Contents

List of Figures

List of Tables

Chapter 1

Tools and applications

In this chapter I will describe the tools and softwares I have utilized to carry out the simulations.

1.1 Radio Propagation Model

A Radio Propagation Model (RPM) is an empirical mathematical formulation used to model the propagation of radio waves as a function of frequency, distance, transmission power and other variables. Over the years various RPMs have been developed, some aiming at modelling a general situation, while other more useful in specific scenarios. For example, we can range from the more general free space model, where only distance and power are considered, to more complex models which account for shadowing, reflection, scattering and other multipath losses. Moreover, it is important to keep into consideration the computational complexity and scalability of the model: some have poor accuracy but are scalable, while others have very good accuracy but can only work for small sets of nodes. As always, it is very important to find the right tradeoff between complexity and accuracy.

The authors of **6298165** classify the propagation models offered by the network simulator ns-3 in three different categories:

* **Abstract** propagation loss models, for example the Maximal Range model (also known as Unit Disk), which establishes that all transmissions within a certain range are all received without any loss
* **Deterministic** path loss models, such as the Friis propagation model, which models quadratic path loss as it occurs in free space, and Two Ray Ground, which assume propagation via two rays: a direct (LOS) one, and the one reflected by the ground.
* **Stochastic** fading models such as the Nakagami model, which uses stochastic distributions to model path loss.

These traditional models, especially the stochastic ones, work quite well to describe the wireless channel characteristics from a macroscopic point of view. However, given the probabilistic nature of the model, single transmissions are no affected by mesoscopic and microscopic effects of the sorrounding environment. To keep these effects into consideration , researchers have utilized Ray-Tracing, a geometrical optics technique used determine all possible signal paths between the transmitter and the receiver, considering reflection, diffraction and scattering of radio waves, suitable both for 2D and 3D scenarios **245274 765022**.

However, a Ray-Tracing based approach, while producing a fairly accurate model, is not very scalable due to its high computational complexity, especially in a real-time scenario. To overcome this problem, the authors of **STEPANOV200861** have resorted to a fairly computationally expensive pre-processing, but this leads to the need of pre-processing every scenario (and also every change in the scenario).

1.2 Obstacle Shadowing propagation loss model

The original thesis **ROM2017**, after having analyzed various works concerning shadowing in urban scenarios **Giordano:2010:CST:1860058.1860065 4020783** used a deterministic RPM called Obstacle Shadowing propagation loss model presented in **5720204** and implemented by the authors of **Carpenter:2015:OMI:2756509.2756512**. This propagation model calculates the loss in signal strength due to the shadowing effect of obstacles such as buildings.

1.3 Network Simulator 3

Network Simulator 3 (ns-3) is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. ns-3 is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use.

ns-3 development began in 2006 by a team lead by Tom Henderson, George Riley, Sally Floyd and Sumit Roy. Its first version was released on June 30, 2008.

ns-3 is the successor of ns-2, released in 1989. The fact that the former was built from scratch makes it impossible to have backward compatibility. In fact, ns-2 used oTCL scripting language to describe network topologies and C++ to write the core of the simulation. This choice was due to avoid the very time consuming C++ code recompilation, exploiting the interpreted language oTCL. ns-2 mixed the “fast to run, slow to change” C++ with the “slow to run, fast to change” oTCL language. Since compilation time was not an issue with modern computing capabilities, ns-3 developers chose to utilize exclusively C++ code (and optional Python bindings) to develop simulations.

1.3.1 Modules and module structure

ns-3 is composed of various modules, which are groups of classes, examples and tests each related to a certain feature. The components of a module work in a cohesive way in order to offer APIs to other modules and users. Some examples of built-in modules are:

- WiFi;
- AODV;
- CSMA.

The obstacle shadowing propagation loss model and the Fast Broadcast algorithm have been implemented as modules too.

