Network Infrastructures

Project:
Epidemic models for VANET message dissemination

Presentation of the problem

- **EPIDEMICS** are critical phenomena, not only from a biological point of view, such as infectious diseases, but also from a technological point of view, as malware propagation.

- In our specific case the propagation of a message in a vehicles urban network

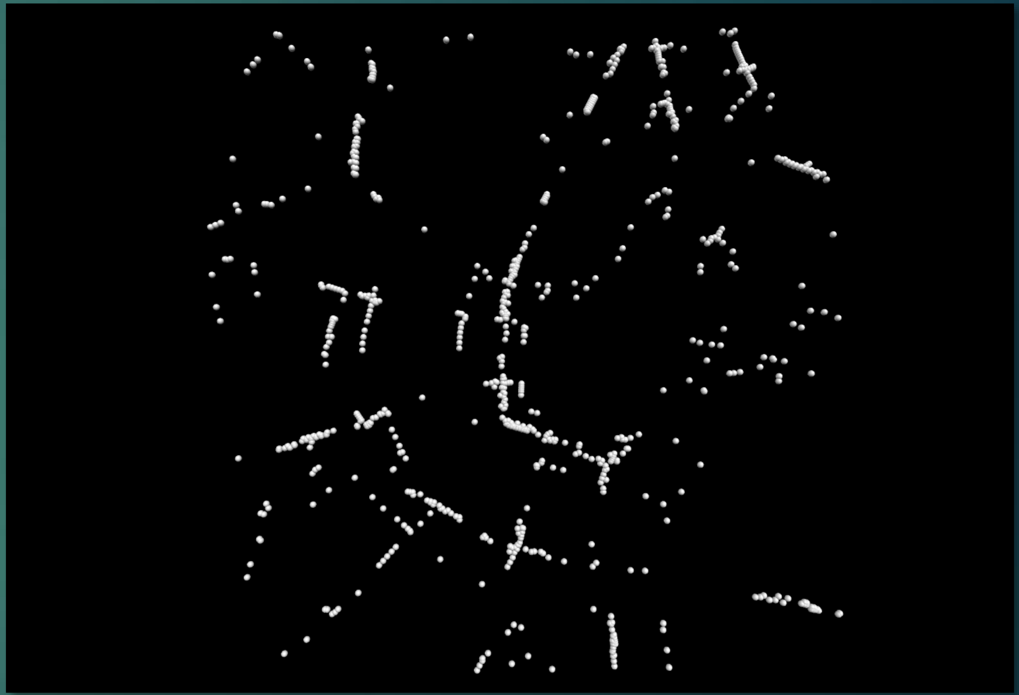
Presentation of the problem

- Main purpose of the project is to develop an **algorithm** for the dissemination of a message in a vehicle network.

- Each vehicle knows the **position** of its neighbors
- Vehicles communicate only in **broadcast** with IEEE 802.11 standard wifi technology

Presentation of the problem

- Lets consider a graph where each node is a vehicle
- Vehicles can communicate only if they are neighbours
- The program provide an interface to simply interact with the graph and make the dissemination start
- We used VPython to represent the graph



Presentation of the problem

- The graphs are taken from real situations settled in various cities (New York, Cologne, Luxemburg) and they are provided in the form of two files
- The first is about the position of vehicles on the map.

pos_time27100Tper50.txt

```
27100 1 6963.9307148737 4361.0158440515 10.963782299209
27100 2 7306.2138804573 4170.0763966782 0.64977075279047
27100 3 7281.3635965167 4088.7782559765 11.351899474092
27100 4 7710.9949400277 5361.3946045766 7.46977184115
```

- The second defines, for each line, the adjacent vehicles from the one about that line.

pos_time27100Tper50.txt adj_time27100Tper50.txt

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
```

Presentation of the problem

- We have created a dissemination algorithm of messages trough our vehicles in order to reach the higher number of “infected” nodes, without overload the network.
- The program will perform the algorithm simply clicking the sphere from where we want to start.
- The result will be a graphic representation of the infected graph and the data results in the terminal.

Presentation of the problem

- We generated a series of graphs to show the variation of the outputs according to the changing of the cities and of the algorithm parameters like R_Max and T_Max.

- The utilized programming language is Python, the libraries are Matplotlib for the graphs and VPython for the user interface.

Algorithm's description

- Our algorithm runs for each vehicle and it returns whether or not a car must re-broadcast a message.
  - If all of this vehicles broadcast, then the network would be overloaded.
  - Else if too few cars broadcast, then few nodes would be reached.

Algorithm's description

- After receiving a message, each car starts a timer which becomes shorter if the distance from the last emitter becomes longer:

$$t = Tmax(1 - (dist / Rmax))$$

  - Long distance, short timer.

During this timer, the car will wait for other messages arriving.

- The lists of these messages' emitters will be the central theme of our algorithm.

Algorithm's description

Algorithm 1 EvaluatePositions returns whether or not a car must re-broadcast a message

```
procedure EvaluatePositions
input:
    messages ← list of messages received during waiting time
    neighbor_positions ← list of positions of my neighbors
    my_pos ← my position

initialization:
    emitters = ∅
    for msg in messages do
        emitters = emitters ∪ msg.emitters

main:
    for emitter in messages do
        if dist(my_pos, emitter.pos) > 2RMIN then
            skip iteration
        for pos in neighbor_positions do
            if dist(emitter, pos) < RMIN then
                neighbor_positions.remove(pos)
    return neighbor_positions.length > 0
```

In the first part of the main function, for each emitter who relayed the message, if it is very far from me, then I will ignore it.

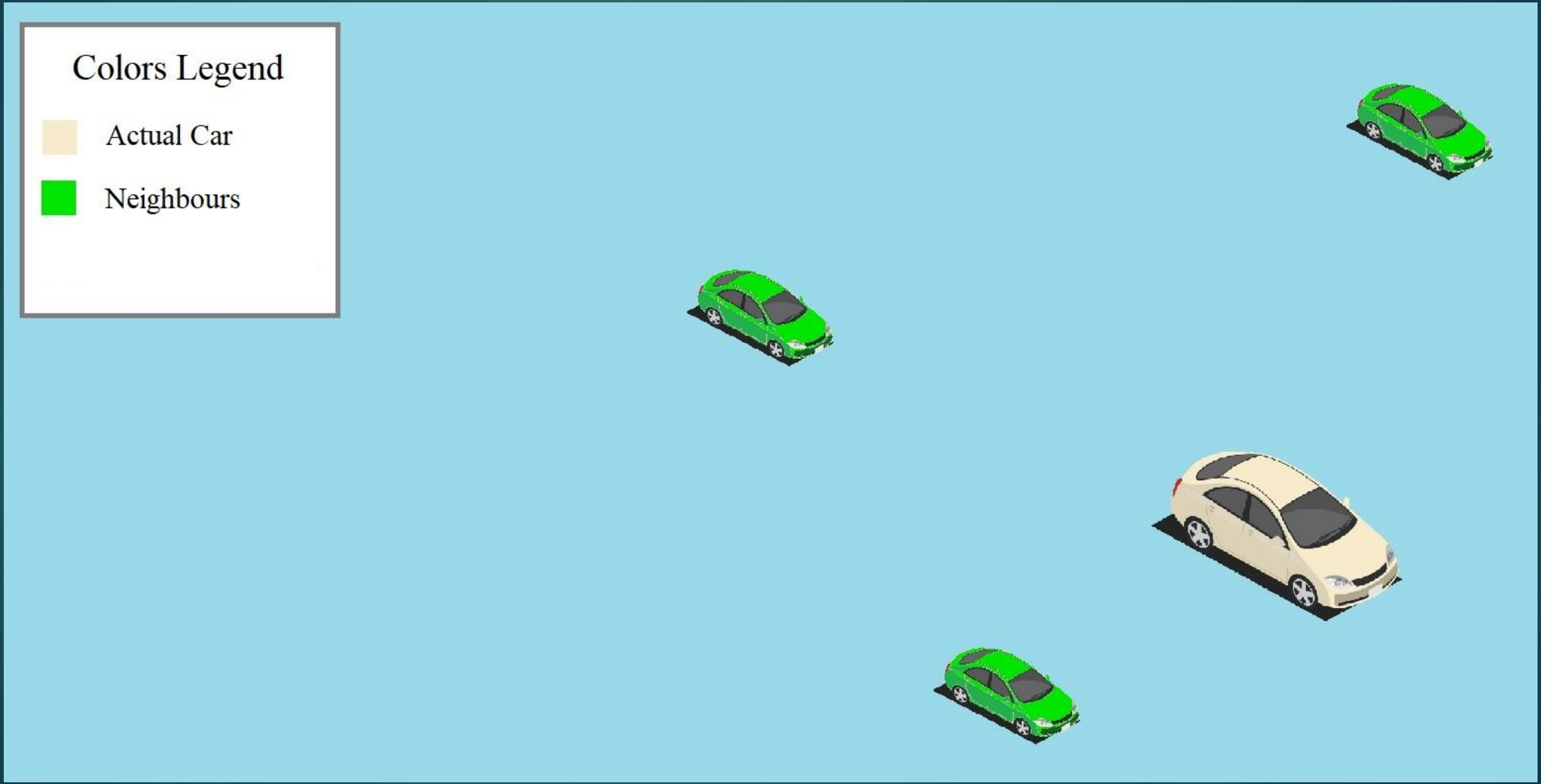
Algorithm's description

Algorithm 1 EvaluatePositions returns whether or not a car must re-broadcast a message

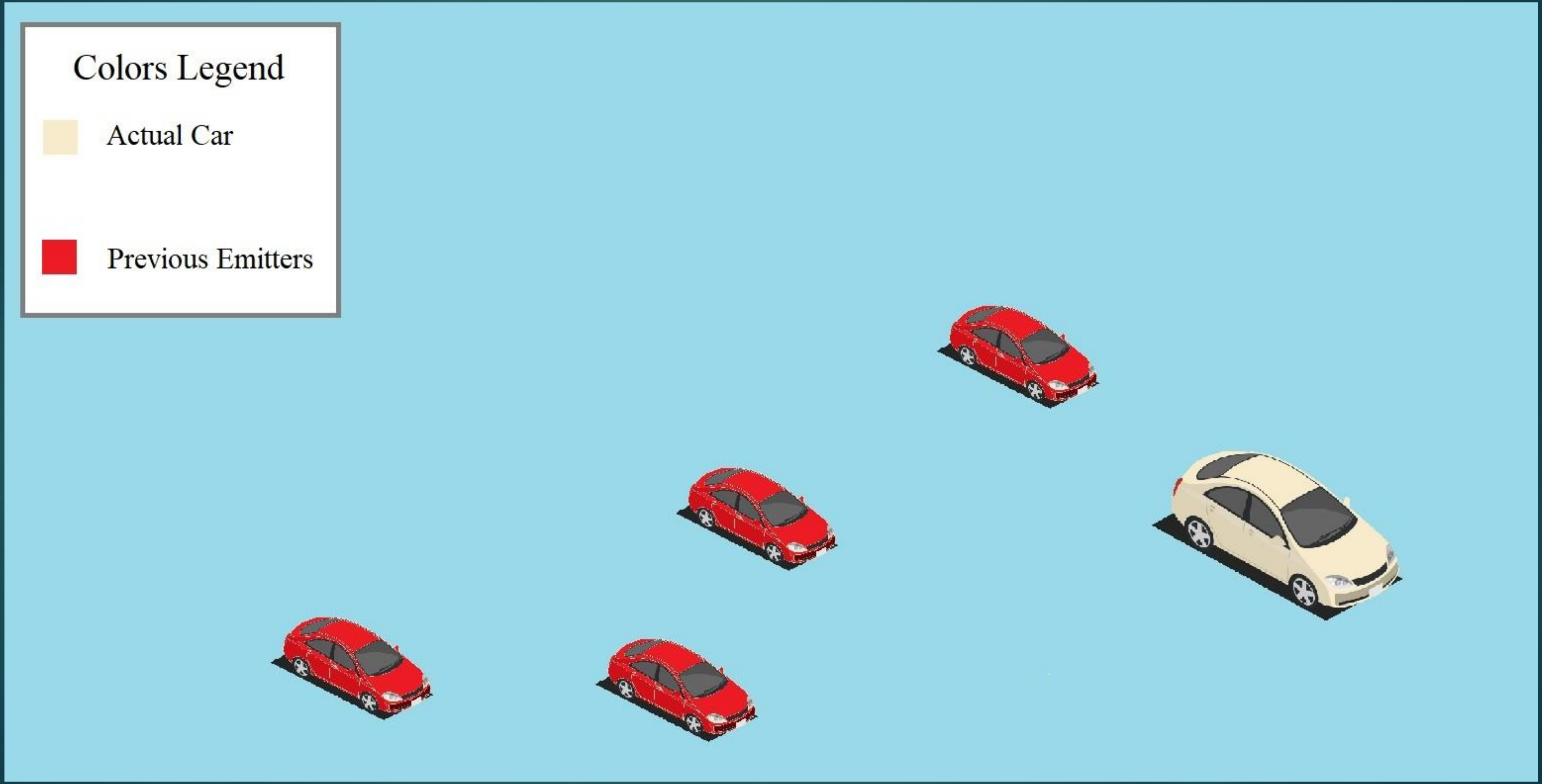
```
procedure EVALUATEPOSITIONS
input:
    messages ← list of messages received during waiting time
    neighbor_positions ← list of positions of my neighbors
    my_pos ← my position

