Department of Information Engineering(DII)
Master of Science in Computer Engineering

Project 1 – Vehicular network: car sensing

Lecturer(s):

Prof. Giovanni Stea
Dr. Giovanni Nardini

Student(s):

Tesfaye Y. Mohammad

Final Project for:
592II PERFORMANCE EVALUATION OF COMPUTER SYSTEMS AND NETWORKS [WCN-LM]

Academic Year 2024/25

Contents

List of Figures

1 Introduction

1.1 Project Requirements

A vehicular system is composed of N vehicles that move randomly within a 2D floor plan of size L $\times$ H, according to a waypoint model. A waypoint is defined by a pair of coordinates (x,y) and a speed s. The coordinates x, y are random variables to be defined later. Vehicles move between waypoints a and b at the constant speed selected together with **b**. As soon as a vehicle reaches a waypoint, it selects a new one and moves towards it.

Vehicles are equipped with a wireless interface and can communicate with other vehicles that fall within their transmission range M. Every T seconds, each vehicle checks how many cars are within its transmission range. The relationship between T and M is expressed as in equation 1.

$$T = \alpha \times M^2 \tag{1}$$

The α is the efficiency of the wireless interface and can assume values between 0 and 1.

Evaluate at least the overall rate of vehicles sensed per second for various values of M and α.

At least the following scenario has to be evaluated:

- uniform distribution of x and y;

In all cases, it is up to the team to calibrate the scenarios so that meaningful results are obtained.

Project deliverables:

(a) Documentation (according to the standards set during the lectures)
(b) Simulator code
(c) Presentation (up to 10 slides maximum)

1.2 Tools Used

Here are the tools used in this project

- **IDE**: OMNeT++ 6.0.2, **found here**
- **Python:3.9.x** for offline analysis.
- **OS**: Win-10.
- **io.draw**: to draw all the images in this document.
- **Overleaf**: the website is used to write this document in LaTeX.
- Programming language:- C++17

1.3 Objective(s)

The main objective of the project is to evaluate at least the overall rate of vehicles sensed per second for various values of M and α.

2 Performance Indexes

To assess the performance of the system, the following is/are taken as a reference:

- **Overall Sensing Rate**: defined here in equation 4. It is the overall sensing rate per second normalized by the total number of vehicles(N) in the simulation area.

2.1 General Assumptions Taken

For this project, the following assumptions/simplifications are considered.

- The **simulation** play ground is arranged on a square $L \times H$ of 500 meters.
- In normal simulation (except degeneracy test), initially all the cars in the simulation are at least 10 meters apart. In this way, at least overlapping is avoided.
- While sensing takes place the distance calculation between the vehicles is based on a method that the INET has provided. Plus to this all the mobility related functionalities are based on INET.
- Other functionalities that, such as vehicle speed, constraint areas of the playground, and initial points for all coordinate axes, are provided by the mobility module from the INET library.
- Simulation results are written to files and an offline analysis is done in python.
- For any vehicle in the simulation playground self sensing is not allowed/skipped.

2.2 Factors

The following factors have some effects on the performance of the system:

- **N** :- It is a parameter that holds the number of vehicles in the simulation playground.
- **M**:- This is the communication range in meters that that a car senses other vehicles.
- α is the a constant that also affects the system. It takes values between 0 and 1.

- **SensingInteval** is a time interval in which every sensing event occurs and it is expressed by the formula in Eq.1.
- **Size** of the play ground is also the one that determines how the simulation environment looks like. In this project it is a square area of 500 by 500 meters.
- The **speed** of the vehicles also has effect in the dynamics of the simulation and it is provided by the INET library.

3 Implementation

In this section I will present the modules that have been defined for the implementation of the Project and their behaviors briefly.

3.1 Modules

The project is designed with two key modules that handle **mobility** and **sensing**, as specified in the problem definition.

- Car:- This is a compound module that represents the system that we are dealing with in the project. It consists of the following sub-modules
  - MobilityModule :- A simple module that extends the **RandomWaypointMobility** from INET and accesses all the necessary parameters to be used in the project.
  - SensingModule :- is a simple module as well. It provides all the parameters that are needed to implement the sensing logic.

