Project work of Mobile Systems M

Mattia Mazzoli

July 15, 2022

1 Introduction

The aim of this project work is to develop a simple vehicular dissemination protocol and simulate it using traces of mobility obtained from real data that the city of Bologna provides as Open Data. Simulations of vehicular communication protocols are very important to be able to improve the protocol and its parameters in the best way, and to evaluate if it will work well in the expected conditions. For this purpose, considering that nowadays the communication between vehicle and infrastructure and vehicle to vehicle (V2X) becomes a key feature for the future, it is increasingly important to be able to simulate these protocols in the most realistic way. The realistic traffic flows play a crucial role in the simulation; in fact, if a simulation is performed with a completely different flow of traffic compared to the real one, it is impossible to know how the protocol will work in real conditions. Therefore, in this project a new approach using Cadyts tool has been tested to retrieve traffic flows from real trace of mobility.

In particular, in the first chapter main software and tools used are described: SUMO as traffic simulator, Cadyts as tool used to retrieve traffic flows and Veins and OMNeT++ to implement dissemination protocol and perform the final simulation. The second chapter contains a deep description of different phases of implementation, from network creation to different simulations performed. Finally, a description of results and a brief conclusion are provided in the last two chapters. All the code developed is freely accessible on GitHub [1].

2 Software and tools

2.1 SUMO

SUMO [2] stands for Simulation of Urban MObility and is an open source, microscopic and continuous multi-modal traffic simulation suite. It is a very powerful tool that permit to simulate vehicular traffic, pedestrians, traffic lights and more, and provides a lot of parameters to evaluate the simulation. For example, it can be used to evaluate V2X protocols and applications, autonomous driving functions, traffic safety analysis, traffic lights performance and calculation of emissions. SUMO is not a single application, but it is composed by different packages, that can be used in combination or standalone.

In order to perform a simulation in SUMO, at least two files must be provided. The first one is a xml file that contains a list of roads and intersections, described in a specific format, that permit to recreate a map of the region of interest. The second one is a file that contains a list of routes that represent a path for each vehicle. A brief example can be seen in figure 1 and figure 2 respectively. In addition, a lot of other information can be provided to SUMO to improve the simulation or to simulate some specific real cases. For example, it is possible to add traffic light information, vehicle type information, pedestrian, bicycle, public transport, electric vehicle and so on.

```
    <type id="highway.trunk" priority="13" numLanes="2" speed="27.78" disallow="pedestrian bicycle tram rail_urban rail rail_electric rail_fast ship" oneway="0"/>
    <type id="highway.trunk_link" priority="8" numLanes="1" speed="22.22" disallow="pedestrian bicycle tram rail_urban rail rail_electric rail_fast ship" oneway="0"/>
    <type id="highway.unclassified" priority="4" numLanes="1" speed="13.89" disallow="tram rail_urban rail rail_electric rail_fast ship" oneway="0"/>
    <type id="highway.unsurfaced" priority="1" numLanes="1" speed="8.33" disallow="tram rail_urban rail rail_electric rail_fast ship" oneway="0"/>
    <type id="railway.highspeed" priority="21" numLanes="1" speed="69.44" allow="rail_fast" oneway="1"/>
    <type id="railway.light_rail" priority="19" numLanes="1" speed="27.78" allow="rail_urban" oneway="1"/>
    <type id="railway.preserved" priority="16" numLanes="1" speed="27.78" allow="rail" oneway="1"/>
    <type id="railway.rail" priority="20" numLanes="1" speed="44.44" allow="rail" oneway="1"/>
    <type id="railway.subway" priority="18" numLanes="1" speed="27.78" allow="rail_urban" oneway="1"/>
    <type id="railway.tram" priority="17" numLanes="1" speed="13.89" allow="tram" oneway="1"/>

    <edge id=":1195439202_0" function="internal">
        <lane id=":1195439202_0_0" index="0" disallow="tram rail_urban rail rail_electric rail_fast ship" speed="3.65" length="4.67" shape="432.72,-0.59 433.91,-1.41 434.80,-1.49 435.40,-0.82 435.70,0.59"/>
    </edge>
    <edge id=":1203429304_0" function="internal">
        <lane id=":1203429304_0_0" index="0" disallow="tram rail_urban rail rail_electric rail_fast ship" speed="3.65" length="4.67" shape="927.92,1188.82 927.30,1190.12 926.57,1190.63 925.72,1190.35
924.76,1189.28"/>
    </edge>
    <edge id=":1336557126_0" function="internal">
        <lane id=":1336557126_0_0" index="0" disallow="tram rail_urban rail rail_electric rail_fast ship" speed="19.21" length="9.30" shape="302.21,1022.30 304.34,1024.07 305.79,1025.37 307.38,1026.36
309.95,1027.20"/>
    </edge>
```

Figure 1: Example of network file.

