetime and limited temporal scope), space (local validity, limited spatial scope and local interest), and agents involved (important to agents in a limited area around the vehicle) [18].

We can expect that communication between nodes has never happen before and will never happen again after first contact, as well as they can communicate for a limited period of time in most of the cases [19].

In the table below, some of the areas where information is applied are shown:

Application	Contents	Local interest	Local validity	Lifetime
Safety warnings	Dangerous Road	All	100 m	10 s
Safety warnings	Accident	All	500 m	30 s
Safety warnings	Work zone	All	1 km	Construction
Public service	Emergency vehicle	All	500 m	10 min
Public service	Highway information	All	5 km	All day
Driving	Road congestion	All	5 km	30 min
Driving	Navigation Map	Subscribers	5 km	30 min

Figure 3.3: Important parameters in a VANET network.

Moreover, the key concept is the centricity of the information and not of the source. The content is important, the source is not since the same data can be transmitted by each node independently.

Taking into account specifics aspects of VANET, several definitions have been provide by different authors [20].

Biggest and most important division can be seen in the fig. 3.4 where VANET applications can be divided between **Safety** (as situation awareness applications and safety messaging applications, including spreading an alarm or warning with the aim of avoiding danger and reducing risk) and **Non Safety** (as comfort driving, enhancing of the driving process and traffic information systems, including information about a new product or business or the shortest path to a destination) applications.

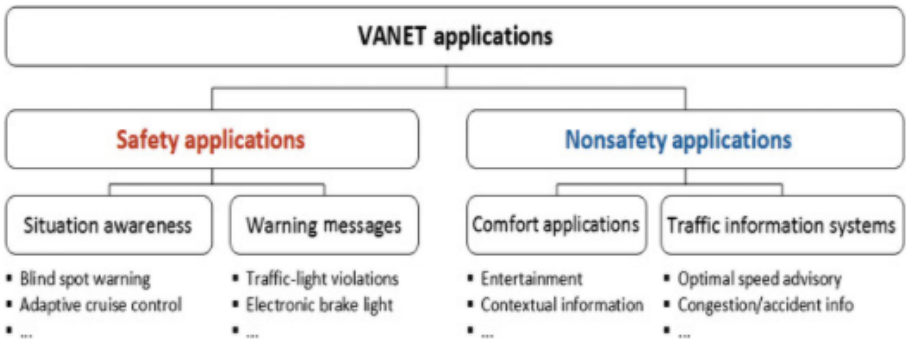


Figure 3.4: Classification of the VANET Applications [1].

After this initial subdivision, we can make a differentiation based on the types of applications, according to their aim. This will lead us to two definitions, that can be considered as separate paradigms or working in cohesion:

- First category:
  1. General Information Services
  2. Vehicle Safety Information Services
  3. Individual Motion Control
  4. Group Motion Control
- Second category:
  1. Active Road Safety apps
  2. Traffic Efficiency and Management apps
  3. Commercial apps

Without going into detail, we can say that each application brings advantages and/or disadvantages, that's why there is a need of cohesion for each of them and not a strong differentiation. Every single application imposes diverse requirements on the supporting technologies, leading to a number of challenges in order to reach data dissemination in the network. The main research challenges to be considered for data dissemination include [21]:

- **Scalability:** focusing on scalable data dissemination, the goal is to reduce data redundancy by detecting traffic density;
- **Security and Trust:** During data dissemination, integrate a security mechanism is necessary to be accepted by costumers. On the other hand, integrating security schemes could increase the delay of message arrival and so it must be necessary to reach a fair tradeoff between pros and cons;
- **Quality of Service and Traffic Characterization:** these are additional important issues to be addressed by data dissemination, since different types of applications are expected to have different QoS requirements;
- **Node Cooperation:** nodes are willing to cooperate and allow the use of their resources to provide connectivity for other nodes, but this assumption might not always be valid and some nodes could make use of services available by other nodes while not allowing similar use of their resources by the same nodes;

- **Simulation:** The computational cost of implementing data dissemination schemes and applications leads the future development to a strong simulation test-bed environment, addressing traffic models on a high levels of complexity to simulate realistic traffic scenarios and realistic driving behavior.

We will see later on how the increasing complexity and difficulties and so the migration on simulated scenarios has raised the need of more powerful tools in order to reach every single request in an efficient way; that is how we will introduce the study of **Gemv²** work.

3.2 Routing Protocols

A routing protocol is obviously required in VANET as well as any other network in order to forward data on a pre-selected route and also to maintain that route while transmitting and then recovering it when errors occur.

The routing protocols are classified as proactive, reactive and hybrid [22]. VANET makes use of a mix of these 3 routing protocols, each of them with a different task to perform and, in the specific, it uses **[a]** Proactive routing protocols to maintain and update information on routing between nodes, **[b,c]** Reactive routing protocols to process route request.

In the detail:

- **OLSR** (Optimized Link State Routing Protocol): it is an optimized protocol for mobile networks applicable to vehicular communication where it is in charge of the exchanging of the topology information with other nodes in the network acquiring firmness on the link state algorithm so that the route will be available immediately when needed. Control messages are forwarded by those nodes designated as multi point relay in the network, comprising also traffic congestion in the process.
- **AODV** (Ad-hoc On Demand Distance Vector): after a on-demand definition of the correct path, the source node can send data do the destination node passing through each connected nodes in between; the source node will broadcast a Route Request to find a route to the destination node that, on the contrary, will send a Route Reply when receiving the packet. The protocol will also maintain route information helping the broadcast of the messages when a node will need the same route, or part of it, again.
- **DSR** (Dynamic Source Routing): sending of the data when needed by first applying for route discovery and route reply using an unique IDs and occasionally sending a Route Error if needed. The difference with the previous protocol is that here it is used by VANET to maintain network information to be sent to roadside unit regarding traffic density.

It still holds the paradigms of the IEEE 802.11p as routing layer of the network.

3.3 RSUs Deployment

When considering a vehicular scenario, not only vehicles and pedestrian's devices play an important role in the transmission of the data, but sensors are installed in several other instruments alongside streets and roads taken by drivers; there are sensors on traffic lights, street lights and along the highway.

A **Road Side Unit (RSU)** is a DSRC (Dedicated Short Range Communications) transceiver that is mounted along a road or pedestrian passageway [23], with the aim of collect real-time data from moving vehicles and then transmit it to traffic information centres for further analysis [17].

Each RSU is connected to several devices in range (see fig. 3.5) to which transmits information regarding traffic conditions, safety warnings messages, weather conditions and so on with many and many other data. Here in this work, only this general main aspect is treated.

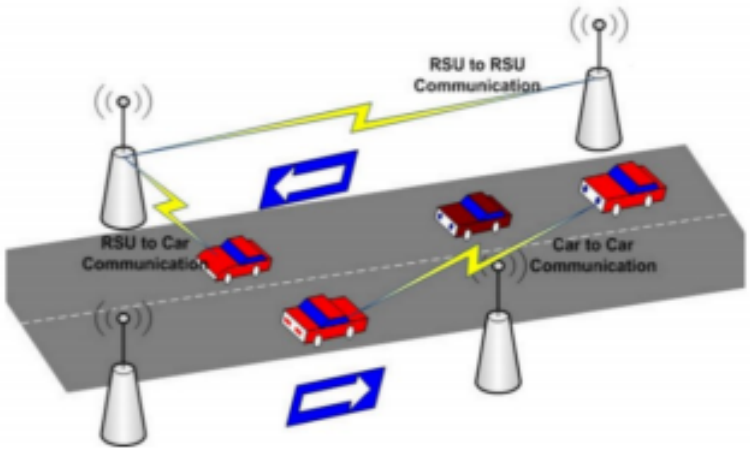


Figure 3.5: Implementation of RSUs in a VANET network.

3.4 Vehicle-to-Everything Communication Protocol

Along with the great growth of ITS and its application, it raised the necessity to develop a specific standard for vehicular transmission, especially when talking about the communication protocol to be involved. The key word is "standard" meaning that every partner needs the same standard paradigm to be able to communicate with all the other devices.

A specific WiFi mode (from the family standard of the IEEE 802.11p basis) operating in the 5.9 GHz frequency band, enables ad hoc communication and the direct exchange of information among vehicles in their vicinity, including the communication between vehicles and the roadside infrastructure (or unit). This approach is commonly referred to as **Vehicle-to-Any or Vehicle-to-Everything (V2X)** communication [24].

V2X communication protocol has four main subdivision (see fig. 3.6):

- **Vehicle-to-Vehicle (V2V)**
- **Vehicle-to-Infrastructure (V2I)**
- **Vehicle-to-Pedestrian (V2P)**
- **Vehicle-to-Network (V2N)**

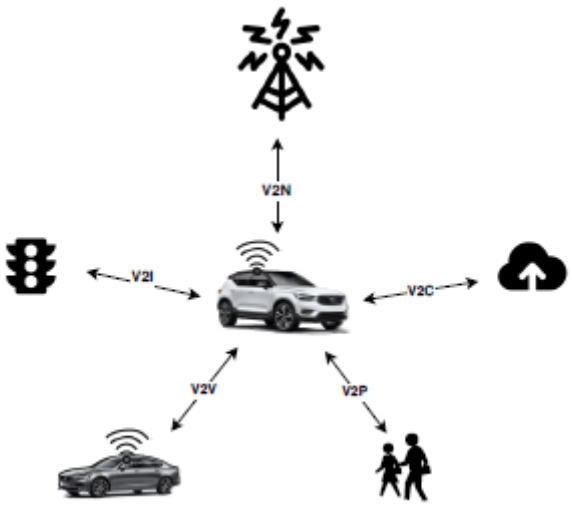


Figure 3.6: Vehicle connections in the V2X paradigm.

The division is self-explanating and it is based on the type of device to which the communicating vehicle is linked to. V2X communication enables a wide range of applications [24].

At the beginning, they were categorized into four groups:

(a) Active road safety

(b) Cooperative traffic efficiency

(c) Co-operative local services

(d) Global Internet services

In all these applications it is assumed that the duration of time for the driver to react is at most in the range of 1 second, so the applications were tolerant to packet loss since the same information is repeatedly transmitted by other vehicles in a redundant way. After this first deployment, connected driving has received great attention in industry and the technology saw a great expansion, bringing new fields of application:

(e) Assistance to driver

(f) Automated driving

(g) Cooperative maneuvering with safety margin

(h) Sharing of the planned trajectory

(i) Help in driving tasks such as steering, accelerating and decelerating

V2X works in a fully distributed way meaning that there is no need for a solid network behind it and there is no need for coordination with an infrastructure; data is directly exchanged among the communicating vehicles in range with a very small delay and a high data rate.

VANET makes use only of the V2V and V2I paradigms while the principles of V2P are mostly used by other technologies. V2V communication is by far the most complex to implement [25] due to the high mobility given by each nodes involved, thus vehicles; while when communicating with an infrastructure or pedestrian the mobility is only to be considered for the source node (that still it is a vehicle), in V2V also the receiving nodes involve a scenario that is continuously changing with the movement of the vehicle. In this case, communication can be either direct from source to destination in a single-hop "network", or can serve intermediate nodes on its path in a multi-hops "network"; from multi-hop comes the difficulty of the implementation: there is a need for an efficient routing protocol.

In general, there are some strong features that are always to be considered when working on a V2V scenario:

(a) the high mobility of vehicles

(b) the very high dynamicity of connections

(c) the reduced duration of communication links

(d) the congestion of the communication channels during packet relaying and retransmissions

(e) the unpredictable characteristics of the environment (interference, obstacles, other vehicles, tunnels, ...)

(f) the embedded energy capacity

3.4.1 LTE and 5G - V2X Implementations

With the growth of the already existing technologies and the spreading of the new ones, also vehicular communication had been adapted and incorporated with the cellular systems to better use all their and its potential. VANET and its paradigm of V2X has been adapted to work in cohesion with the **Long Term Evolution (LTE)** system enabling a fast deployments of all its services; not only that, the **Third Generation Partnership Project (3GPP)**, which is in intended to encompass all the different features in the same usable way, has made a specific standard for LTE-based V2X services and, from 2016, it is continuously expanding the properties of LTE systems to give a better and better experience to the end user. It took the name **LTE eV2X** [26].

The cooperation with the automotive industry led the LTE system to work with New Radio (NR) solutions, creating a new branch of V2X-based solutions called **NR-V2X** that complement the **LTE eV2X service**.

The cooperation of the two technologies gave the basis for some vehicle platooning applications and, most important, for limited automated driving applications. This first approach has nowadays a strong implementation in V2X services and the combination of these technologies with the 5G networks leads to faster application services, better communication protocols, fully automated driving applications, better support to connected mobility and road safety applications, in general it fulfills the concepts of smart cities and intelligent transportation. The requirements fully satisfies the needs of VANET networks for a stringent reliability, low latency, high data rates and larger communication ranges even in a high traffic density scenario, handling the rapid mobility of vehicles and the required QoS for each application service.

In VANET, 5G technology is usually implemented first to establish the Vehicle-to-Infrastructure (V2I) and Vehicle-to-Pedestrian (V2P) connections (see fig. 3.7). In an urban environment, it is reasonable to have a big number of pedestrians w.r.t. RSU giving the system the possibility to choose the best device for the transmission. The pedestrian's mobiles will be the gateways of the connection and choosing the best one will improve the data transmission efficiency.

To support dynamic network management mechanism, a cluster is formed before starting the transmission in order to prevent non-necessary connection outside the usefull range of connection.

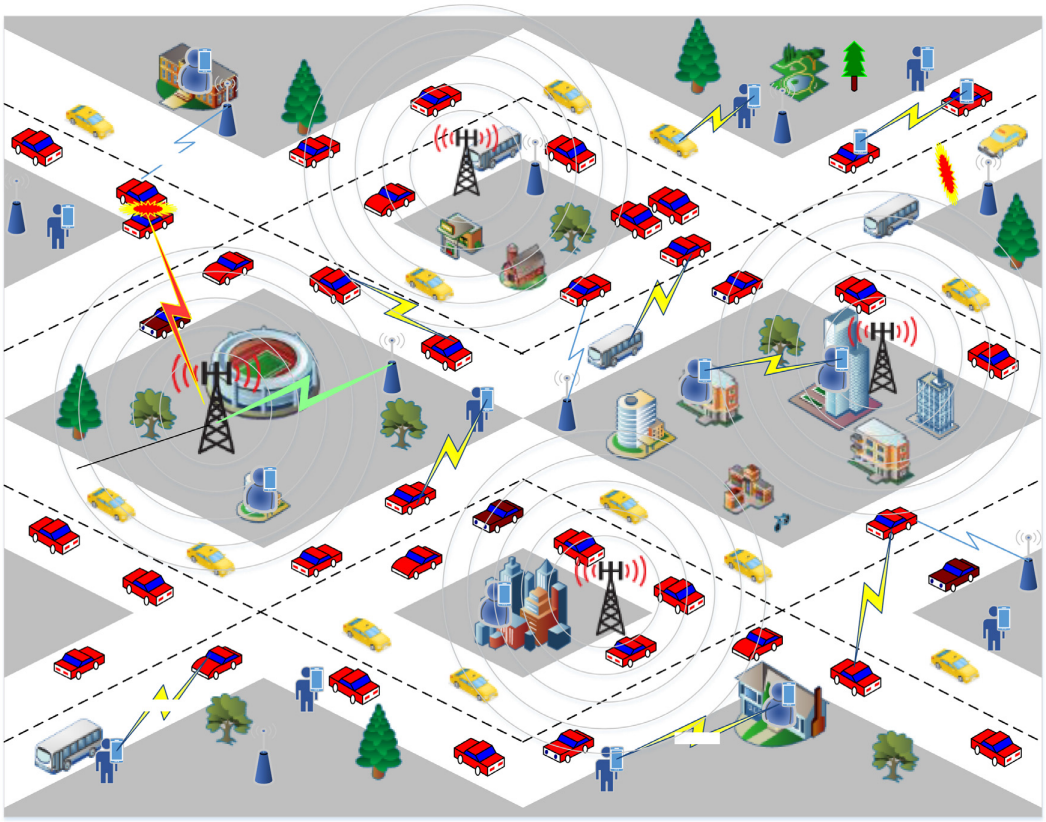


Figure 3.7: Cellular-5G VANET model.

Once the clusters are formed, both V2V and D2D mechanism can be used to achieve the communication goals and to reach the exact messages dissemination it is needed in the simulation.

3.5 Internet of Vehicles

The most dominant issue when dealing with the exponential growth of vehicles is the rising number of road accidents that can occur everyday. We have seen that with the spreading technologies that are rising nowadays, there are plenty of methods to overcome to the problem assuring efficiency when dealing with road security but, of course, everything is running at the speed of light and the technologies need an update as soon as it is possible.

In this fast scenario, the concept of **Internet of Vehicles (IoV)** came into being. The conventional VANET technology is evolved to the Internet of Vehicles in a way to do not only provide traffic security while driving, but as a system where each vehicle is seen as an intelligent object, equipped with sensors, computing facilities, control units and storage units and is connected to any entity (other vehicles, RSUs, stations, cloud, etc..) by exploiting the concept of the Vehicle-to-everything (V2X) communications [27].

IoV is thus a dynamic mobile communication system between vehicles and resources with high controllability, operability and credibility which is expected to provide real-time road traffic information, protect the travel comfort, and improve the transportation efficiency [28].

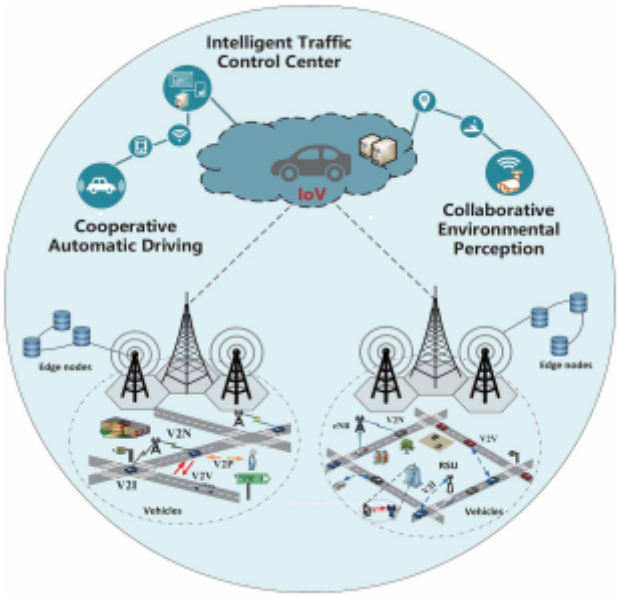


Figure 3.8: IoV illustration.

The deployment of the IoV will lead the technology of VANET a step further, closer to the implementation with the cellular 5G technology allowing more advanced applications to optimize traffic control, sharing more exact information quickly, enabling assisted and autonomous driving, and many more. Each vehicle will generate a continuous flood of data in constant growth; not a simple issue to deal with but something to take into account when deploy the methods. IoV is massively working with **Big Data** concept since modern vehicle consumes and generates large amount of data to compute the information needed. Not merely this way to see the view, but reversely also the fact that nowadays Big Data are easily accessible and analysable is something that can help in the spreading of the IoV concept.

The major challenge that IoV has to go through with Big Data (see fig. 3.9) is that the information could arrive from multiple sources and at different time each to another. For example, the safety data, such as motion status or brake alert, should be transmitted from one vehicle to its neighbors with stringent low packet loss rate and end-to-end transmission delay, while the infotainment data could be transmitted from vehicles to the RSUs connected with a longer delay tolerance [29].

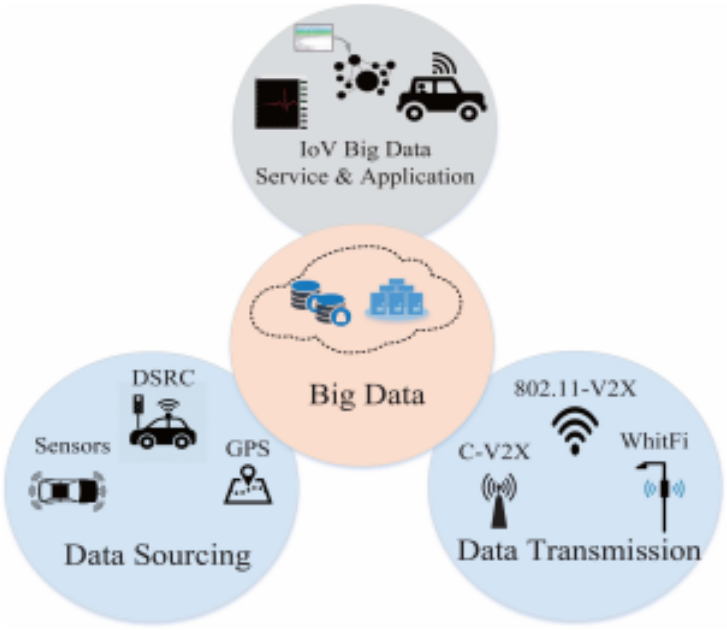


Figure 3.9: Mutual relationship between IoV and big data.

Moreover, we should remember that data are coming from moving vehicles and are forwarded to moving vehicles, without any (possibly) loss and information integrity in the transmission. The coverage scale of Iov is therefore addressed to the large propagation range and to the high signal penetration capability in order to reach

each device in a wide area. The data may (most of the times) require different QoS while transmitted, meaning different propagation delays and transmission data rates, with a need of massive management to be implemented. In general, we can say that IoV requires high bandwidth to be able to implement all the services it is mention to support.