initialization:
    emitters = ∅
    for msg in messages do
        emitters = emitters ∪ msg.emitters

main:
    for emitter in messages do
        if dist(my_pos, emitter.pos) > 2RMIN then
            skip iteration
        for pos in neighbor_positions do
            if dist(emitter, pos) < RMIN then
                neighbor_positions.remove(pos)
    return neighbor_positions.length > 0
```

Instead, about emitters not so far from me, if some of my neighbours are NOT contained in emitters' range of action, then I will broadcast.

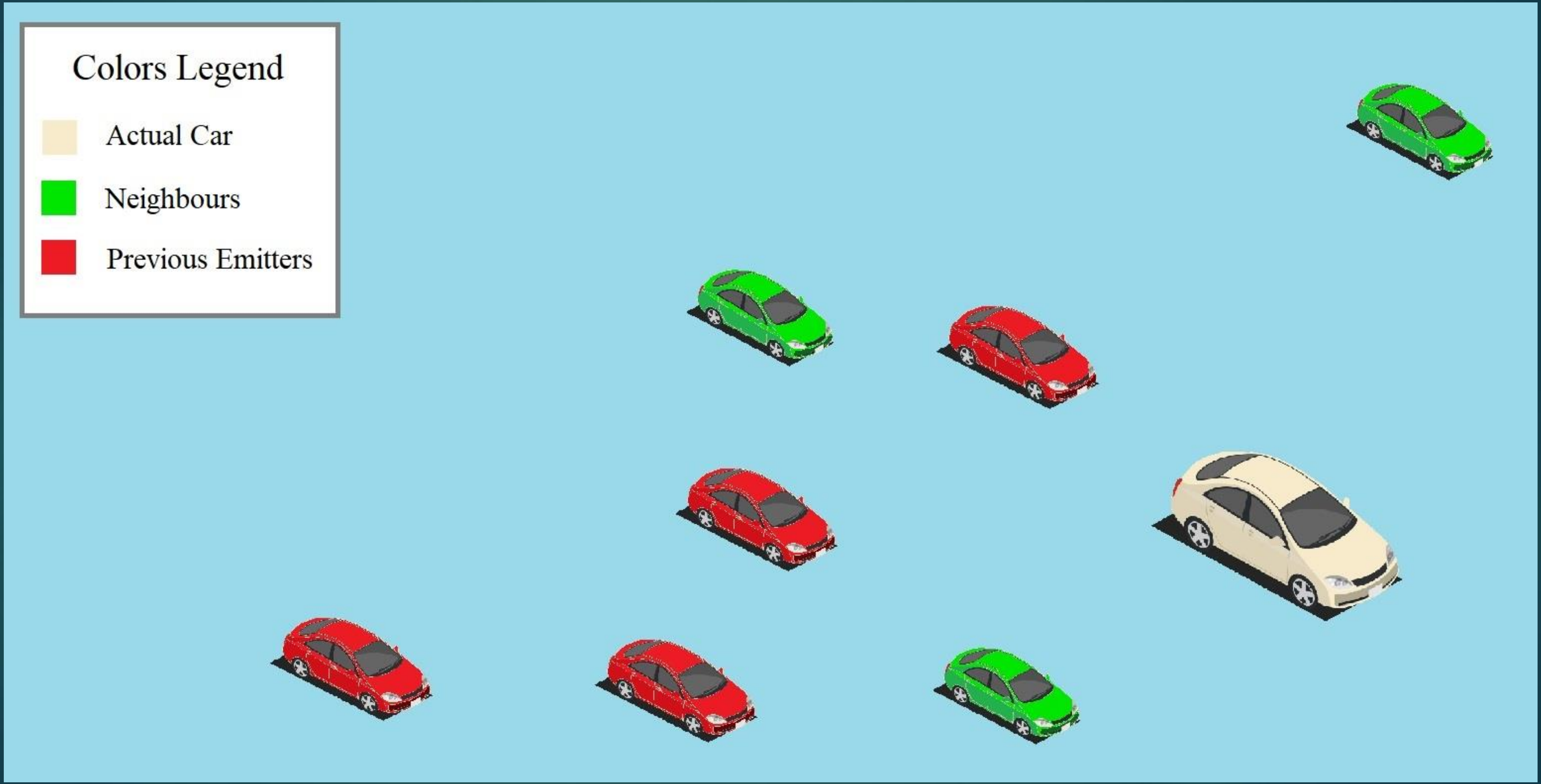
Algorithm's description - Neighbours