Modules follow a prototypical structure in order to promote clarity and offer built-in documentation. Figure 1.1 shows the typical module structure.

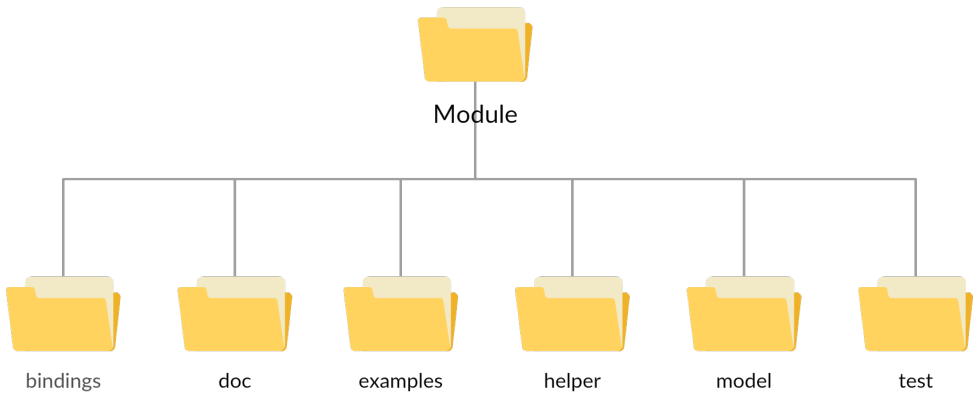


Figure 1.1: ns-3 module structure

The following directories can be found inside a module's root directory:

* **bindings:** Python bindings used to make the module's API compatible with Python;
* **doc:** documentation of the module;
* **examples:** examples and proof of concepts of what can be done using the module;
* **helper:** higher level APIs to make the module easier to use;
* **model:** headers and source files which implement the module's logic;
* **test:** test suite and test cases to test the module.

1.3.2 Key elements

The element at the base of ns-3 is called *node*, instance of `ns3::Node`. A node can be thought of as a shell of a computer. Various other elements can be added to nodes, such as:

* NetDevices (e.g. NICs), which enable nodes to communicate over *channels*;
* protocols;
* applications.

It is the last those which implement the logic of a simulation. For example, the `UdoEchoClientApplication` and `UdpEchoServerApplication` can be used to implement a client/server application which exchange and print the packets' content over the network. The Fast Broadcast protocol has been implemented as an application as well.

The *channels* model various type of transmission media, such as the wired and the wireless ones.

1.3.3 Structure of a simulation

A simulation can be implemented in many ways, but in most cases the following steps are executed:

* manage command line arguments (e.g. number of nodes to consider in the simulation, transmission range, etc.);
* initialize all the necessary fields in classes;
* create nodes;
* set up physical and MAC layers;
* set up link layer, routing protocols and addresses;
* configure and install applications on nodes;
* position nodes and (optionally) give them a mobility model;
* schedule user defined events, such as transmissions of packets;
* start the simulation;
* collect and manage output data.

1.3.4 NetAnim

Netanim is an offline animator tool based on the Qt toolkit. It collects an XML tracefile during the execution of a simulation and can be used to animate the simulation, analyzing packet transmissions and contents.

