Implementation and Evaluation of a Mobility Model in a Peer-to-Peer Simulator

Bachelor Thesis

by

Tobias Korfmacher

born in
Düsseldorf

submitted to

Technology of Social Networks Lab
Jun.-Prof. Dr.-Ing. Kalman Graffi
Heinrich-Heine-Universität Düsseldorf

Mai 2015

Supervisor:
Jun.-Prof. Dr.-Ing. Kalman Graffi
Prof. Dr. Martin Mauve

Abstract

This bachelor thesis is about the implementation of a mobility model to a peer-to-peer simulator. Therefore we use the simulator *PeerfactSim.KOM* [Pee15]. Originally it contains a simple mobile movement model.

The original idea was to create a foundation for realistic mobility movements. In the simulator *PeerfactSim.KOM* we implement a new network layer which implements the *NetPosition* for every host. In order to create realistic models in the future we consider a new design of the package.

The implementation of these interfaces establishes a basis for the future. Thereby it is possible to add some new models without changing the source code in other files. We simply have to implement the predefined functions in the interface.

To verify the correctness of the implementation we create some basic models and evaluate them.

Conclusively we can assume that the implementation enables more realistic mobility simulations and evaluations.

Acknowledgments

First of all, I would like to give my appreciation to Jun.-Prof Dr.-Ing. Kalman Graffi who gave me the opportunity to write my thesis.

Additionally, I would like to thank my advisor Tobias Amft. Exchanging ideas advanced my implementation and was very contributing, too.

I am also grateful to Ahmad Reza Cheragi for pushing me.

I wish to express my sincere thanks to Sandra Hodißen, Alessa Köhne and Michael Girps for their commitment.

Finally, I thank my family and friends for being patient and supportive during the last months.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In section 1.1 we start with the introduction by giving the motivation for this bachelor thesis. Thereafter the structure is described in section 1.2.

1.1 Motivation

Peer-to-peer networks are normally decentralized and consist of equal network nodes. That means every participant has the same rights and takes part in the services offered, for example file sharing, direct communication or voice over IP. These are the main differences to server - client architectures, whereby the servers offers these services. The clients are only users. It is often centralized.

These peer-to-peer networks are more popular due to safety aspects and its possibilities.

The use of simulators for peer-to-peer networks and distributed systems allows the user to analyze overlays, routing protocols or applications.

Nowadays mobile phones are popular devices. They contain strong processors and high bandwidth communication possibilities. Furthermore, the cellphone providers offer increasingly high traffic volume to use internet at a low price. In order to test movement behavior, working of routing algorithms and other services of moving nodes we use simulations. Since we have up to 10.000 nodes used, research would be too expensive. Moreover, new protocols would have to be tested.

1.2 Structure

A main part of the bachelor thesis addresses the implementation of different interfaces in order to realize the mobility model.

In chapter 2 the simulation software and the mobile model from Carsten Snider as well as its drawbacks are presented.

Considerations regarding building a new mobility model and its advantages are described in chapter 3.

Chapter 4 includes a description of the drafted interfaces and realized models.

In chapter 5 the settings for testing the implementation are presented. The simulation results are also given.

Ultimately, chapter 6 summarizes the results. Additionally, we discuss some ideas for possible future work on *PeerfactSim.KOM* with the *MobilityModel*.

Chapter 2

Related Work

In this chapter the mentioned peer-to-peer simulator *PeerfactSim.KOM* is explained in section 2.1 and the mobile network model which was implemented by Carsten Snider is described in section 2.2.

2.1 PeerfactSim.KOM

PeerfactSim.KOM [Pee15] is an open source event-based simulation software with focus on large peer-to-peer networks. The *Technical University (TU) Darmstadt* [Tec15] first developed the simulator. Continuing work and support has been done by the *University of Paderborn (UPB)* [Uni15] and the *Heinrich-Heine-University Düsseldorf (HHU)* [Hei15]. The simulator generates different layers for every host shown in 2.1.

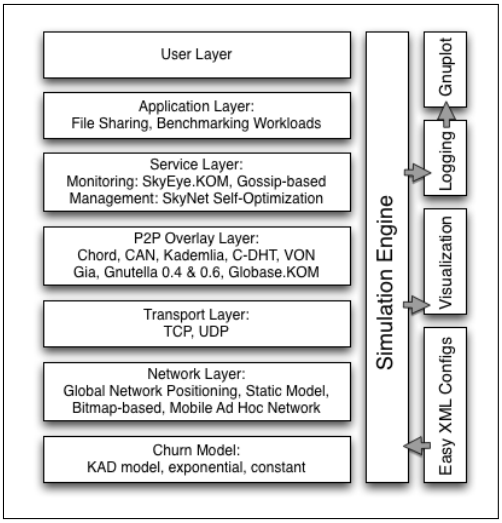


Figure 2.1: PeerfactSim.KOM architecture
[Ka15]