Furthermore, each vehicle will be a kind of server for people to implement the parallel technology of the **Internet-of-Things (IoT)**, whose devices on the road are increasing exponentially in the last years. It is no more a secret that IoV and IoT will be the technologies that will dominate the market in the coming years and their cooperation will support any type of industry that based its expansion on communications.

3.6 Summary VANET Technology

In the scheme below are shown all the technologies mentioned in chapter 3:

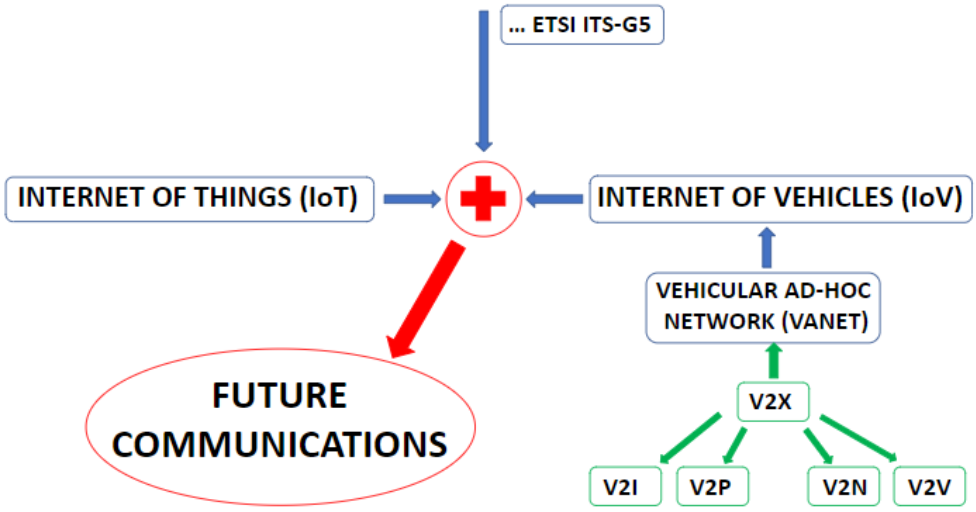


Figure 3.10: Scheme 2: VANET technology.

Chapter 4

Geometry-based Efficient Propagation Model for Vehicle-to-Vehicle Communication

4.1 Geometry-based Models

When dealing with mathematical problems there could be the need for a method that uses shapes to better understand the elements that are involved in the situation. Geometric modeling is that part of math which making use of algorithms for the mathematical description of these elements, which are indeed any kind of shapes necessary for the computational counts. The shapes used are two-dimensional (in technical drawing) or three-dimensional (in computer design SW), but a general definition can be seen as whichever volumetric model with finite dimension able to describe the model in the best possible way.

A geometry-based model can be adapted also in the vehicular communication by defining those shapes similar to real-world scenario: we are talking about the communicating nodes taken at a certain moment, i.e. communicating vehicles; we are talking about all types of obstructions that can occur during communication, whether they other vehicles, buildings or foliage on the streets simulating traffic conditions.

The geometry-based model used in our scenario takes into account the vehicle-to-vehicle communication and fits perfectly with the use of VANET as the network topology at the basement.

It is the **Geometry-based Efficient propagation Model for Vehicle-to Vehicle communication (Gemv^2)**.

Before entering in the detail of the tool, it is better to understand the reasons why VANET needs this implementation to work.

We have said VANET facilitates the development of the Intelligent Transport System and its role in supporting the Internet of Vehicle's services to which VANET is evolving to; it is nowadays fully integrated with the 5G technology, it takes advantages from Device-to-Device paradigm to reach the correct Safety Message dissemination in the scenario, making use of the V2X (focus on V2V and V2I) as the communication method for the purpose.

So **why there is a need for something different to fully exploit the advantages of such network?** First, it has to be considered the high cost of deployment for a traffic scenario that involve a massive use of vehicles and objects in the simulation; then, there is no account for specific link conditions, let's say that the concept of obstructing/non-obstructing objects in the path is only consider in the theory, thus not all the traffic conditions are really take into account. Last, the already existing models currently used in VANET cannot utilize geographic descriptors to enable specific features of the V2V communication. These are the reasons why to fully simulate a real traffic scenario, VANET and GEMV^2 are used in cohesion on a SW simulator (OMNeT++, explained later on).

4.2 Tool Features

When dealing with Gemv^2 [30], there are some features to take into account:

- it models the outlines of objects in the model and use them to differentiate traffic condition based on a link classification: **Line-of-sight (LOS)** and **Non-LOS (NLOS)** are introduced;
- calculates the signal variations in the communication capturing both small-scale and large-scale signal variations (due to the high dynamic nature oh vehicular simulation) and using different propagation methods, such as Path-loss model, diffraction/reflection characteristics, shadowing or fading communication;
- it considers different urban scenarios and environments: urban, suburban, highway, open space, etc..
- it uses external tools to simulate real-world scenarios and GPS locations of vehicles: **SUMO** and **OpenStreetMap** software;
- it stores the outlines of the shapes used, being them vehicles, buildings, foliage, etc.. in a custom structure: the **R-Trees** structure;

- it exports the visualization of the simulation to KML format, CVS file (a format similar to the Excel file) or SQL database;
- it can be used in MATLAB or OMNeT++ software, depending on the operating system (the OMNeT++ implementation works better under a Linux environment).

4.3 Path Loss Type: Free Space and Obstructed View

When simulating a real world scenario it has to be taken into account the environment in which vehicles are running, and so the scenario where we are working on. Since vehicles have relatively low heights of the antennas mounted on them (it cannot go so high even when considering a van or a truck), it is reasonable to say that the connection between the communicating vehicles will be obstructed by obstacles, either static (buildings, hills, foliage) or mobile (other vehicles on the road) [31].

It is reasonable to say that a significant portion of the communication will be lost and it is important to create a methodology to better understand this impact in the transmission and how it makes the signal worst than the original.

The concept of Line-of-sight (LOS) and Non-Line-of-sight (NLOS) is introduced. A LOS condition occurs when the two communicating vehicles have no obstacles between them, thanks to the fact that most probably they are running on the same straight lane meeting each other (reverse driving way) or going in the same direction. A NLOS condition occurs when something it is in between the direct path pf the two communicating vehicles, being it static or dynamic. Of course, during a transmission, a single node (vehicle) can experience both LOS and NLOS conditions multiples time, that's way it is important to introduce an appropriate counts when running a simulation.

Moreover, it is important to distinguish the different "types" of NLOS conditions the vehicles has encountered because each of them has different parameters that are used to make further computations (i.e. scattering, propagation, reflections, etc..). The Non-Line-of-sight is so divided in:

- **NLOSv**: Non-LOS due to the obstruction from other vehicles in the path;
- **NLOSb**: Non-LOS due to buildings in the scenario;
- **NLOSf**: Non-LOS due to the foliage on the path (vegetation such as trees, bushes, shrubbery, etc..).

Another aspect to consider when working on real scenario is the scenario itself, meaning the environment in which the considered vehicle is running; there are scenarios where we will experience an higher counts of vehicles running and obstructing the connection, there are other scenarios where the impact of buildings if more significant w.r.t. to other obstructing objects. And, not negligible feature, different obstacles lead to different behaviour of the transmission being it a direct/reflected/diffracted path. The figure 4.1 shows the conditions offered by Gemv^2 in which we can being when simulating:

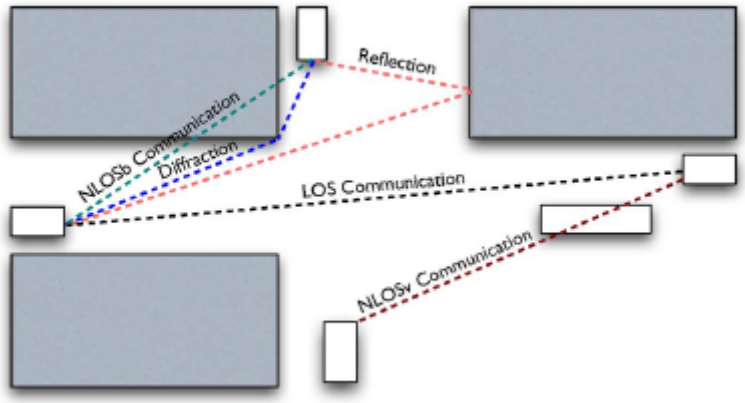


Figure 4.1: Link types and propagation effects captured by Gemv^2. White rectangles represent vehicles, gray rectangles represent buildings.

We can mainly divide the distinguishing in four areas [31]:

- **Highway**: only NLOSv is considered in a high mobility environment (obstructing vehicles are moving);
- **Parking lot**: again, only NLOSv is considered but here obstructing vehicles are mostly still;
- **Suburban area**: wide streets are typically lined with small buildings and trees, with some occasional crests or blind corners;
- **Urban canyon**: streets cut through dense blocks of tall buildings that affect the transmission.

When running a simulation, sometime also the Time of the day is taken into account; in fact, the same scenario gives different LOS or NLOS conditions if evaluated in rush hours or night hours.

4.4 Summary Gemv^2 Technology

In the scheme below are shown all the technologies mentioned in chapter 4:

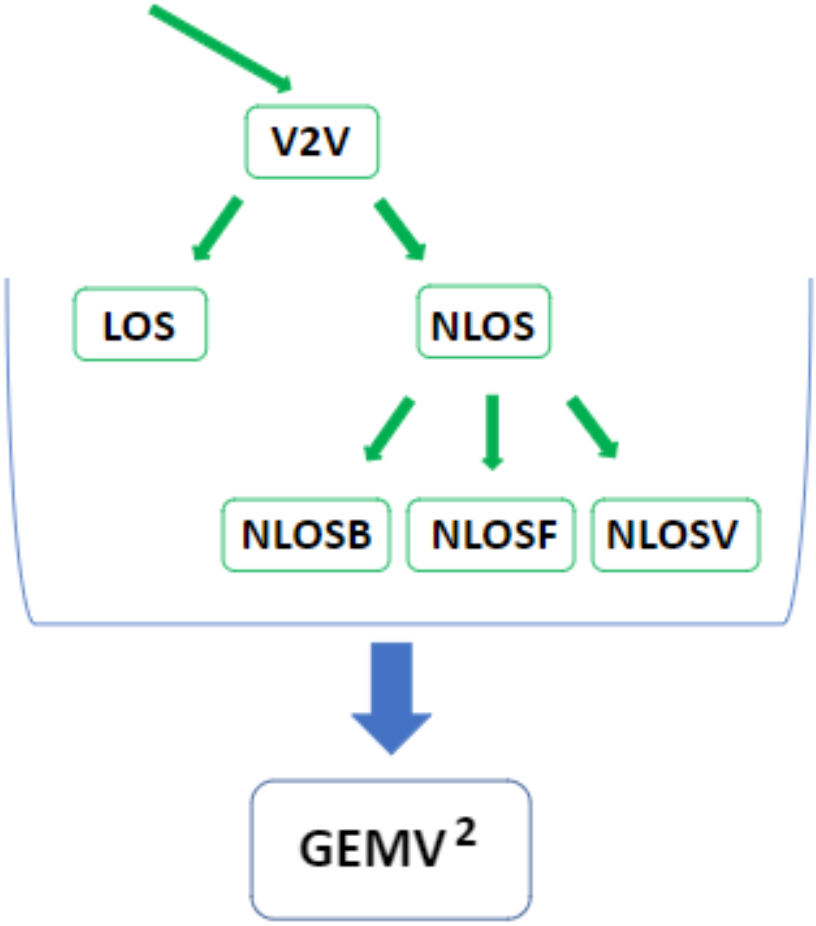


Figure 4.2: Scheme 3: Gemv^2 technology.

Chapter 5

Frameworks and Tools Supporting the Discrete Event Simulator

In this chapter it is shown the whole set of Frameworks and Tools supporting the simulation of VANET and its expansion using Gemv^2. They are not explained in deep detail since some of them (e.g. OMNeT++) would need pages to fulfill every argument.

The detail about how to install the every mentioned tools to be used in the simulation to run correctly are shown in the appendix at the end of the work.

5.1 OMNeT++

OMNeT++ is a discrete event network simulator software providing to the user all the modules necessary to compose a real world scenario [32].

Tools and modules available from OMNeT++ are coded in C++ language and the user can add his own features if a new implementation from an already existing one is needed. Different modules, or entities (users, packets, vehicles in this work) communicate with each other through a message exchange system, the basic principle of the simulator. A complete suite of libraries is given to the user to perform simulations in different environments and conditions.

The main areas where OMNeT++ is used include: network protocols, queuing systems, multiprocessor systems with parallel tasks, evaluation of the performance of a complex system such that of vehicular communication with different agents and objects involved [33].

From here, it starts an overview about the OMNeT++ structure (see fig. 5.1).

5.1.1 Model

An OMNeT++ work is called "project" and every project follows the same model when opened in the simulator:

- **Modules**: each part of the project is a set of different modules with a specific task to be performed.
  (a) Simple module: active component of the model that perform one or more specific tasks;
  (b) Compound module: a set of Simple Modules performing more complex tasks.
- **Messages**: the way how modules communicate each other;
- **Connections**: link from one module to another used for the transmission of messages;
- **Gates**: doors from where a connection is established;
- **Network**: the series of Simple and Compound Modules of the project.

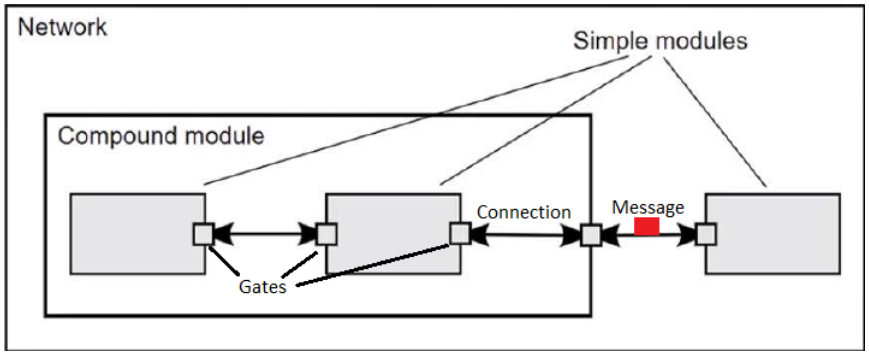


Figure 5.1: OMNeT++ model.

5.1.2 Structure

As done for the model, also the structure of the project has a fixed definition for each simulation. In order to run a simulation, these components must always be present (see fig. 5.2):

Network Description (NED) file
Definition of the network and of the modules (both simple and compounds), set up and managing of the connections between modules, initialization of the module's gates. This can be done through a graphical view or acting directly on the C++ source code.

Source (.cc) file
The source file contains all the logic of the project. In particular, it defines the classes used by the modules initializing the functions that will be called by them. It imports libraries available from the **Header (.h) file**. Moreover, it defines and initializes two fundamental methods for the exchange of messages: the "initialize" method to create the message and perform actions on it and the "handle" method to manage the message during its lifecycle.

Network Configuration (.INI) file
Sets parameters and configures the behaviour of the simulation, both for single and common variables. It sets up the parameters used by the modules (size, range, position, etc..), sets up of the simulation parameters (number of repetition of an event, timer, etc..), enable/disable of the records for what concerning the collected statistics and results of the simulation.

The combination of these files lead to a complete project that can now be run.

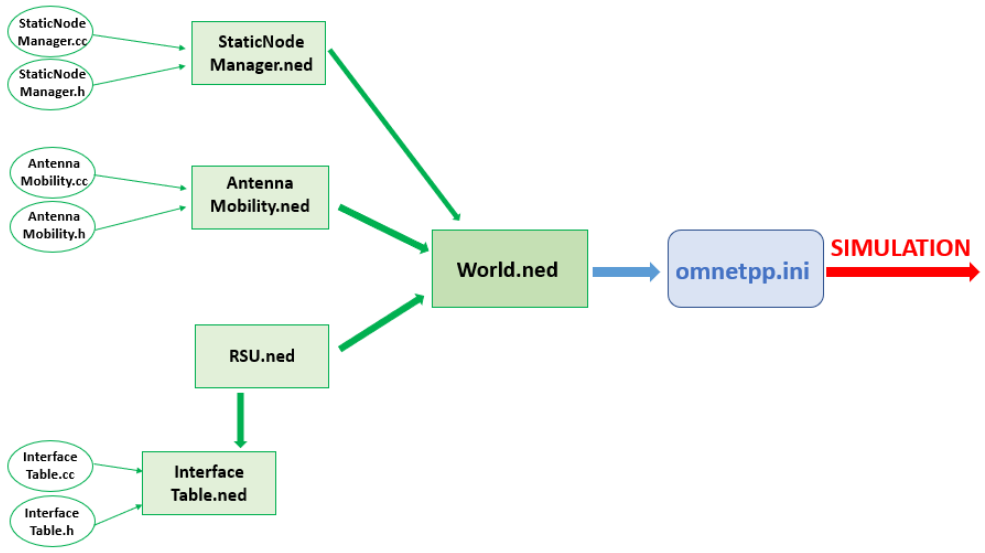


Figure 5.2: Relations of the files in OMNeT++. The example comes directly from the files used in this work.

5.1.3 Results

At the end of the simulation it is possible to analyse the results that have been stored [34][35].

5.1.3.1 What it is stored

The system is able to record vector or scalar values by default, then also histograms can be recorded if enabled in the .INI file.

- **Output vectors** are time series data, recorded from modules or channels. They are used to record parameters such as delays, times related to packets or queues, states, packet drops, etc..;
- **Output scalars** are summary results, integer or real numbers, representing summary statistics as counts, maximums/minimums, standard deviations, sums, etc..;
- **Output histograms** are recorded as collection of more scalar results. Most of the time, rates are recorded here.

5.1.3.2 How to record results

There are essentially two possible ways to record results:

Declared statistics
This method combines the signal mechanism and the NED properties. Statistics about what it must be recorded are declared in the NED files with @statistic property, then the correct module emits a @signal that is heard by the statistic and then uploaded with the value.

Direct result recording
This other method needs the user knowledge of the C++ language since it acts directly on the code using the simulation libraries. Statistic results are collected in class variables inside modules, then a call to the functions is performed invoking the recordScalar() or recordVector() functions; using the classes cStdDev it is possible to store summary statistics like mean, standard deviation, min/max, etc.. while with other histogram-like classes (cHistogram, cPSquare, cKSplit, etc..) it is possible to record distributions of values.

5.1.3.3 Enable/disable the records

Now that the values are recorded, the user can decide to see it or not via the enable/disable functions from the .INI file.

(a) All recording from a @statistic can be enabled/disabled together using the statistic-recording option;

(b) Recording of a scalar or a statistic object can be controlled with the scalar-recording option;

(c) Recording of an output vector can be controlled with the vector-recording option;

(d) Recording of the bins of a histogram object can be controlled with the bin-recording option.

The method to enable/disable would be:
module-path.statistic-name.statistic/scalar/vector/bin-recording = true/false

Example of usage:
***.queueLength:max.scalar-recording = true*

Another method is to tell the system which modes must be recorded, where "modes"

could be all (count, mean, max, vector), - (none of that), or a combination using - or + (-vector, +histogram).

Example:
***.result-recording-modes = all, -vector, +histogram*

The collected results are stored in the Results folder. OMNeT++ provides a graphical interface for the analysis of such results:

Result Analysis (.ANF) file
Default results can be retrieved in scalar (.sca) and vector (.vec) files, but with some C++ command lines it is possible to export them in a tabular file (.csv) or in a SQLdatabase. Besides scalar and vector files, also a log file can be created containing all the significant events occur during the simulation.

Opening the scalar and vector files simultaneously, the .anf file is created. Here it is possible to act on the recorded values defined by the signals emitted and the related collected statistics. Through the **Browse Data** button it can be shown the vectorial and scalar data recorded, as well as the histogram data from which it is possible to plot the results on a graph. Through the **Dataset** button it is possible to create pre-defined functions to work on the data and create extra information coming from an equation or a combination of different data.

5.2 Artery

Before introducing Artery, some words have to be spent about "Veins".

Vehicles in Network Simulation (Veins) is an open source network simulation framework specifically created for vehicular communication, giving simulation models and tools to be executed on an event-based network simulator (e.g. OMNeT++), while interacting with a road traffic simulator (e.g. SUMO), taking care of setting up, running, and monitoring of the simulation [36] (see fig. 5.3).

Veins is designed to serve as an execution environment for user written code, which typically includes an application, created for the needs of a simulation; the framework is in charge of modeling lower protocol layers and node mobility, taking care of the setting up of the simulation, collecting results during and after the simulation. It contains a lot of pre-defined simulation models that can be used on a simulation or not, it is not mandatory to use them all at the same time.

As an extension of Veins, in 2014 **Artery** born.

Veins put emphasis on WAVE protocol, which is the U.S. counterpart of the ITS-G5 developed in Europe (both coming from the IEEE 802.11p project). Starting from this division and from the fact that Artery framework works primarily on European standard, Artery took the name just as an association of names between the veins and the arteries of a human body.

As a general definition, Artery enables V2X simulations based on ETSI ITS-G5 protocols. Single vehicles can be equipped with multiple ITS-G5 services through Artery's middleware, which also provides common facilities for these services. Some basic services, such as Cooperative Awareness (CAMs) and Decentralized Notification (DENMs), are already included [37].