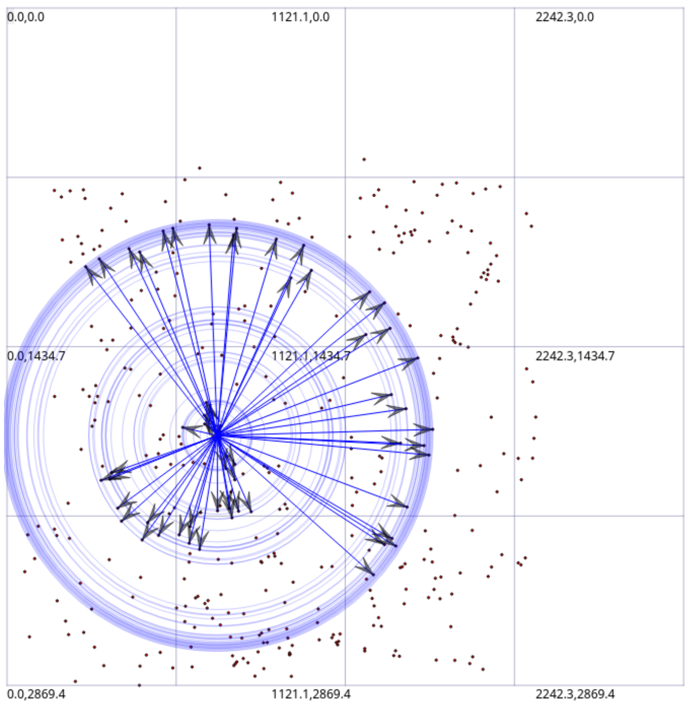


Figure 1.2: Packet transmission in NetAnim

1.4 Simulation of Urban MObility

Simulation of Urban MObility is an open-source road traffic simulation package. It is written in C++ and licensed under GPLv3.

It offers different tools to analyze and manage real maps from the urban mobility point of view, including pedestrian movement and various types of vehicles.

The original work **ROM2017** utilized SUMO to produce, starting from real maps obtained from OpenStreetMap (OSM), two files:

1. a `.poly` file using the SUMO tool Polyconvert. This file contains information about all the obstacles, such as buildings, useful for the Obstacle Shadowing module.

2. a `.ns2mobility` file using the SUMO tool TraceExporter. This file contains information about the vehicles and their positioning.

The process of generating these two files necessary for ns-3 simulations requires some intermediate steps. The full process is represented in figure 1.3.

The original work considered a only a distance of 25 meters between vehicles; this work considers various distances, ranging from 15 to 45 meters. Figure 1.4 shows the same scenario (Padua) with different distances between vehicles.