It is configured in an *XML - file*. We have to set up all different layers by various modules. The network layer is the relevant layer used in this thesis. We give an exemplary extract of this config file in Listing 2.1. It sets up the configuration of the *MobilityNetLayer*. With the simulator and the *NetLayer* it is possible to receive realistic data for delay, loss and host positions.

```
<NetLayer
class="org.peerfact.impl.network.wireless.impl.MobilityFactory"
downBandwidth="5000" upBandwidth="1000" >
<World
class="org.peerfact.impl.network.wireless.impl.world.HardWorld"
worldX = "$world_X" worldY ="$world_Y"/>
<MobilitySubnet
class="org.peerfact.impl.network.wireless.impl.MobilitySubnet"/>
<ObstacleModel
class="org.peerfact.impl.network.wireless.impl.obstacle.RandomObstacle"
tally="20"/>
<PlacementModel
class="org.peerfact.impl.network.wireless.impl.placement.RandomPlacement"/>
<AccelerationModel
class="org.peerfact.impl.network.wireless.impl.acceleration.ConstantSpeed"
speed="35"/>                    <!-- in mm/sec-->
<PacketLossModel
class="org.peerfact.impl.network.wireless.impl.sending.packetLoss.SquaredPacketLoss"
stop ="100000"/>
<LatencyModel
class="org.peerfact.impl.network.wireless.impl.sending.latency.MobilityLatencyModel" />
<MovementModel
class="org.peerfact.impl.network.wireless.impl.movement.RandomMovement"/>
</NetLayer>
```

Listing 2.1: Example config file for *MobilityNetLayer*

We want to assign the *NetLayer* to every host. Therefore, we choose the string *<NetLayer*. Thus we start the configuration. We choose the *MobilityFactory* with the variables of bandwidth.

The following interfaces are necessary for this specific factory.
We first choose the *World - interface* with the selected *HardWorld* and a dimension. The first dimension is described by world_X, the second by world_Y.
In the next step we pick *MobilitySubnet* for the simulation subnet. The next interface creates 20 random obstacles. Additionally, the *PlacementModel* and the *AccelerationModel* are assigned. At this point the moving speed is 35 $\frac{mm}{sec}$. Then we set up the *SquaredPacketLoss* for the simulation. At a certain distance we decide whether the message should be sent or dropped on the bases of the value at the variable “stop”. Finally, the *LatencyModel* and the *MovementModel* are configured.

All created interfaces and models are described in chapter 4.

Furthermore, the simulations are managed by the simulation event queue. The queue contains all planed and created events which have a time stamp. The simulator starts with the simulation and looks for an event with the simulated time. If there is an event with the time signature the simulator automatically executes it.

Additionally, you can select an analyzer in the *XML - file*, too. For example the *DefaultNetAnalyzer* which creates metrics of sent messages or online hosts during the simulated time. After the simulation, the chosen analyzer automatically writes *png-files* and *pdf - files* containing the results if *GnuPlot* [Tho15] is already installed. Figure 2.2 gives an example. It shows the online hosts during a simulation of 120 min simulation time and 50 hosts.

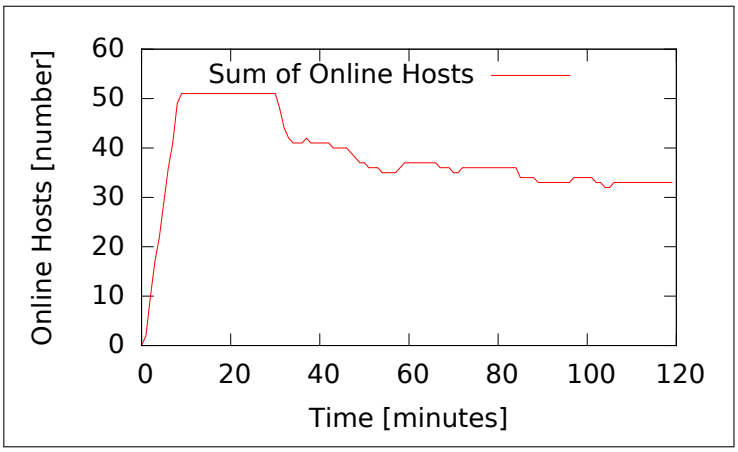


Figure 2.2: Mapping of online hosts during a simulation

All details about *PeerfactSim.KOM* refer to the **Community Edition** released on 17.10.2013.

2.2 Mobile Network Model by Carsten Snider

An implementation of mobile movement is already given in the simulator *PeerfactSim.KOM* [Pee15]. It was developed by Carsten Snider during his bachelor thesis “Mobility Aware Peer-to-Peer Networking” [Car09] submitted on 03.03.2009 at the *Technical University (TU) Darmstadt* [Tec15].
He generated a new network layer which simulates a mobile network with different hosts. This implementation computes the delay between a sender and a receiver. The delay is represented by a certain number of hops between sender and receiver.