In this project, SUMO has been used with traces of mobility obtained by real data of the city of Bologna. Real data have been elaborated using Cadyts (a tool that will be described in detail in section 2.2). Also the xml file, that contains roads and intersections, has been obtained from real data, in particular from OpenStreetMap data.

```
    <vehicle id="5" depart="4.00" departLane="best" departPos="last">
        <route edges="23290112 23290114#0 475866696 230599229 479678077#0 479678076 479678075 479678074 325532353#0 325532353#1 163787528 248200387#0 248200387#1 248200387#5 163787530#1 163787530#3
163787530#4 163787530#6 163787530#7 163787530#8 163787530#9 163787530#10 163787530#11 163787530#12 163787530#14 163787530#15 163787530#16 163787530#17 163787530#19 163787530#20 163787530#21 163787530#23
81470426304#0 2328894#0 2328894#3 2328894#5 2328894#6 2328894#7 2328894#8 23837911#0 23837911#1-AddedOnRampEdge 23837911#1" exitTimes="10.41 29.82 34.41 37.43 39.07 39.23 39.31 39.82 40.63 41.84 43.29
46.85 50.93 53.43 55.34 59.57 63.79 65.42 66.93 68.14 69.06 69.70 71.61 72.20 73.18 74.56 78.73 82.24 85.44 85.62 88.67 110.79 125.35 140.74 149.78 151.33 151.85 161.72 179.94 182.32 188.62 198.49"/>
    </vehicle>
    <vehicle id="6" depart="4.80" departLane="best" departPos="last">
        <route edges="-35933885 238873859 238873860 -96199157#1 914176829#0 914176829#2 914176829#3 35898872#0 35898872#2 35898872#3 35898872#4 35898872#5 35898872#6 914176830 -395970161#1 -914176831#1
230601119 23600621 -96199160#1 -96199160#0 -454925330 -914127386 -231109180 2328894#3 2328894#5 23288944#6 2328894#7 2328894#8 23837911#0 23837911#1-AddedOnRampEdge 2383791l#1" exitTimes="16.32 18.86
23.18 35.61 48.16 59.05 65.23 69.59 72.56 76.50 80.24 86.87 96.63 100.84 117.43 127.17 128.49 130.14 138.80 143.90 152.24 156.94 185.21 194.25 195.80 196.32 206.19 224.41 226.79 233.09 242.96"/>
    </vehicle>
```

Figure 2: Example of routes file.

Another important part of SUMO that has been used is TraCI (Traffic Control Interface); it uses a TCP based client/server architecture to provide access to a running simulation, allowing to retrieve values of simulated objects and to manipulate their behaviour online. In this way other applications (OMNeT++ and Veins in this case) can use SUMO to perform the road traffic simulation and implement over it a dissemination protocol.

2.2 Cadyts

Cadyts [3] is a Java tool developed by Gunnar Flötteröd and it estimates disaggregate demand models of dynamic traffic assignment simulators from traffic counts and vehicle re-identification data. It has been used to calibrate traffic simulators such as SUMO, MATSim and DRACULA. So, it has been developed to be a solution for the dynamic traffic assignment (DTA) problem, which is the process of generating route flows based on an origin-to-destination (OD) demand model. Cadyts adopts a mathematically well-defined stochastic process view on the simulation and calibrates its behavioral parameters in a Bayesian setting from traffic counts and/or vehicle re-identification data.

The general idea is to provide some information about traffic (average flow rate or total vehicle count in a given time interval) to Cadyts, and the tool tries to match generated route file with the real one. This is done in a iterative way: first Cadyts selects a route for each vehicle and then performs a simulation to check if the result is realistic or not. In this way a more realistic simulation should be obtained, because in SUMO default tool uses a method that can't take into account real data but consider only shortest and fastest route.
Cadyts performs three different steps in an iterative way:

1. Before the simulation is started, it needs to be initialized and must be provided the measurement data (real flow data). This is called the initialization step;

2. It must have a chance to affect the agents' plan choice and/or their behavioral parameters, for example changing starting time. This is called the choice step.

3. It must observe the simulated network conditions in order to compare them to the sensor data. This is called the update step.