The network of the simulation is named as **VehicularSensingNetwork** and all these modules work together to simulate the movement and sensing capabilities of the vehicles, ensuring that the system meets the specified requirements.

In Fig.1 the network of the project and all the modules are depicted.

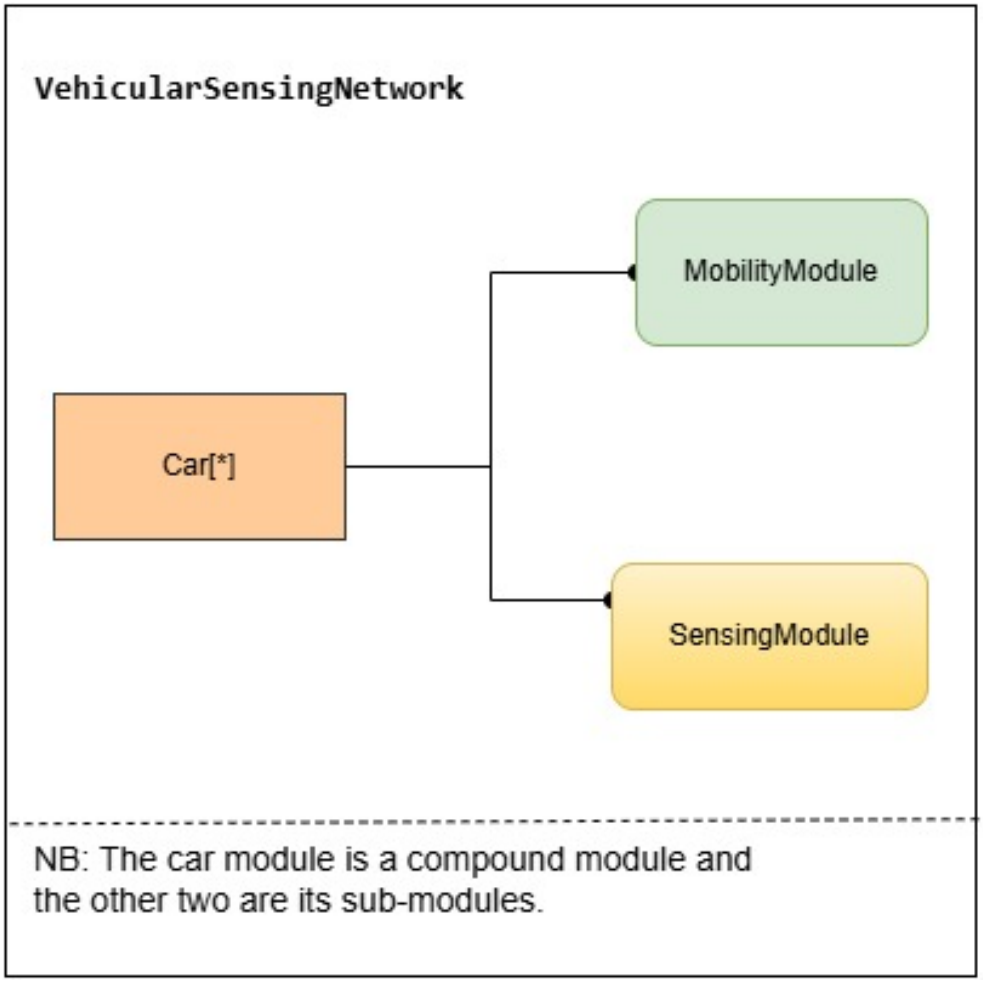

Figure 1: The modules that are defined in the project.

3.2 Modules Behavior Description

3.2.1 Car Module

This module is a compound module and performs the following tasks.

- Define the **carIndex** which is an important parameter as the car is an array in the network level.
- Defines the **H**eight and the **L**ength of the play ground of the simulation.
- Provides the min and the max **speed** that the vehicles are moving during the simulation. Noting that the speed is a uniform distribution.
- Includes the mobilityModule and the sensingModule as its sub-modules.

3.2.2 Mobility Module

Initially to avoid the overlapping of the vehicles the module places them at least 10m apart.

This is the module that handles all the logic regarding to the mobility. For this to happen it extends the **RandomWaypointMobility** from the INET.