Even if Artery is an extension of Veins, in the years it is grown as a separate framework leaving the necessity to be added to a complete Veins environment to run. Nowadays, Artery can be run without Veins when the radio model from INET framework is emplied as explained later on. It is possible to install Artery framework on a network simulator, OMNeT++ in this work, and let it run with the external extensions of INET and others directly without the need to call them every time a simulation is performed. Further details on how to install it can be found in the Appendix.

5.2.1 Features

Artery framework [38] enables the V2X mechanisms in a simulation environment, the Cellular-V2X protocols and the V2X applications, providing the communication links, the interfaces for the nodes, the message communication mechanisms, the set of features to work with 5G mobile networks and the instruments to pass from one cellular mode to another (mode 3 and mode 4 of the 3GPP standards) dynamically. It provides a complete set of components to integrate the different part of a vehicular transmission, such as vehicle mobility, the environment in which vehicles are running, the radio environment for the communications, the V2X services and communication protocols with a strong separation between the facilities. It also gives support for those features that have to be implemented with an external tools (e.g. those coming from INET framework) without any notices for the user that has to be do nothing to use them.

Artery supports three modes in a common scenario, switching from one to another in a dynamic way: (i) uplink/downlink, (ii) network-assisted sidelink mode 3 and (iii) out-of-coverage sidelink with distributed resource allocation and management mode 4. In this way, the simulation takes a more realistic condition of vehicle mobility, using both up/downlink transmission in a scenario where it can go from network coverage (mode 3) to out-of-coverage (mode 4), supporting also the network-assisted D2D when needed.

Artery gives modularity to its layers, which are clearly separated each other and communicate using an exchange of messages mechanism:

(a) The user-plane layers are implemented as simple modules and encapsulated into compound modules in OMNeT++, called Network Interface Card (NIC);

(b) The control-plane acts as an independent layer that communicates through messages;

(c) Traffic model scenario of V2X (i.e. V2V as major paradigm used here but also the others) are developed through the use of road traffic simulator such as SUMO, constructed inside the simulator (OMNeT++);

(d) The link path that a vehicle experiences is given by Gemv^2, implemented in Artery as a separate environment can be run autonomously;

(e) The message mechanism used is the already cited CAM and DENM system;

(f) The support for an heterogeneous traffic enables both IP-based and non-IP-based transmissions;

(g) The conventional scheduling schemes are implemented: Round Robin, Deficit Round Robin, Maximum Carrier over Interference, Proportional Fair queuing.

The implementation design consists of two floors: on the lower floor, there are the Cellular-V2X protocols including the **Access Modules**, the **Control Plane** with

the Radio Resource Control **(RRC)**, the **Mode-switching Control Module**, the **User Plane**, the Physical Module Component **(PHY)** (information about antenna power, channels, cells, etc..), the Medium Access Control Module **(MAC)**, the Packet Data and Convergence Protocol **(PDCP)** (to connect components between Cellular-V2X and network) and the Radio Link Control **(RLC)** (for the multiplexing/de-multiplexing of MAC units). On the upper floor is used to generate and receive V2X messages and make available the usage of the ITS facilities.

5.2.2 Architecture

Running one of Artery's tool (i.e. Gemv^2 in this work), two programs are launched simultaneously working hand-in-hand during the simulation [2].

On one side is the (vehicle) traffic simulator, thus SUMO. It reads a configuration file at start, which refers to XML files defining the road network, the traffic demand (where and how many vehicles are on their way etc..), and optionally miscellaneous polygons describing the outline of building or vegetation.

On the other side is the event simulator giving the environment where the simulation is running, thus OMNeT++. It executes a target file (run-gemv2), it launches the TraCI manager responsible for launching the SUMO process and connecting the traffic command interface via TCP. Then, Artery retrieves the mobility parameters of SUMO vehicles via this TraCI connection for each SUMO simulation step.

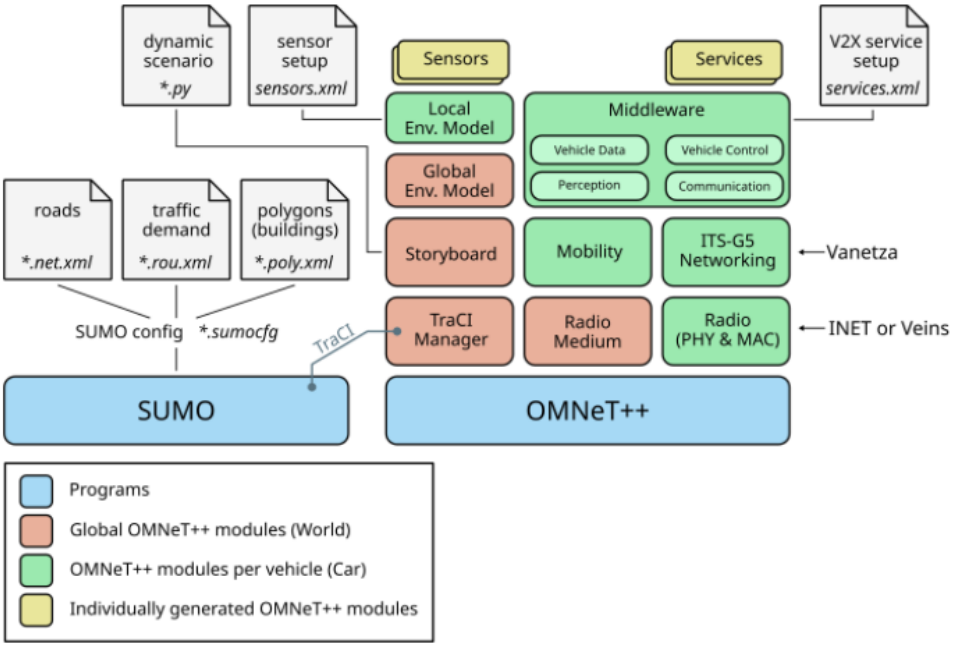


Figure 5.3: Artery architecture [2].

5.2.3 Middleware

The Middleware can be described as a central hub module mounted on each vehicle in charge of provide V2X services (applications) dynamically during the simulation, such as giving the lifecycle of V2X modules, forwarding of V2X messages from lower layers, sharing facilities among services thus allowing one module to get access in another module [39]. The services are created according to an XML configuration file.

One important task of the Middleware is to store a so call "LocalDynamicMap" for saving the received CAMs messages in order to be able to forward them again in the case a new station come in range during the simulation right after the previous sending station (the one which first transmitted the message) has already gone at the time the new station come.

Another important task is to give access to vehicle data such as position, speed, etc.. or to compute an hypothetical path to provide through the VehicleDataProvider.

5.2.4 Environment Model

Each vehicle in the simulation is equipped with sensors attached to the front, rear, left or right of its body which are in charge of understand how many and which other vehicles are in the range and in the field-of-view for a transmission. Ranges and field-of-views can be configured individually, but usually they are defined in the path-loss file inside Gemv^2 link type's files; buildings and other vehicles can block the line-of-sights [40].

In the figure 5.4 we can see vehicles running in the environment filled with blue color; each vehicle has a cone which define the range on transmission, while line-of-sights are drawn as dashed lines in the same colour as the corresponding sensor cone. Static objects such as buildings and foliage (not present here) are coloured in green.

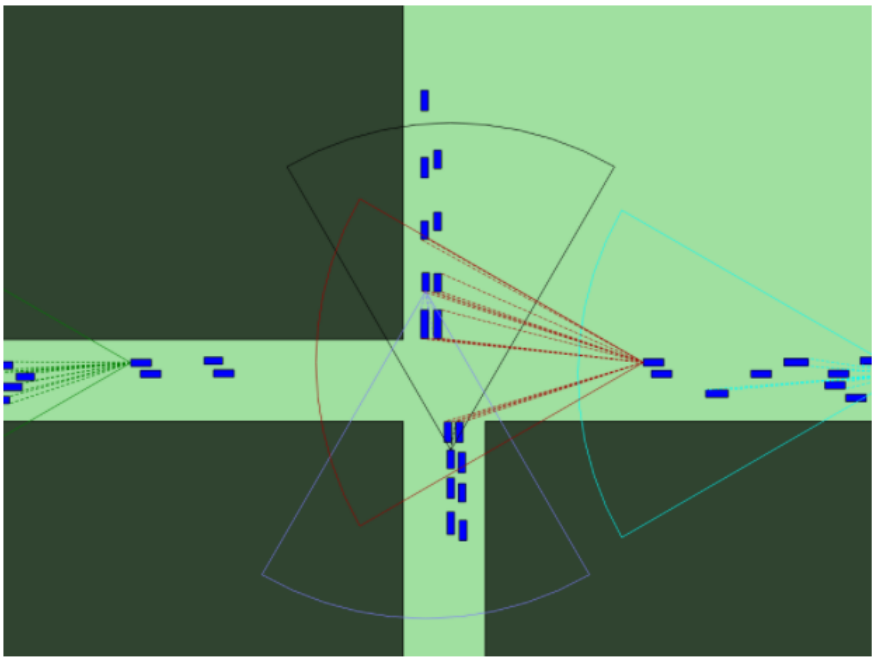


Figure 5.4: Example of simulation running. Each vehicle has its range of transmission (visualized as a cone) and the field-of-view (visualized as dash lines).

Each vehicle has its LocalEnvironmentModel, which is the vehicle's local perspective on the environment. The local environment model is composed of a set of sensors where filter rules are defined by XML files similarly to what happen in the Middleware.

5.2.5 Storyboard

The Storyboard module is necessary due to the dynamic behaviour of vehicles during the simulation: traffic density varies, traffic jams may build up, weather conditions affect the speeds, vehicles run in different direction, accidents can occur and SUMO has no notification of such changes; all these changes are modeled in the Storyboard [41]. Multiple stories can be registered, each consisting of different conditions and related effects, alone or as a combinations of different events.

5.2.6 Gemv^2 in Artery

The C++ implementation of Gemv^2 has been written from scratch by Thiago C. Vieira (Universidade Federal do Parana - UFPR) and Raphael Riebl (Technische Hochschule Ingolstadt - THI). However, Mate Boban's Matlab code and Artery's

C++ version share some ideas, such as the usage of R-trees. While the Matlab implementation computes received power for each communication pair per time step at once, in Artery the signal attenuation per transmission is computed by implementing INET's IPathLoss interface. Hence, you can only use Gemv^2 with the INET radio model at the moment [42].

5.3 Inet

INET Framework is a simulator specifically designed for communication networks.

INET is a set of protocols and models useful when there is the need to try an already existing network or a new one, maybe trying new paradigms and scenarios. It provides models for network layer (TCP, UDP, IPv4, IPv6, OSPF, etc..), wired and wireless link layer protocols (Ethernet, PPP, IEEE 802.11, etc..), support for mobility, MANET protocols, DiffServ, MPLS with LDP and RSVP-TE signalling, application models, and many others [43]. Its primary concept is the communication between modules by messages. Its single components are combined to form more complex modules working together, that could be hosts for the user or routers, switches and network devices for the network itself.

INET has a strong base for the development of LTE and vehicular networks, and that is how it is used in the OMNeT++ simulator: as an open source model library supporting the simulation environment. Moreover, it makes use of the services provided by OMNeT++ for its purpose, meaning that it is possible to evaluate the results coming from INET's components directly in the OMNeT++ environment, using the graphical or command-line view.

5.3.1 Features

Here are shown some of the major INET's features that can be used while working with the framework. Then, some of the modules (packages) mostly used in this work are explained more in detail (references lead to the specific pages in the INET framework site).

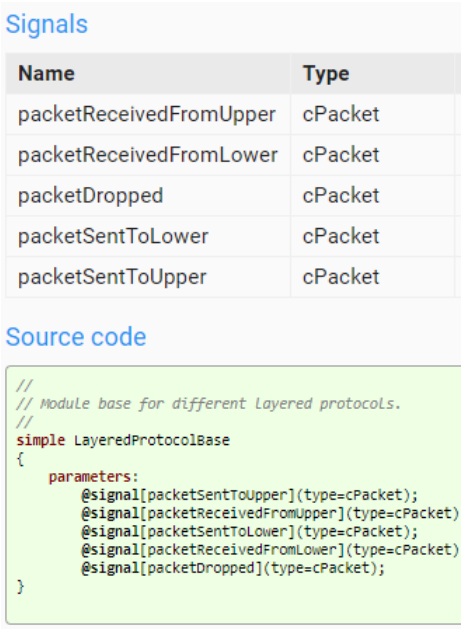
- **OSI layers**: physical (scalable level of detail), link, network (IPv4/IPv6), transport (TCP, UDP, SCTP), application models;
- **Additional plug-ins** for further protocol implementation;
- **Routing protocols**: wired/wireless and ad-hoc;
- **Wired/Wireless interfaces**: Ethernet, PPP, IEEE 802.11, etc..;
- **Mobility** support;
- **Network emulation** support;
- **Visualization** support;
- **Physical environment modeling** support.

Each feature has several packages in charge of implement it. There are packages for the specifics required by the protocol, packages implementing rules to be applied, packages that link different part of the same module or that link several module, packages which develop the interfaces, etc..

5.3.1.1 Layered Protocol Base

This module takes an important role since it is where a first definition of the signals (the method used to register significant events and then record them) is placed (see fig.5.5). Here, the mechanism of packet "sent-to-upper" and "sent-to-lower" (layers) is mentioned, as well as packet "received" and packet "drop" [44].

A more precise definition of what signals are and how to use them as well as which signals are recorded in this work to analyze them as results of the simulation will be provided later on in the simulation chapter.



Signals

Name	Type
packetReceivedFromUpper	cPacket
packetReceivedFromLower	cPacket
packetDropped	cPacket
packetSentToLower	cPacket
packetSentToUpper	cPacket

Source code

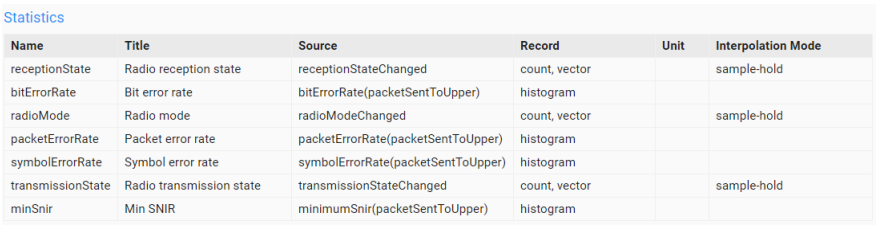
```
//
// Module base for different layered protocols.
//
simple LayeredProtocolBase
{
    parameters:
        @signal[packetSentToUpper](type=cPacket);
        @signal[packetReceivedFromUpper](type=cPacket);
        @signal[packetSentToLower](type=cPacket);
        @signal[packetReceivedFromLower](type=cPacket);
        @signal[packetDropped](type=cPacket);
}
```

Figure 5.5: INET-LayeredProtocolBase definition of signals.

5.3.1.2 Physical Layer Base

Beside another definition of the signals already mentioned in the previous sub-section, here what it is really interesting is the implementation of the Radio Module and Shortcut-Radio module [45].

The **Radio** module describes the physical device that is capable of transmitting and receiving signals on the medium. It contains the sub-module for the antenna, the transmitter, the receiver and the energy consumer models. It also supports changing in the radio mode (mode 3-4 and viceversa) and transmission power. It has the definition of new signals and related collected statistics to be used in the result analysis [46].



Statistics

Name	Title	Source	Record	Unit	Interpolation Mode
receptionState	Radio reception state	receptionStateChanged	count, vector		sample-hold
bitErrorRate	Bit error rate	bitErrorRate(packetSentToUpper)	histogram		
radioMode	Radio mode	radioModeChanged	count, vector		sample-hold
packetErrorRate	Packet error rate	packetErrorRate(packetSentToUpper)	histogram		
symbolErrorRate	Symbol error rate	symbolErrorRate(packetSentToUpper)	histogram		
transmissionState	Radio transmission state	transmissionStateChanged	count, vector		sample-hold
minSnir	Min SNIR	minimumSnir(packetSentToUpper)	histogram		

Figure 5.6: INET-PhysicalLayer-Radio definition of signals and collected statistics.

The **Shortcut-Radio** module implements a simple shortcut to peer radio protocol that completely bypasses the physical medium, thus it permits to directly sends packets to the other radio module without any physical layer processing in the radio medium.

To be reported is also the module interface **IRadio** in charge of the transmission of frames over a wireless medium. In transmission, upper layers can send frames to the radio module which encapsulates them into signal messages to be distributed to other network nodes in the communication range (another interface communicates the list with the node positions and interference with other nodes); In reception, the received signals are sent to the gate of the radio module and, if they have been received correctly, they are decapsulated and the retrieve frame is sent to the upper layers [47].

5.3.1.3 Mobility Base

Module used to setting up mobility parameters and areas in which the mobility is constrained [48].

Parameters:

(a) position

(b) velocity

(c) speed

(d) acceleration

(e) angular position

(f) angular velocity

(g) angular speed

(h) angular acceleration

5.3.1.4 Antenna Base

Related to the Mobility Module through the Mobility Interface, this module gives all the possible set of antennas the user can use in the simulation [49]. In this work, only **Isotropic** [50] and **Parabolic** [51] antennas are used.

5.3.1.5 Physical Environment

All the elements that affect the integrity of the transmission are listed in this module: objects and surfaces influence propagation, absorption, refraction and reflection of signals [52].

The module defines a set of physical objects which are loaded from an XML configuration file. The elements that compose the objects are:

- **object**: id, name, position, orientation, color, outline width, opacity, texture;
- **shap**e: id, type, size, radius, height, points of the polygons;
- **material**: id, name, resistivity, relative permittivity, relative permeability.

5.3.1.6 Flat Ground

It is the typical view related to the ground environment that is used in a simulation: considering the earth curvature as a flat ground is much easier and best fits with the real scenario where a little portion of city or whatever considered scenario could not be affected by the curvature of the Earth [53].

5.3.1.7 Scenario Manager

Used for setting up and controlling simulation experiments, scheduling events and their time, changing parameter values and/or rates, removing or adding connections, removing or adding routes, etc.. It executes an XML script [54].

5.4 Vanetza

Vanetza is an open-source implementation of the ETSI C-ITS protocol suite [55]. It is basically a series of C++ libraries working both autonomously and together adding features to the simulator that it is used, OMNeT++ in this case.

Vanetza's most important features added are:

- **GeoNetworking (GN)** over IP multicast, among the communication technologies;
- **Basic Transport Protocol (BTP)** as transport layer protocols;
- **Decentralized Congestion Control (DCC)** as traffic management system;
- **Security**;
- **Support for ASN.1 messages**, such as CAM and DENM.

Here are the major features listed in a self-explanating table:

Component	Depends on	Features
access	net	Access layer, helpers for IEEE 802.11 PHY and MAC
asn1	-	Generated code and wrappers for ASN.1 based messages, e.g. CAM and DENM
btp	geonet	Headers and interfaces for BTP transport layer
common	-	General purpose classes used across Vanetza components, including serialization and timing
dcc	access, net	Algorithms for DCC cross-layer
facilities	asn1, geonet, security	Helpers to generate and evaluate ITS messages
geonet	dcc, net, security	GeoNetworking layer featuring geographical routing
gnss	-	Satellite navigation integration for positioning
net	common	Utilities for socket API and packet handling
security	common, net	Security entity to sign and verify packets

Figure 5.7: Components, dependances and features of Vanetza's tools.

The combinations of these major features (and other minors) with the ITS-G5 channel model of VANET (under the IEEE 802.11p paradigm) give how Vanetza is used in the Artery framework.

5.5 SUMO and TraCI

Simulation of Urban MObility (SUMO) is a set of applications to enable traffic simulation of a real world scenario. It is an open source traffic simulation package that helps to investigate several research topics such as route choice, traffic light algorithms, vehicular communications, etc.. allowing the simulation of automatic driving or traffic management strategies [56].

SUMO can generate the network of roads, buildings and traffic density by using an application called "netgen", which it is in charge of create a virtual map, or importing portion or road maps coming from real world (e.g. a city or part of it) with the support of other traffic simulators such as VISUM, MATsim or the most important and famous **OpenStreetMap**.

In SUMO, every object has an identifier helping the visualization of the behaviour during the simulation; in particular, each vehicle has a departure and arrival time, velocity, dimensions, dynamic position, a pre-defined route to follow during the simulation; Also the buildings have their own features as well as the streets which have dimension and traveling speed. The movement on the street is typically provided by the intelligence navigation systems that nowadays are present on every vehicle in use.

In the vehicular communication, each object (vehicle) in SUMO receive extra information from an external communication simulation, the **Traffic Control Interface (TraCI)**, that allows to retrieve values of simulated objects and to manipulate their behavior "on-line". TraCI uses a TCP based client-server architecture to provide access to SUMO [57]. Basically, TraCi gives an architecture to couple two simulators: a road traffic (SUMO) and a network (OMNeT++) simulator. The traffic and network simulators are connected in real time by TraCI thus enabling the control of mobility attributes of each simulated vehicle [58].

In the figure 5.8, an example of possible exchange of messages in the TraCI architecture is shown:

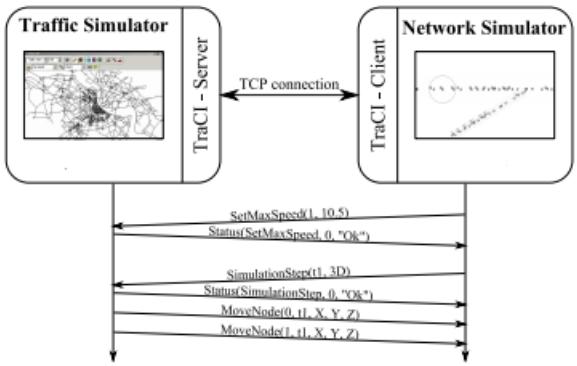


Figure 5.8: The example shows an exchange of TraCI messages.