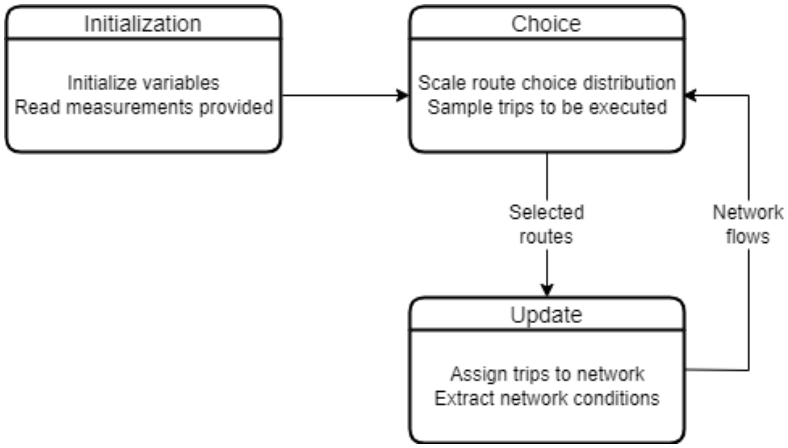

Figure 3: Cadyts steps.

2.3 Veins & OMNeT++

OMNeT++ [4] is a simulation library and framework. It is very extensible, modular and primarily used for building network simulators. With "network", it is intended a wide kind of networks that includes wired and wireless communication networks,

on-chip networks, queueing networks, and so on. Domain specific functionality is provided by model frameworks, developed as independent project. OMNeT++ uses a component architecture for models and each component is programmed in C++ and then assembled into larger components using a high level language. The main network protocols are already available as external frameworks, in particular INET is one of the most used and well known. It contains models for the Internet stack (TCP, UDP, IPv4, IPv6, etc.), wired and wireless link layer protocols (Ethernet, PPP, IEEE 802.11, etc), support for mobility, MANET protocols, etc.

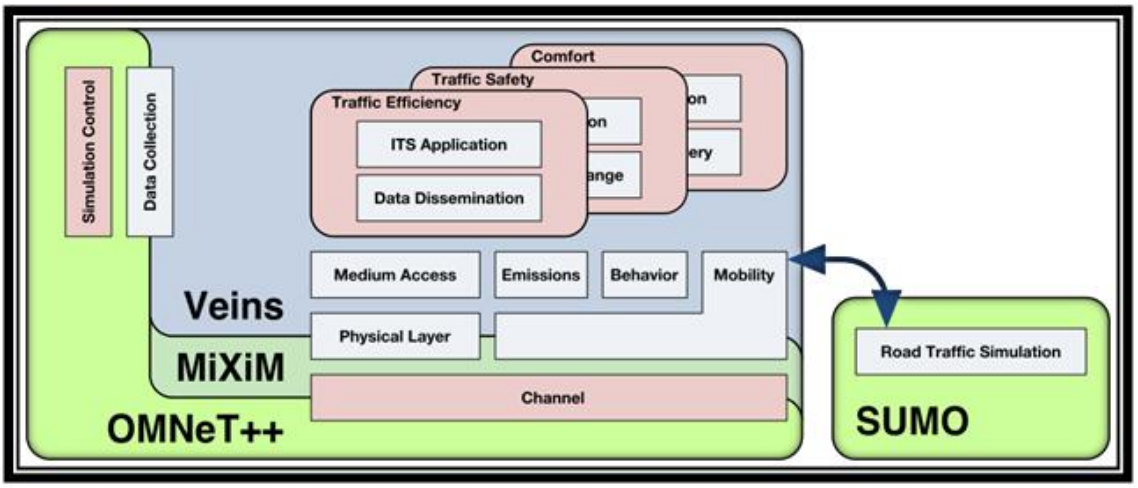

Figure 4: Veins architecture.

Veins [5] is an open source framework utilized also in running vehicular network simulations. It is based on two well-established simulators: OMNeT++ and SUMO. It relies on fully-detailed models of IEEE 802.11p that is, with 5G, the main candidate to become the protocol used for V2X communication in near future. 802.11p is a variant of WiFi with some improvements to reduce association time with AP. It is the protocol chosen to be used for the dissemination protocol.

3 Implementation

The project work has been developed and implemented using the virtual machine [6] provided by Veins that already contains all the necessary software and tools (except Cadyts). In particular, to developed this project version 5.2-i1 of the virtual machine has been used. It is based on Debian 11 and it contains:

- Veins 5.2
- INET framework 4.2.8
- OMNeT++ 5.7
- SUMO 1.11.0
- Cookiecutter 1.7.3

The whole project has been divided into two main phases. The objective of the first one was to produce a quite realistic simulation on SUMO using real traffic data and real roads network. The second phase involves implementation of a simple probabilistic dissemination protocol and evaluates how the protocol works in SUMO simulation previously developed. The next sections provides a detailed description of the steps followed to developed each part of the project.