Here Snider’s model generates a static link between two hosts and saves the number of hops to reach a destination. If no path has been found, the sender is set offline for the rest of the simulation. Concerning moving hosts this procedure does not make sense. He designed a mobile movement

which generates a random position in the simulation area called *Random Waypoint Mobility Model*. We will refer to this aspect in subsection 3.2.2. The procedure is repeated every minute during the simulation. Thus every movement is of limited duration. If the new position cannot be reached in time, the node moves to an intermediate position. It continues moving in the next time section.

Furthermore, Carsten Snider divided the simulation area into equal squares each called *Quadrant*. With rising number of *Quadrants* it is easier to find the path to the receiver node.

Figure 2.3 gives a graphical example for latency calculation between node 0 and node 9. The latency model counts the number of hops to compute the delay. Thus we also recognize that the path finding algorithm does not necessarily find the shortest path.

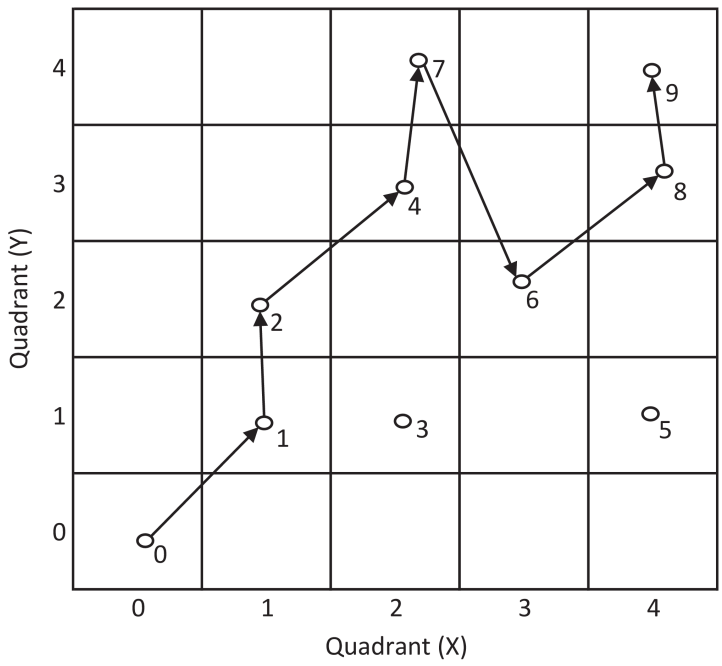


Figure 2.3: Finding a final path [Car09]

The next chapter presents the design of the implementation and includes a short overview of some movement models and routing algorithms.

Chapter 3

Design

We discuss the idea of building a new mobility model in section 3.1. Furthermore, we introduce some movement models in section 3.2 and some routing algorithms in section 3.3.

3.1 Considerations

First of all, we decide to use the *PeerfactSim.KOM* [Pee15] for it is event based and has a hierarchical structure. This structure allows to implement a new model of the *NetLayer*. As we said above this model is chosen in the *XML-file*. By means of using this *XML-file* it is possible to select other models for the simulation. Furthermore, new modules can be developed and tested. The *MobilityModels*' structure allows to create new models, e.g. new movement models. Thereby, the source code of other models does not have to be changed. Several models will be presented in the following chapter.

One possibility of such a movement model would be a *MobilityHost* asking his *MovementModel* for a new position. In doing so it is possible to give various models to different nodes. Some models calculate the new position considering the previous one. For this reason the node transmits the new position to its *MovementModel* in case of request.

In his mobile movement model, Carsten Snider asks for a new position every minute. The *Mobility-Model* does not aks ask for a new position before the previous prescribed position is reached. Therefore, this new model is more realistic regarding the movements of nodes.

A short example:
A node sends a message to another node. Afterwards, the subnet asks the *NetlatencyModel* for the latency of the message. This new model creates a *MobileMovementManager* that updates the position of all nodes. The Snider Model can only deliver the position information in an interval of one minute.

On the contrary, the position information in our *MobilityModel* are generated in real time. Here the *NetLayer* is the sender. It calculates the distance towards the receiver by means of real time positions. This procedure is more precise.

It is necessary to implement a *Move_Event*. The *EventQueue* list contains this event. The *MobilityNetLayer* asks its *MovementModel* for a new position. In relation to the recent speed it calculates the movement time. After this calculation the simulator schedules this event with the adequate time stamp to the event queue list. Similar to a car that stops at a traffic light, a movement pause is possible. Furthermore, this implementation enables to send messages between each other given that the distance between sender and receiver is not too big. This decision is made by the *PaketLossModel*. As soon as nodes move the distance changes. For this reason the distance can be reduced in the next sending-receiving process and the sending is possible.