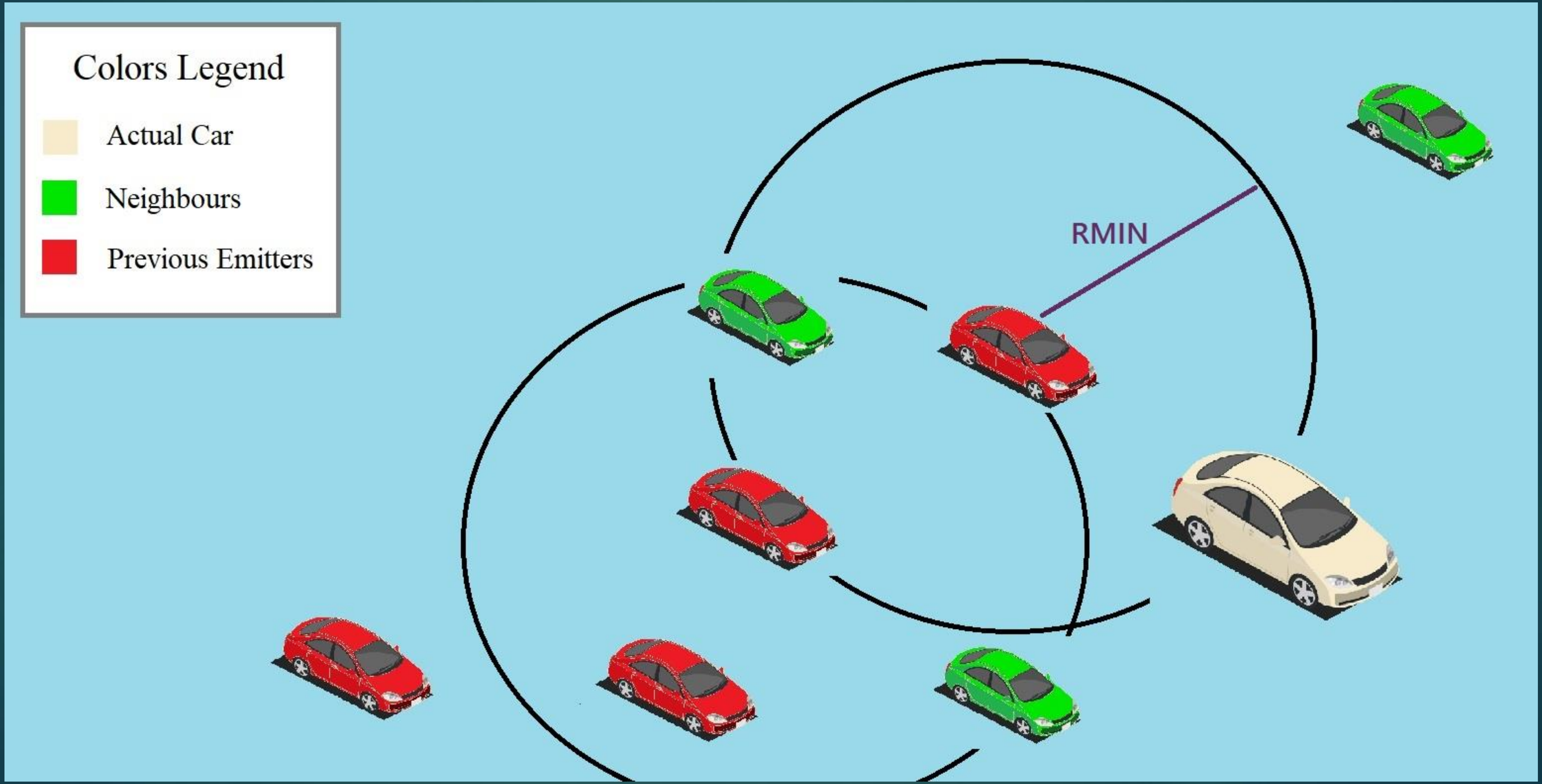
Algorithm's description - Previous Emitters



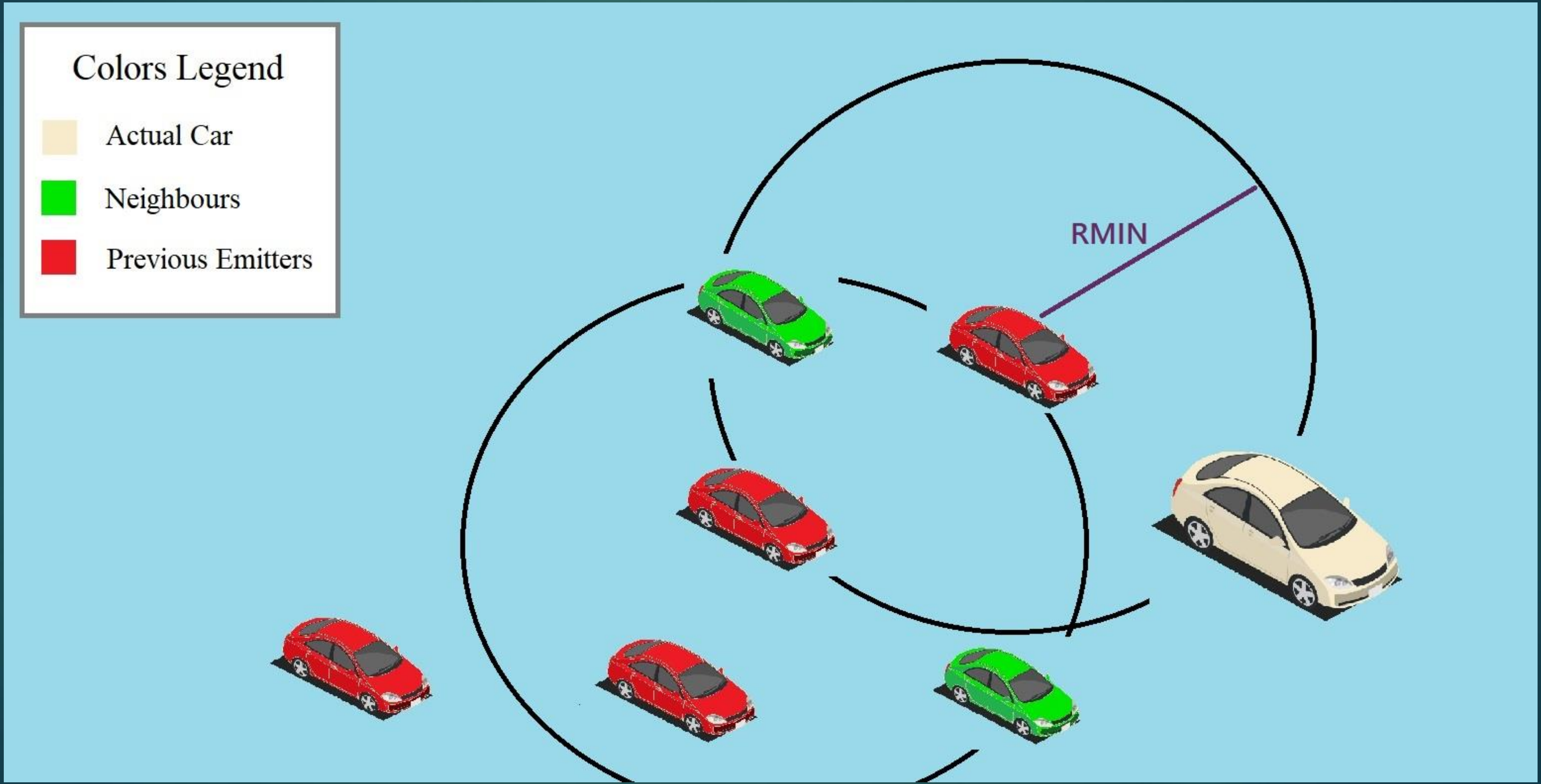
Algorithm's description - Previous Emitters & Neighbours



Algorithm’s description - Re-Broadcast



Algorithm's description - NO Re-Broadcast



Results
data available

- We have real world data: position and adjacency matrix for Luxemburg, Cologne and New York, counting several hundred cars.

- Multiple graphs for a single city, more and less dense.

- Cities have different topology: impact on performances.

Results simulations

- We built a simulator capable of understanding those data and emulating the behavior of the cars.