3.1 Generate the network

To generate the network (the representation of streets) has been used the osmWebWizard.py tool [7] provided by SUMO. This is a python tool that permits to use OpenStreetMap data and generate the network, selecting the area of interest and which are the types of roads to import (car, pedestrian, bicycles, train, ecc). It also permits to create a random demand of vehicular traffic.

When the script is launched a web page is shown and on it is possible to select the area of interest and modify different parameters. Parameters and settings used for this project can be seen in figure 5. In particular are selected the "Car-only network" checkbox that permits to export car roads only and not pedestrian and bicycles roads that are out of interest in this case. Finally, all the options to export a random demand are unchecked because in this case traffic demand will be generated with Cadyts.

After that the network has been generated and saved in xml format, it can be visualized in SUMO to check the result of selection. Before proceeding with the next step, the network has been opened in netedit to remove some unconnected roads.

3.2 Generate routes with Cadyts

First of all, to use Cadyts it is necessary to download the tool from the official repository [8] and compile it. After compiling Cadyts utilities.jar and cadyts.jar should be obtained. As already mentioned, in order to perform a SUMO simulation two fundamental files are necessary:

- **Network file:** an xml file representing a part of a map, in particular roads and intersections;

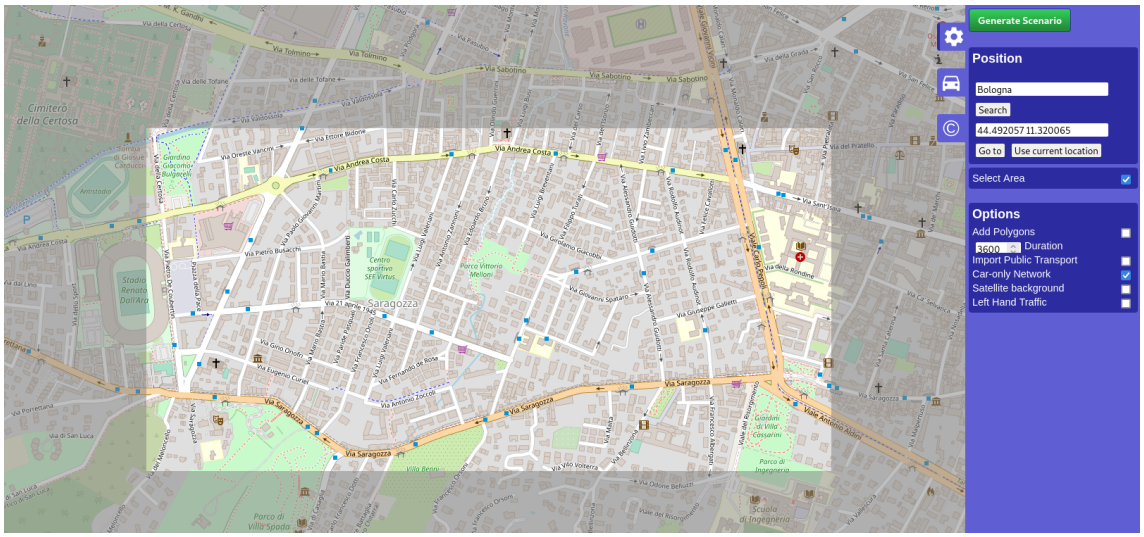

Figure 5: Area of Bologna that has been used in the simulation.

- **Routes file:** an xml file containing all vehicles with their type and the route (sequence of roads and intersections) that they must follow.

The first file has been created in the section 3.1. The routes file creation is shown in this section.
Cadyts is used to retrieve routes file. Cadyts usage is simplified by the python script provided by SUMO. Following the documentation [9] provided by SUMO, three different files are necessary to use the script:

- Network file that describes roads topology;
- Realflow file that indicates where detectors points are placed and their value;
- Route alternative file that provides a list of trips and for each trips a list of routes and their probability.

The network file has been already generated in section 3.1, instead the second and the third will be created in the following sections.

3.2.1 Realflow file creation

In order to create the realflow file, data from real measurements are necessary. Luckily, for the city of Bologna these data are freely accessible on this web page [10].

In the available web page, all detectors of the city of Bologna can be seen and easily filtered for date, time and position. Each detector provides number of vehicle pass through every hour. For this project, a total of 26 detectors have been selected and in particular, interval from 1PM to 2PM of 15 March 2022 has been selected as time interval. Then, all measurements have been inserted in a xml file where each row represents a detector, and a single row is composed as follow:

<singlelink link="825" start="0" end="3600" value="50" stddev="8" type="COUNT_VEH"/>

Where:

- **link:** street identifier where the detector is placed;
- **value:** number of vehicles detected;
- **start and end:** starting and ending time (in the simulation) during which the value is measured;
- **stddev:** standard deviation of measured value. In this case a default value (8) has been used for simplicity instead of calculate it considering multiple measurements;
- **type:** type of measurement performed, in this case count vehicles number.