We built a new factory. The factory allows to implement new algorithms on the basis of the modular structure. The following graphic 3.1 shows the design.

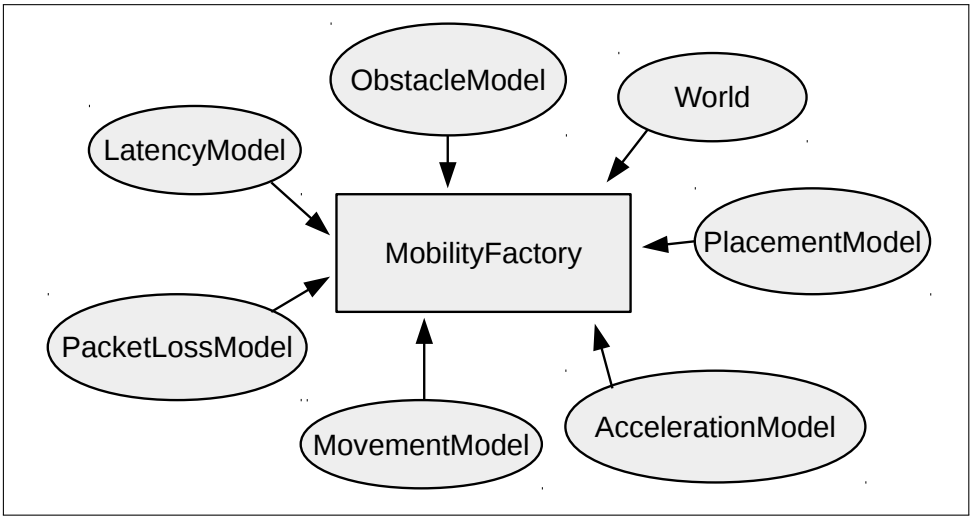


Figure 3.1: *MobilityFactory* design

This *MobilityFactory* issues the *NetLayer* and transfers all modules that have been chosen in the *XML - file* to it.

Carsten snider developed the first mobile movement model for the *PeerfactSim.KOM* simulator. With

its implementation the simulator works pretty well. Therefore, Snider describes it as a

> "[...] foundation for the modelling of a mobile multi-hop network in the simulator *PeerfactSim.KOM*." [1]

Nevertheless, a *PaketLoss* is not possible. On top this model does not feature obstacles and prohibits the implementation of further models such as new world models or placement models.

In this case a possible model would be e.g.the *ObstacleModel*: An obstacle is a figure that is created in the world. The host has to pass. Obstacles can be of different shape. Furthermore, obstacles can be created by means of a data base.

A further problem is described in the following:

If Carsten Snider's algorithm does not find any path between sender and receiver the host turns itself offline during the rest of the simulation. In this case it is not possible to send or receive further messages.

Mobile nodes move. If the distance between two hosts grows too big, they are not able to send messages to each other. But, after further movement it might be possible again. However some overlay algorithms delete those nodes from their routing table. Thus they are not able to receive messages for the rest of the simulation time. It happens for example in a *Chord - Overlay*. To avoid this, the overlay algorithms have to be changed in the future in order to test mobile movements.

The above-mentioned mobility model design enables implementations of *WayPoints*. It is possible to choose a predefined path for the movement. Furthermore, streets can be defined such as paths for cars or pavements for human beings. *OSM* [osm15] uses *WayPoints* as well. Therefore, they are also useful for the development of a new world namely *OSM*. They support waypoints in order to mark streets. *OSM* will be described particularly in section 6.2.

A further implementation of the world interface is *SoftWorld*. *SoftWorld* understands the world as a globe. Due to this implementation edges do not represent a border anymore as the nodes can go beyond and send their messages. In comparison to the *HardWorld* the sending and the movement beyond the edges happens faster. A useful algorithm that is also used by *OSM* converts Cartesian coordinates to Polar coordinates (ρ, θ).

Another possibility would be to implement a position-based *PlacementModel*. Regarding this model or a similar model it can be possible that nodes have to move back to their origin in case of low

[1] [Car09, S. 37]

battery in order to charge it. Sending and movement processes consume electricity. It may happen that a message cannot be sent while moving towards the origin.

In the mobility model radio masts can be defined as *MobilityHost*. They can be allocated all over the simulation area as they can have higher sending ranges. This enables to reach receivers by multi hop routing.