From listing.1 one can see the parameters that are defined in this module. The constraints of area

```
customMobility: MobilityModule {
    totalCars = parent.allCars;
    // x
    constraintAreaMinX = parent.minCoordValue;
    constraintAreaMaxX = parent.maxCoordValue;
    // y
    constraintAreaMinY = parent.minCoordValue;
    constraintAreaMaxY = parent.maxCoordValue;
    //z
```

```
        constraintAreaMinZ = 0m;
        constraintAreaMaxZ = 0m;
        //Initials
        initialX = uniform(parent.minCoordValue,
            parent.maxCoordValue);
        initialY = uniform(parent.minCoordValue,
            parent.maxCoordValue);
        initialZ = 0m;
        // Choose a uniform speed for each car
        speed = uniform(parent.minSpeed, parent.maxSpeed);
        //
        waitTime = (0s);
        @display("p=$x,$y");
    }
```

Listing 1: Parameters defined in the Mobility Module

3.2.3 Sensing Module

This module handles all the sensing logic. It defines the following parameters.

- M, the communication range, in meters in which a vehicle tries to sense other vehicles in the simulation area.
- The alphaValue, α , which is the efficiency of the wireless interface and can assume values between 0 and 1.
- The **sensingInterval** which is defined in equation 1.So,a sensing event is triggered by the sensingModule every T seconds.
- An outputFile where all the sensing results are stored in **.csv** format.

4 Simulation Results Analysis

As mentioned in sec.1.1 the objective of the project is to evaluate the over all sensing rate of the vehicles in the simulation time.

Every time a sensing event is triggered, the instantaneous sensing rate, r, is calculated based on the equation that has a form as depicted at 2. It is the inverse of the sensingInterval, T. Here, the approximate symbol is important.

$$r \approx \frac{1}{T} \tag{2}$$

In the cpp implementation, the **performSensing** method grabs a vehicle and calculates its distance from all other vehicles in the simulation area and then **counts** how many vehicles are within the range **M** iteratively.

Once we get the count the equation in 2 takes the following form i.e the **equal** sign is applied.

$$r = \frac{count}{T} \tag{3}$$

So the over all sensing rate is calculated as the sum of all the instantaneous rates based on the following formula in equation 4. The N is the number of vehicles in the simulation.

$$R = \frac{1}{N} \sum_{i=1}^{N} r_i \tag{4}$$

In listing 2 it has been depicted how the sensing logic implemented.

```
void SensingModule::performSensing() {
    // [The code is cropped for brevity]
    Coord currentPosition = mobility->getCurrentPosition();
    int count = 0; // Holds the number of vehicles sensed in the
        current sensing event.
    for (int i = 0; i < totalCars; i++) {
        if (i == carID)
            continue; // Skip to next car if self sensing occurs.
        cModule *otherCarModule =
                getParentModule()->getParentModule()->getSubmodule("car",
                    i);
```

```
    if (!otherCarModule) {
        EV << "otherCarModule submodule NOT found!\n";
        return;
    }
    auto otherCarMobility = check_and_cast<IMobility*>(
            otherCarModule->getSubmodule("customMobility"));

    if (!otherCarMobility) {
        EV << "otherCarMobility is NOT of type IMobility!\n";
        return;
    }
    Coord otherCarPosition =
        otherCarMobility->getCurrentPosition();
    double distance =
        currentPosition.distance(otherCarPosition);
    if (distance <= communicationRange) {
        count++;
    }
}

int runNumber = getEnvir()->getConfigEx()->getActiveRunNumber();
double rate = (sensingInterval > 0) ? (count / sensingInterval)
    : 0.0000; // Instantaneous sensing_rate per sensing event.
outputFileStream << simTime() << "," << runNumber << "," <<
    carID << ","
        << count << "," << rate << "\n";
}
```

Listing 2: The sensing logic and how it calculates the rate.

4.1 Effect of α and M on R

As per the requirements of the project the overall sensing rate is calculated as follows for every parameter combination.