5.5.1 Gemv^2 in SUMO

In the Gemv^2 folder we can find the files useful to upload the traffic scenario to be launched by SUMO.

Each of the link type proposed by Gemv^2 is composed by one SUMO configuration file and two/three XML files to set up general parameters.

- **LinkType.sumo.cfg**: permits SUMO to load the network and road files for traffic scenario;
- **LinkType.net.xml**: it contains parameters of the vehicles (length, width, speed, shape) and of the street borders (lines) and it creates the links between nodes representing the network in which they are connected;
- **LinkType.rou.xml**: it contains parameters regarding the route edge followed by vehicles during their mobility in the scenario, it sets up the departure speed and time foe each street and the maximum allowed speed in each street;
- **LinkType.poly.xml**: additional file not always present which defines parameters regarding buildings and foliage's lines.

5.6 R-Trees structure

When working with such a complicated simulation, something that helps to recognize what objects are inserted in and classify them in dynamic/static objects is mandatory.

Since the simulation required a strong design and geo-data applications, spatial data are used and so an index mechanism that helps in retrieving data items quickly according to their locations has to be used: a dynamic index structure called **R-Tree** structure is what meets the needs and give algorithms for searching and updating objects [59].

The R-Trees structure is used to manipulate traffic objects using a hierarchical view based on their locations in the map's space. Moreover, it can distinguish moving objects (here vehicles, but also pedestrians and drones in other works where they are used) w.r.t. static objects (here buildings and foliage, but also different infrastructures and networks in other works).

The structure is constructed with a top-down approach similar to what is done with a binary-tree structure: every leaf node contains a pointer to a data object; a spatial

search is performed when needed but visiting only a limited number of nodes since every index is completely dynamic and there's no need for a reverse path in order to jump from one node to another located far away from the considered branch. Each tuple in this structure (a database indeed) represents a spatial object with a unique identifier that is used to retrieve information regarding its leaf nodes to which it is linked to in the form (I, tuple-ID) where I is an n-dimensional rectangle representing the "box" of the considered object, and tuple-ID identify the tuple in the database.

We won't go into the details of the algorithms (searching, updating, insertion, etc..), a good explanation can be found in Guttman Antonin work from 1984 [59] which despite the little dated work, still remains current regarding the mechanism of the R-Tree structure.

In the figure below, it is presented an R-Tree structure (fig. [a]) and the overlapping relationships that can exist between the rectangles of the structure (fig. [b]).

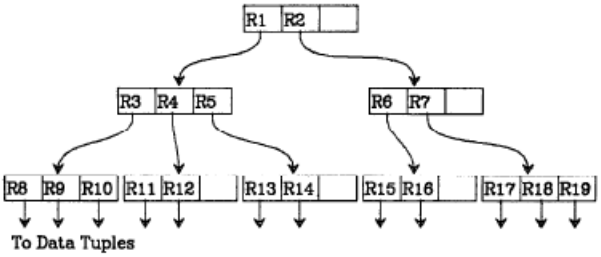


Figure 5.9: [a] R-Tree structure.

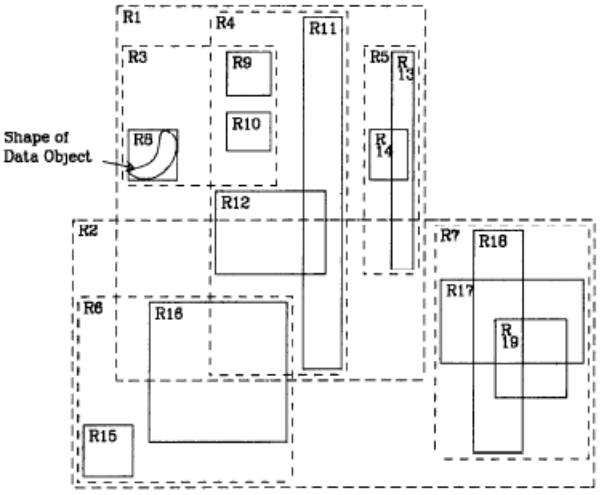


Figure 5.10: [b] Relationships between rectangles in the structure.

5.7 Summary Frameworks and Tools Technologies

In the scheme below are shown all the technologies mentioned in chapter 5:

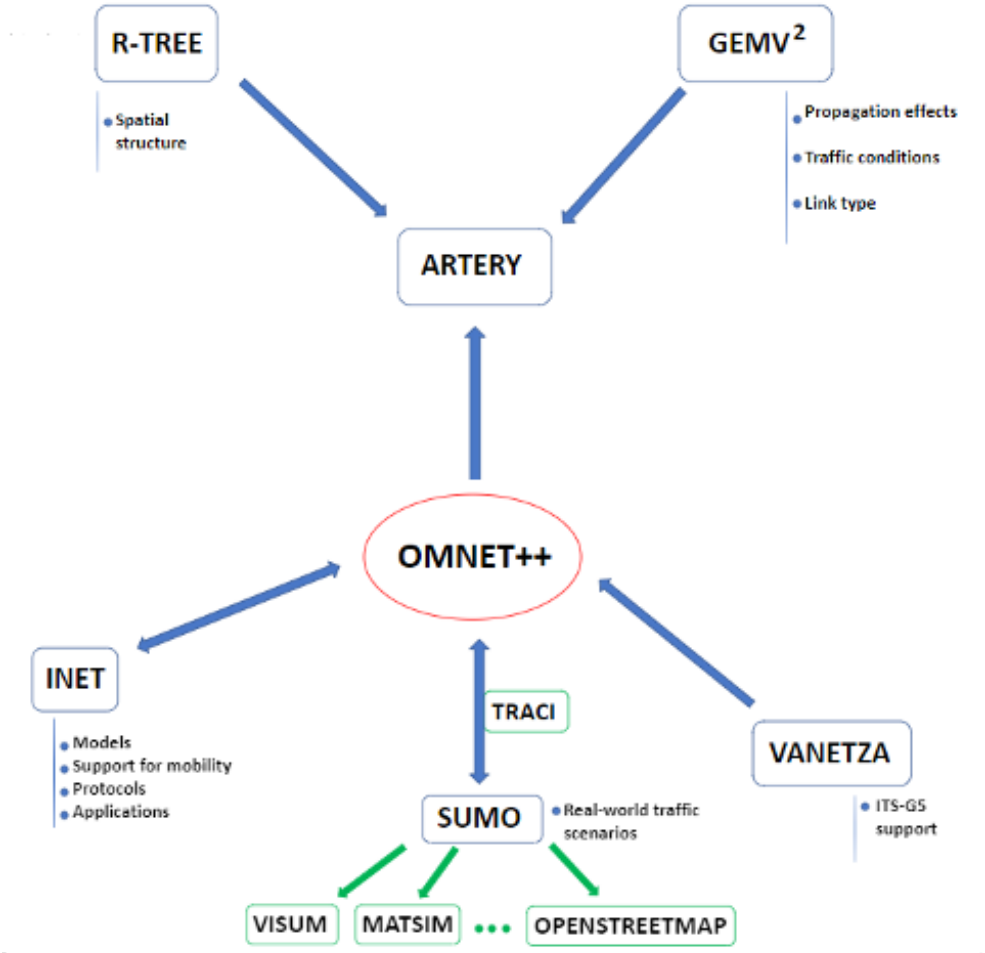


Figure 5.11: Scheme 3: Tools and Frameworks.

Chapter 6

Simulation Environment

In this chapter, we will go through each step of the set of instructions that has to be followed to launch the Gemv^2 tool in Artery project while working with OMNeT++.

The main purpose of this work is to understand how an already existing tool created for the aim of vehicular communication behaves and how it can be usefull for the needs of the end user. How the results given can help understanding a real-world situation?

The code is already created, the end user run it and sees the behaviour of different scenarios attributable to real world situations and make further analysis by using the results file the software gives back or by using any of the external software available on the market.

Further, the end user can take action on the code changing the main parameters for new environments, such as transmitted power or range of transmission, or by storing new parameters acting on the emitted signals.

6.1 Simulation flow

The simulation flow (see fig. 6.1) is the set of operations that are performed after the launch of the program. Some of them are performed automatically by the system (e.g. checking the already existing of previously loaded R-Trees), some other are choices to be taken by the user (e.g. LOS/NLOS link type).

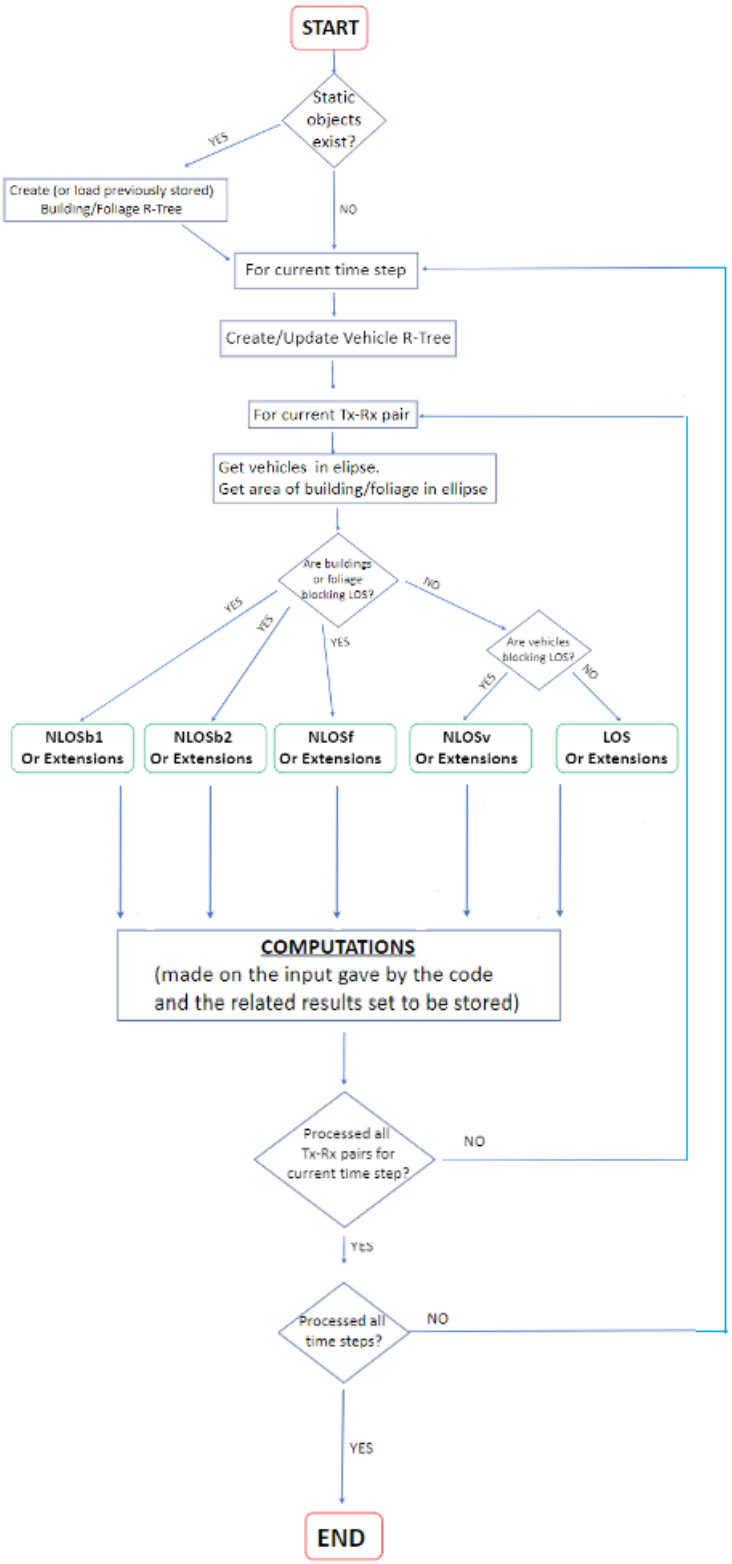


Figure 6.1: Simulation flow.

6.2 INI file, NED files and SUMO files

We have already talked about the role of the INI file in the section "OMNeT++" in chapter 5.
Here is the INI file from Gemv^2 where we can: set the initial parameters, load the first NED file (from there all the others NED files are sequentially loaded) which is the World.ned file, view the SUMO files eventually launched when the link type will be selected.

However, there are some assumptions that are made when using GEMV^2 in the network simulator and that typically are not changed in the configuration files involved: (i) buildings are too tall for any meaningful amount of power to be received over them; (ii) in environments where other objects (besides vehicles, buildings and foliage) e.g. lamp posts, signs, railing, etc.. are placed in the scenario, GEMV^2 does not consider them as obstructing objects; (iii) scattering is typically non considered due to its potential "large-number" nature; (iv) it is assume a flat-ground earth as terrain for the simulation.

omnetpp.ini simulation file:

omnetpp.ini

```
[General]
network = artery.inet.World

cmdenv-express-mode = true
cmdenv-autoflush = true

**.scalar-recording = false
**.vector-recording = false


*.traci.launcher.typename = "PosixLauncher"


*.withPhysicalEnvironment = true
*.physicalEnvironment.groundType = "FlatGround"

*.radioMedium.pathLossType = "Gemv2"
*.radioMedium.pathLoss.withSmallScaleVariations = false
*.radioMedium.pathLoss.withVisualization = true

*.node[*].wlan[*].typename = "VanetNic"
*.node[*].wlan[*].radio.channelNumber = 180
*.node[*].wlan[*].radio.carrierFrequency = 5.9 GHz
*.node[*].wlan[*].radio.transmitter.power = 200 mW

*.node[*].withAntennaMobility = true
*.node[*].antennaMobility.offsetX = -2.0 m
*.node[*].antennaMobility.offsetZ = truncnormal(2.0m, 0.4 m)
*.node[*].mobility.antennaHeight = 0.0m


*.node[*].middleware.updateInterval = 0.1s
*.node[*].middleware.datetime = "2017-10-26 15:05:00"
*.node[*].middleware.services = xmldoc("services.xml")
*.node[*].middleware.CaService.fixedRate = true
*.node[*].middleware.CaService.withDccRestriction = false


[Config LOS]
*.traci.launcher.sumocfg = "LOS.sumo.cfg"
*.node[*].antennaMobility.offsetZ = 0m

[Config LOS_lowAntennas]
extends = LOS
*.node[*].mobility.antennaHeight = 1.0 m

[Config LOS_mediumAntennas]
extends = LOS
*.node[*].mobility.antennaHeight = 1.5 m

[Config LOS_highAntennas]
extends = LOS
*.node[*].mobility.antennaHeight = 4.0 m

[Config NLOSv]
*.traci.core.startTime = 5s
*.traci.launcher.sumocfg = "NLOSv.sumo.cfg"
```

```
[Config NLOSb1]
*.traci.launcher.sumocfg = "NLOSb1.sumo.cfg"

[Config NLOSb2]
*.traci.launcher.sumocfg = "NLOSb2.sumo.cfg"

[Config NLOSb1_diffractionReflection]
extends = NLOSb1, NLOSb_diffractionReflection

[Config NLOSb1_diffractionReflectionWithoutVisualization]
extends = NLOSb1_diffractionReflection, noVisualization

[Config NLOSb1_distanceSwitch]
extends = NLOSb1, NLOSb_distanceSwitch

[Config NLOSb2_diffractionReflection]
extends = NLOSb2, NLOSb_diffractionReflection

[Config NLOSb1_smallScaleVariations]
extends = NLOSb1_diffractionReflection
*.radioMedium.pathLoss.withSmallScaleVariations = true
*.radioMedium.pathLoss.smallScaleVariations.maxVehicleDensity = 0.00025
*.radioMedium.pathLoss.smallScaleVariations.maxObstacleDensity = 0.035

[Config NLOSb2_smallScaleVariations]
extends = NLOSb2_diffractionReflection
*.radioMedium.pathLoss.withSmallScaleVariations = true
*.radioMedium.pathLoss.smallScaleVariations.maxVehicleDensity = 9e-05
*.radioMedium.pathLoss.smallScaleVariations.maxObstacleDensity = 0.035

[Config NLOSf]
*.traci.launcher.sumocfg = "NLOSf.sumo.cfg"

[Config NLOSf_noVisualization]
extends = NLOSf, noVisualization

[Config NLOSb_diffractionReflection]
*.radioMedium.pathLoss.NLOSb.typename = "NLOSb"
*.radioMedium.pathLoss.smallScaleVariations.minStdDevNLOSb = 0 dB

[Config NLOSb_distanceSwitch]
*.radioMedium.pathLoss.NLOSb.typename = "DistanceSwitchPathLoss"
*.radioMedium.pathLoss.NLOSb.thresholdDistance = 50 m
*.radioMedium.pathLoss.NLOSb.near.typename = "NLOSb"
*.radioMedium.pathLoss.NLOSb.near.maxRange = 50 m
*.radioMedium.pathLoss.NLOSb.far.typename = "FreeSpacePathLoss"
*.radioMedium.pathLoss.NLOSb.far.alpha = 2.9

[Config noVisualization]
*.radioMedium.pathLoss.withVisualization = false
```

Figure 6.2: omnetpp.ini simulation file

The **World.ned** file (see fig. 6.3, 6.4 and 6.5) is the initial NED file that is loaded by the system. It contains the basic parameters and sub-modules that are needed to go through each step of the simulation.

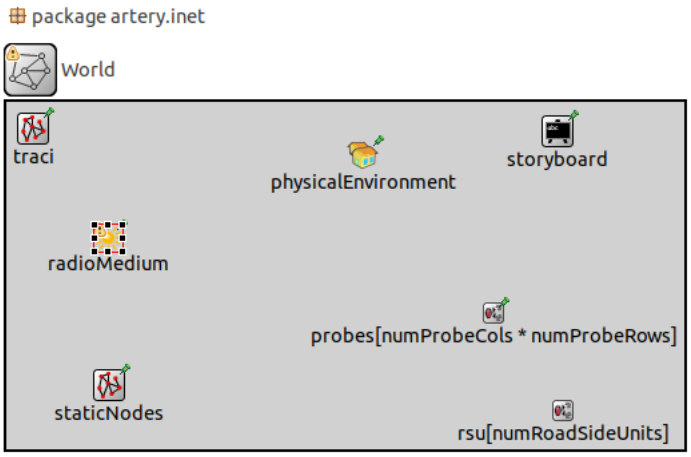


Figure 6.3: World.ned file: Design view.

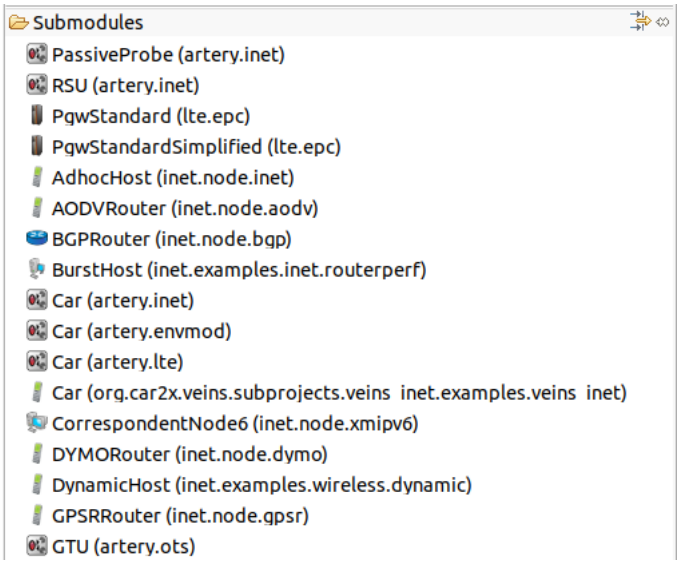


Figure 6.4: World.ned file: Submodules.

World.ned

```
package artery.inet;

import artery.StaticNodeManager;
import artery.storyboard.Storyboard;
import inet.environment.contract.IPhysicalEnvironment;
import inet.physicallayer.contract.packetlevel.IRadioMedium;
import traci.Manager;

network World
{
    parameters:
        bool withStoryboard = default(false);
        bool withPhysicalEnvironment = default(false);
        int numRoadSideUnits = default(0);
        traci.mapper.personType = default("artery.inet.Person");
        traci.mapper.vehicleType = default("artery.inet.Car");
        traci.nodes.personSinkModule = default(".mobility");
        traci.nodes.vehicleSinkModule = default(".mobility");
        storyboard.middlewareModule = default(".middleware");

        int numProbeCols = default(0);
        int numProbeRows = default(0);
        double probeInterval @unit(m) = default(25m);

    submodules:
        traci: Manager {
            parameters:
                @display("p=20,20");
        }

        radioMedium: <default("Ieee80211ScalarRadioMedium")> like IRadioMedium {
            parameters:
                @display("p=76,102");
                mediumLimitCache.carrierFrequency = 5.9GHz;
        }

        physicalEnvironment: <default("PhysicalEnvironment")> like IPhysicalEnvironment if withPhysicalEnvironment {
            parameters:
                @display("p=267,40");
        }

        storyboard: Storyboard if withStoryboard {
            parameters:
                @display("p=413,22");
        }

        rsu[numRoadSideUnits]: RSU {
            parameters:
                mobility.initFromDisplayString = false;
        }

        staticNodes: StaticNodeManager {
            parameters:
                @display("p=77,212");
                waitForTraCI = default(true);
        }

        probes[numProbeCols * numProbeRows]: PassiveProbe {
            parameters:
                mobility.numHosts = numProbeCols * numProbeRows;
                mobility.columns = numProbeCols;
                mobility.rows = numProbeRows;
                @display("p=365,158");
        }
}
```

Figure 6.5: World.ned file: Source view (C++ code).