3.2 Movement Models

Beside the precision of the network implementation simulation a realistic simulation and evaluation of algorithms and applications in mobile Ad-Hoc-Networks depends on the nodes' movement. On the one hand "real" scenarios with mobile users and their movement data can be used for simulations. However this procedure is quite expensive as people's movement in a large area during a long period of time had to be acquired. As a consequence most simulations of mobile Ad-Hoc-Networks use problematic movement models. The validity of analysis mainly depends on the closeness to reality of the movement models as unrealistic connectivity relations can lead to false suggestions and evaluations. For the application in field trials they cannot be verified and might lead to a failure of the application. During the past years several movement models developed and were used by different scientific groups. The journal from Tracy Camp et al. [TCD02] offers a well overview. In principle the movement models can be divided into two clusters.

- ***Entity Mobility Models*** are models where hosts move independently
- ***Group Mobility Models*** have a node-depending-movement

Often used *Entity Mobility Models* are *Random Walk Mobility Model* and *Random Waypoint Mobilty Model*. The following subsections introduced them.

3.2.1 Random Walk Mobility Model

The *Random Walk Mobility Model* is based on a simple algorithm. With reference to the current position a node chooses his movement direction from the interval $[0, 2\pi]$ and the velocity from a predefined interval randomly. Once the direction is set the node moves either for a certain period of time or towards a certain distance. As a host moves towards an edge it bounces off and continues moving corresponding to the angle of incidence. In some variations of the *Random Walk Mobility Model* the simulation area is set as a sphere. Here the nodes can move beyond the edges. As a node escapes the simulation area it appears on the opposite edge.

3.2.2 Random Waypoint Mobility Model

The *Random Waypoint Mobility Model* [Dav01] is similar to the *Random Walk Mobility Model*. A mobile node randomly selects a target position on the simulation area and a speed from an arbitrary chosen interval. Thus the node moves on the basis of this data. For an arbitrary period of time the

node stays at the target position before it selects a new one. As a high number of nodes crosses the center of the simulation area the density of nodes in the center is generally higher than towards the edges [Bet01]. Therefore, the model requires a longer attack time. The longer the simulation runs the more spreading of nodes occurs. Neglecting the attack time leads to distorted results especially during short simulations. In [JYN] it is proofed that the average velocity decreases gradually if the velocity interval starts at zero. The reason behind is that lower velocity can be chosen and therefore it might take more time to reach the target position than the simulation lasts. By using the *Random Waypoint Mobility Model* one has to regard these issues in order to avoid distorted results. In [YN03] it is proven, that this problem appears in every "Random Mobility" models since target position and velocity are chosen separately. A default construction is introduced in order to solve this problem concerning these types of movement model.

3.2.3 Gauss-Markov Mobility Model

The *Gauss-Markov Mobility Model* avoids sudden changes of direction and breaks [LH03] through changing every host's direction and velocity at fixed points in time. The new position bases on the previous one as they vary by two normally distributed random values. This leads to gradual, natural movement. To assure that a host does not stay in a corner too long its movement is set up towards the center automatically as it approaches one. At this point it should be mentioned that some movements take action with particular probability. It is calculated by an algorithm like the Metropolis criterion [BS00]

3.2.4 Smooth Random Mobility Model

A similar approach pursues the *Smooth Random Mobility Model* [Bet01]. Even in this model the new chosen direction and speed is based on previous data. Additionally, the model works with target speeds. That means that there is a given target speed for each hop to mobile nodes. From the beginning of each hop a node tries to reach the given target speed. A suitable choice of target speeds enables to control the break and movement periods. Nodes do not change their direction having the full speed but rather decelerate and accelerate again afterwards. This is used for the simulation of vehicles.

3.2.5 Graph-Based Mobility Model

The *Graph-Based Mobility Model* [THB+02] is based on the realization that the random choice and the direct heading for target points within the common mobility models do not refer to the circum-

stances in reality. In the real world obstacles like houses exist and prohibit a direct overcome. People can move on paths and streets. Therefore, this model makes use of a graph. Its nodes define reliable targets and its edges define reliable paths between these targets. The mobile hosts start on a randomly chosen node on the graph and pick a target node coincidentally. They just move along the edges towards the target node and remain for an accidentally chosen period of time before they choose the next target node. The speed is chosen at random from a speed interval. Another imaginable routing of target nodes and speeds would be a schedule. Each device heads for the target points given by the schedule successively and remains for the given period of time. In this way for example the students' behavior on an university campus attending different lectures within one day can be simulated.

3.2.6 Manhattan Grid Mobility Model

The movement of mobile nodes in metropolis are simulated in the *Manhattan Grid Mobility Model* [Umt]. The nodes move on checkered streets separated by squared obstacles. The hosts are only able to move on the street. The houses cannot be negotiated. At every crossroad a node decides (with a certain probability) whether to turn left or right or to go on straightforward. Additionally the nodes can change their velocity in a certain interval.