Now the realflow file is created and can be used to run the python script of cadyts.

3.2.2 Route alternatives file creation

First of all, concepts of "trips" and "routes" in SUMO must be explained to understand this section. In SUMO a trip is composed by an identifier, a departure time, a starting position and an ending position. Instead, a route is defined as a trip where also a list of roads and intersections are specified. These lists represent the path that the vehicle must follow to reach the destination.

With these definitions, it is possible to describe what a route alternatives file is. It is a xml file that contains a list of trips and for each trips is given a list of possible routes and probabilities to chose that given route. To generate this file, SUMO documentation suggest to use duarouter application or *duaIterate.py* script [11]. Python script has been chosen because it provides a simpler interface to the duarouter application. A network file (the same created in section 3.1) and a xml file containing a list of trips must be given in input to the script.

To generate this trips file an easier and faster approach is to generate it randomly. Even in this case, a python script provided by SUMO can be used and it is called *randomTrips.py* [12]. Script generates a set of random trips for a given network and it does this by choosing source and destination edges randomly. A lot of parameters are present in order to modify the distribution; for example numbers of trips generated, minimum distance of trips in straight-line, the probability of a trip to start or end from an edge of the map and so on can be set. A description of all parameters and values used for this project can be found in section 3.2.3.

Now, randomly generated trips file can be given in input to the *duaIterate.py* script. The script recalls duarouter application iteratively to perform the computation of a dynamic user assignment (DUA). It works by running the simulation to discover travel times and then by assigning alternative routes to some of the vehicles according to these travel times. This is repeated for a defined number of iteration steps.
At each step the script produces a folder that contains the route alternative file. In figure 6 an example of this type of file can be seen.

3.2.3 Cadyts usage

Now, all necessary files have been created, and therefore routes file can be generated using Cadyts. Also in this case, SUMO provides a python script called *cadytsIterate.py* [9] that permit an easier use of cadyts tool. In this step, before proceeding

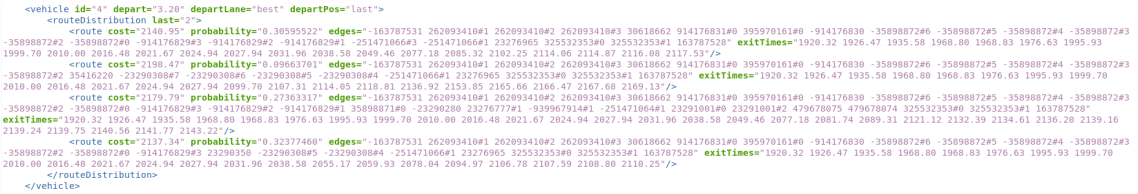

Figure 6: Example of route alternatives file.

to tune cadyts parameters, how different trips files can influence cadyts results has been tested. To evaluate the results, a parameter returned by cadyts has been used. The parameter is called log-likelihood and it allows to monitor the convergence of the calibration; the higher is the value the better would the calibration is.

The first test has been done changing time period of *randomTrips.y* script. This parameter changes how frequently a trip is generated and so how many trips are created. For example, with a value of 1 a trip every second is generated, so in an hour (as in this case) 3600 trips will be generated. In total three different tests have been performed, with three different values of parameter. The results can be seen in table 1.

Period	Log-likelihood value
1.1	-48998 +/- 6863
0.8	**-45517 +/- 4283**
0.4	-56509 +/- 1000

Table 1:

Then, other two parameters have been tested:

- Distance: minimum distance in a straight-line from starting to ending edge;
- Fringe-factor: probability that trips will start/end at the fringe of the network. For example, with a value of 10, edges that have no successor or no predecessor will be 10 times more likely to be chosen as start or endpoint of a trip.

A total of four different tests have been performed, with two different values for each parameter. Values and results can be consulted in table 2.

Distance & fringe-factor	Log-likelihood value
500 & 10	-52347 +/- 11073
500 & 25	**-39036 +/- 4015**
1000 & 10	-49511 +/- 1192
1000 & 25	-47087 +/- 2133

Table 2:

After that all tests have been performed, the best combination of values is: 0.8 for the period, 500 as minimum distance and 25 for the fringe-factor. Obviously not all parameters have been tested and not a huge number of tests have been performed, but the idea is that also the generation of trips must be tuned in order to obtain better results.