Import is the way OMNeT++ links different NED files to work in cohesion during the simulation. It means that not only one single NED file (the World.ned file) is uploaded at the beginning and then nothing else happen, but a bunch of NED files are loaded and simultaneously running for a more realistic simulation.
Several NED files are loaded for the managing of the transmission, the connections between nodes, the antenna and radio parameters, and so on..
(In the figure above we can clearly see an example of "import" of NED file, right after the name of the package file the file is referred to).

As initial parameters, the default NED files are loaded (e.g. the Car.ned file from the artery-inet path).
As sub-modules, the basic elements in charge of controlling the situation while the simulation is running are created.

- **Traci Manager**: Management parameter (e.g. startingTime), launcher files (e.g. configuration file, SUMO launcher file), parameters related type of nodes and type of mapper to be used (e.g. people and vehicles involved)(see fig. 6.6).

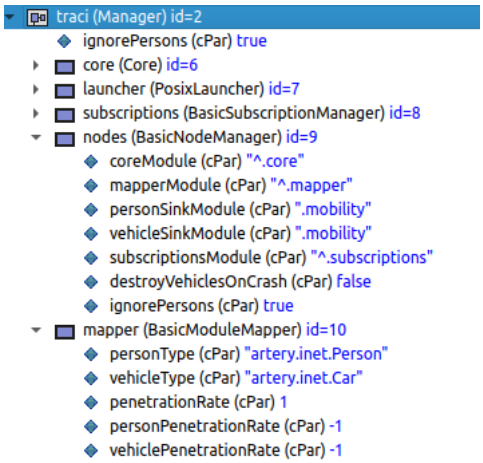


Figure 6.6: Traci Manager.

Here is where the SUMO files related to the link type are first loaded by the Traci Manager with the following line of code:

**.traci.launcher.sumocfg = "LinkType.sumo.cfg".*

- **Radio medium**: all the sets of parameters used to define the medium channel that is used, such as pathLossType (in our case, it is always Gemv^2), background noise parameters, propagation parameters, etc.. (see fig. 6.7).

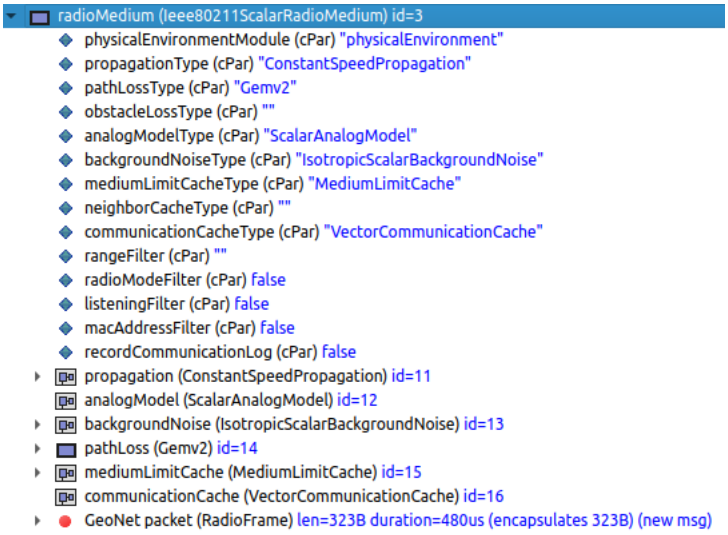


Figure 6.7: Radio medium.

- **Physical Environment**: groundType file, temperature, spaces (see fig.6.8).

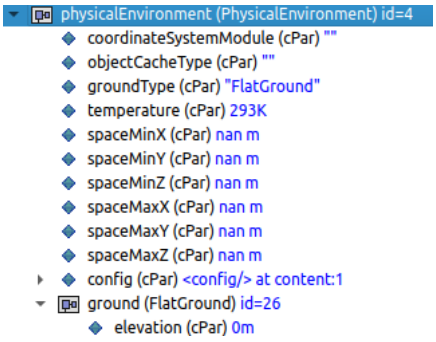


Figure 6.8: Physical Environment

- **Static Nodes and Nodes**: parameters related to the vehicles involved (antennas, belonging RSU, mobility parameters, etc..)(see fig. 6.9).

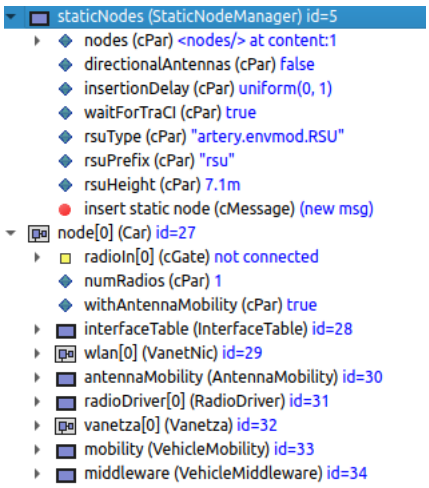


Figure 6.9: Static Nodes and Nodes.

Still in the INI file, we can see the different link type that can be chosen when using Gemv^2.
LOS, NLOSb1, NLOSb2, NLOSv, NLOSf and their extensions, such as the differentiation of the antenna heights when selecting the LOS type.

From that, what happens is that the configuration file of SUMO will go through its code sequentially launching the others SUMO files.

- **LinkType.sumo.cfg**: the configuration file where .net and .rou xml files are launched (see fig.6.10).

LOS.sumo.cfg

```
<configuration>
    <input>
        <net-file value="LOS.net.xml"/>
        <route-files value="LOS.rou.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="79"/>
    </time>
</configuration>
```

Figure 6.10: LinkType.sumo.cfg. Here the LOS type is shown.

- **LinkType.net.xml**: as explained is chapter 5, in the .net file we can find all the parameters about vehicles, such as length, width, etc.., about the link from one vehicles to the others creating the "network" of communicating vehicles and the parameters regarding the lane's shapes and borders (see fig.6.11).

LOS.net.xml

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

<!-- generated on Mon Feb 26 13:49:33 2018 by SUMO netconvert V
<?xml version="1.0" encoding="UTF-8"?>

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst

    <input>
        <node-files value="LOS.nod.xml"/>
        <edge-files value="LOS.edg.xml"/>
    </input>

    <output>
        <output-file value="LOS.net.xml"/>
    </output>

</configuration>
-->

<net version="0.27" xmlns:xsi="http://www.w3.org/2001/XMLSchema

    <location netOffset="500.00,-10.00" convBoundary="0.00,0.00

    <edge id=":1_0" function="internal">
        <lane id=":1_0_0" index="0" speed="13.90" length="7.45"
    </edge>
    <edge id=":2_0" function="internal">
        <lane id=":2_0_0" index="0" speed="13.90" length="7.45"
    </edge>

  <edge id="a" from="1" to="2" priority="-1">
      <lane id="a_0" index="0" speed="13.90" length="1000.00" w
          <neigh lane="b_0"/>
      </lane>
  </edge>
  <edge id="b" from="2" to="1" priority="-1">
      <lane id="b_0" index="0" speed="13.90" length="1000.00" w
          <neigh lane="a_0"/>
      </lane>
  </edge>

  <junction id="1" type="priority" x="0.00" y="0.00" incLanes="
      <request index="0" response="0" foes="0" cont="0"/>
  </junction>
  <junction id="2" type="priority" x="1000.00" y="0.00" incLane
      <request index="0" response="0" foes="0" cont="0"/>
  </junction>

  <connection from="a" to="b" fromLane="0" toLane="0" via=":2_0
  <connection from="b" to="a" fromLane="0" toLane="0" via=":1_0

  <connection from=":1_0" to="a" fromLane="0" toLane="0" dir="s
  <connection from=":2_0" to="b" fromLane="0" toLane="0" dir="s

net>
```

Figure 6.11: LinkType.net.xml. Here the LOS type is shown.

- **LinkType.rou.xml**: route edges followed by vehicles , speeds and times (see fig.6.12).

LOS.rou.xml

```
<routes>
    <vehicle id="1" depart="0.0">
        <route edges="a"/>
    </vehicle>

    <vehicle id="2" depart="0.0">
        <route edges="b"/>
    </vehicle>
</routes>
```

Figure 6.12: LinkType.rou.xml. Here the LOS type is shown.

- **LinkType.poly.xml**: additional parameters regarding building and foliage shapes (see fig. 6.13).

NLOSf.poly.xml

```
<additionals>
    <poly id="build_x" type="building" color="230,230,230" fill="1" layer="1.00" sha
    <poly id="forest1" type="forest" color="green" fill="1" layer="0.00" shape="-45.
    <poly id="tree1" type="tree" color="green" fill="1" layer="0.00" shape="18.40,86
    <poly id="tree2" type="tree" color="green" fill="1" layer="0.00" shape="35.43,85
    <poly id="tree3" type="tree" color="green" fill="1" layer="0.00" shape="19.19,16
    <poly id="tree4" type="tree" color="green" fill="1" layer="0.00" shape="20.95,85
    <poly id="tree5" type="tree_row" color="green" fill="1" layer="0.00" shape="27.9
</additionals>
```

Figure 6.13: LinkType.poly.xml. Here the NLOSf type is shown.

6.2.1 Modifications to the code

Some modifications have to be done to the code of the INI file (see fig. 6.14) in order to store some data that are not automatically stored but that they already have the signal declarations, emissions and listening methods implemented in some other NED files, especially from the INET framework.

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

**.bitrate.param-record-as-scalar = true
**.packetBytes.param-record-as-scalar = true
**.packetDiscarded.param-record-as-scalar = true
**.throughput.param-record-as-scalar = true
**.symbolErrorRate.param-record-as-scalar = true
**.packetErrorRate.param-record-as-scalar = true
**.bitErrorRate.param-record-as-scalar = true

**.scalar-recording = true
**.vector-recording = true
**.bin-recording = true
**.statistic-recording = true
```

Figure 6.14: Modifications.

6.3 Launching from the terminal

Here it is shown the schema needed at the beginning to launch the simulation:

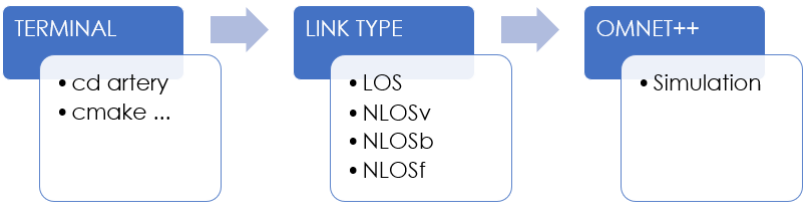


Figure 6.15: Simulation schema.

So first thing to do is to open a new terminal, move from "home" directory to "Artery" and launch the Gemv^2 tool with the **cmake** command; A better explanation of the set of operations to perform the command is given in the Appendix A: Installation Guide.

The simulator will automatically upload the frameworks it needs (OMNeT++, traci, SUMO, etc..) and, at the end, a window of the simulator OMNeT++ will be displayed.

6.4 LinkType choice

Once the simulator is ready, a multiple choice menù will be displayed on screen (see fig. 6.16). Here is where the user can decide in the bunch of possibilities which link type he want to simulate; the possibilities are the ones proposed by Gemv^2 tool: LOS, NLOSv, NLOSb, NLOSf and all their extensions related to them.

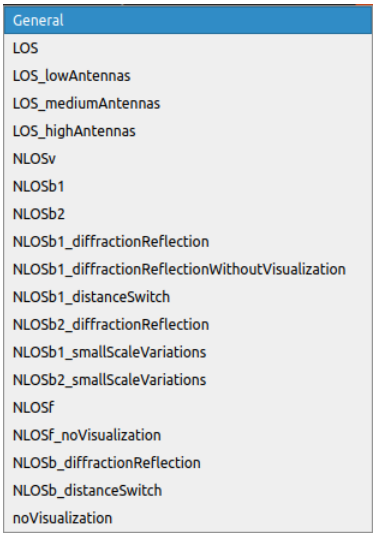


Figure 6.16: Gemv^2 menù.

6.4.1 Gemv^2 link type

As we have already said, Gemv^2 link choice is not only related to the non-obstructed/obstructed path but there also some other features that can be performed in the simulation.

LOS type has 3 extensions related to the different antenna heights mounted on on top of the vehicles (low, medium, high).
NLOSb is first divided in "b1" and "b2" where it can be found a different expansions of number of buildings in the path, number of streets and number of vehicles involved. Then, we have 2 extensions for b2 type and 3 for b1 type: [a] Diffraction-Refrection of the path (with a different minStdDev w.r.t. the basic NLOSb type), [b] the implementation of the SmallScaleVariations (besides to the already running LargeScaleVariations) and, only for b1 type, [c] DistanceSwitch extension (with a more strictly range of transmission and the implementation of the free space loss parameter set to 2.9dB).
NLOSv and NLOSf have no extensions.

6.5 Graphical view

When the link type to be simulated is chosen, the OMNeT++ ide (the graphical interface) is shown (fig. 6.17).

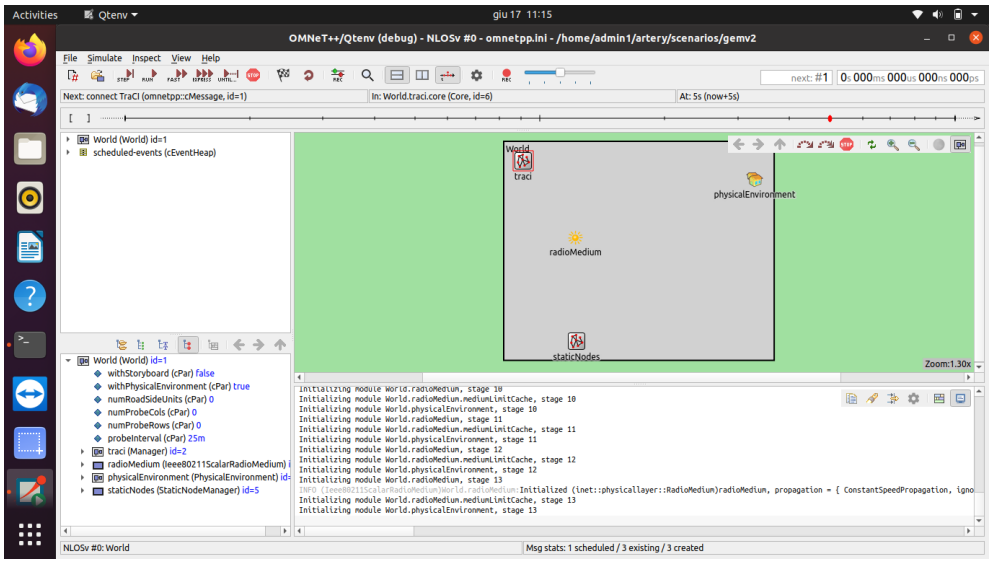


Figure 6.17: OMNeT++ graphical view.

Here we can find all the buttons to run the simulation in normal speed or fasten, some others to search specific features while running, the counter of the number of events occurred during the transmission and the time passed and 4 windows showing some parameters:

1. Top left shows the major NED file used, the world.ned file, and the directory containing the scheduled events.
2. Bottom left shows the modules implemented and their parameters when highlighted and opened. It varies while the simulation proceeds adding and/or removing items (vehicles) when they reach their final destination.
3. Top right shows the modules used (defined in the INI file).
4. Bottom right shows the events occurred while running and significant characteristics.

6.5.1 Visualization in SUMO

With the play button the simulation begins. But first, if the traci manager sees that the SUMO graphical view is required, it launches it and the user is able to better understand what it is going on through a more realistic scenario representing the selected link type.

- **LOS**: 1 highway, 2 vehicles running in opposite directions meeting halfway.

Figure 6.18: LOS view: standard visualization.

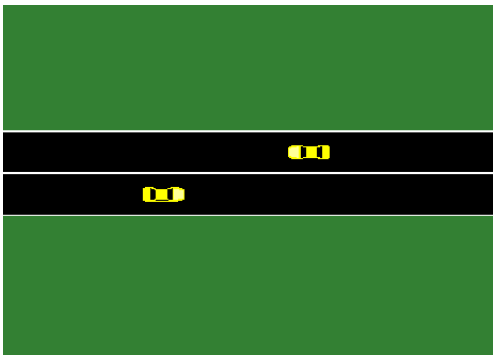


Figure 6.19: LOS view: real-world visualization.

- **NLOSv**: 1 highway, 4 lanes but only 2 are used (and running in the same directions), xxx vehicles already in the scenario since the beginning.

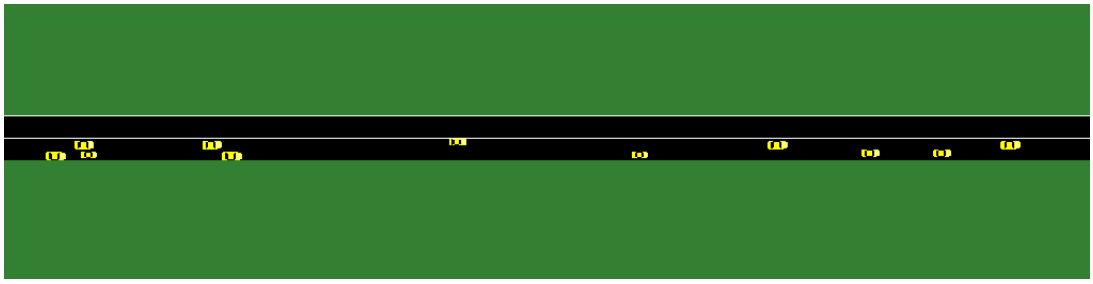


Figure 6.20: NLOSv view: real-world visualization.

- **NLOSb1**: urban scenario with 6 streets (4 with opposite directional lanes, 2 with single directional lane), increasing number of vehicles up to 19 vehicles entering and exiting the scenario, 4 buildings.

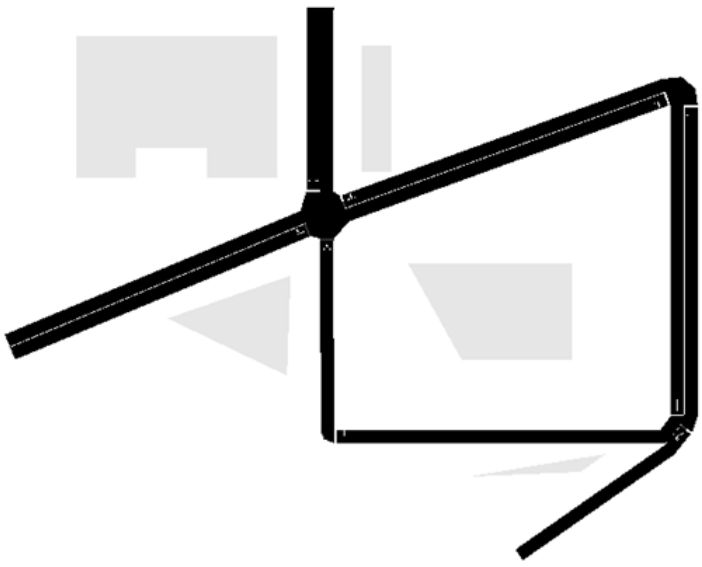


Figure 6.21: NLOSb1 view: standard visualization.

- **NLOSb2**: urban scenario with 4 streets (each having double directional lanes), increasing number of vehicles up to 8 vehicles entering and exiting the scenario, 4 buildings.

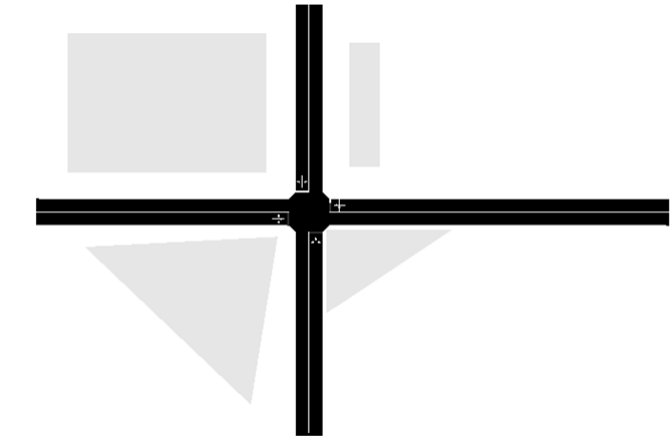


Figure 6.22: NLOSb2: standard visualization.

- **NLOSf**: urban scenarios with the introduction of foliage concept; 4 streets (each having double directional lanes), increasing number of vehicles up to 8 vehicles entering and exiting the scenario, 1 forest and 5 bushes, 1 building.

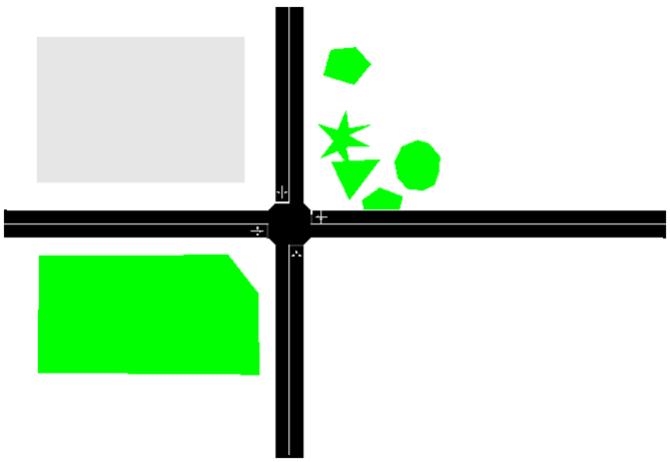


Figure 6.23: NLOSf view: standard visualization.