- We can model the simulation with different parameters, both for the environment and the dissemination algorithm.

- With the obtained results, we can evaluate the performance of the algorithm and how these differ when we change some parameters.

Results evaluation

- We evaluated different metrics:
  - Cars reached;
  - Broadcasted messages;
  - Delay to reach last car;
  - Hops to reach last car.

- We ran multiple simulations for different cities for different graphs for different parameters: lots of data!

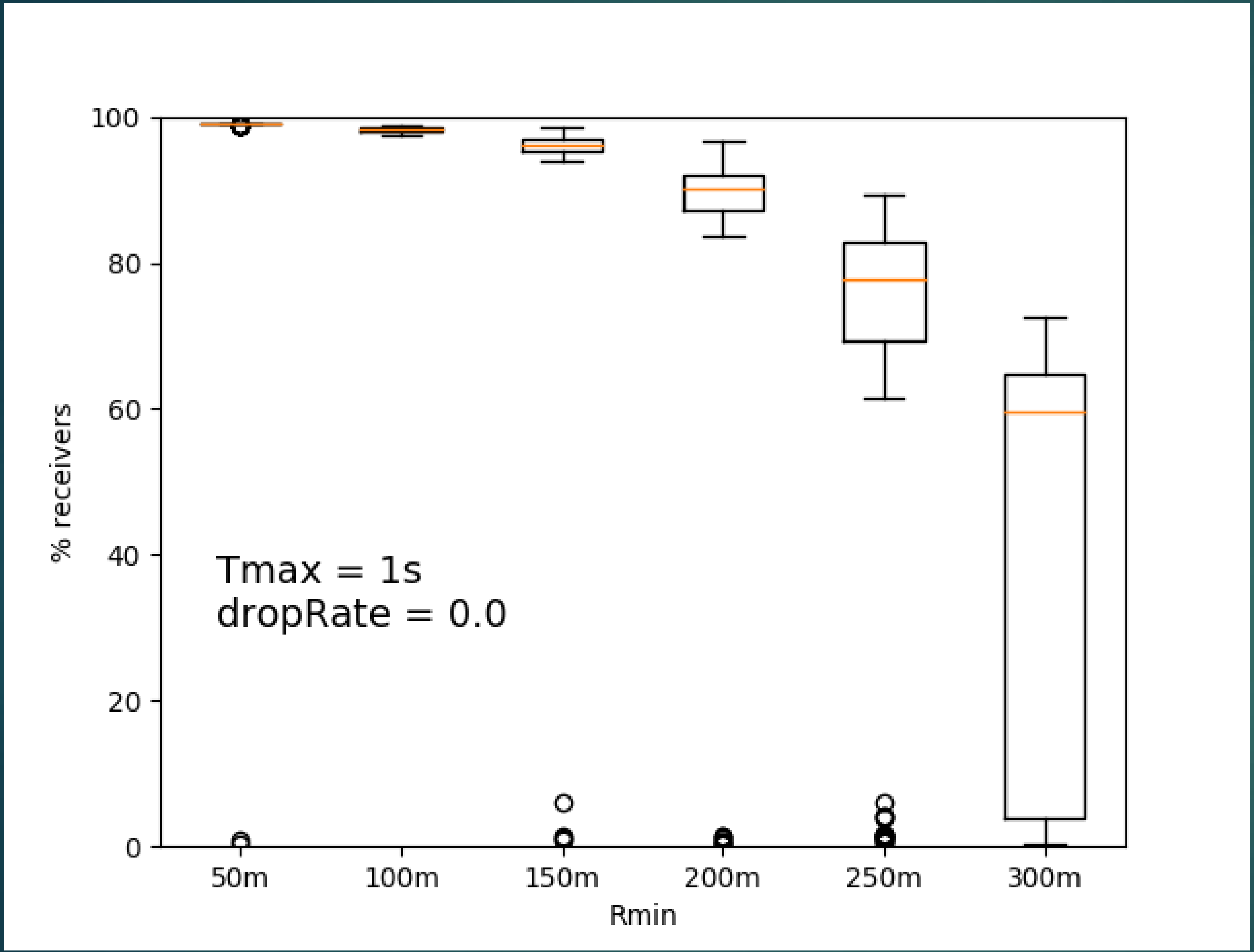
Results evaluation

- We evaluated different metrics:
  - **Cars reached;**
  - **Broadcasted messages;**
  - Delay to reach last car;
  - Hops to reach last car.

- We ran multiple simulations for different cities for different graphs for different parameters: lots of data!

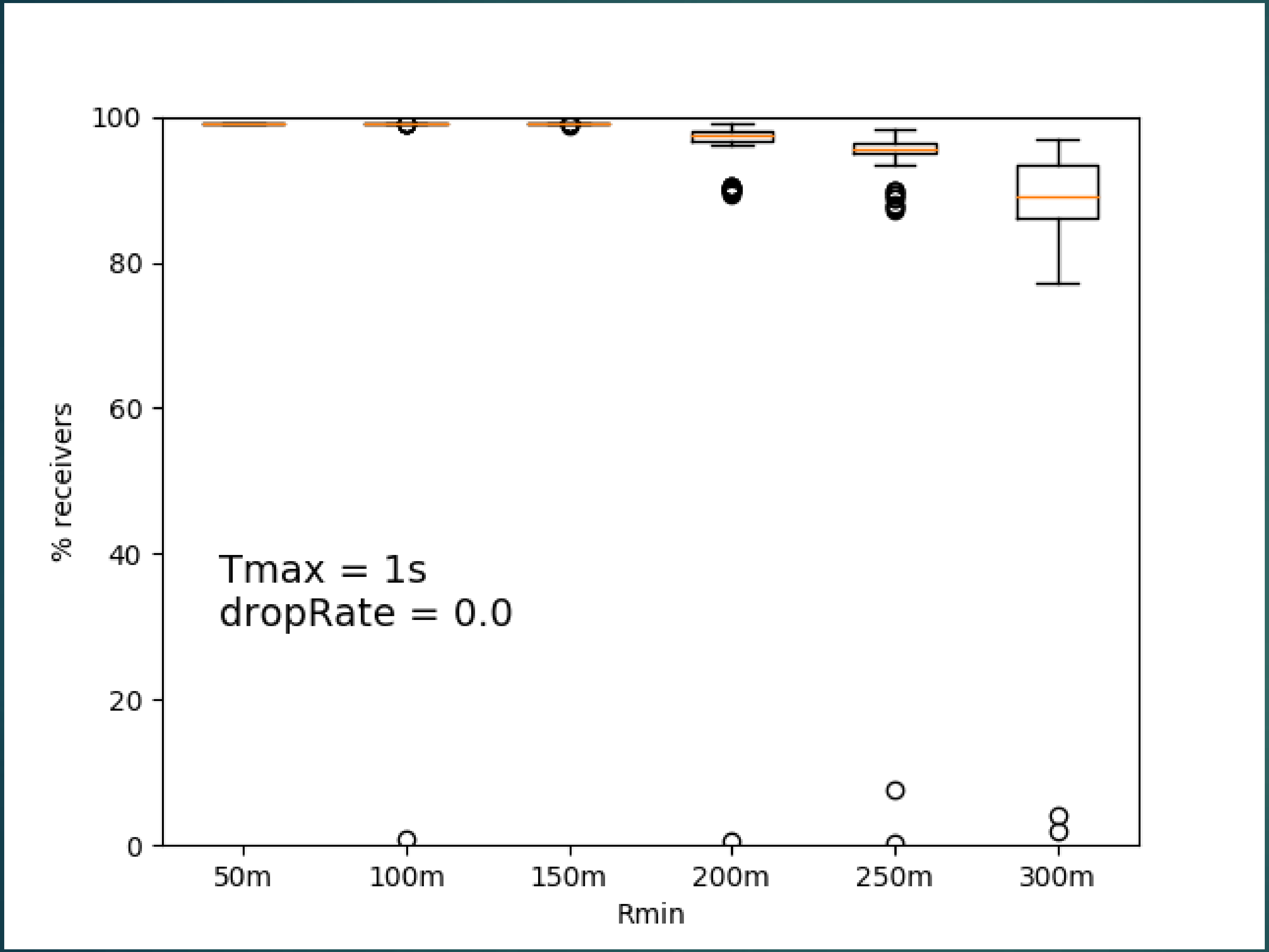
Results
impact of RMIN

- Among all the parameters, RMIN impacted performance the most.
  - RMIN small: cars may decide to broadcast even if it is not necessary, impacting network traffic;
  - RMIN big: cars may decide to not broadcast even if it is necessary, the message won't reach some cars.
- Balance between performances and network congestion.