Finally, after using *duaIterate.py* script with default parameters to generate alternatives route files, some tests on cadyts parameters have been done. In particular, only two parameters have been tested: freezeitIteration and overridett. The first one sets the number of iterations steps after which cadyts disables the adaptivity (default 85). It should be tuned manually by checking in cadyts log after how many steps the calibration seems to stabilize, and then sets the value just a little bit higher. The second one, if turned on, allows the script to override the depart time according to updated link travel times. Performed tests and obtained results are shown in table 3 and 4.

Freezeit	Log-likelihood value
18	**-38000 +/- 5000**
85	-39036 +/- 4015

Table 3:

Overridett	Log-likelihood value
True	**-32000 +/- 2700**
False	-38000 +/- 5000

Table 4:

The value for freezeit parameter has been retrieved manually following indications in the documentation, but neither improvement nor worsening are present, but the parameter has been used anyway. Instead, the second parameter permits to improve calibration and so also this one has been used.

All the steps described in chapter 3.2 have been unified in a single bash script. In this way, it will be simpler to reuse cadyts without the need to execute every single step, hand by hand.

3.3 Implement dissemination protocol

As already mentioned, OMNeT++ and Veins have been used to implement the dissemination protocol. Cookiecutter has been used to generate basic project setup (as suggested by the documentation). This project provides a ready to use simulation in which is implemented a vehicular communication protocol over 802.11p. Each vehicle sends a message if itself has been standing still for too long. This basic project has been used as starting point, but some modifications have been done.

First of all, predefined network and route file have been replaced and some unused configurations have been removed from omnetpp.ini file. Then, all application logic of the protocol has been implemented in *ApplicationLayer.cc* file. In this file there are three main functions:

- **initialize:** called when vehicle is initialized. It initializes some variables used in other functions;
- **onWSM:** called when vehicle receive a message. It sets a variable used to control the coverage over time;
- **handlePositionUpdate:** called when a position update is provided by sumo. Here is implemented protocol logic.

The logic of the implemented protocol is the following: every vehicle tries at predefined time interval to send the message. The message will be sent if and only if a random number (from 0 to 1) is below a predefined threshold. If it is lower, the message will be sent in broadcast to all vehicles in the range. Below, pseudo code of protocol logic is provided.

if $simTime() - lastSent \geq sendingPeriod$ **then**
  if $randomNumber \leq alpha$ **then**
    sendMessage()
  end if
  $lastSent \leftarrow simTime()$
end if

The variables in pseudo code have the following meaning:

- **sendingTime:** define time interval to send the message (seconds). In all the simulation it has been set to 10;
- **randomNumber:** generated with a uniform distribution from 0 to 1 at each iteration;
- **alpha:** threshold that randomNumber must exceed to send the message.

Then, OMNeT++ gives the possibility to record all variables you want, so that the simulation can be analyzed in all aspects. By default, this project already records a lot of information for every vehicle such as: acceleration, position, speed, distance travelled, total time in the simulation and so on. With some of these values some considerations on how the behaviour of vehicle changes using cadyts or not will be done, but to evaluate how the protocol works using different alpha values an indicator must be introduced. The selected indicator is coverage: how many vehicles received at least a message in a given time interval, compared to the total number of vehicles.

$$Coverage = \frac{received}{received + nreceived}$$

To do that, a boolean variable has been introduced for each vehicle and it has been set to true when the vehicle receives a message. Finally, every time interval the variable has been recorded and then it has been reset to false.
Pseudo code used can be seen below.

if $simTime() - lastRecord \geq recordingPeriod$ **then**
  recordVariable()
  $receivedMessage \leftarrow false$
  $lastRecord \leftarrow simTime()$
end if

In all simulations performed, *recordingPeriod* has been set to 5 seconds and *receivedMessage* is the variable used to track if at least one message has been received by the vehicle in the last 5 seconds.

Other minor changes have been performed to simulation parameters and some additional information must be given before analyzing the results. In the simulation is not present an obstacle map, so wireless signal is not affected by buildings.

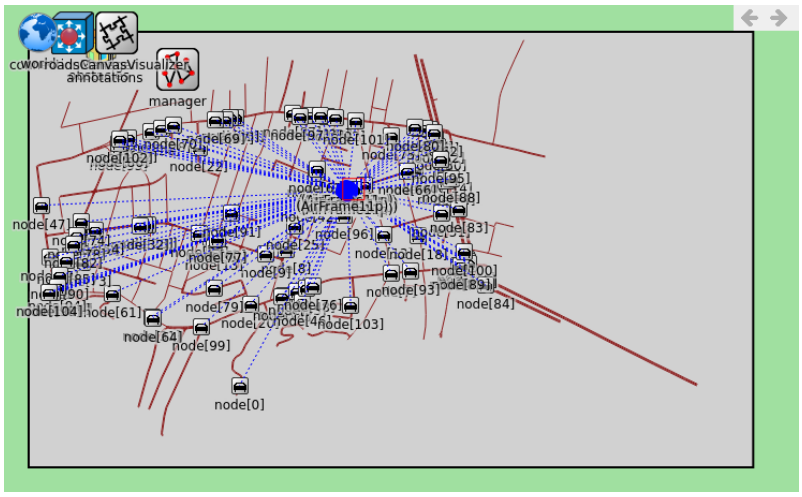

Figure 7: Screenshot of simulation in progress.