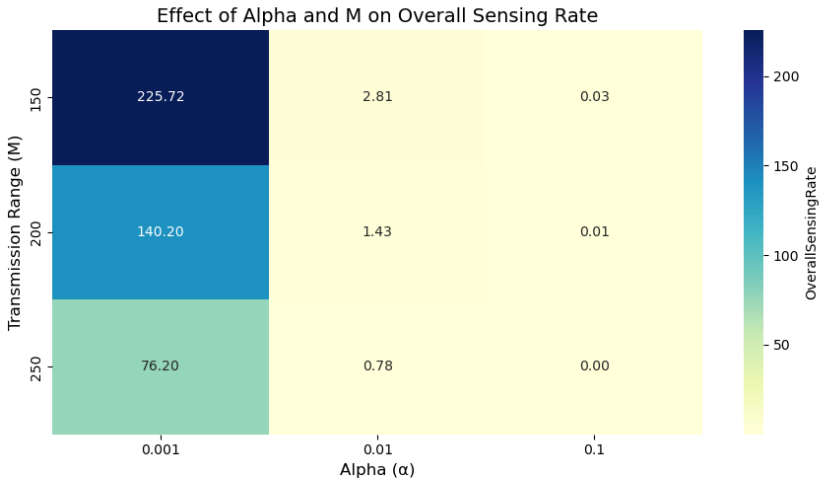

Figure 2: Results of the overall Sensing rate

From Fig.2 we can conclude that if the **sensingInterval** is very big the sensing events are triggered less frequently and the overall sensing rate is really low.

On the other hand, if the **sensingInterval** is small then the sensing evens are scheduled frequently and the overall sensing rate becomes very high.

It is **recommended** to choose combinations of parameters of M and α that produce moderate results, ensuring that R is neither too low nor too high.

5 Verification

This section provides the different tests that are implemented to show how much the code that is developed represents the vehicle-sensing model.

5.1 Consistency Test

The consistency test checks whether the system and its output behave in a consistent manner. Essentially, we anticipate the system to display comparable behavior when given two equivalent inputs. Thus, during this test, we expect the system's behavior to remain largely the same. At the very least, we do not anticipate significant discrepancies between the results of the two tests.

For this reason, the model is tested using a simulation time limit of 6000 seconds and triggering an event every 100 seconds, then in a second test doubling the simulation time to 12000 seconds and scheduling events every 200 seconds, will yield a nearly identical overall sensing rate.

```
[Config Verify_Consistency]
*.numCars = 10
repeat = 10
warmup-period = 100s
*.car[*].customSensing.alphaValue = 1
*.car[*].customSensing.communicationRange = 10m
```

Listing 3: Consistency test

5.1.1 Test 1

```
[Config Ver_Consistency_Test1]
extends = Verify_Consistency
sim-time-limit = 6000s
*.car[*].customSensing.sensingInterval = 100s
```

Listing 4: Consistency test: test_1

5.1.2 Test 2

```
[Config Ver_Consistency_Test2]
extends = Verify_Consistency
sim-time-limit = 12000s
*.car[*].customSensing.sensingInterval = 200s
```

Listing 5: Consistency test: test_2

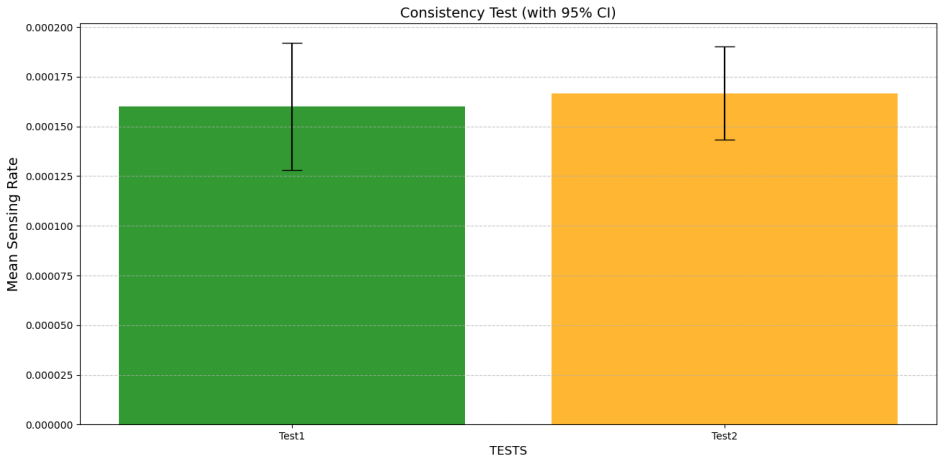

Figure 3: Consistency Test

As we can see from Fig.3 the results show that there is no wild behavior in the two tests outputs. The test results are as expected. We can say that the model passes the **consistency** test.