3.2.7 Obstacle Mobility Model

The *Obstacle Mobility Model's* [AJS03] primary goal is to replace the different movements of other mobility models by more realistic movement patterns. Polygons (for example buildings) can serve as obstacles which can be placed on the simulation area. Between these buildings we can calculated paths on which the node moves. In the beginning the hosts are set up randomly on the paths. Each node chooses a target building and takes the shortest path towards it. After reaching the destination the host pauses for an arbitrary period of time before choosing a new target building.

Those paths are calculated by using the *Voronoi-Diagram*[2] which generates the corners of the buildings. Hereby the intuitive idea that paths run almost centered between two buildings is nearly realized. Furthermore, this model regards the obstruction of radio beams across buildings.

[2] cf.: `http://de.wikipedia.org/wiki/Voronoi-Diagramm`

3.2.8 Reference Point Group Mobility Model

In contrary to the previously documented mobility models the node movements of the *Reference Point Group Mobility Model* [XHC99] depend on each other. This model defines movements of a group on a predefined path. This path consists of a number of checkpoints which have to be reached within a specified time limit. By using this model a wide spread spectrum of scenarios can be executed. Every group of nodes has a center node. This node moves on the group's path. Every single node has its own reference point. It moves towards the group's path. After every step the nodes are placed nearby the reference point. Besides the groups' movement along paths a random behavior of the group members can be accomplished.

Among this selection of common used mobility models there exist a multitude of other models of different purposes. Nevertheless the development of further realistic movement models continues.

Within this thesis we implement the *Random Waypoint Mobility Model*. Future work deals with the implementations of the other models.

3.3 Routing Algorithms

Contrary to static networks routing algorithms in Ad-Hoc-networks have to handle steadily changing topologies what it aggravated by a missing central server. Routing protocols can be divided into two groups: Position-based [Mar01] and topology-based protocols [Eli]. The latter protocol can be subdivided into proactive and reactive protocols.

3.3.1 Topology - Based

Proactive topology - based protocols react to network-topology-changes and update the routing information permanently. The use of proactive protocols may be problematical as the steady updates of routing information at high node mobility wastes important communication bandwidth and energy. Examples for proactive protocols are:

- *Destination Sequenced Distance- Vector Routing (DSDV)* [PB94]
- *Wireless Routing Protocol (WRP)* [Mur96]

Reactive topology - based protocols do not try to update the latest routing information constantly but determine these information just in case of demand. Often the route is planned through network information input. This can lead to a high appearance of messages. Searching for a suitable route can take a longer period of time which can cause delays. One advantage is that reactive protocols just lead to a high appearance of messages in case that a node wants to communicate. This has high effects in case of low communication frequency. Common reactive protocols are:

- *Ad Hoc On-Demand Distance Vector Routing (AODV))* [Per03]
- *Temporally Ordered Routing Algorithm (TORA)* [Par97]
- *Dynamic Source Routing(DSR)* [Dav01]

3.3.2 Position - Based

The first heuristic position-based routing protocols emerged in the mid-eighties [Hid03] and assumed that the own position and the direct neighbor's position is known. The direct neighbor's position is updated generally by periodical broadcast whereas the target node's position is calculated by a self-organized location service [WKM04] within the Ad-Hoc network. Instead of calculating the route single nodes choose the route to transmit packages solely based on the target position and the position of the direct neighbor. Possible strategies of transmission are:

- *Most Forward within Radius(MFR)* [Kha10]
  Packages are transmitted to the neighbor which lays next to the target node within the transmitting range.
- *Nearest with Forward Progress(NFP)* [Dio10]
  Packages are transmitted to the closest neighbor towards the target node.

The following chapter describes the implemented interfaces and models.

Chapter 4

Implementation

This chapter presents the implementation of the *MobilityModel* for the p2p simulator *PeerfactSim.KOM*. First, we enumerate the different interfaces used for the mobility movement in section 4.1. Afterwards, we specify the implemented models in section 4.2. An evaluation of a simulation with broadcast messages is described in section 4.3.

Basically we extend the simulator and build a new network model. The whole *MobilityModel* is implemented in the folder:
`src\org\peerfact\impl\network\wireless`.

The files are sorted into a folder as follows:

The “api”- folder contains whole interfaces, the “impl” - folder contains implementations. Therefore, everyone who wants to test the model is able to do so, as one can copy the whole package into the network of the simulator. The simulator *PeerfactSim.KOM* is hierarchically structured as well.

4.1 The Interfaces

World:

The world is the main component and the running surface. It is given in [mm;mm]. The *MobilityFactory* creates a *MobilityHost* which is added to the *World* and saved in a list. During the simulation it is possible to merely create one *World*. Moreover it extends the *SimulationEventHandler* which initiates the hosts’ movement. As a *MOVE_EVENT* occurs in the simulation *EventQueue* the world handles it. Figure 4.1 shows the modules. In order to build a new instance of this interface the given functions have to be implemented.