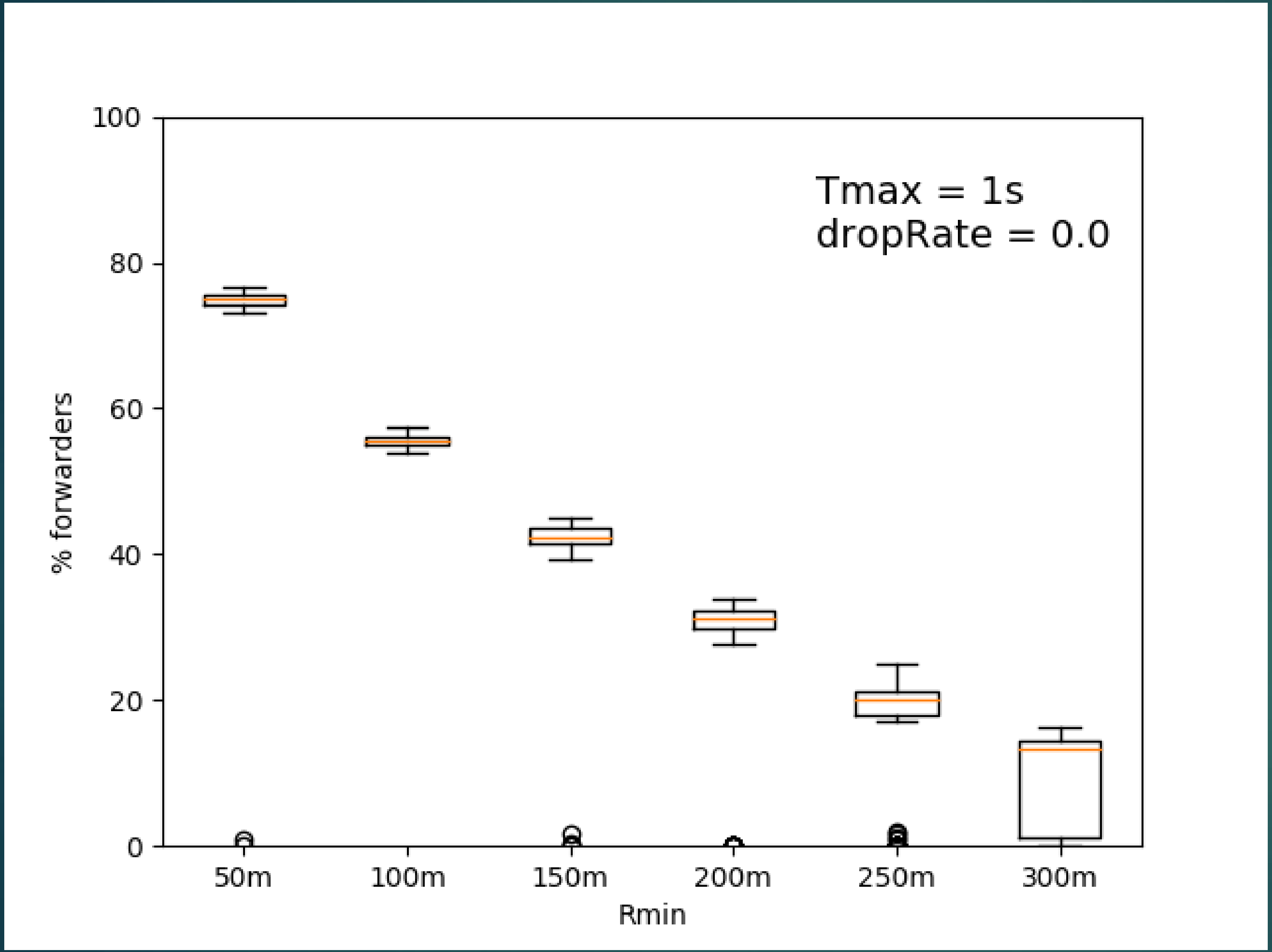
Luxemburg

- Percentage of receivers and RMIN.
- Least dense graph



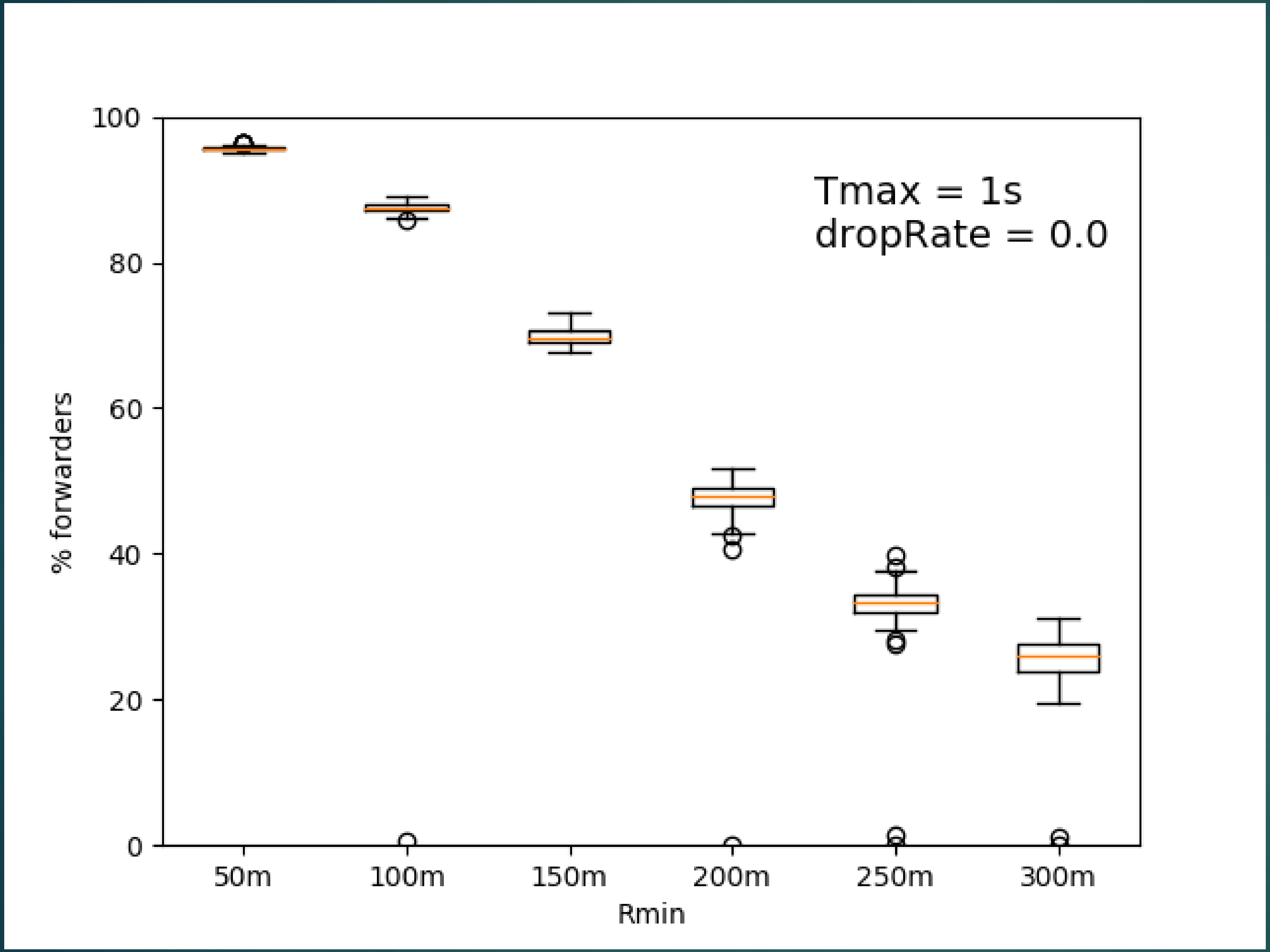
Luxemburg

- Percentage of receivers and RMIN.
- Denser graph



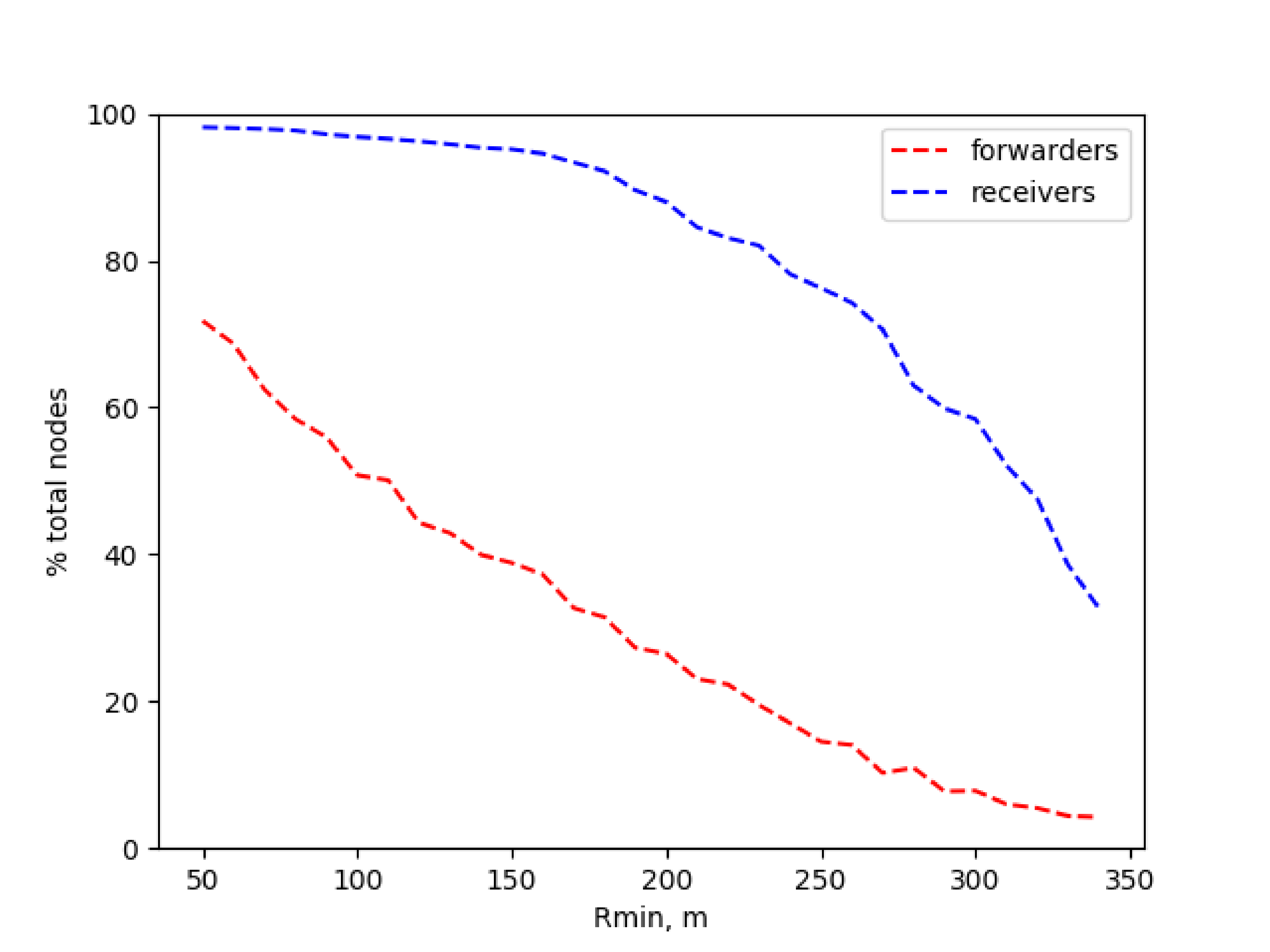
Luxemburg

- Percentage of forwarders and RMIN.
- Least dense graph



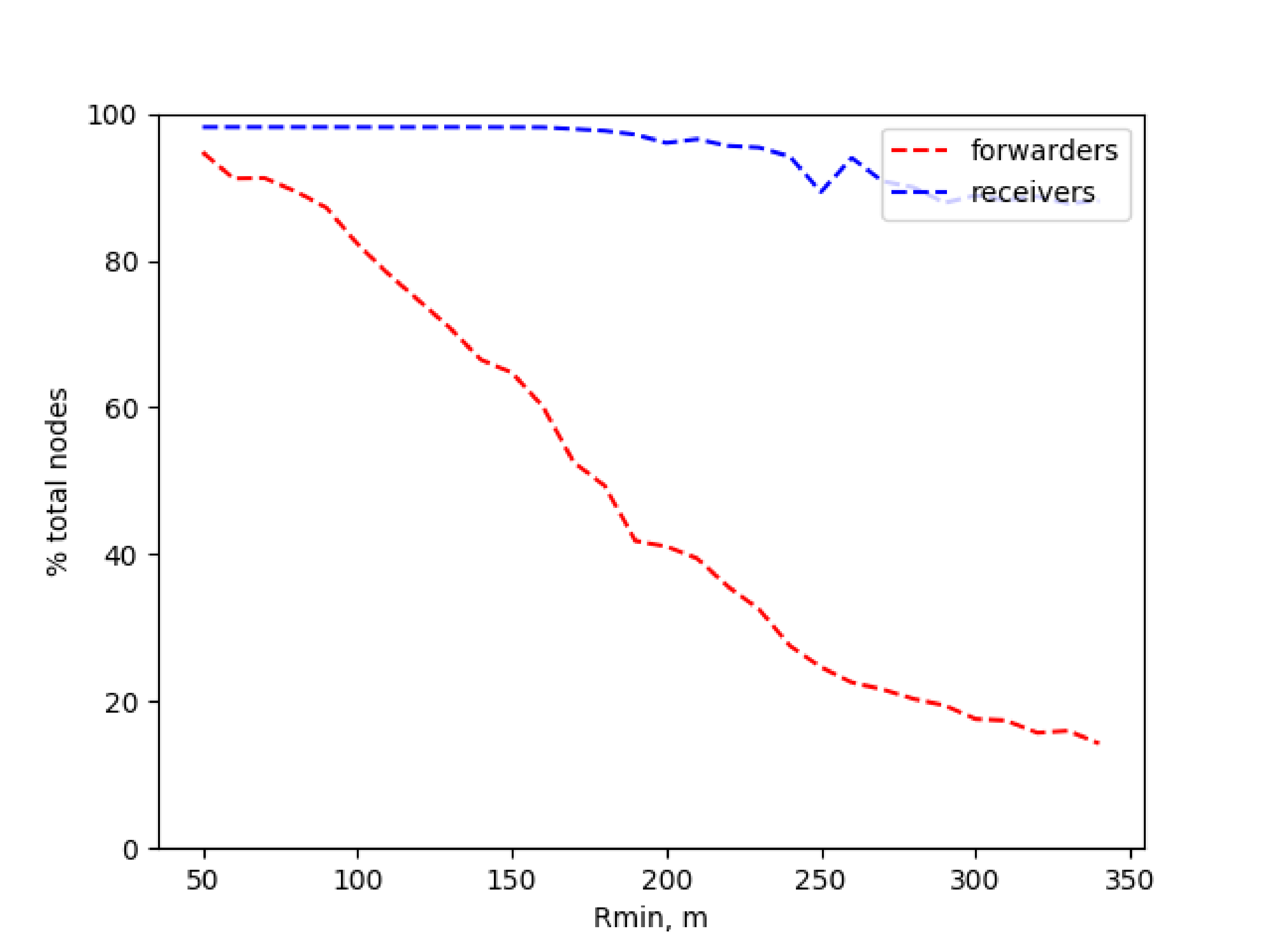
Luxemburg

- Percentage of forwarders and RMIN.
- Denser graph



Luxemburg

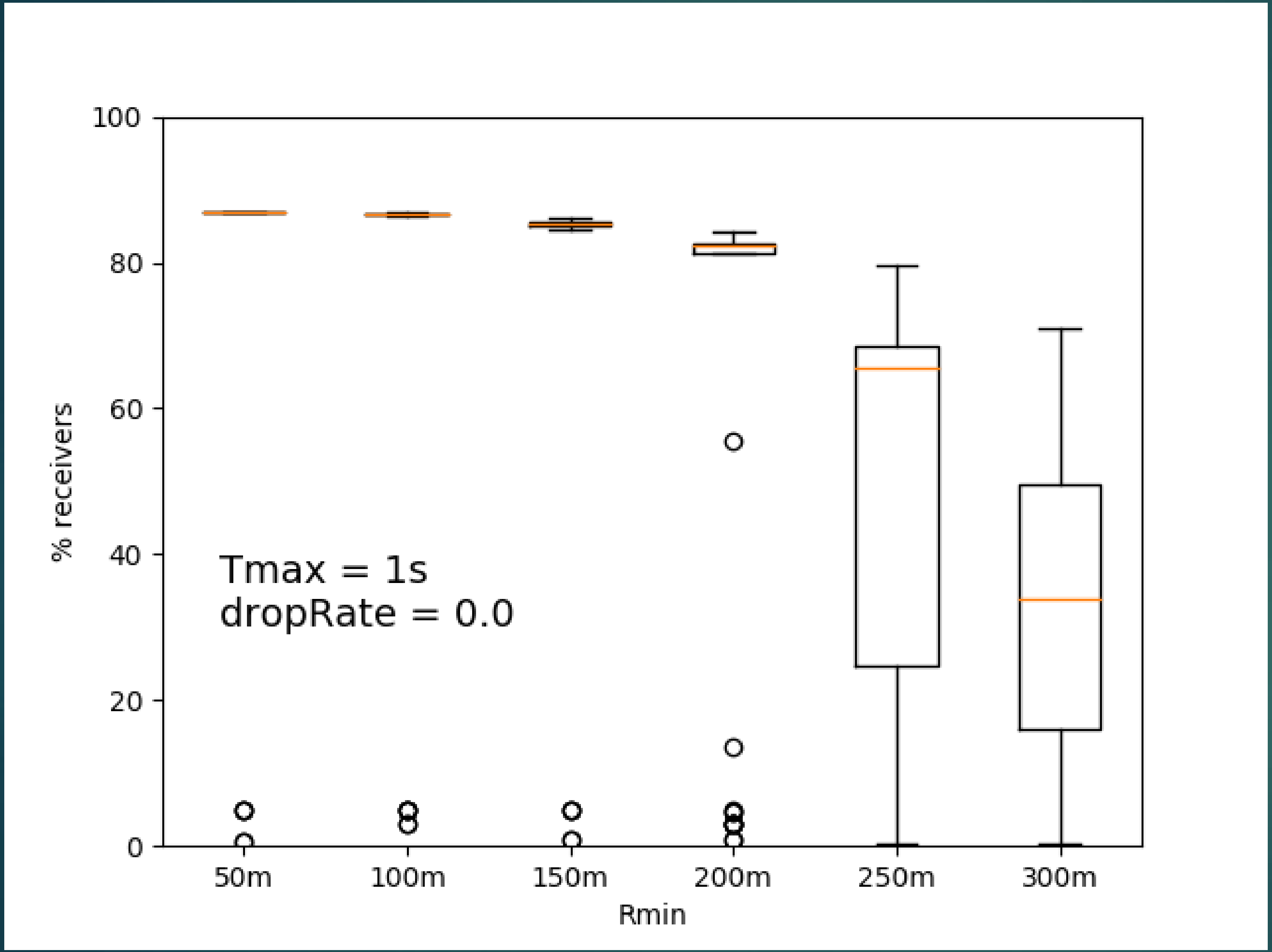
- Forwarders and receivers compared
- Least dense graph



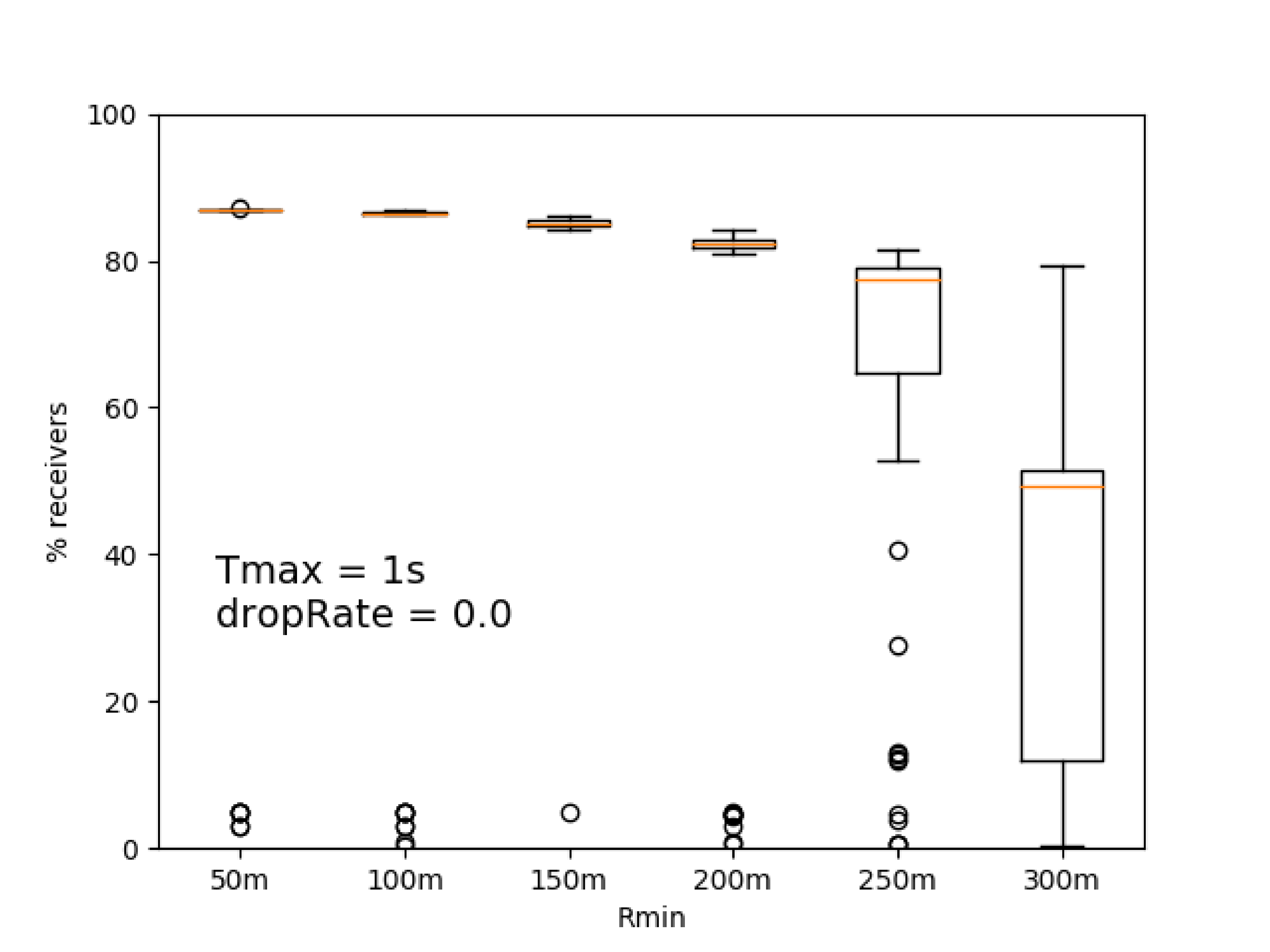
Luxemburg

- Forwarders and receivers compared
- Denser graph