5.2 Continuity Test

This test is all about to verify if changing slightly the input affects slightly the output.

As can be seen from the configurations for the three test, the input that is N(numCars) increases slightly, so we expect that the output ,overallrate, for this test to increase slightly or at least there is no wild differences in the test results.

```
[Config Verify_Continuity]
sim-time-limit = 2000s
repeat = 10
*.car[*].customSensing.alphaValue = 0.01
*.car[*].customSensing.communicationRange = 200m
```

Listing 6: Continuity test configuration

5.2.1 Test-1

```
[Config Ver_Continuity_Test1]
extends = Verify_Continuity
*.numCars = 20
```

Listing 7: Continuity test: test_1

5.2.2 Test-2

```
[Config Ver_Continuity_Test2]
extends = Verify_Continuity
*.numCars = 21
```

Listing 8: Continuity test: test_2

5.2.3 Test-3

```
[Config Ver_Continuity_Test3]
extends = Verify_Continuity
*.numCars = 22
```

Listing 9: Continuity test: test_3

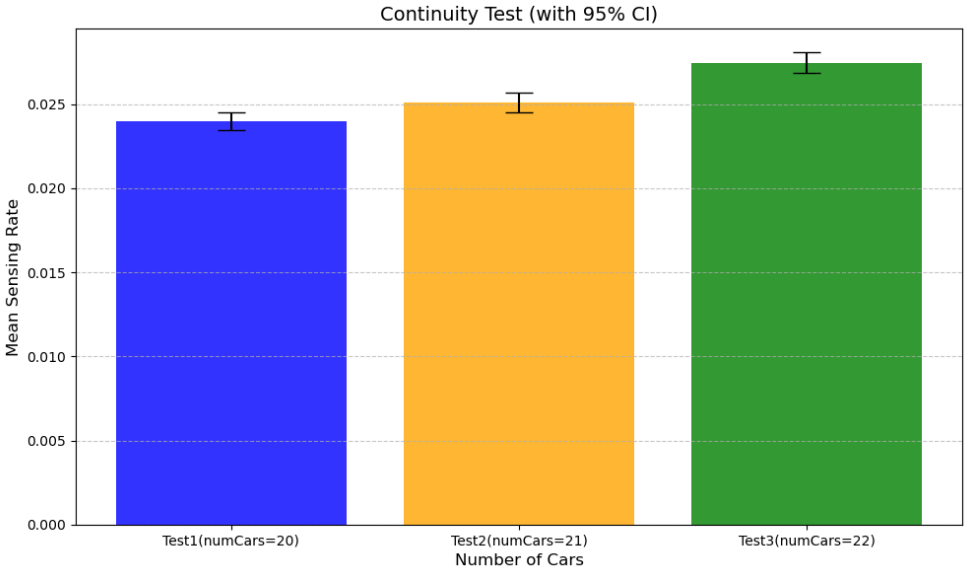

Figure 4: Continuity Test

From Fig.4 we can see that as the number of vehicles (**input**) increases slightly the mean sensing rate (**output**) also increases.Moreover, there are no outliers from the results, so the model passes this test.

5.3 Degeneracy Test

This test is used to analyze the behavior of the system when the parameters are set to 0 or to other extreme values.

In this test the following configurations are used.

```
[Config Verify_Degeneracy]
sim-time-limit = 2000s
repeat = 5
*.numCars = 10
*.car[*].customSensing.alphaValue = 0.01
*.car[*].customSensing.communicationRange = 200m

//
[Config Ver_Degeneracy_Test1]
#numCars set to one
extends = Verify_Degeneracy
*.numCars = 1
//
[Config Ver_Degeneracy_Test2]
# The play-ground is almost a point i.e (LxH == 1m x 1m)
extends = Verify_Degeneracy
*.car[*].customMobility.minInitDistance = 0m # the min intial
    distance between cars in meter.
*.car[*].minCoordValue = 0m
*.car[*].maxCoordValue = 1m
```

Listing 10: Degeneracy test configuration

5.3.1 Test_1

In this test we set the $N = 1$ i.e there is only one car in the simulation. Based on the logic of the sensing module, there is no **self-sensing** so all the results that we got for the overall sensing rate is **zero** in every 'repetition' as expected.