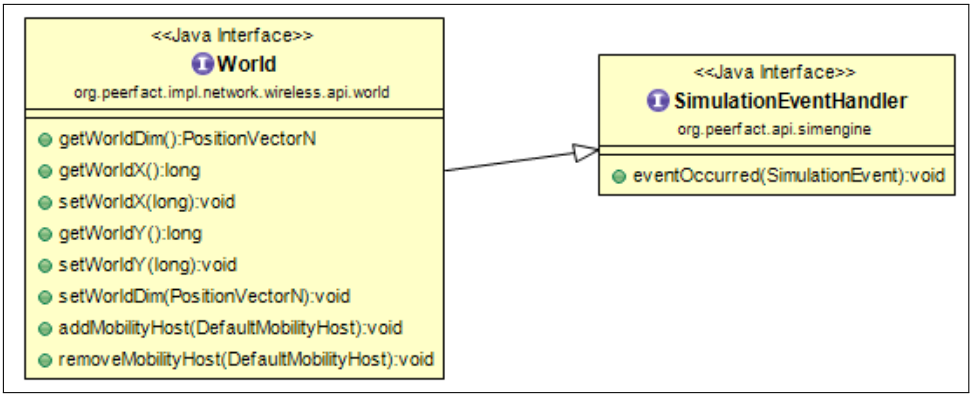


Figure 4.1: World interface

ObstacleModel:

The *ObstacleModel* creates a number of obstacles on the world which is determined in the *XML - file*. A host cannot move towards a position within a square of [100 mm;100 mm] around an obstacle.

MobilityHost:

The *MobilityHost* represents an element of a layer in the simulator *PeerfactSim.KOM*. Here it creates a *MobilityNetLayer* which implements the *NetLayer*. The battery state is implemented. The host's movement reduces the battery storage. The *MobilityHost* executes the movement. Figure 4.2 gives an overview of the *MobilityHost's* tasks referring to the movement of nodes. Besides the *MobilityHost* is extended by the *Component - class*.

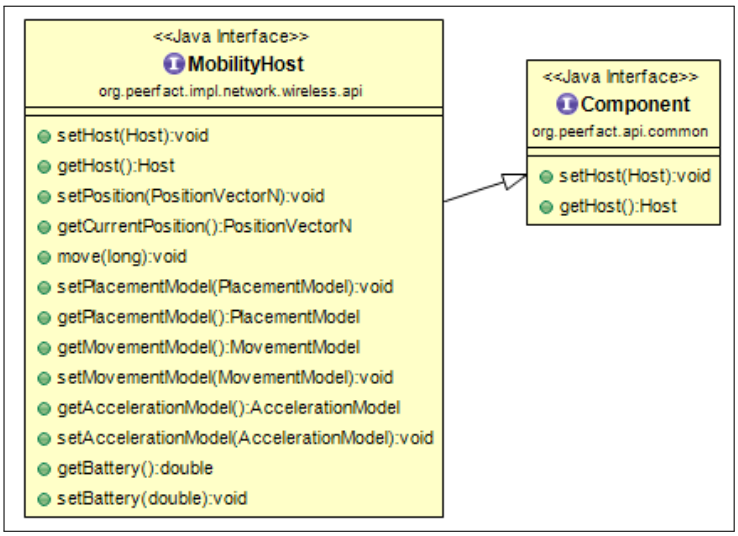


Figure 4.2: MobilityHost interface

PlacementModel:

The *PlacementModel* sets the starting point of the *MobilityHosts*. Here different algorithms can be used. Section 4.2 shows the currently implemented models.

PacketLossModel:

It computes the distance between sender and receiver and generates a packetloss through different models. Here the decision is made whether a packet should be send or rather be dropped. In the *XML - config* we can define the distance which allows the sending of a message.

The movement-strategy is implemented in two interfaces:

MovementModel:

In order to compute the new movement position, the *MobilityHosts* calls the *MovementModel* interface. Thereby it conveys its current position in terms of a *PositionVectorN*.

AccelerationModel:

Testing different types of nodes, we create the *AccelerationModel*. It calculates the motion speed. Those different types could represent humans, cars, or motorcycles. As a person starts moving her / his motion speed is constant. However a car needs time to accelerate until it reaches the target speed.

4.2 The Models

The main model is the *MobilityFactory*. It implements a *ComponentFactory* in the simulator. The Simulator is configured in the *XML - file*. The mobility environment sets up the *Netlayer*.

The *DefaultConfigurator* creates an instance of *MobilityFactory*. Then the other modules are configured according to the setup. The first is the *World* interface. We implement an instance called *HardWorld*. It represents a rectangle with fixed edges. No node can move beyond. The second model is the *MobilitySubnet* which registers the *NetLayer*. Hereafter the *ObstacleModel* has to be selected.