Cologne

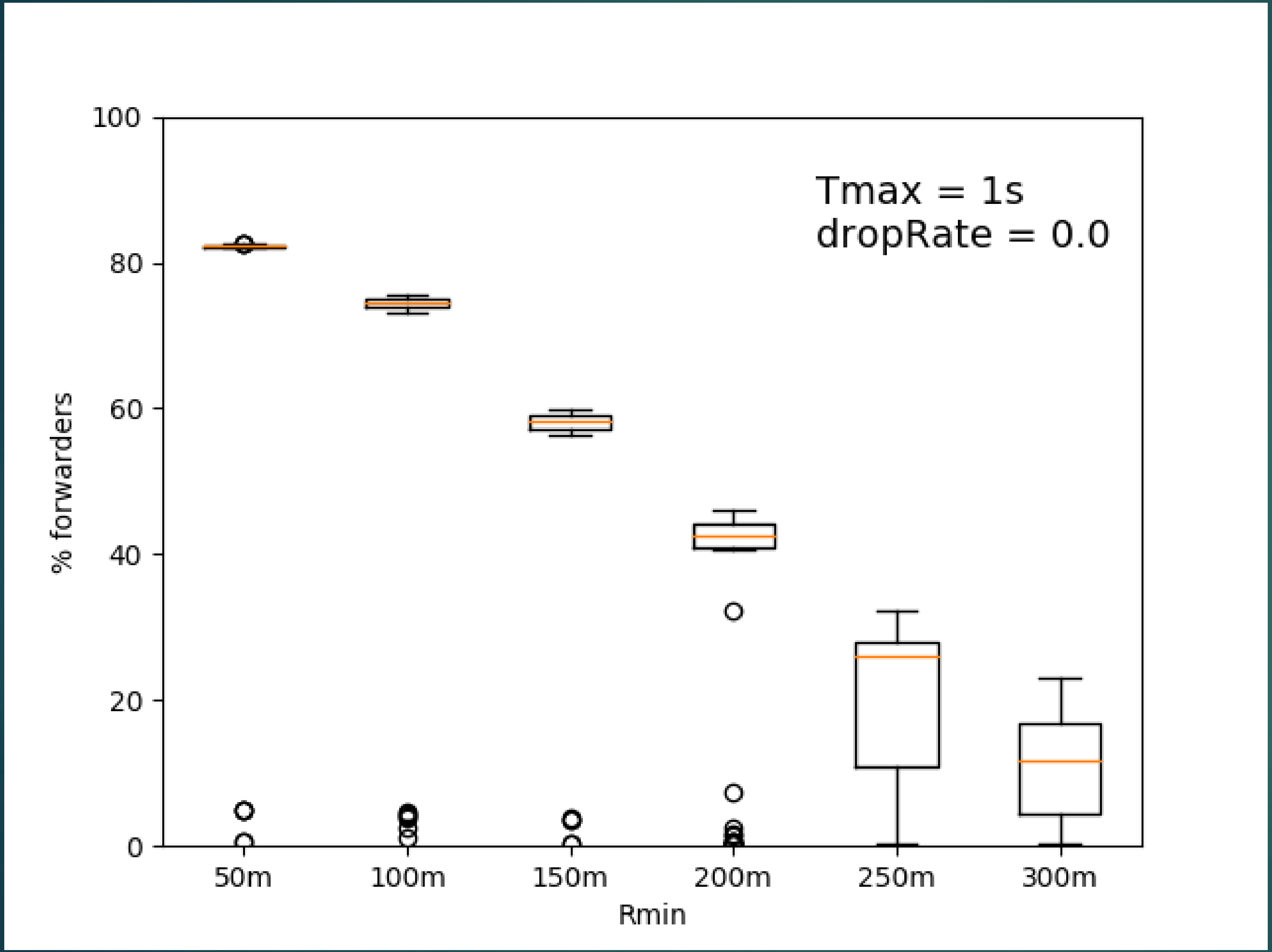


- Percentage of receivers and RMIN.
- Least dense graph



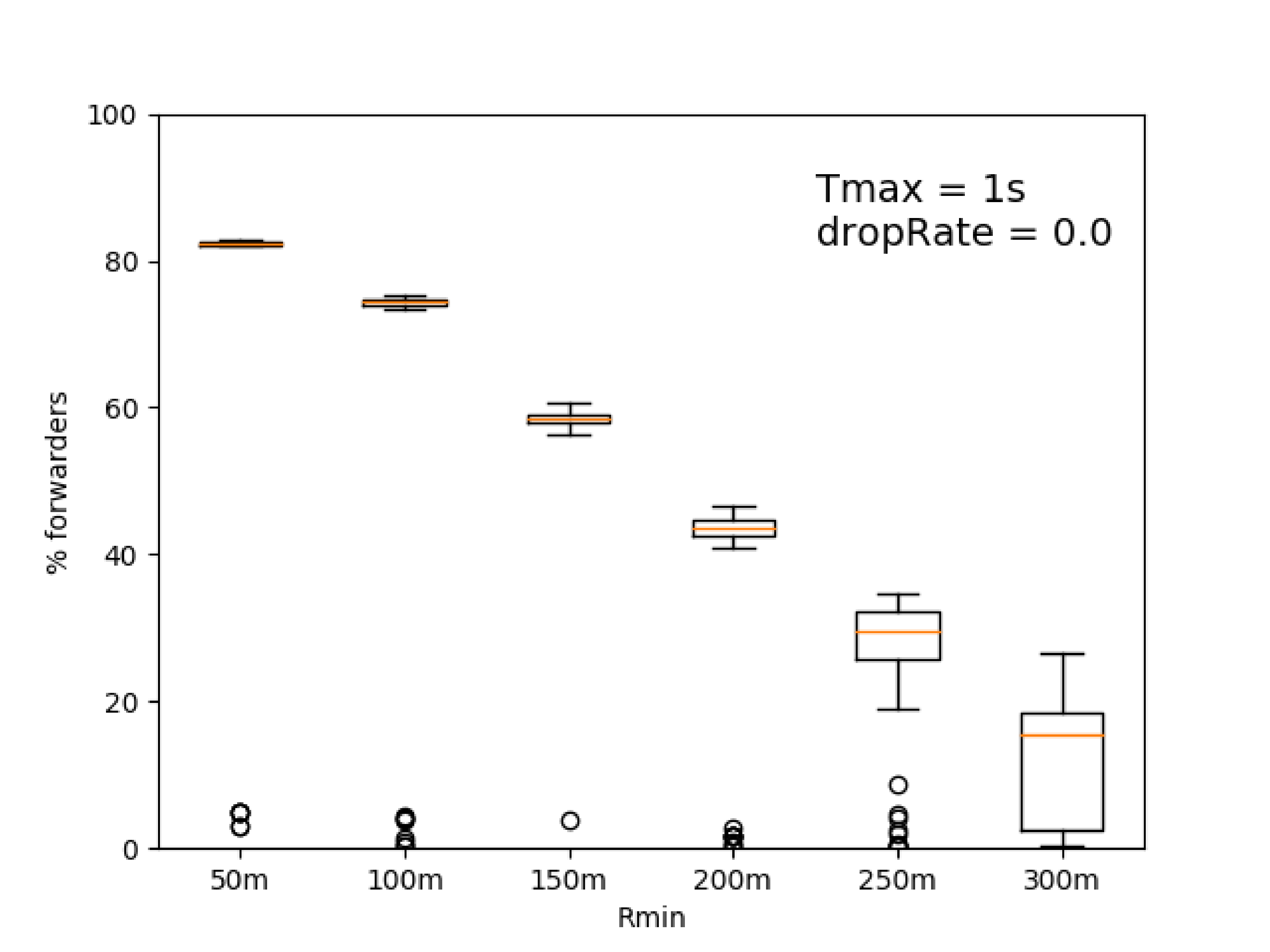
Cologne

- Percentage of receivers and RMIN.
- Denser graph



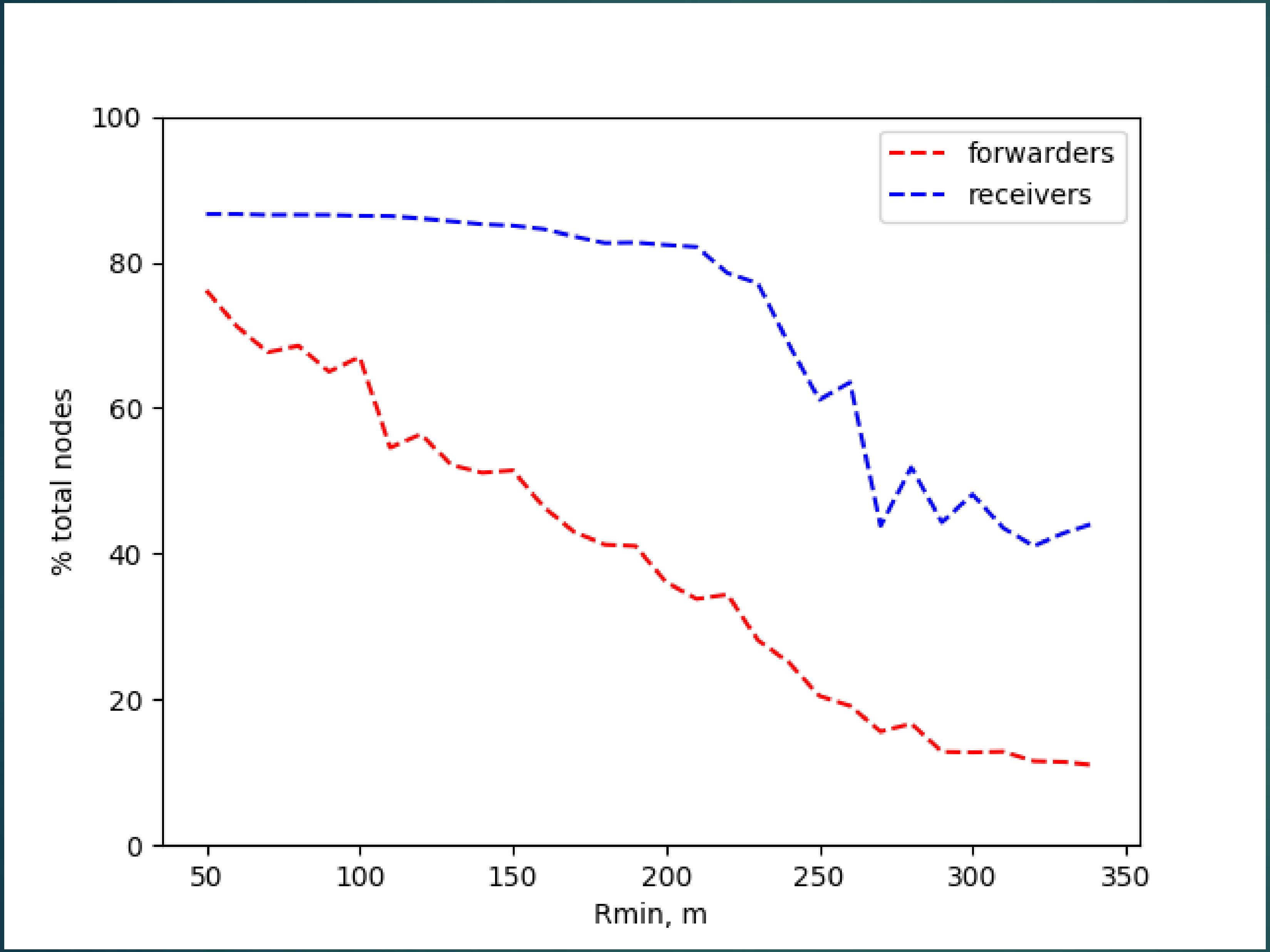
Cologne

- Percentage of forwarders and RMIN.
- Least dense graph



Cologne

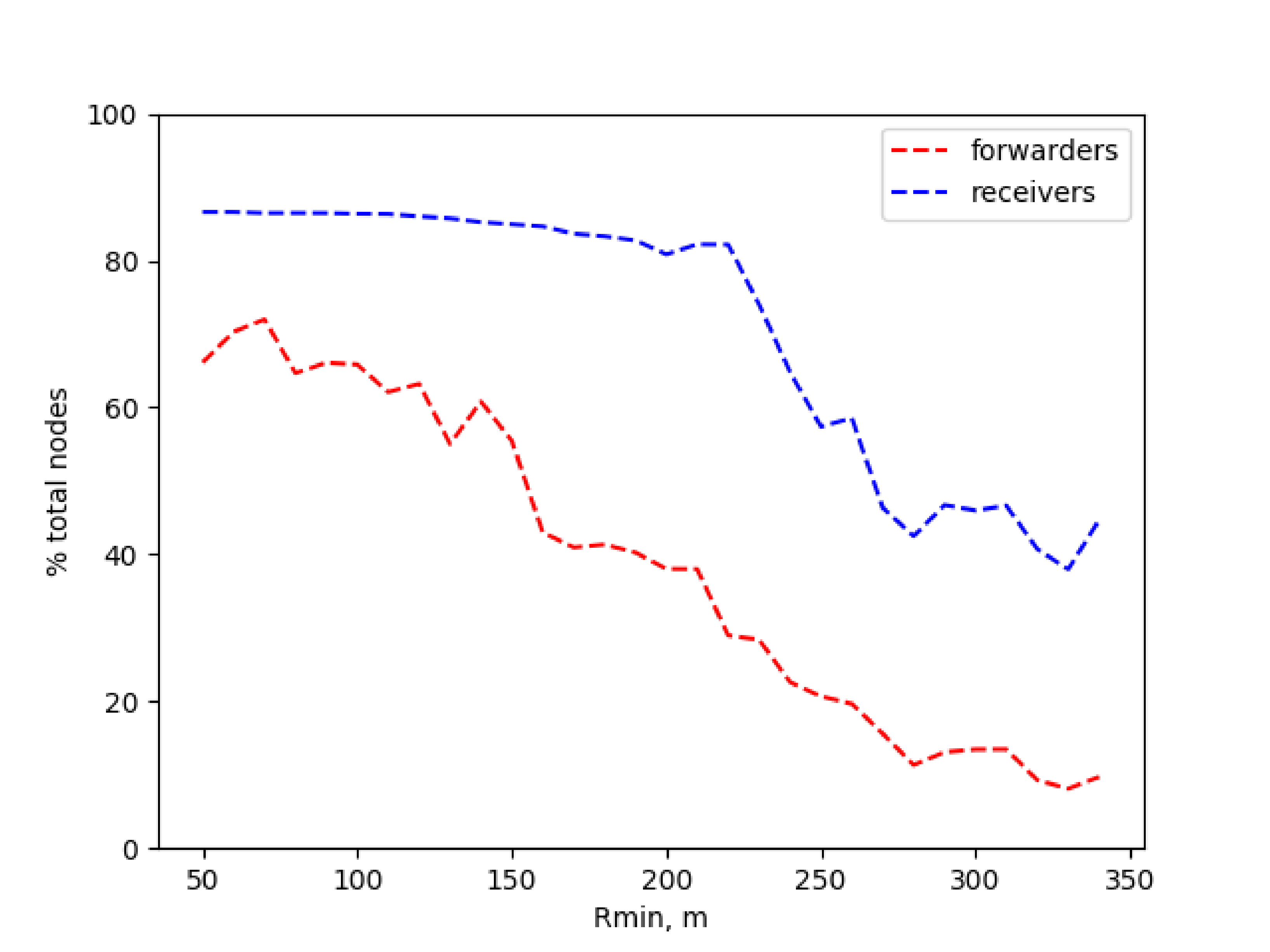
- Percentage of forwarders and RMIN.
- Denser graph



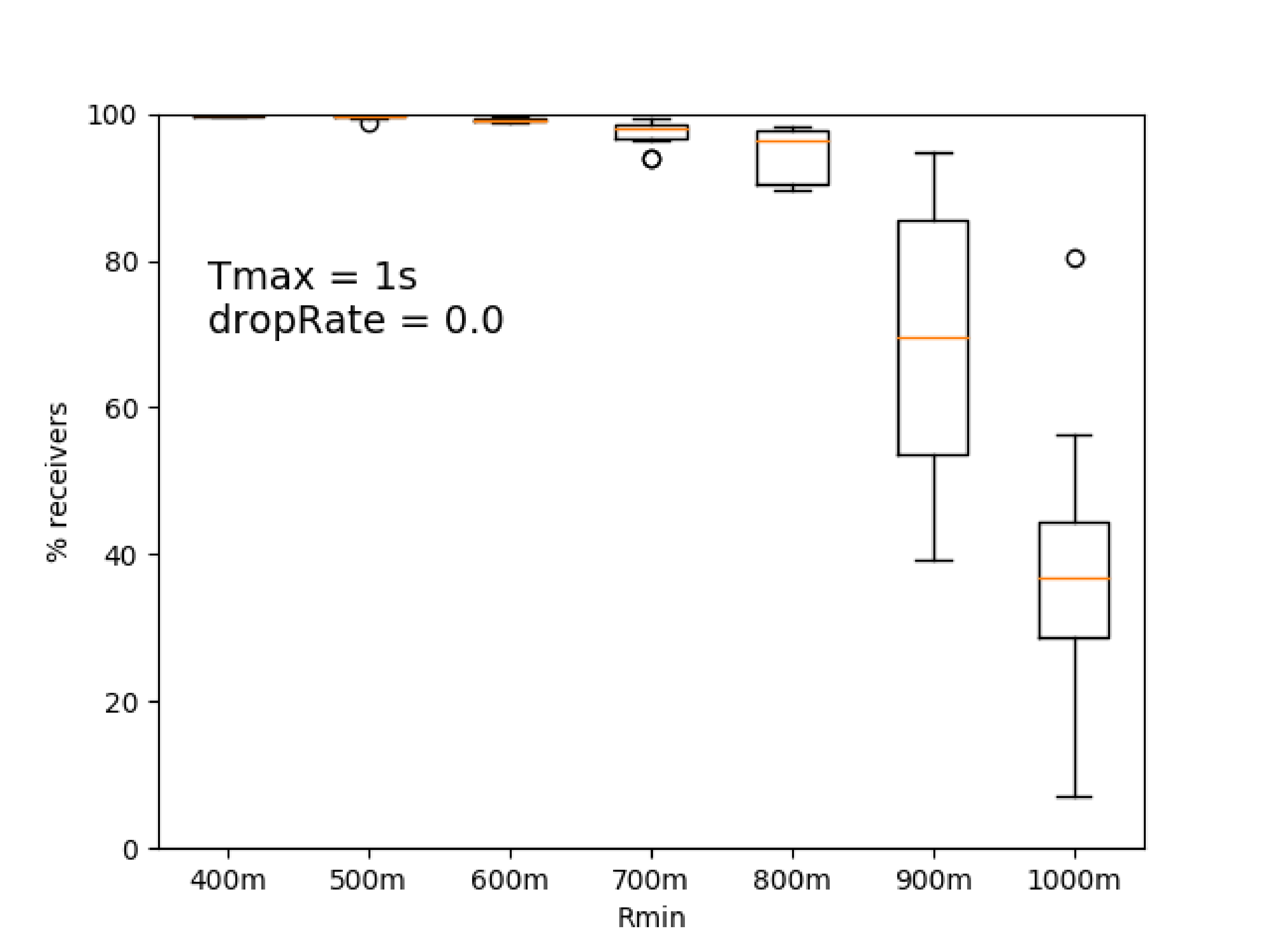
Cologne

- Forwarders and receivers compared
- Least dense graph

Cologne

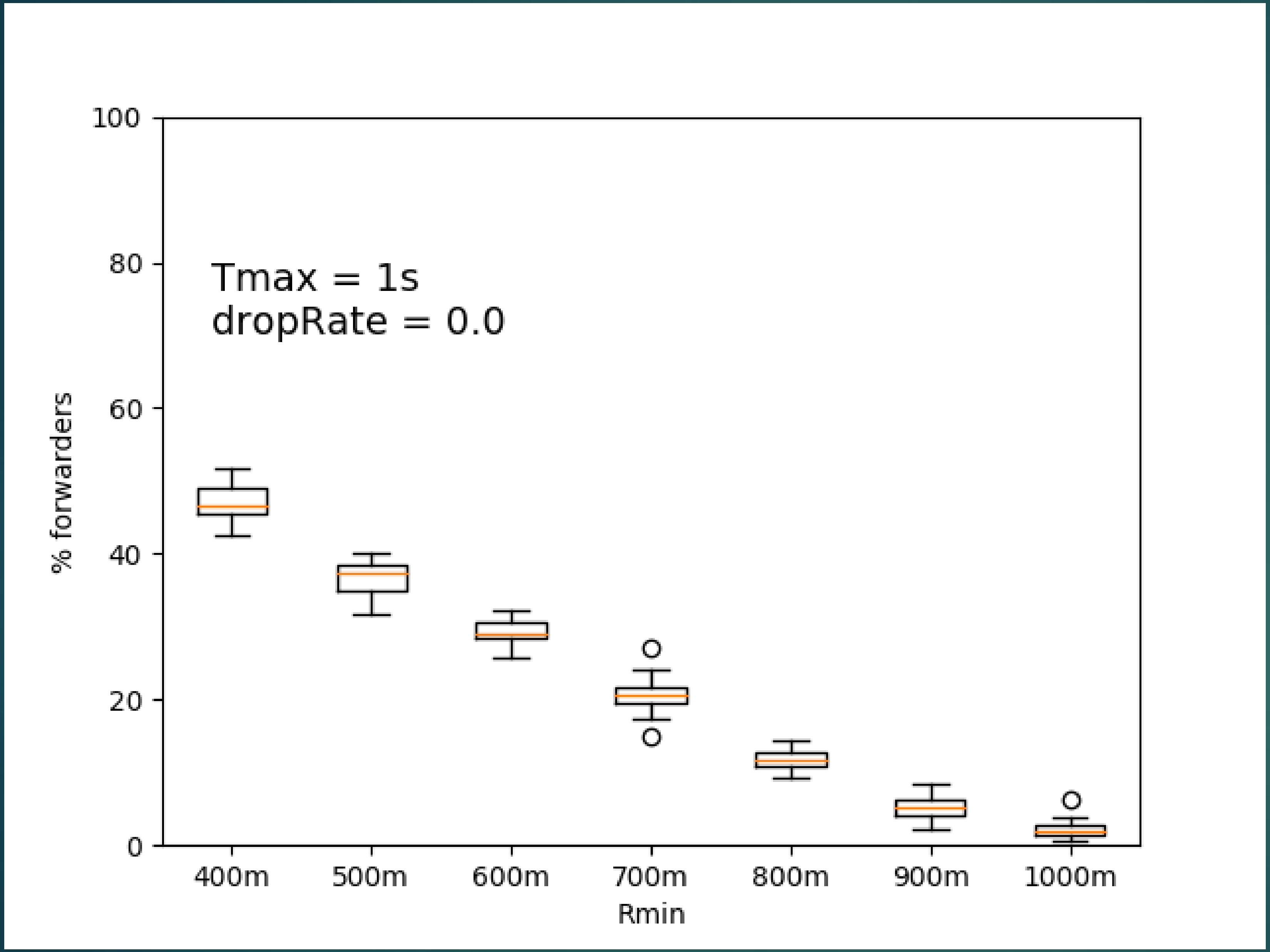


- Forwarders and receivers compared