Now, pushing the play button from the SUMO ide, the simulation can start.

6.6 While running

We can observe the progress of the simulation both in the SUMO ide and OMNeT++ ide.
The first one is more representative of what it is going on in the scenario meaning that we can see the exact path where a vehicle is running and which are the other vehicles it encounters during the simulation or when it enters or exits the scenario.
Using the OMNeT++ visualization we can track the events occurring in the simulation or see which node or RSU is transmitting at a certain moment: stopping the simulation, a typical situation is when we observe a node (fig. 6.25) or RSU (fig. 6.24) tracking the paths of its messages (to which nodes transmit them) and after that see the effective paths (fig. 6.26) follow by each of the packet containing the information; plus, we can observe also those nodes not hit by the transmission and so not receiving anything at that moment.

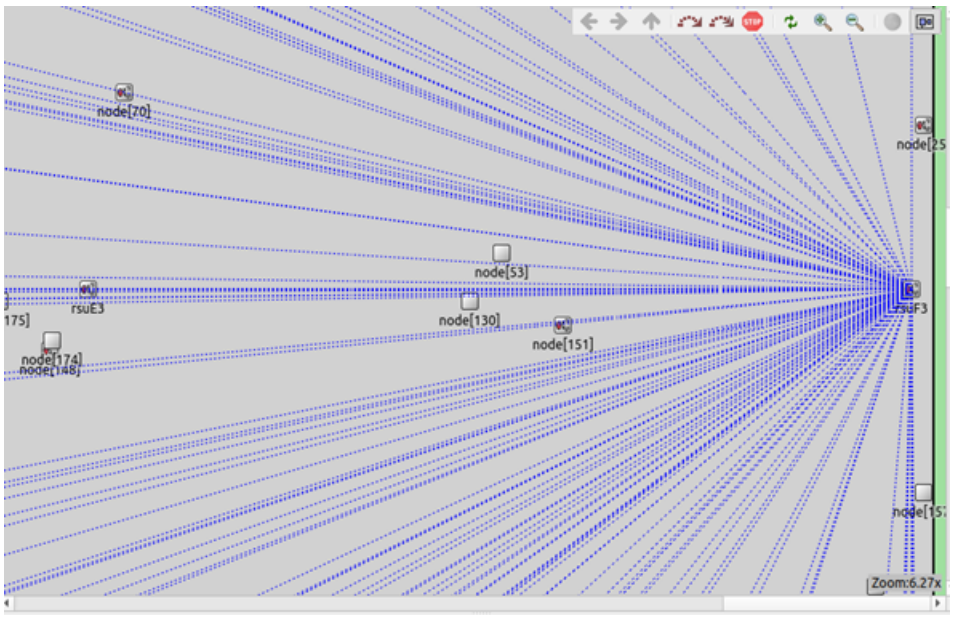


Figure 6.24: RSU path.

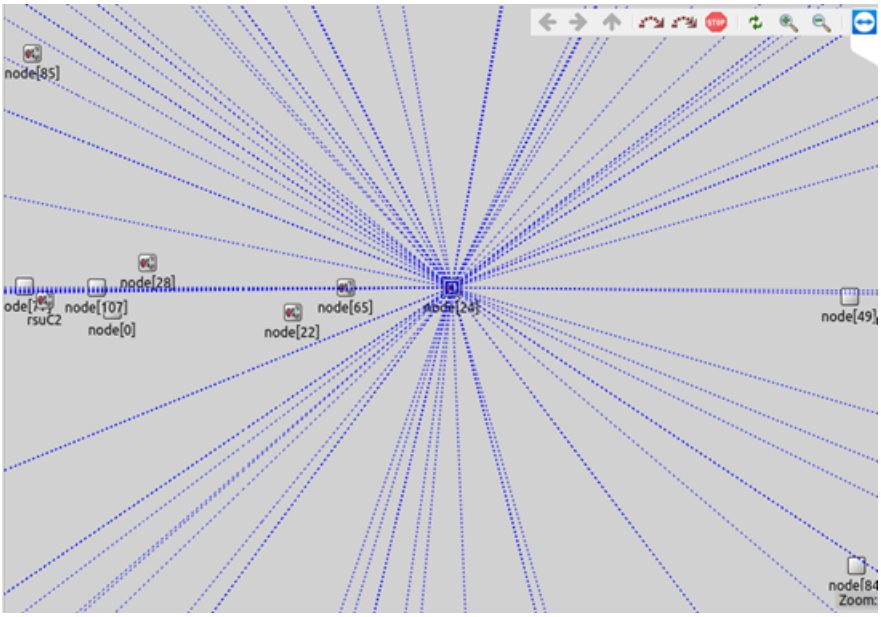


Figure 6.25: Node path.

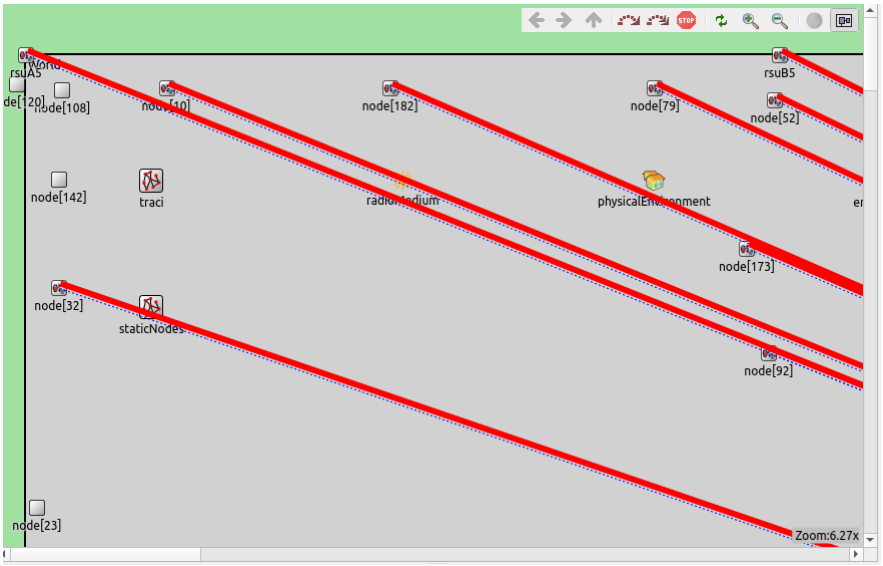


Figure 6.26: Target nodes.

6.7 End of the simulation

Depending on the link type chosen, the simulation has different duration times.

At the end of the simulation, the results are stored in the "results" directory in the Artery - scenarios - Gemv^2 directory; it can be access directly from the the directories path or from the OMNeT++ ide typing *$omnetpp* in the console and opening a new OMNeT++ window.

From the OMNeT++ view, we switch to the "results" directory of Gemv^2 and double-clicking on the .sca and .vec files, which are the two files where the system has stored scalar and histogram data in the .sca file and vectorial data in the .vec file.

Chapter 7

Numerical Results and Analysis

In this chapter we will see which results are automatically stored by the tool and which others can be stored with the changes to the code mentioned in the "Modifications" section of chapter 7.

Few comments before going on: Gemv^2 is not a complete tool.
It was created by students as a thesis work [60], some of its parts have not been completed while others are only mentioned and suggested but not coded yet. This could be a good hint for any future works.

7.1 Parameters and Files Involved

Since Gemv^2 is an already existing tool, there are some parameters that are taken as always true by default and set by the system automatically and some others the end user can work on to change the prediction on the scenario.

7.1.1 Fixed Parameters

- **Flat ground**: the physical environment takes as true a flat ground type, thus not considering the earth's curvature. That's why the simulated scenarios are not so big and, as it happens in real world, the curvature can be neglected in such scenarios;

- **5.9 GHz**: the carrier radio frequency on which Gemv^2 works is the 5.9 GHz, a transmission band that nowadays is contested between the wireless and the 5G communications and not allocated yet;
- **Band**: the band use is the so called 5 GHz and the bandwidth is 10 MHz;
- **Permittivity**: relative permittivity for LOS environment = 1, by default. Relative permittivity of vehicles in NLOS environments = 6. Relative permittivity of buildings in NLOSb and NLOSf environments = 4.5;
- **Path loss exponent**: = 2.9 dB;
- **Polarization**: vertical type;
- **Antenna height**: = 1.5 m from the ground. It changes in the LOS extensions;
- **RSU height**: = 7.1 m.

7.1.2 Changing Parameters

To best evaluate different possibilities in the same scenario, some parameters have been taken with different values:

- **Transmitter power**: it varies from 0.01 mW (-20 dBm), to 1 mW (0 dBm), endlessy to 200 mW (23 dBm) which is the value given by default;
- **LOS/NLOS ranges**: from all ranges equal to 100 m, to the default values and finally to the default values multiplied by 10.

The different combinations used (different "replicas") are:

SIMULATION PARAMETERS						
REPLICA		RADIO TRANSMITTER POWER		LOS RANGE	NLOSB RANGE	NLOSV RANGE	NLOSF RANGE
0		200 mW → 23 dBm		500 m	300 m	400 m	500 m
1		0,01 mW → -20 dBm		500 m	300 m	400 m	500 m
2		200 mW → 23 dBm		100 m	100 m	100 m	100 m
3		200 mW → 23 dBm		5000 m	3000 m	4000 m	5000 m
4		0,01 mW → -20 dBm		100 m	100 m	100 m	100 m
5		0,01 mW → -20 dBm		5000 m	3000 m	4000 m	5000 m
6		NOT USED					
7		1 mW → 0 dBm		500 m	300 m	400 m	500 m
8		1 mW → 0 dBm		100 m	100 m	100 m	100 m
9		1 mW → 0 dBm		5000 m	3000 m	4000 m	5000 m

Figure 7.1: Combinations of simulated parameters.

7.1.3 .ANF file

Once the simulation is completed, the results can be accessed (1°) directly from the "Results" directory in Artery path, (2°) exporting them in a CSV file or (3°) by looking at them through the use of OMNeT++ graphical view. The third method is the one we will use here.

Opening OMNeT++, we will search in the Artery - scenarios - Gemv^2 - results directories.
Here there will be a set of files, typically 4 files for each scenario simulated, named (i) LinkType.sca, (ii) LinkType.vec, the (iii) log file containing all the events occurred while running and a (iv) LinkType.vci containing the data that the vectors will use.

What we are interested in is the .anf file (see fig. 7.2) containing all the results: select all the file of the relative LinkType chosen for the anlysis, double click on the selection and create the .anf file.

The **.ANF** file has 3 possible views attributable to the buttons placed at the bottom-left side of it:

- **Inputs**: it resumes the file contained in the .anf file, indeed the .sca and .vec files;
- **Browse Data**: it contains all the values recorded, from vectors to scalars to histograms data;
- **Datasets**: where automatic computations can be performed at the end of the simulation. The end user creates the dataset, adding values and performing calculations, and see the results of the dataset on a graph or new file.

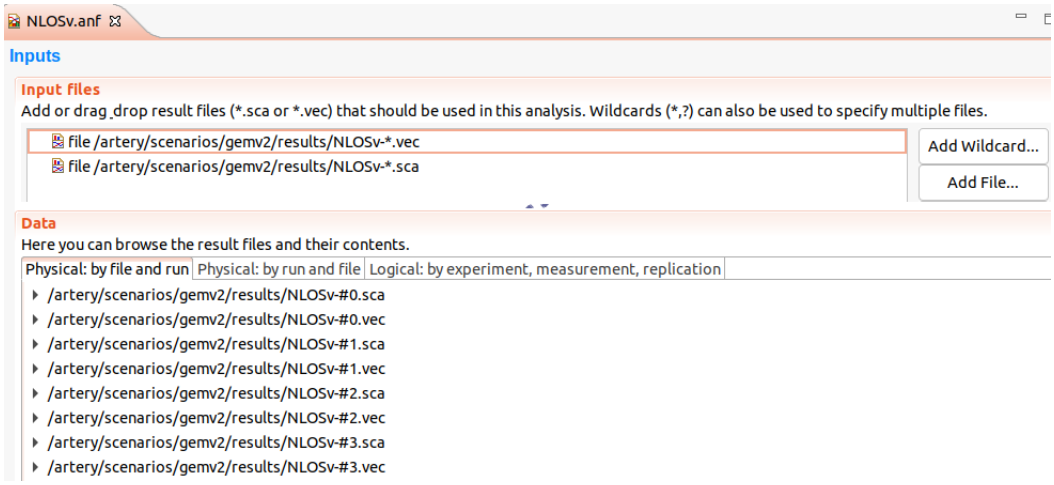


Figure 7.2: .anf file view. Here NLOSv type is shown.

7.2 Values Types

7.2.1 Browse Data: Vectors values

The values recorded here are series of values, but the single data cannot be accessed individually. Infact, Gemv^2 does not handle and has no concept of individual packets.

- **radioChannel**: number of the channel used in transmission, by default channel n° 180;
- **radioMode**: counts of the switches from radio mode 3 to radio mode 4 and viceversa for each node;
- **transmissionState**: number of occurrences that passes through each node;
- **receptionState**: number of occurrences received by each node;
- **passedUpPk(packetBytes)**: packets (in terms of Bytes) the highlighted node forwards to the upper layers of its pile;
- **sentDownPk(packetBytes)**: packets (in terms of Bytes) the highlighted node forwards to the lower layers of its pile;
- **rcvdPkFromHL(packetBytes)**: packets (in terms of Bytes) the highlighted node receives from the upper layers of its pile;
- **rcvdPkFromLL(packetBytes)**: packets (in terms of Bytes) the highlighted node received from the lower layers of its pile.

Few comments about the last four values.

Remembering the ISO/OSI stack (see fig. 7.3), we know that in each node the layer in charge to deal with the packets is the second layer, the **Link layer**.

Here, the link layer receives packets from the upper layer (rcvdPkFromHL) when the application needs to transmit a packet to any other node and sent it down to the lowest layer (sentDown) that will transmits the packet on the network.
On the other side, the link layer receives packets from the lower layer (rcvdPkFromLL) when the node has received a new one from any other node, and then it transmits this packet to the upper layer of the stack (passedUp) with a flag set to true (when the received packet coming from the network has some errors and it's not correct) or false (when everything went smoothly and run correctly).

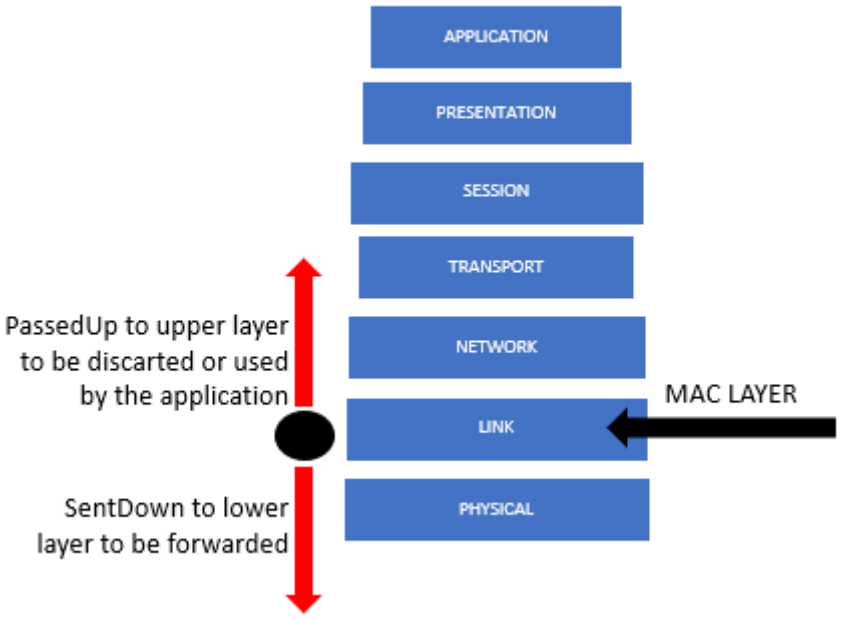


Figure 7.3: ISO/OSI stack and the PassedUp/SentDown mechanism handling the packets.

7.2.2 Browse Data: Scalars values

- **Arrival computation count**: total number of occurrences propagates by the radio medium;
- **bitrate**: Gemv^2 is specifically designed to operate at the 6 GHz band;
- **countLOS/NLOSb/NLOSv/NLOSf**: number of times the transmission has experienced a communication without any obstacles or has been hampered by a building/vehicle/foliage on its path;
- **Interference computation count**: number of times the radio medium has experienced any kind of interference of the channel;
- **port**: on which the traci launcher has opened a connection;
- **radio frame send count**: number of occurrences the radio medium has forwarded a frame on a transmission;
- **passedUpPk(packetBytes)** and **passedUpPk(count)**: with the same mechanism already explained as vectors data;
- **sentDown(packetBytes)** and **sentDown(count)**: with the same mechanism already explained as vectors data;

- **rcvdPkFromHL(packetBytes)** and **rcvdPkFromHL(count)**: with the same mechanism already explained as vectors data;
- **rcvdPkFromLL(packetBytes)** and **rcvdPkFromLL(count)**: with the same mechanism already explained as vectors data.

7.2.3 Browse Data: Histograms values

This section is supposed to be the place where the system automatically computes **Error Rates** to better understand what's going on in the transmission.
Unfortunately, only a couple of rates really work while the other are not able to record the right flag for correct/incorrect packets and so it cannot compute any rates.

- **packetErrorRate**: total number of occurrences involved in the computation of packets transmitted wrongly;
- **minSNIR**: total number of occurrences involved in the computation of the minimum signal-to-noise plus interference ratio and SUM of the mean value of each (focus on sum, this is not the mean minSNIR of all the transmitted packets!);
- **bitErrorRate** and **SymbolErrorRate**: not recorded.

7.3 Recorded Data

Here we will make some analysis of the given results trying to catch something related in order to better understand what the tool can give to the end user to be replicated in real-world situation.

7.3.1 Data not influenced by parameters

Some values do not change when transmitter power and/or ranges are set to different values.

- **count LOS/NLOS**: number of times the communication has experienced a clear transmission (LOS type) or something has blocked it (NLOS due to whatever thing).
  It does not change even if there is a change in the range values and this is a big limitation of the tool.

Where not necessary, the count of a NLOS due to something missing is considered as 0; indeed, it is reasonable that in LOS type with no obstacles in between the count of NLOS occurrences is 0 (see fig. 7.4) or in NLOS due to the vehicles the counts of NLOS due to the buildings or NLOS due to the foliage is 0 (see fig. 7.5) since the scenario is running on a highway with only vehicles as obstacles, while in NLOS due to the foliage all the available NLOS types have a count (see fig.7.6).

Module	Name	Value
World.radioMedium.pathLoss.classifier	countLOS	1498.0
World.radioMedium.pathLoss.classifier	countNLOSb	0.0
World.radioMedium.pathLoss.classifier	countNLOSf	0.0
World.radioMedium.pathLoss.classifier	countNLOSv	0.0

Figure 7.4: Counts of LOS/NLOS in LOS type.

Module	Name	Value
World.radioMedium.pathLoss.classifier	countNLOSb	0.0
World.radioMedium.pathLoss.classifier	countNLOSf	0.0
World.radioMedium.pathLoss.classifier	countNLOSv	7098.0
World.radioMedium.pathLoss.classifier	countLOS	6294.0

Figure 7.5: Counts of LOS/NLOS in NLOSv type.

Module	Name	Value
World.radioMedium.pathLoss.classifier	countLOS	5868.0
World.radioMedium.pathLoss.classifier	countNLOSb	295.0
World.radioMedium.pathLoss.classifier	countNLOSf	2132.0
World.radioMedium.pathLoss.classifier	countNLOSv	1042.0

Figure 7.6: Counts of LOS/NLOS in NLOSf type.

- **Transmission state**: the number of packets sent by each node does not change if we change the value of transmitting power or ranges. The node tries to forward the same number of packets every time (fig. 7.7).

Replica	Module	Name	Count
#0	World.node[4].wlan[0].	transmissionState:vector	312
#1	World.node[4].wlan[0].	transmissionState:vector	312
#2	World.node[4].wlan[0].	transmissionState:vector	312
#3	World.node[4].wlan[0].	transmissionState:vector	312
#4	World.node[4].wlan[0].	transmissionState:vector	312
#5	World.node[4].wlan[0].	transmissionState:vector	312
#7	World.node[4].wlan[0].	transmissionState:vector	312
#8	World.node[4].wlan[0].	transmissionState:vector	312
#9	World.node[4].wlan[0].	transmissionState:vector	312

Figure 7.7: Transmission state.

- **Arrival computation count**: the number of occurrences that the radio medium handle is the same no matter the transmission powers or ranges. Infact, has seen for Transmission state, each node forward the same number of packets on the radio medium each time (fig. 7.8).

Replica	Module	Name	Value
#0	World.radioMedium.propagation	Arrival computation count	13392.0
#1	World.radioMedium.propagation	Arrival computation count	13392.0
#2	World.radioMedium.propagation	Arrival computation count	13392.0
#3	World.radioMedium.propagation	Arrival computation count	13392.0
#4	World.radioMedium.propagation	Arrival computation count	13392.0
#5	World.radioMedium.propagation	Arrival computation count	13392.0
#7	World.radioMedium.propagation	Arrival computation count	13392.0
#8	World.radioMedium.propagation	Arrival computation count	13392.0
#9	World.radioMedium.propagation	Arrival computation count	13392.0

Figure 7.8: Arrival computation count.