5.3.2 Test_2

In this case we set the play-ground from 500 by 500 to 1m by 1m, almost a **point**. We expect that every vehicle senses every other vehicle in the simulation play-ground.

As can be seen in the results folder the results found fits with our expectations. Hence, the model passes the **degeneracy** test!

5.3.3 Test_3

The system also works as expected when the $N = 0$.

5.4 Monotonicity Test

This test consists in assessing the monotonicity of some performance indexes using different combinations of factors.

In this test we are going to analyze if the overall sensing rate decrease as we increase the communication range in 4 different tests. All the rest of the factors remains the same.

```
[Config Verify_Monotonicity]
sim-time-limit = 2000s
repeat = 5
*.numCars = 10
*.car[*].customSensing.alphaValue = 0.01

[Config Ver_Monotonicity_Test1]
extends = Verify_Monotonicity
*.car[*].customSensing.communicationRange = 50m

[Config Ver_Monotonicity_Test2]
extends = Verify_Monotonicity
```

```
*.car[*].customSensing.communicationRange = 100m

[Config Ver_Monotonicity_Test3]
extends = Verify_Monotonicity
*.car[*].customSensing.communicationRange = 150m

[Config Ver_Monotonicity_Test4]
extends = Verify_Monotonicity
*.car[*].customSensing.communicationRange = 200m
```

Listing 11: Monotonicity test: test

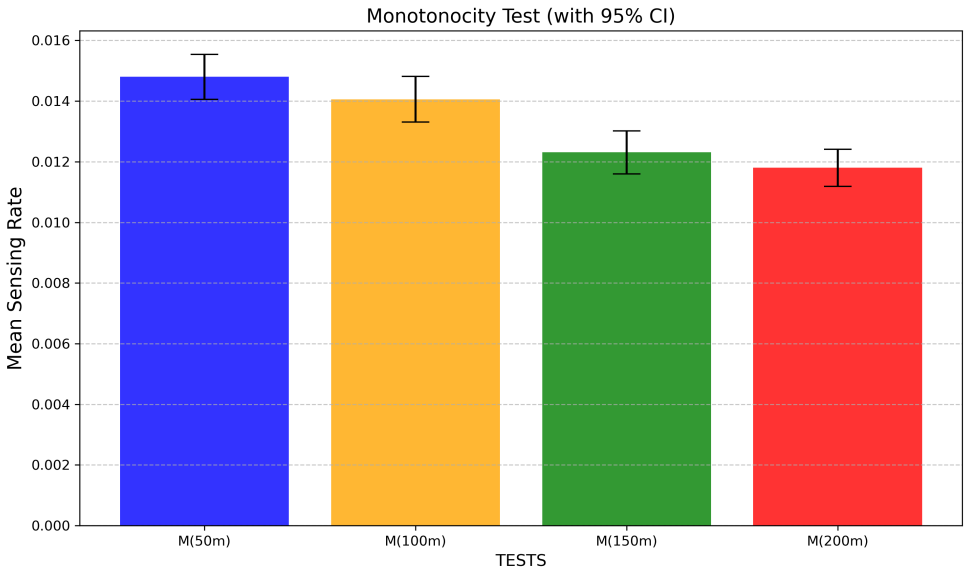

Figure 5: Monotonicity test results.

As can be seen from Fig.5 we can observe that with increasing the communication range M, the overall mean sensing rate decreases as we already expected. Hence, the model passes the test.

6 Conclusion

As a final conclusion, I would like to emphasize that the overall sensing rate is significantly influenced by the parameter combinations of α and the communication range, M. A larger sensing interval results in fewer sensing events, while a smaller sensing interval leads to more frequent sensing events, so the over all sensing rate is greatly affected by the calibration of the sensing interval.

It is recommended to choose parameter combinations α and the communication range M that could result in a moderate sensing interval, ensuring that the overall sensing rate is neither too low nor too high.

7 Github

The project code and documentation can be found here: **github repo**.