For this reason, antenna transmission power has been reduced and alpha value of simplePathLossModel ha been increased to ensure that the wireless signal does not spread too much.

4 Results

In total, eight different simulations have been performed. Four of them have been executed with the route file generated with cadyts and different alpha values. the other four have been executed with the route file generated by duarouter (so without cadyts calibration) and the same alpha values of the first four. In this way, how the coverage changes using different alpha values can be seen, and also how the protocol coverage changes with different route files. In figure 8 and 9 simulations have been named with a progressive number and corresponding parameters configuration can be seen in table 5.

Simulation	alpha
Sim 1	0,4
Sim 2	0,6
Sim 3	0,8
Sim 4	1,0

Table 5: Alpha values used in different simulations

In figure 8 and 9 on the left is shown how the coverage changes during all the time of the simulation, instead on the right is shown the mean coverage for each simulation.

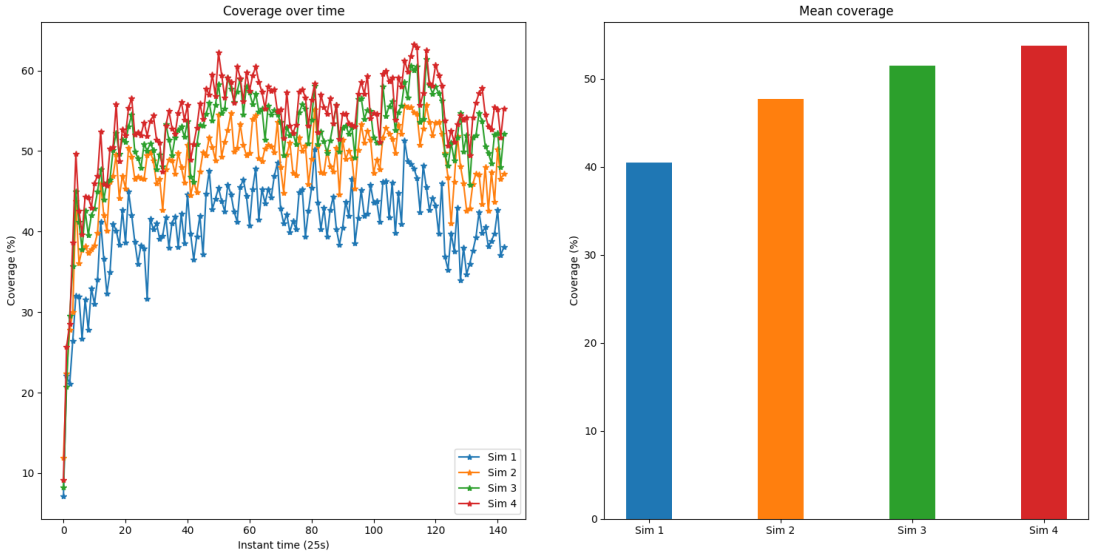

Figure 8: Coverage of different simulations generated by Cadyts.

The graphs show that all cadyts simulations have lower coverage. This is probably due to the fact that cadyts tries to spread all vehicles in such a way that their distribution is as similar as possible to the traffic flows indicated in the realflows file. Instead, duarouter computes the path of each vehicle using shortest path computation considering travel time (Dijkstra or A*) and so much more vehicles will follow the same path to reach their destination. In this way, the shortest path in the network (or also the fastest one, since travel time is considered) will be congested and so the probability for a vehicle to receive a message will be higher.