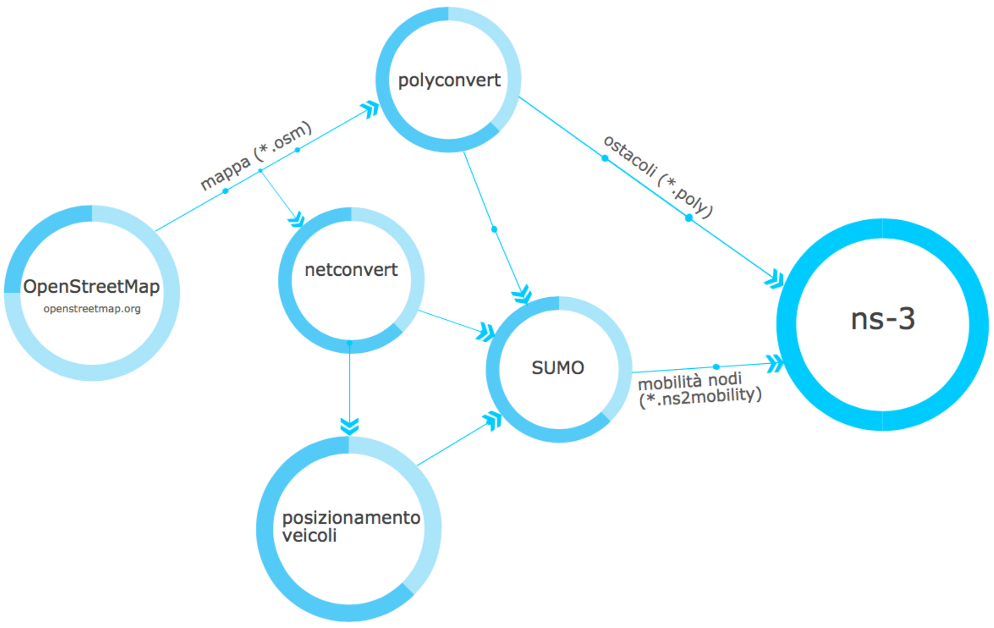
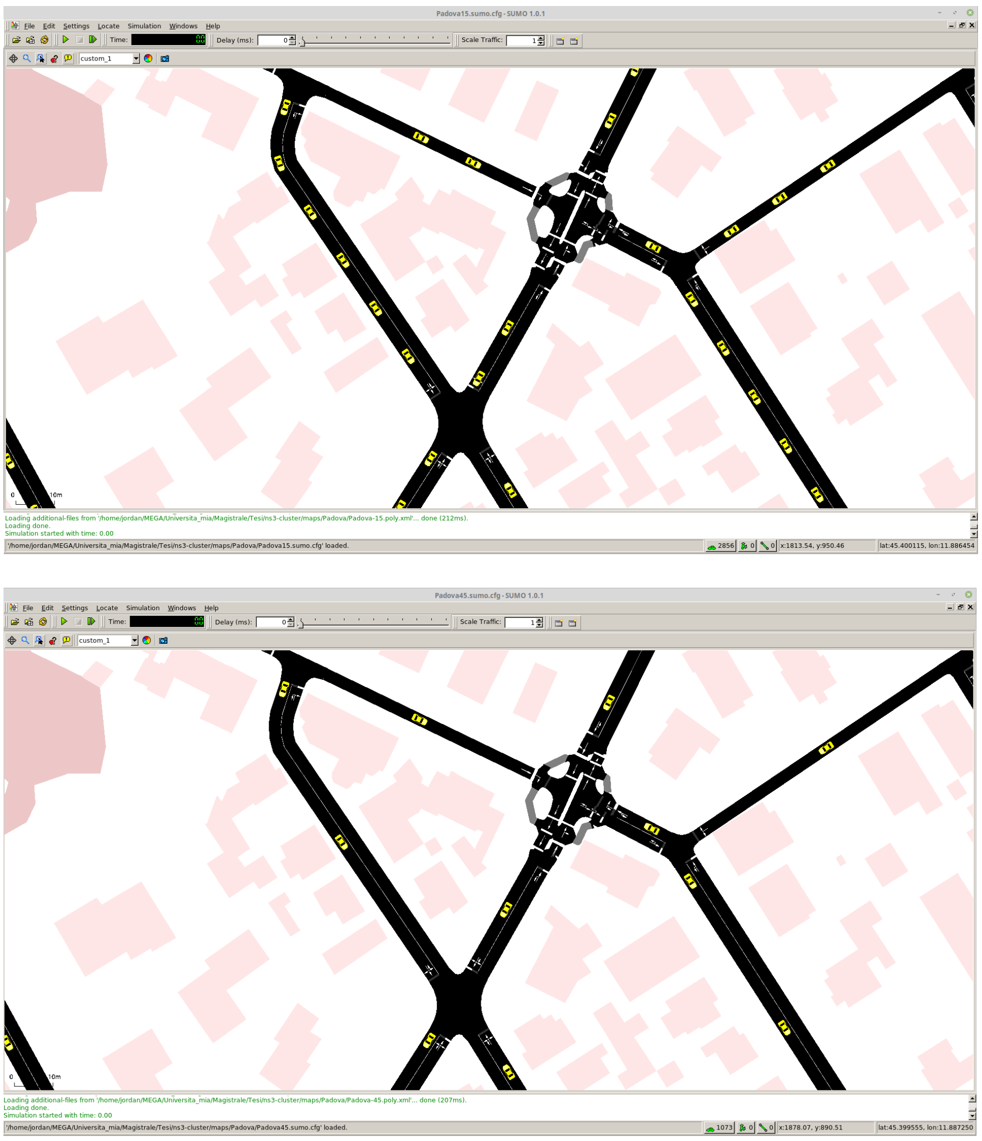