- **Radio frame send count**: as the Transmission state, also the Radio frame forward on the radio medium is always the same no matter about the already mentioned parameters (fig. 7.9).

Replica	Module	Name	Value
#0	World.radioMedium	radio frame send count	13392.0
#1	World.radioMedium	radio frame send count	13392.0
#2	World.radioMedium	radio frame send count	13392.0
#3	World.radioMedium	radio frame send count	13392.0
#4	World.radioMedium	radio frame send count	13392.0
#5	World.radioMedium	radio frame send count	13392.0
#7	World.radioMedium	radio frame send count	13392.0
#8	World.radioMedium	radio frame send count	13392.0
#9	World.radioMedium	radio frame send count	13392.0

Figure 7.9: Radio frame send count.

- **Radio mode count**: the number of times the one node has switched its status from mode 3 to mode 4 does not change (fig. 7.10).

Replica	Module	Name	Value
#0	World.node[5].wlan[0].rac	radioMode:count	260.0
#1	World.node[5].wlan[0].rac	radioMode:count	260.0
#2	World.node[5].wlan[0].rac	radioMode:count	260.0
#3	World.node[5].wlan[0].rac	radioMode:count	260.0
#4	World.node[5].wlan[0].rac	radioMode:count	260.0
#5	World.node[5].wlan[0].rac	radioMode:count	260.0
#7	World.node[5].wlan[0].rac	radioMode:count	260.0
#8	World.node[5].wlan[0].rac	radioMode:count	260.0
#9	World.node[5].wlan[0].rac	radioMode:count	260.0

Figure 7.10: Radio mode count.

- **Reception computation count**: as already said for Arrival computation count, here the concept is the same; each node forward the same number of packets on the radio medium each time and thus the radio medium module receive the same number of packets each time (fig. 7.11).

Replica	Module	Name	Value
#9	World.radioMedium	reception computation count	13392.0
#8	World.radioMedium	reception computation count	13392.0
#7	World.radioMedium	reception computation count	13392.0
#5	World.radioMedium	reception computation count	13392.0
#4	World.radioMedium	reception computation count	13392.0
#3	World.radioMedium	reception computation count	13392.0
#2	World.radioMedium	reception computation count	13392.0
#1	World.radioMedium	reception computation count	13392.0
#0	World.radioMedium	reception computation count	13392.0

Figure 7.11: Reception computation count.

- **Transmission count, TransmissionState count**: each node forwards the same number of packets on the radio medium each time (fig. 7.12).

Replica	Module	Name	Value
#0	World.radioMedium	transmission count	1310.0
#1	World.radioMedium	transmission count	1310.0
#2	World.radioMedium	transmission count	1310.0
#3	World.radioMedium	transmission count	1310.0
#4	World.radioMedium	transmission count	1310.0
#5	World.radioMedium	transmission count	1310.0
#7	World.radioMedium	transmission count	1310.0
#8	World.radioMedium	transmission count	1310.0
#9	World.radioMedium	transmission count	1310.0

Figure 7.12: Transmission count.

- **rcvdFromHL, sentDown**: as already said for Transmission count, each node forwards the same number of packets and thus receives the same number of occurrences from higher layers sending them down to the physical layer to transmit them, no matter about transmitting power or ranges (fig. 7.13).

Replica	Module	Name	Value
#0	World.node[1].wlan[0].mac	sentDownPk:count	26.0
#1	World.node[1].wlan[0].mac	sentDownPk:count	26.0
#2	World.node[1].wlan[0].mac	sentDownPk:count	26.0
#3	World.node[1].wlan[0].mac	sentDownPk:count	26.0
#4	World.node[1].wlan[0].mac	sentDownPk:count	26.0
#5	World.node[1].wlan[0].mac	sentDownPk:count	26.0
#7	World.node[1].wlan[0].mac	sentDownPk:count	26.0
#8	World.node[1].wlan[0].mac	sentDownPk:count	26.0
#9	World.node[1].wlan[0].mac	sentDownPk:count	26.0

Figure 7.13: SentDown.

7.3.2 Data changing with parameters

Here are reported all those data influenced by the different replicas applied:

- **Reception state**: a vector containing the count of packets received by each node; this number grows with the power, thus more the power more the probability of receiving the packet (fig. 7.14).

Replica	Module	Name	Count ▲
#4	World.node[2].wlan[0].radio	receptionState:vector	629
#5	World.node[2].wlan[0].radio	receptionState:vector	631
#1	World.node[2].wlan[0].radio	receptionState:vector	631
#7	World.node[2].wlan[0].radio	receptionState:vector	2951
#9	World.node[2].wlan[0].radio	receptionState:vector	3039
#8	World.node[2].wlan[0].radio	receptionState:vector	3047
#2	World.node[2].wlan[0].radio	receptionState:vector	5179
#3	World.node[2].wlan[0].radio	receptionState:vector	6190
#0	World.node[2].wlan[0].radio	receptionState:vector	6221

Figure 7.14: Reception state.

- **rcvdFromLL, passedUp**: directly related with the reception of packets, it changes with the increasing value of power meaning that if the power is low the node is not able to forward the packet and thus to receive it; if it does not receive the packet, it does not have something to bring up to upper layers (fig. 7.15).

Replica	Module	Name	Value
#5	World.node[3].wlan[0].mac	passedUpPk:count	301.0
#1	World.node[3].wlan[0].mac	passedUpPk:count	305.0
#4	World.node[3].wlan[0].mac	passedUpPk:count	305.0
#7	World.node[3].wlan[0].mac	passedUpPk:count	3110.0
#8	World.node[3].wlan[0].mac	passedUpPk:count	3750.0
#9	World.node[3].wlan[0].mac	passedUpPk:count	3780.0
#2	World.node[3].wlan[0].mac	passedUpPk:count	6316.0
#3	World.node[3].wlan[0].mac	passedUpPk:count	6811.0
#0	World.node[3].wlan[0].mac	passedUpPk:count	6840.0

Figure 7.15: PassedUp.

- **minSNIR, packetErrorRate**: directly related with the reception of packets, it changes with the increasing value of power meaning that if the power is low the node is not able to forward the packet and thus to receive it and thus the number of errors and interference ratio is lower than in those cases where the number of transmitted packets is higher (fig. 7.16).

Replica	Module	Name	Count ▲
#1	World.node[3].wlan[0].radio	packetErrorRate:histogram	0
#5	World.node[3].wlan[0].radio	packetErrorRate:histogram	0
#4	World.node[3].wlan[0].radio	packetErrorRate:histogram	0
#7	World.node[3].wlan[0].radio	packetErrorRate:histogram	104
#8	World.node[3].wlan[0].radio	packetErrorRate:histogram	149
#9	World.node[3].wlan[0].radio	packetErrorRate:histogram	150
#2	World.node[3].wlan[0].radio	packetErrorRate:histogram	183
#3	World.node[3].wlan[0].radio	packetErrorRate:histogram	480
#0	World.node[3].wlan[0].radio	packetErrorRate:histogram	560

Figure 7.16: packetErrorRate.

7.4 Analysis

In this section, we analyse the results given trying to make the most complete information we can, further doing some assumptions (based on what it is clear from the values) and completing with some hypothesis (when there is not a clear data explaining what we are looking for).

The mechanisms we will see are valid for all the different scenarios involved, even when running any of the available extensions. The values change, the paradigms to which we reach the information are the same.

We will use the NLOSb1 type just as a random choice among the link types.

- **Arrival computation count → Reception computation count**
  They both refer to what happen in the radio medium, but the first one is also referred to the propagation mechanism of the radio medium and has values that are a little higher than the Reception counts due to the fact that at the end of the simulation some of the vehicles that propagate a transmission are trying to reach some others vehicles that already left the scenario, leading to a loss in the total count (fig. 7.17).

		REPLICA								
	ATTRIBUTE	0	1	2	3	4	5	7	8	9
MODULE: World.radioMedium.propagation	Arrival computation count	104764	104759	104771	104770	104781	104759	104766	104780	104768
MODULE: World.radioMedium	Reception computation count	104759	104759	104759	104759	104759	104759	104759	104759	104759
	Difference	5	0	12	11	22	0	7	21	9

Figure 7.17: Arrival computation count → Reception computation count.

- **Reception computation count == Radio frame send count**
  Again, they both refer to what happen in the radio medium module and thus the number of occurrences sent/received must be the same (radio medium is different to the correctness of the communications between nodes).

- **Radio frame send count == ∑(countLOS + countNLOS)**
  LOS/NLOS is something referred to the radio medium and, has already said before, it is not related to the correctness of the transmission but it occurs every time with the same counts (each node always tries to forward the same number of occurrences encountering a LOS and NLOS type link) (fig. 7.18).

	ATTRIBUTE	VALUE
MODULE: World.radioMedium.pathLoss.classifier	countLOS	45206
	countNLOSv	23278
	countNLOSb	36275
	countNLOSf	0
	SUM:	104759
MODULE: World.radioMedium	Radio frame send count	104759

Figure 7.18: Radio frame send count == ∑(countLOS + countNLOS).

- **Received Packet from Higher Layer == Sent Down Packet**
  At level 2 of the ISO/OSI stack (see figure 8.3 in Values Type section) packets are received from upper layers and then they are sent down to level 1 to be forwarded on the channel (fig. 7.19).

MODULE: World.node[nodeX].wlan.mac					
	NODE X	VALUE		NODE X	VALUE
ATTRIBUTE: rcvdPkFromHL:count	0	270	ATTRIBUTE: sentDownPk:count	0	270
	1	718		1	718
	2	269		2	269
	3	616		3	616
	4	271		4	271
	5	271		5	271
	6	778		6	778
	7	245		7	245
	8	410		8	410
	9	763		9	763
	10	241		10	241
	.....			.....	

Figure 7.19: rcvdPkFromHL == sentDownPK.

- **Received Packet from Lower Layer == Passed Up Packet**
  As we have said for HigherLayer/sentDown, here the principle is the same but from level 1 of the stack the pakctes arrive at level 2 and then from level 2 they are passed up to the upper layers.
  Before the forwarding to upper layers, a flag is set to true/false based on a correct/wrong reception of packets during the transmission.

- **Transmission count == $\sum$(rcvdPkFromHL) == $\sum$(sentDown)**
  If packets from upper layers arrive at level 2 of the ISO/OSI stack (received from higher) and then from here are forwarded to level 1 (sent down) in order to be transmitted to other nodes, then it is reasonable that the sum of this packets sent down to be transmitted is equal to the number of occurrences in the communication, indeed the transmission count (fig. 7.20).

MODULE: World.node[nodeX].wlan.mac					
	NODE X	VALUE		NODE X	VALUE
ATTRIBUTE: rcvdPkFromHL:count	0	270	ATTRIBUTE: sentDownPk:count	0	270
	1	718		1	718
	2	269		2	269
	3	616		3	616
	4	271		4	271
	5	271		5	271
	6	778		6	778
	7	245		7	245
	8	410		8	410
	9	763		9	763
	10	241		10	241
	.....			.....	
	18	552		18	552
	SUM	9260		SUM	9260
ATTRIBUTE: TRANSMISSION COUNT	9260				

Figure 7.20: Transmission count == $\sum$(rcvdPkFromHL) == $\sum$(sentDown).

7.5 Graphs

Here a couple of interesting graphs are shown.

7.5.1 Path Loss Type

In the first graph (see fig. 7.21) we can see the relation between the link type run in the simulation and the percentage of LOS/NLOS-due-to-X that has influenced the transmissions.

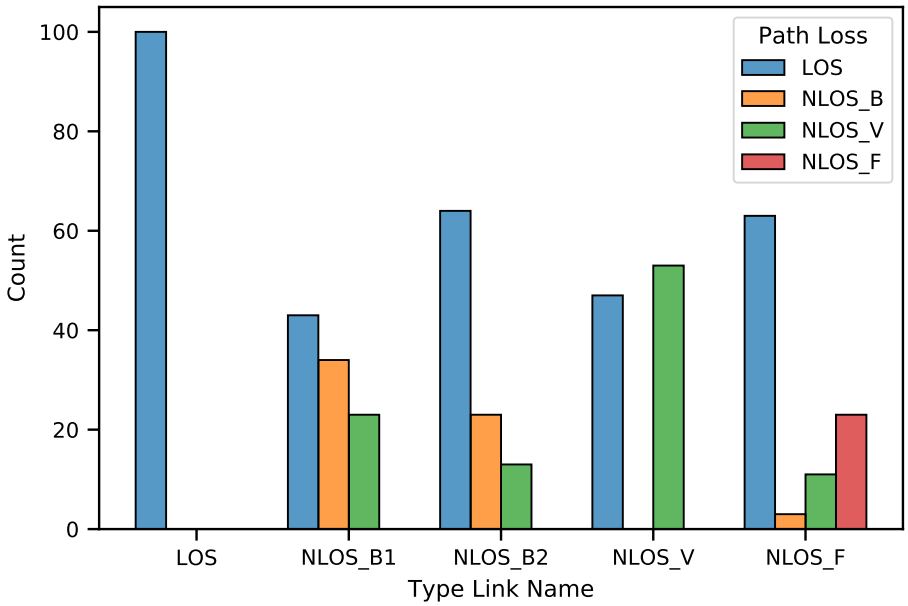


Figure 7.21: Path Loss Type graph.

On the X-axis there are the different link types that Gemv^2 is able to run; for each of them, a bar shows the percentage of path loss for each link type.
As we can see, in LOS type, where there are no obstacles blocking the communication, there is a 100% of LOS as path loss, while the foliage's obstacles is only introduced while running a NLOSf type.

7.5.2 Passed Up Packets count

In the second graph (see fig. 7.22) we can see the total amount of packets that are forwarded to the upper layer by level 2 of the ISO/OSI stack and that is represented by the passedUpPk:count data.

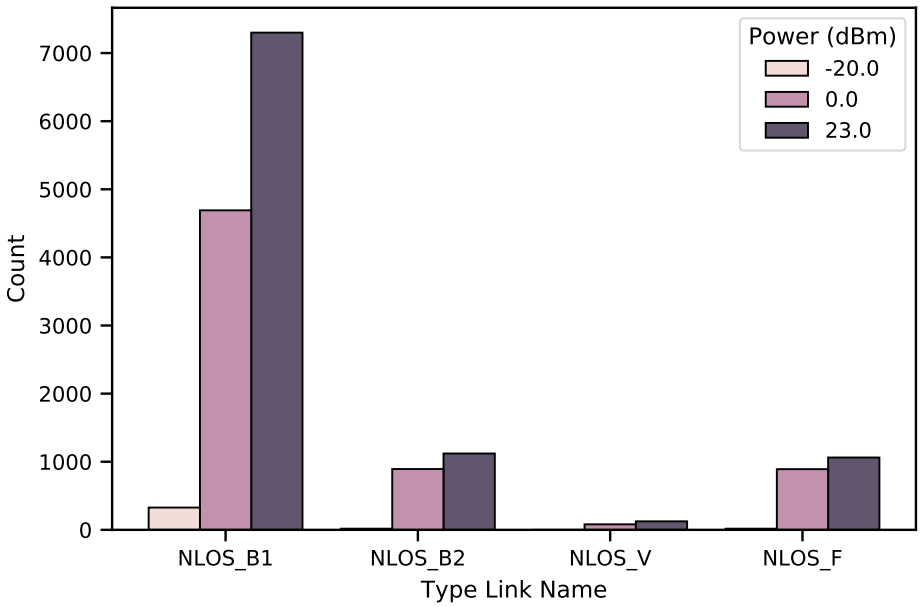


Figure 7.22: PassedUpPk:count graph.

As we have already seen in the analysis of the results, the count of the packets raises with the power: the more the transmitted power, the more the packets passing through each node.

7.6 Hypothesis

In this section, a couple of hypothesis will be done from the comparisons made on different results file.

(1) **Packet length (Bytes):**

We have seen in the previous section that packets received from Higher/Lower layers (and by consequence packets sentDown/passedUp) are the effective occurrences running through the single node and then eventually transmitted to other nodes.
By looking at the results we can see that there is a "mean" value (see fig. 7.26) when considering the vector data of such parameters, plus or minus a standard deviation. In particular, this mean value is related to the sum of Packet Bytes as mentioned by the recorded values (see fig. 7.24 and 7.25).

The 1° hypothesis we can do is indeed that the mean value is referred to the mean length in terms of Bytes that each packet has in the simulation, and which is equal to 163,... Bytes ± the standard deviation of 80,... Bytes meaning that the packet length varies from 83 Bytes to 243 Bytes.

The hypothesis takes strong confirmation directly looking at the results (see fig. 7.23): taking, from scalars values, the values given by each node at the quote sentDownPk:sum(packetBytes) and divide them by sentDownPk:count, we obtain exactly the mean value given by sentDownPk:vector(packetsBytes) as mean value.

ATTRIBUTE: Sent Down Packet					
	SUM (PACKET_BYTES)	COUNT		VECTOR (PACKET_BYTES)	
NODE	VALUE	VALUE	DIFFERENCE	MEAN VALUE	STD DEV
0	44010	270	163	163	80,...
1	117114	718	163,1114206128	163,1114206128	80,...
2	43887	269	163,1486988848	163,1486988848	80,...
3	100568	616	163,2597402597	163,2597402597	80,...
4	44333	271	163,5904059041	163,5904059041	80,...
5	44333	271	163,5904059041	163,5904059041	80,...

Figure 7.23: Tabulated values.

Module	Name	Value
World.node[0].wlan[0].mac	sentDownPk:sum(packetBytes)	44010.0
World.node[1].wlan[0].mac	sentDownPk:sum(packetBytes)	117114.0
World.node[2].wlan[0].mac	sentDownPk:sum(packetBytes)	43887.0
World.node[3].wlan[0].mac	sentDownPk:sum(packetBytes)	100568.0
World.node[4].wlan[0].mac	sentDownPk:sum(packetBytes)	44333.0
World.node[5].wlan[0].mac	sentDownPk:sum(packetBytes)	44333.0

Figure 7.24: Scalar values - sentDownPk:sum(packetBytes).

Module	Name	Value
World.node[0].wlan[0].mac	sentDownPk:count	270.0
World.node[1].wlan[0].mac	sentDownPk:count	718.0
World.node[2].wlan[0].mac	sentDownPk:count	269.0
World.node[3].wlan[0].mac	sentDownPk:count	616.0
World.node[4].wlan[0].mac	sentDownPk:count	271.0
World.node[5].wlan[0].mac	sentDownPk:count	271.0

Figure 7.25: Scalar values - sentDownPk:count.

Module	Name	Mean	StdDev
World.node[0].wlan[0].mac	sentDownPk:vector(packetBytes)	163.0	80.14856094504341
World.node[1].wlan[0].mac	sentDownPk:vector(packetBytes)	163.11142061281	80.13927108705182
World.node[2].wlan[0].mac	sentDownPk:vector(packetBytes)	163.14869888475	80.26063076572923
World.node[3].wlan[0].mac	sentDownPk:vector(packetBytes)	163.25974025974	80.25931996695736
World.node[4].wlan[0].mac	sentDownPk:vector(packetBytes)	163.59040590405	80.58824321605135
World.node[5].wlan[0].mac	sentDownPk:vector(packetBytes)	163.59040590405	80.58824321605135

Figure 7.26: Vectors values - sentDownPk:vector(packetBytes).

(2) **Effective Transmission (percentage of correctness):**

From scalars values we can see 2 data related to the count of occurrences in transmission: [a] TransmissionState:count for each node and [b] Transmission:count for the radio medium.

As we have said before, each node tries to forward a total number of occurrences (thus, packets) which is larger than the total number of correct occurrences detected at the end. Some of the packets can be lost in transmission while some others can arrive but with some errors.

We can suppose that the difference of such parameters is the percentage of correctness of the transmission that can be simply calculated with these values (fig. 7.27).

NODE	TX STATE		
0	4		
1	104		
2	212		
3	180		
4	312		
5	516		
6	20		
7	384		
8	564		
9	596		
10	368		
11	460		
12	116		
13	440		
14	36		
15	252		
16	152		
17	8		
18	516		
SUM	5240		
		TX_COUNT : SUM = X : 100	X = 25%
TX COUNT	1310		

Figure 7.27: Analysis of Transmission values in the NLOSv file.

Repeating the same analysis on all the other files (LOS, NLOSb1, NLOSb2 and NLOSf), we can observe that the value 25% is recurring in every scenario.
We can make the 2° hypothesis that Gemv^2 has a percentage of effective transmission correctness of 25%.

Chapter 8

Conclusions and Future Works

The simulation can be considered a valid instrument to simulate a real-world situation that the end-user can make use of to reduce the cost of the implementation and the difficulties rising when considering different scenarios and traffic conditions.

The Artery project gives an excellent environment where to test all the possible features and the integration with the Gemv^2 tool is optimal to complete all the set of information there is a need of, where among them stands out, for sure, the need for a path classification while transmitting.

The integration of SUMO simulator with OMNeT++ simulation platform gives a bunch of possibilities to consider non only the predefined scenarios given by Gemv^2 but also real scenarios coming from real-world, by simply updating the information about the location of the simulated landscape.

For sure, Gemv^2 has some lack of functions in its code that should be implemented to better work.