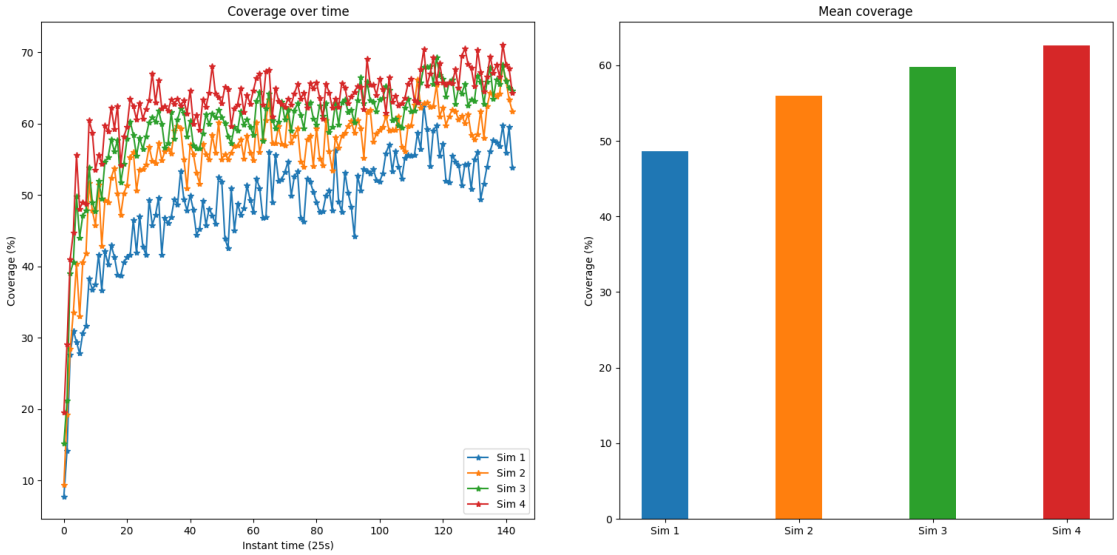

Figure 9: Coverage of different simulations generated by duarouter.

Then, some considerations on the protocol itself can be done. Simulation with $\alpha = 1.0$ gives the maximum coverage that can be reached with this network topology and these protocol parameters. It is easy to see that starting from Sim 3 ($\alpha = 0.8$), the coverage is not increasing too much and so it is not so advantageous to use a value of alpha higher than 0.8. In fact, a higher value of alpha will cause more congestion on the network with only a small advantage for the coverage.

Finally, in figure 10 is shown how the mean speed of all vehicles changes from a simulation calibrated with cadyts and one using only duarouter.

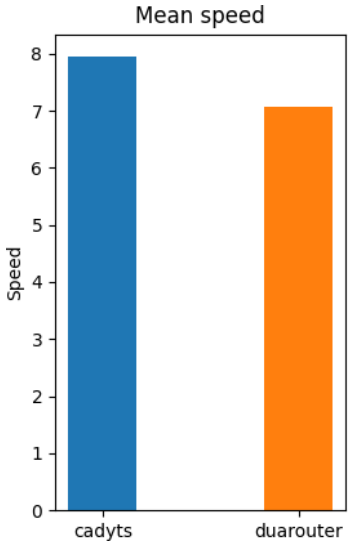

Figure 10: Mean speed in Cadyts (blue) and duarouter (orange) simulation.

These data should be compared to another one obtained from real observations to understand which of the two is more realistic. The mean speed is strictly related to the real traffic conditions and the roads topology, and so only some superficial considerations can be done. In this case, network's topology presents three main roads where a large majority of vehicles travel, so a higher mean speed could means

that a higher number of vehicles go through these roads as in the real conditions. Unfortunately, with just these data, it is difficult to establish if cadyts calibration is working well or not, but of course there are differences between these two simulations.

5 Conclusion

The initial aim of this project was to try to use cadyts with SUMO, verify how well it calibrates route file to match real measurements, and implement a probabilistic dissemination protocol. All the objectives have been achieved and, in particular, some initial considerations on cadyts performance have been done.
Certainly, it is very important to provide cadyts with the "right" alternative route file. "Right" means a file generated taking into account the type of traffic that cadyts should calibrate. In fact, section 3.2 shows that changing parameters of *randomTrips.py*, cadyts results changes themselves. Obviously, number of performed tests, considered traffic and network type are very limited but this can be a starting point to further analysis. In addition, using the same network and detector points used in this project a huge number of additional tests can be done, for example with a larger number of parameters and/or changing other parameters not used in this project.

References

[1] URL: https://github.com/mazzo98/Mobile-Systems-M.

[2] URL: https://www.eclipse.org/sumo/.

[3] URL: https://people.kth.se/~gunnarfl/cadyts.html.

[4] URL: https://omnetpp.org/.

[5] URL: https://veins.car2x.org/.

[6] URL: https://veins.car2x.org/documentation/instant-veins/.

[7] URL: https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html.

[8] URL: https://github.com/gunnarfloetteroed/java.

[9] URL: https://sumo.dlr.de/docs/Contributed/Cadyts.html.

[10] URL: https://opendata.comune.bologna.it/explore/dataset/rilevazione-flusso-veicoli-tramite-spire-anno-2022/.

[11] URL: https://sumo.dlr.de/docs/Tools/Assign.html#duaiteratepy.

[12] URL: https://sumo.dlr.de/docs/Tools/Trip.html.