- Denser graph



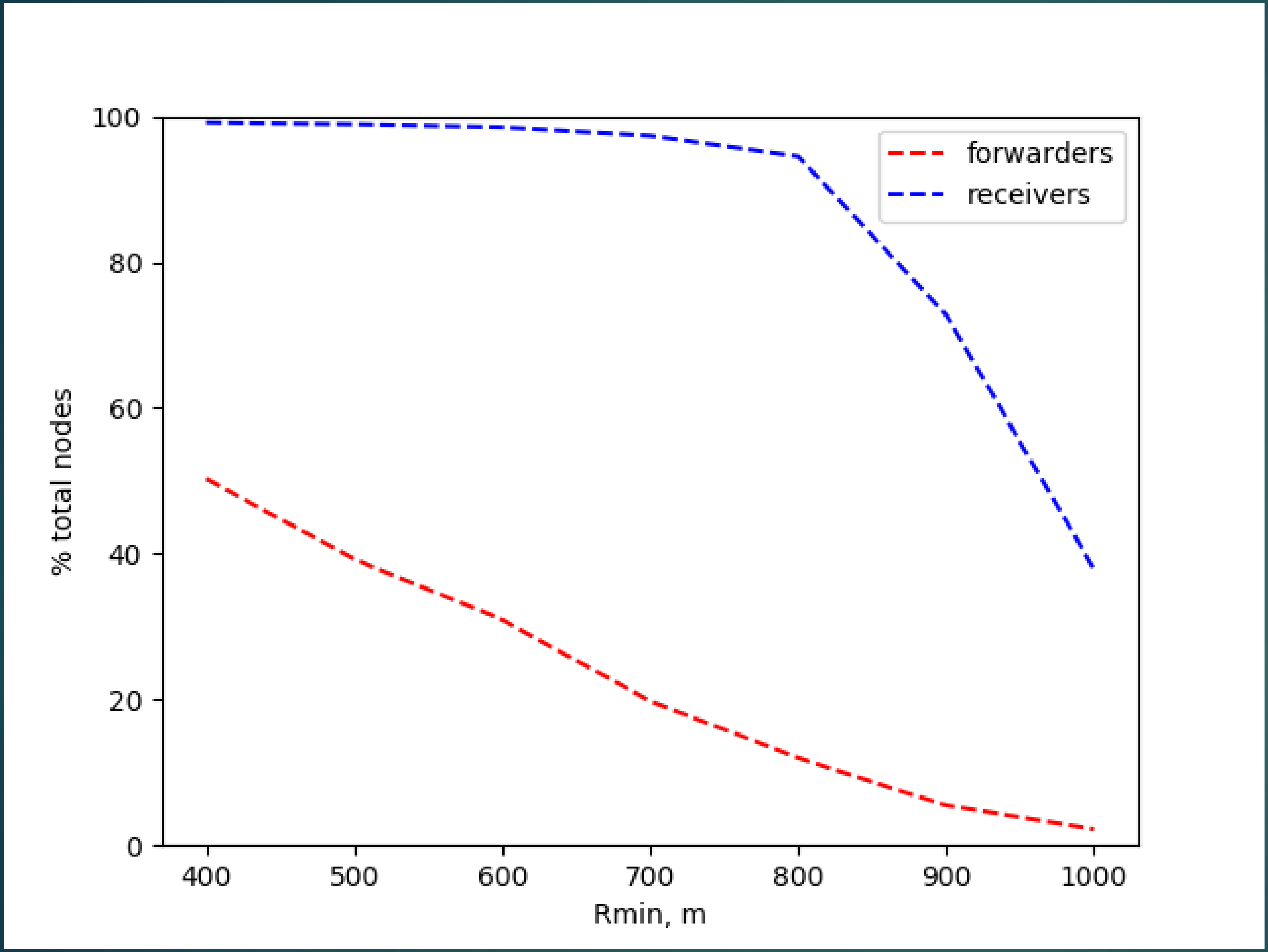
New York

- Percentage of receivers and RMIN.



New York

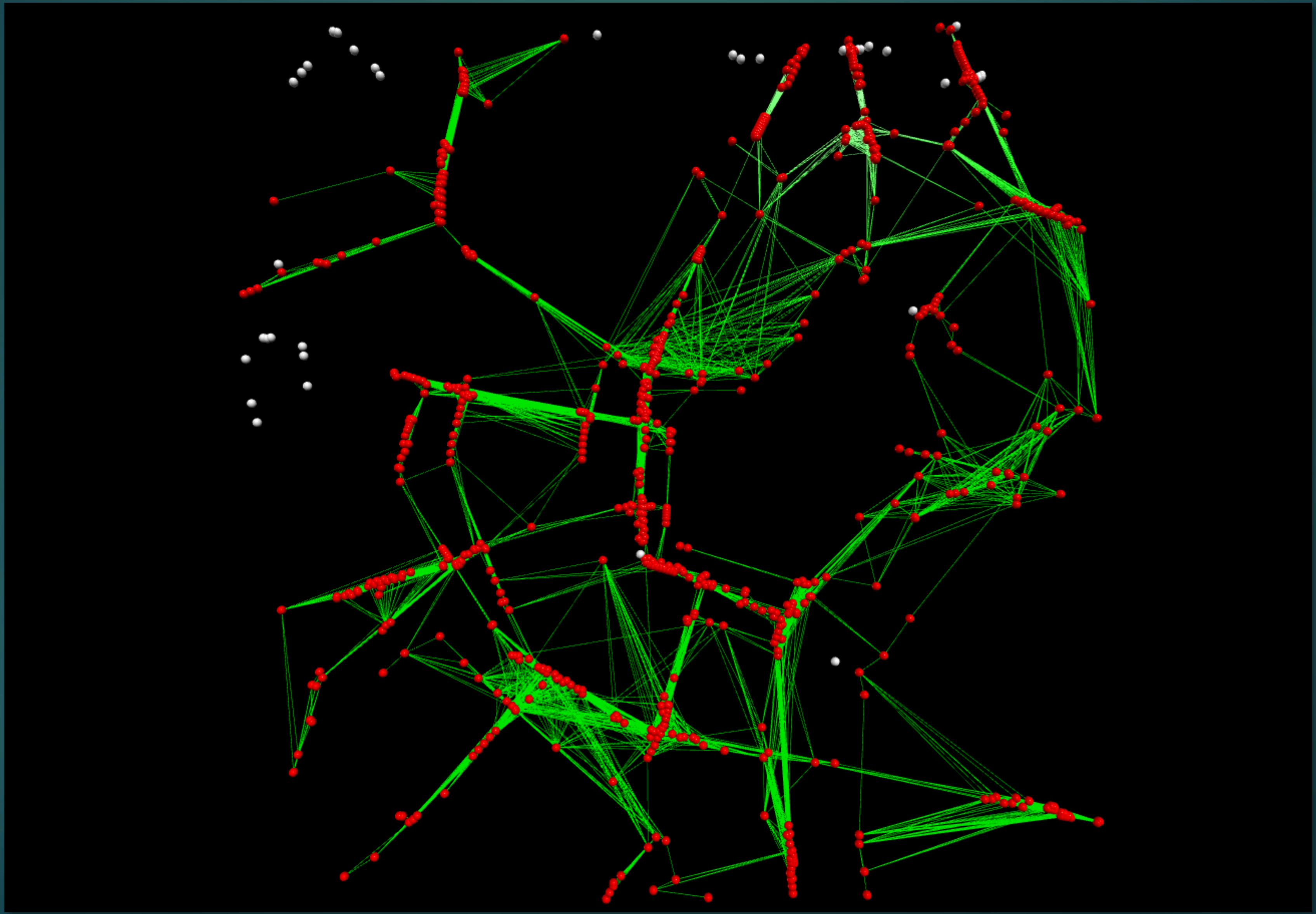
- Percentage of forwarders and RMIN.



New York

- Forwarders and receivers compared

Result's Presentation - Visual Algorithm



Result's Presentation - Visual Algorithm

```
<7478.82, 5261.88, 0>
400
Simulation ended
Vulnerable:  36
Infected:  0
Recovered:  767


Average metrics with rmin = 200
#sent messages:  305.0
#received messages:  7864.0
time of last car infection:  7.300000000000001
#hops to reach last infected car:  10.0
Cars infected ratio: 97.38%
```

DEMO