Moreover, there is a lack in what concerning the results it is able to store and strong knowledge of C++ programming language is suggested in order to complete those parts where a better storage of values is needed to complete then also the results file.

The second suggestion and purpose to continue with this work is to verify the hypothesis I personally did to confirm and, in case, complete the analysis of the tool.

What for sure has to be done is to replicate the same experiments in other simulation

platforms under the same conditions: take all the devices and items used in this work (not only the simulation parameters but also the radio environment given by INET or the control features given by Traci), put them in the new simulator and see the values it gives as output. From the output, make some comparisons with the values of this work checking the hypothesis and eventually giving some new ones.

Then, with the new results, my suggestion is to somehow "open" the vectors given by Gemv^2 in order to see the single packet and not all the set of packets that the tool is not able to automatically divide while storing. By looking at all the positions of the vector, the user should be able to better understand what it is effectively transmitted from one node to another. In addition, what could be very usefull is to reach the information regarding the flag that, at level 2 of the ISO/OSI stack, is added to the packet to report any error before pass it up to the upper layers; by looking at the flag, the user should be able to understand how many packets are effectively correct/wrong computing indeed all the error rates are needed: packet error rate, then bit error rate (knowing the packet length in terms of bytes) and symbol error rate (by knowing the modulation type used).

The software that could accomplish to this work are Network simulator, Jupiter Notebook, Pandas for the analysis of the results and the matplotlib library to plot them under a Python like environment.
The use of SQL database could be usefull too as analysis tool of the results.
The Gemv^2 used in Matlab software could also be a good starting point to replicate the simulation under the same conditions.

Finally, the implementation of the dataset formulas in the OMNeT++ simulator could be usefull to verify the information and to add some new one not recorded by the system, such as the throughput or whichever performance parameters needed by the user.

Chapter 9

Ringraziamenti

Arrivato fin qui, è doveroso esprimere al meglio la mia più profonda gratitudine a tutte quelle persone che mi sono state accanto, non solo durante il periodo di tesi ma durante tutto il lungo cammino intrappreso ormai qualche anno fa.

Un sentito grazie al mio relatore Prof. Maurizio Magarini che ha saputo guidarmi durante tutto l'arco di ricerca e stesura del lavoro, con i suoi consigli indispensabili e le sue conoscenze pratiche permettendomi di proseguire anche quando sembrava non si potesse andare oltre.

Ringrazio i dottorandi che mi hanno affiancato e che con le loro conoscenze in diversi ambiti hanno saputo darmi preziosi consigli: un grazie sincero a Dott. Francesco Linsalata, Dott. Eugenio Moro, Dott. Mehdi Haghshenas e Dott.ssa Francesca Ratti.

Ringrazio di cuore i miei genitori, Tiziana e Sergio. Grazie per avermi sempre sostenuto nei momenti più bui ed avermi permesso di portare a termine questo percorso, incoraggiandomi a non mollare mai e a mettercela sempre tutta. Sono valori che porterò sempre con me in ogni ambito del futuro che mi aspetta. Un grazie anche a tutta la mia famiglia per aver creduto in me, sempre.

Un enorme grazie a tutti quei pazzi furiosi dei miei amici, sempre pronti a festeggiare al meglio ogni traguardo.

Ringrazio la saggezza dei Pollici e la premura delle donne di Apez. Mi avete dato sempre e solo il meglio di voi, mi avete appoggiato e mi avete tollerato, mi avete reso meno testardo e più ragionevole, mi avete seguito nelle mie follie ed io vi seguirei in capo al mondo. Non chiedetemi con largo anticipo cosa farò il prossimo weekend, perchè la risposta io NON LA SOOOOOO!

Ringrazio la serenità degli Eventi Cazzuola. Mi avete trasmesso sicurezza e fiducia nei miei mezzi, mi avete dato la giusta dose di spensieratezza per godere al meglio di ogni momento passato insieme. Da ognuno di voi ho preso il meglio e se sono oggi una persona migliore è anche merito vostro. Un consiglio per il futuro? L'impepata di cozze MAI prima di un calcetto.

Ringrazio gli Ing. Seri, Ilaria e Simone, presenti fin dal primo anno. L'esperienza universitaria non sarebbe stata la stessa senza di voi. Grazie per tutte le emozioni vissute prima e dopo un esame, per le scorribande in giro per l'Europa e per il mondo, per avermi fatto emozionare dei vostri traguardi e per aver gioito insieme dei miei.

Ringrazio i miei più vecchi e cari amici, Mara e Luca, per avermi fatto rivivere i bei ricordi d'infanzia e avermi fatto sentire tutto il loro appoggio in questi mesi di tesi.

Special thanks to Ramiro 24 for giving me the chance to learn from people of such different backgrounds with their beautiful minds and personalities. Thanks to Fabio, Gianluca, Florian and Kaan to have known and lived me in all my multiple versions and still always making me feel like home. Not just roomies, but hermanos. You know bros, THAT'S LIFE!

Grazie a Ginevra, Eriona, Roberta e Greta, donne forti, determinate, eleganti. Vi ringrazio per avermi sopportato anche quando non lo meritavo, per aver portato sempre il sorriso in casa Ramiro, per avermi fatto sentire meno vecchio, per essere state indispensabili nel percorso di crescita durante tutta l'esperienza Valenciana. Vi scriverò per i programmi della serata, ma rigorosamente alle 12 in punto. Mi troverete ad aspettarvi lì, sempre a Colòn.

Bibliography

[1] C. Sommer and F. Dressler, *Vehicular Networking*. Cambridge University Press, 2014.

[2] "Artery-architecture section in artery web site."

[3] D. Jiang and L. Delgrossi, "Ieee 802.11p: Towards an international standard for wireless access in vehicular environments," in *VTC Spring 2008 - IEEE Vehicular Technology Conference*, pp. 2036–2040, 2008.

[4] V. Mannoni, V. Berg, S. Sesia, and E. Perraud, "A comparison of the V2X communication systems: ITS-G5 and C-V2X," *IEEE Vehicular Technology Conference*, vol. 2019-April, 2019.

[5] S. Eichler, "Performance evaluation of the ieee 802.11p wave communication standard," in *2007 IEEE 66th Vehicular Technology Conference*, pp. 2199–2203, 2007.

[6] M. Choudhary, "What is intelligent transport system and how it works?," 2019.

[7] "Il ruolo degli ITS Intelligent Transport Systems Architettura Telematica Italiana per il Sistema dei Trasporti," 2006.

[8] J. Santa, F. Pereniguez-Garcia, A. Moragón, and A. Skarmeta, "Experimental evaluation of cam and denm messaging services in vehicular communications," *Transportation Research Part C: Emerging Technologies*, vol. 46, p. 98–120, 09 2014.

[9] S. Kuehlmorgen, P. Schmager, A. Festag, and G. Fettweis, "Simulation-Based Evaluation of ETSI ITS-G5 and Cellular-VCS in a Real-World Road Traffic Scenario," *IEEE Vehicular Technology Conference*, vol. 2018-August, pp. 0–5, 2018.

[10] A. Turley, K. Moerman, A. Filippi, and V. Martinez, "C-ITS: Three observations on LTE-V2X and ETSI ITS-G5-A comparison," *Npx*, 2018.

[11] N. Lyamin, *Performance evaluation of safety critical ITS-G5 V2V communications for cooperative driving applications*. No. 53, 2019.

[12] D. E. S. Its, "in the 5 GHz frequency band," vol. 0, pp. 1–27, 2010.

[13] IEEE-spectrum, "Applications of Device-to-Device Communication in 5G Networks," pp. 5–7, 2020.

[14] U. N. Kar and D. K. Sanyal, "An overview of device-to-device communication in cellular networks," *ICT Express*, vol. 4, no. 4, pp. 203–208, 2018.

[15] B. Lorenzo, "5g: le tecnologie messe in campo," 2019.

[16] H. Badis and A. Rachedi, "Chapter 23 - modeling tools to evaluate the performance of wireless multi-hop networks," in *Modeling and Simulation of Computer Networks and Systems* (M. S. Obaidat, P. Nicopolitidis, and F. Zarai, eds.), pp. 653–682, Boston: Morgan Kaufmann, 2015.

[17] T. Yeferny and S. Hamad, "Vehicular Ad-hoc Networks : Architecture , Applications and Challenges," vol. 20, no. 2, pp. 1–7, 2020.

[18] D. C. Marinescu, "Chapter 5 - cloud access and cloud interconnection networks," in *Cloud Computing (Second Edition)* (D. C. Marinescu, ed.), pp. 153–194, Morgan Kaufmann, second edition ed., 2018.

[19] A. Paul, N. Chilamkurti, A. Daniel, and S. Rho, "Chapter 9 - future trends and challenges in its," in *Intelligent Vehicular Networks and Communications* (A. Paul, N. Chilamkurti, A. Daniel, and S. Rho, eds.), pp. 185–210, Elsevier, 2017.

[20] M. Soyturk, K. Muhammad, M. Avcil, B. Kantarci, and J. Matthews, "Chapter 8 - from vehicular networks to vehicular clouds in smart cities," in *Smart Cities and Homes* (M. S. Obaidat and P. Nicopolitidis, eds.), pp. 149–171, Boston: Morgan Kaufmann, 2016.

[21] A. Paul, N. Chilamkurti, A. Daniel, and S. Rho, "Chapter 4 - evaluation of vehicular network models," in *Intelligent Vehicular Networks and Communications* (A. Paul, N. Chilamkurti, A. Daniel, and S. Rho, eds.), pp. 77–112, Elsevier, 2017.

[22] V. D. V, A. Chekima, F. Wong, and J. A. Dargham, "A study on vehicular ad hoc networks," in *2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, pp. 422–426, 2015.

[23] D. Yuen, "The future begins with the road side unit," 2020.

[24] A. Festag, "Standards for vehicular communication—from IEEE 802.11p to 5G," *Elektrotechnik und Informationstechnik*, vol. 132, no. 7, pp. 409–416, 2015.

[25] D. Dhoutaut, A. Régis, and F. Spies, "Impact of radio propagation models in vehicular ad hoc networks simulations," *VANET - Proceedings of the Third ACM International Workshop on Vehicular Ad Hoc Networks*, vol. 2006, pp. 40–49, 2006.

[26] S. Chen, J. Hu, Y. Shi, Y. Peng, J. Fang, R. Zhao, and L. Zhao, "Vehicle-to-Everything (v2x) Services Supported by LTE-Based Systems and 5G," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 70–76, 2017.

[27] X. Shen, R. Fantacci, and S. Chen, "Internet of Vehicles," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 242–245, 2020.

[28] S. Wan, R. Gu, T. Umer, K. Salah, and X. Xu, "Toward offloading internet of vehicles applications in 5g networks," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–9, 2020.

[29] H. Zhou, W. Xu, J. Chen, and W. Wang, "Evolutionary v2x technologies toward the internet of vehicles: Challenges and opportunities," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 308–323, 2020.

[30] M. Boban, J. Barros, and O. K. Tonguz, "Geometry-Based Vehicle-to-Vehicle Channel Modeling for Large-Scale Simulation," vol. 63, no. 9, pp. 4146–4164, 2014.

[31] R. Meireles, M. Boban, P. Steenkiste, O. Tonguz, and J. Barros, "Experimental study on the impact of vehicular obstructions in VANETs," *2010 IEEE Vehicular Networking Conference, VNC 2010*, pp. 338–345, 2010.

[32] "Omnet++ web site."

[33] F. Saremi, "OMNet ++ OMNet," *Computer Engineering*, 2009.

[34] S. learn developers, "OMNeT++ userguide," *Computer*, no. September, pp. 1–20, 2020.

[35] "Omnet++ simulation manual, omnet++ version 5.6.1."

[36] "Veins web site."

[37] "Artery web site."

[38] A. Hegde and A. Festag, "Artery-C: An OMNeT++ Based Discrete Event Simulation Framework for Cellular V2X," *MSWiM 2020 - Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 47–51, 2020.

[39] "Artery-middleware section in artery web site."

[40] "Artery-environment section in artery web site."

[41] "Artery-storyboard section in artery web site."

[42] "Artery-gemv^2 section in artery web site."

[43] "Inet web site."

[44] "Inet-layeredprotocolbase module in inet web site."

[45] "Inet-physicalbase module in inet web site."

[46] "Inet-radio module in inet web site."

[47] "Inet-iradio module in inet web site."

[48] "Inet-mobility module in inet web site."

[49] "Inet-baseantenna module in inet web site."

[50] "Inet-isotropicantenna module in inet web site."

[51] "Inet-parabolicantenna module in inet web site."

[52] "Inet-physicalenvironment module in inet web site."

[53] "Inet-flatground module in inet web site."

[54] "Inet-scenariomanager module in inet web site."

[55] "Vanetza web site."

[56] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO–simulation of urban mobility: an overview," *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*, no. October, 2011.

[57] G. A. C. (DLR) *et al.*, "Sumo."

[58] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J. P. Hubaux, "TraCI: An interface for coupling road traffic and network simulators," *Proceedings of the 11th Communications and Networking Simulation Symposium, CNS'08*, no. April, pp. 155–163, 2008.

[59] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, (New York, NY, USA), p. 47–57, Association for Computing Machinery, 1984.

[60] M. Boban, J. Barros, and O. Tonguz, "Geometry-based vehicle-to-vehicle channel modeling for large-scale simulation," *IEEE Transactions on Vehicular Technology*, vol. 63, pp. 4146–4164, Nov 2014.

Appendix A

Installation Guide

A.1 Introduction

A.1.1 Ubuntu 20.04 LTS

The following installation guide is performed under the assumptions of Ubuntu 20.04 LTS as the Operating System.

- **Ubuntu 20.04 LTS**

A.1.2 Suggestions

Before starting with the installation of the programs, it is always better to start from the very beginning in order to delete possible packages that could be counterproductive in our installation. If you are using a Linux-based pc, re-initialize the system with a new installation of Ubuntu. If you are using a Virtual machine, delete it and create a new one from zero.

When downloading anything, move it to the Home folder and always work from there.

A.1.3 Synaptic

First thing to do, refresh the database of available packages. Type in the terminal:

$ sudo apt-get update

Install the Synaptic application for a better control when installing some of the needed packages. Open a new terminal and type:

$ sudo apt install synaptic

Since software installation requires root permissions, it will ask you to type your password.

A.1.4 Git

Install the Git tool to clone programs from extern (e.g. GitHub). Type in the terminal:

$ sudo apt install git

A.2 OMNeT++

A.2.1 Installing the Prerequisite Packages

OMNeT++ requires several packages to be installed on the computer. These packages include the C++ compiler (gcc or clang), the Java runtime, and several other libraries and programs. You can perform the installation using the graphical user interface or from the terminal, whichever you prefer. Here, the Command-Line Installation is shown.

First thing to do, refresh the database of available packages. Type in the terminal:

$ sudo apt-get update

To install the required packages, type in the terminal:

$ sudo apt-get install build-essential gcc g++ bison flex perl python python3 qt5-default libqt5opengl5-dev tcl-dev tk-dev libxml2-dev zlib1g-dev default-jre doxygen graphviz

At the confirmation question (Do you want to continue? [Y/N]), answer Y.

Open the Synaptic application typing "sudo synaptic" in the terminal. From the

searching window, type the following package name: "libweb". Click the squares before EACH and EVERY names where "libweb" is shown, then choose Mark for installation. If the Mark additional required changes? dialog comes up, choose the Mark button. When the installation of the packages is complete, close synaptic and go back to the terminal.

Now you have to add the ubuntugis/ppa repository manually to your software sources:

```
$ sudo add-apt-repository ppa:ubuntugis/ppa
$ sudo apt-get update
```

To use Qtenv with 3D visualization support, install the development packages for OpenSceneGraph (3.2 or later) and the osgEarth (2.7 or later) packages:

```
$ sudo apt-get install openscenegraph-plugin-osgearth libosgearth-dev
```

To enable the optional parallel simulation support you will need to install the MPI packages:

```
$ sudo apt-get install openmpi-bin libopenmpi-dev
```

The optional Pcap library allows simulation models to capture and transmit network packets bypassing the operating system's protocol stack. It is not used directly by OMNeT++, but models may need it to support network emulation.

```
$ sudo apt-get install libpcap-dev
```

At this point, it is better to restart the system.

A.2.2 Downloading and Unpacking

Download OMNeT++ from http://omnetpp.org. Make sure you select to download the generic archive, omnetpp-5.6.2-src-linux.tgz. Copy the archive to the directory where you want to install it. This is usually your home directory, /home/you. Open a terminal, and extract the archive using the following command:

$ tar xvfz omnetpp-5.6.2-src-linux.tgz

This will create an omnetpp-5.6.2 subdirectory with the OMNeT++ files in it.

A.2.3 Environment Variables

OMNeT++ needs its bin/ directory to be in the path. To add bin/ to PATH temporarily, change into the OMNeT++ directory and source the setenv script:

$ cd omnetpp-5.6.2
$. setenv

To set the environment variables permanently, edit .bashrc in your home directory. Use your favourite text editor to edit .bashrc, for example gedit:

$ gedit ~/.bashrc

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

export PATH=$HOME/omnetpp-5.6.2/bin:$PATH

You need to restart the pc for the changes to take effect.

A.2.4 Configuring and Building

In the top-level OMNeT++ directory, type:

$ cd omnetpp-5.6.2
$ ./configure

The configure script detects installed software and configuration of your system. It writes the results into the Makefile.inc file, which will be read by the makefiles during the build process.

When ./configure has finished, you can compile OMNeT++. Type in the terminal:

$ make

A.2.5 Verify the Installation

You can now verify that the sample simulations run correctly. For example, the dyna simulation is started by entering the following commands:

$ cd samples/dyna
$ ./dyna

By default, the samples will run using the Tcl/Tk environment. You should see nice gui windows and dialogs.

A.2.6 Starting the IDE

You can launch the OMNeT++ Simulation IDE by typing the following command in the terminal:

$ omnetpp

When the dialog window comes up, go to the "Workspace" option.
Then, let OMNeT++ install the INET framework automatically (it would take some minutes to install it).

A.3 SUMO

A.3.1 Installing the Prerequisite Packages

To install the required packages, type in the terminal:

$ sudo apt-get install cmake python g++ libxerces-c-dev libfox-1.6-dev libgdal-dev libproj-dev libgl2ps-dev libavformat-dev libswscale-dev libopenscenegraph-dev python3-dev swig libgtest-dev libeigen3-dev python3-pip python3-setuptools default-jdk

$ sudo pip3 install texttest

A.3.2 Installation

Install the basic version of SUMO. Type in the terminal:

$ sudo apt-get install sumo sumo-tools sumo-doc

Then, update the ppa repository for a more recent version:

$ sudo add-apt-repository ppa:sumo/stable
$ sudo apt-get update

Then redo the installation:

$ sudo apt-get install sumo sumo-tools sumo-doc

Now the SUMO-1.8.0 version is installed and run correctly.

A.4 Artery

A.4.1 Requirements

Install some prerequisite packages. Type in the terminal:

$ sudo apt-get install python python3 libgeographic-dev libcrypto++-dev libboost-dev libboost-date-time-dev libboost-system-dev

At this point we need to be sure that all requirements needed by Artery are fullfiled before proceding with its installation.
Open Synaptic application, search the following packages and do what it is written for each of them:

- **cmake:** Mark and Install every package containing the word "cmake" in the name.
- **python:** Mark and Install every package containing the word "libpython" + the version of the package already installed with this name (e.g. if libpython3.8 has already some packages installed, Mark and Install the remaining libpython3.8 packages).
- **geographicLib:** Mark and Install every package that it is found when searching "geographiclib".
- **crypto++:** Mark and Install every package that it is found when searching "crypto++".

A.4.2 Cloning the Artery repository

All source code of Artery is stored in a git repository on GitHub. This repository contains compatible versions of INET, SimuLTE, Veins, Vanetza and some other third-party dependencies as git submodules. You can find those submodules in the extern subdirectory.

The command below will clone Artery's master branch along with matching versions of its submodules onto your machine's file system. Type in the terminal:

$ git clone –recurse-submodule https://github.com/riebl/artery.git
(before "recurse", two minus signs are written: - - but with no space in them).

A.4.3 Create build directory

From ARTERY_PATH create a build directory for Artery, configure the build directory with CMake and finally build Artery there. Type in the terminal:

$ cd artery
$ mkdir build
$ cd build
$ cmake ..
$ cmake –build .
(before "build", two minus signs are written: - - but with no space in them).

A.5 Getting started

A.5.1 Import Artery project

Now that all the tools and packages are correctly installed on your pc, you need to import the Artery project into the OMNeT++ workspace.
Open a new OMNeT++ session, typing in the terminal:

$ omnetpp

Now follow these instructions:

File - import - General - existing project into workspace - root dir:Browse - home:Artery

- open - Options(check the square box): search for nested projects - Options(check the square box): copy projects into worspace - Finish

At the end of the process (be sure that everything is uploaded, e.g. C/C++ indexer at the bottom of the window that takes much time), do:

File - Open Projects from File System... - import source:directory/artery - Finish

Now you can see that in the set of available projects that can be run in OMNeT++ (left window) there is the Artery project and also other related projects such as Veins, Lte, etc...

Click on:

Project - Build All

Now you are able to run the Artery tool from the terminal and then see the file and the results by using OMNeT++ ide.

To run artery type in the terminal:

$ cd artery
(to move from home to artery directory)
$ cmake –build build –target run_name
(before "build" and "target", two minus signs are written: - - but with no space in them).

Where "name" is one of the available scenarios you can find in the Artery - build - scenarios directory.