Figure 1.3: Steps to generate necessary files using SUMO (**ROM2017**)


Padova15.sumo.cfg - SUMO 1.0.1
File Edit Settings Locate Simulation Windows Help
Time:
Delay (ms): 0
Scale Traffic: 1
custom_1
0 10m
Padova45.sumo.cfg - SUMO 1.0.1
File Edit Settings Locate Simulation Windows Help
Time:
Delay (ms): 0
Scale Traffic: 1
custom_1
0 10m

```
Loading additional-files from '/home/jordan/MEGA/Universita_mia/Magistrale/Tesi/ns3-cluster/maps/Padova/Padova-15.poly.xml'... done (212ms).
Loading done.
Simulation started with time: 0.00
'/home/jordan/MEGA/Universita_mia/Magistrale/Tesi/ns3-cluster/maps/Padova/Padova15.sumo.cfg' loaded.   2856   0   0   x:1813.54, y:950.46   lat:45.400115, lon:11.886454
```

```
Loading additional-files from '/home/jordan/MEGA/Universita_mia/Magistrale/Tesi/ns3-cluster/maps/Padova/Padova-45.poly.xml'... done (207ms).
Loading done.
Simulation started with time: 0.00
'/home/jordan/MEGA/Universita_mia/Magistrale/Tesi/ns3-cluster/maps/Padova/Padova45.sumo.cfg' loaded.   1073   0   0   x:1878.07, y:890.51   lat:45.399555, lon:11.887250
```

Figure 1.4: Padua scenario with vehicle distance equals to 15 meters (top) and 45 meters (bottom). Vehicles painted yellow

Chapter 2

Fast Broadcast

Fast Broadcast **4199282** is a multi-hop routing protocol for vehicular communication. Its main feature consists in breaking the assumption that all vehicles should know, a priori, their fixed and constant transmission range. This assumption is often unreasonable, especially in VANETs and urban environments, where electromagnetic interferences and obstacles such as buildings heavily influence the transmission range.

Fast Broadcast employs two different phases:

1. the **Estimation Phase**, during which cars estimate their frontward and backward transmission range;
2. the **Broadcasts Phase**, during which a car sends an alert message and the other cars need to forward it.

2.1 Estimation Phase

During this phase, cars try to estimate their frontward and backward transmission range by the means of Hello messages. These beacons are sent periodically via broadcast to all the neighbors of a vehicle.

Time is divided into turns and, in order to keep estimations fresh, data collected during a certain turn is kept for the duration of the next turn, then discarded. The parameter *turnSize* specifies the duration of a turn: the authors suggest a duration of one second. A bigger *turnSize* could guarantee less collisions to the detriment of freshness of information. On the other hand, the effects of smaller *turnSize* are specular to those just presented.

Using this approach, vehicles can estimate two different values:

∗ ***Current-turn Maximum Front Range*** **(CMFR)**, which estimates the maximum frontward distance from which another car can be heard by the considered one;

∗ ***Current-turn Maximum Back Range*** **(CMFR)**, which estimates the maximum backward distance at which the considered car can be heard.

When the turn expires, the value of these variables is stored in the LMFR and LMBR variables (*Latest-turn Maximum Front Range* and *Latest-turn Maximum Back Range*, respectively). The algorithm uses both last turn and current turn data because the former guarantees values calculated with a larger pool of Hello messages, while the latter considers fresher information.

When sending a Hello Message (algorithm 1), the vehicle inially waits for a random time between 0 and *turnSize*. After this, if it has not heard another Hello Message or a collision, it proceeds to transmit a Hello Message containing the estimation of its frontward transmission range.

When receiving a Hello Message (algorithm 2), the vehicle retrives its distance and the sender's distance, calculates the distance between these two positions and then updates the CMFR field if the message comes from ahead, otherwise CMBR is updated. The new value is obtained as the maximum between the old CMFR or CMBR value, the distance between the vehicle and the sender and the sender's transmission range estimation included in the Hello Message.

Algorithm 1 Hello message sending procedure

```
1: for each turn do
2:     sendingTime ← random(turnSize)
3:     wait(sendingTime)
4:     if ¬ (heardHelloMsg() ∨ heardCollision()) then
5:         helloMsg.declaredMaxRange ← max(LMFR, CMFR)
6:         transmit(helloMsg)
7:     end if
8: end for
```

Algorithm 2 Hello message receiving procedure

```
1: mp ← myPosition()
2: sp ← helloMsg.senderPosition
3: drm ← helloMsg.declaredMaxRange
4: d ← distance(mp, sp)
5: if receivedFromFront(helloMsg) then
6:     CMFR ← max(CMFR, d, drm)
7: else
8:     CMBR ← max(CMBR, d, drm)
9: end if
```

2.2 Broadcast Phase

This phase is activated once a car, the Anomalous Vehicles (AV), sends an Alert Message. The cars can exploit the estimation of transmission ranges to reduce redundancy. Each car can exploit this information to assign itself a forwarding priority inversely proportional to the relative distance: the higher the relative distance, the higher the priority.

When the Broadcast Phase is activated , a vehicle sends an alert message with application specific data. Broadcast specific data is also piggybacked on the AM, such as:

* **MaxRange:** the maximum range a transmission is expected to travel backward before the signal becomes too weak to be received. This value is utilized by following vehicles to rank their forwarding priority;

- **SenderPosition**: the coordinates of the sender.

Upon reception, each vehicle waits for a random time called *Contention Window* (CW). This window ranges from a minimum value (CWMin) and a maximum one (CWMax) depending on sending/forwarding car distance (Distance) and on the estimated transmission range (MaxRange), according to formula 2.1. It is quite easy to see that the higher the sender/forwarder distance is, the lower the contention window is.

$$\left\lfloor \left(\frac{\text{MaxRange} - \text{Distance}}{\text{MaxRange}} \times (\text{CWMax} - \text{CWMin}) \right) + \text{CWmin} \right\rfloor \qquad (2.1)$$

If another forwarding of the same message coming from behind is heard during waiting time, the vehicle suppresses its transmission because the message has already been forwarded by a another vehicle further back in the column. On the contrary, if the same message is heard coming from the front, the procedure is restarted using the new parameters. The vehicle can forward the message only if the waiting time expires without having received the same message.

Algorithm 3 and 4 describe the logic behind the Broadcast Phase.

Algorithm 3 Alert Message generation procedure

```
1: alertMessage.maxRange ← max(LMBR, CMBR)
2: alertMessage.position ← retrievePosition()
3: transmit(alertMessage) ← helloMsg.declaredMaxRange
```

Algorithm 4 Alert Message generation procedure

```
1: cwnd ← computeCwnd()
2: waitTime ← retrievePosition()
3: wait(waitTime
4: if sameBroadcastHeardFromBack() then
5:     exit()
6: else if s thenameBroadcastHeardFromFront()
7:     restartBroadcastProcedure()
8: else
9:     maxRange ← max(LMBR, CMBR)
10: end if
```

2.3 Two dimensions extension

The original work **4199282** considered only a strip-shaped road, where it was easy to define directions and establish when a message came from the front or the back. In **BAR2017** an extension considering two dimensions was proposed.

The modifications to the Fast Broadcast algorithm are the following:

1. Utilizing only one parameter between CMBR and CMFR (thus considering only CMR):

2. Including the position of the vehicle which originally generated the Alert Message in addition to the position of the sender of the message.

When a vehicle receives an Alert Message, the origin-vehicle distance is confronted with the origin-sender distance: the vehicle can forward the message only if the former is greater than or equal to the latter, otherwise it simply discards the message.

Acronyms

RPM	Radio Propagation Model. 1, 2
VANET	Vehicular Ad-Hoc Network. 7