We implement two basis models. *NoObstacle* does not create any obstacles. *RandomObstacle* creates a defined number of obstacles of random positions. Listing 4.1 shows an example call with 20 obstacles.

```
Creating new Obstacle:  1 : X: 365483 Y: 120268
Creating new Obstacle:  2 : X: 318708 Y: 275218
Creating new Obstacle:  3 : X: 298772 Y: 166609
Creating new Obstacle:  4 : X: 192594 Y: 492420
Creating new Obstacle:  5 : X: 439591 Y: 470624
Creating new Obstacle:  6 : X: 137476 Y: 64448
Creating new Obstacle:  7 : X: 73300 Y: 11619
Creating new Obstacle:  8 : X: 273369 Y: 482243
Creating new Obstacle:  9 : X: 52245 Y: 312573
Creating new Obstacle: 10 : X: 205398 Y: 388156
Creating new Obstacle: 11 : X: 495361 Y: 243616
Creating new Obstacle: 12 : X: 373120 Y: 366576
Creating new Obstacle: 13 : X: 408648 Y: 419445
Creating new Obstacle: 14 : X: 263349 Y: 449667
Creating new Obstacle: 15 : X: 66969 Y: 41531
Creating new Obstacle: 16 : X: 489287 Y: 361178
Creating new Obstacle: 17 : X: 357515 Y: 71610
Creating new Obstacle: 18 : X: 231478 Y: 2242
Creating new Obstacle: 19 : X: 35749 Y: 174210
Creating new Obstacle: 20 : X: 169384 Y: 429678
```

Listing 4.1: Creating random obstacles

The next interface which has to be selected is the *PlacementModel*. We build a *RandomPlacement* to generate random start positions.

Additionally, we implement the *EvalPlacement* for the evaluation. It sets up the start position in the center of the world.

The hosts can have various speed and acceleration. We implement the *ConstantSpeed*. The velocity has to be set up in the *XML - file* as well.

In order to send a *MobilityNetMessages* two interfaces are necessary: *PacketLossModel* and *LatencyModel*. We create three implementations for *PacketLossModel*.

NoPacketLoss
: This model repudiates no messages.

LinearPacketLoss
: This one rejects a message as soon as the distance between two hosts is larger then the “stop”

distance defined in the *XML - file*.

SquaredPacketLoss

The last packetloss model allows to send a message up to the "stop" distance. The next 200 m an algorithm decides whether a message is sent or dropped. If the value of $\frac{1}{distance^2}$ is lower than a random number between zero and one the model repudiates the message. Any larger distance leads to a packet loss. These calculations find application in physics science, see figure 4.3[1].

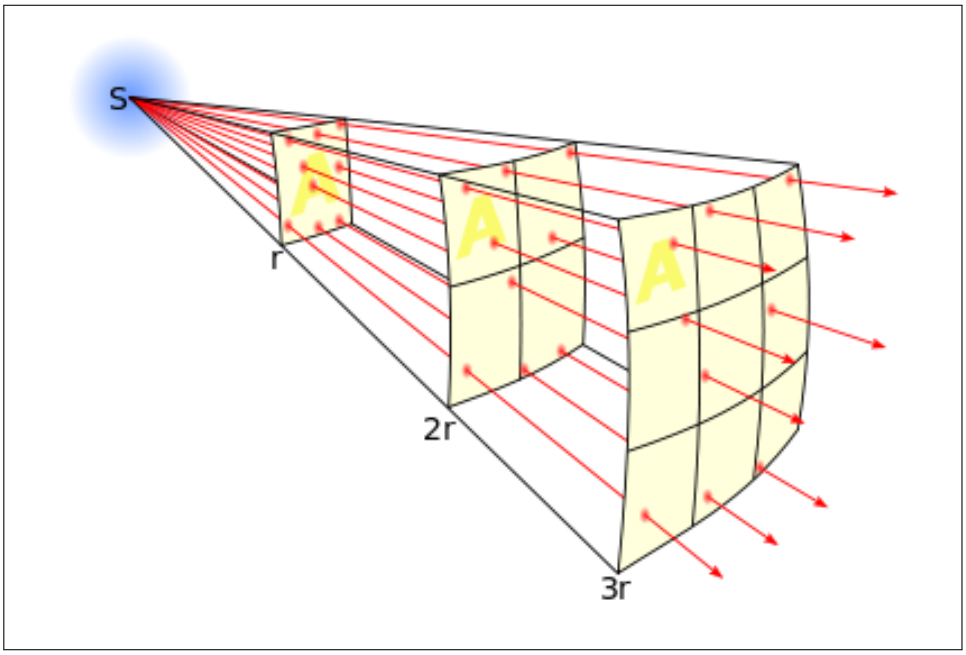
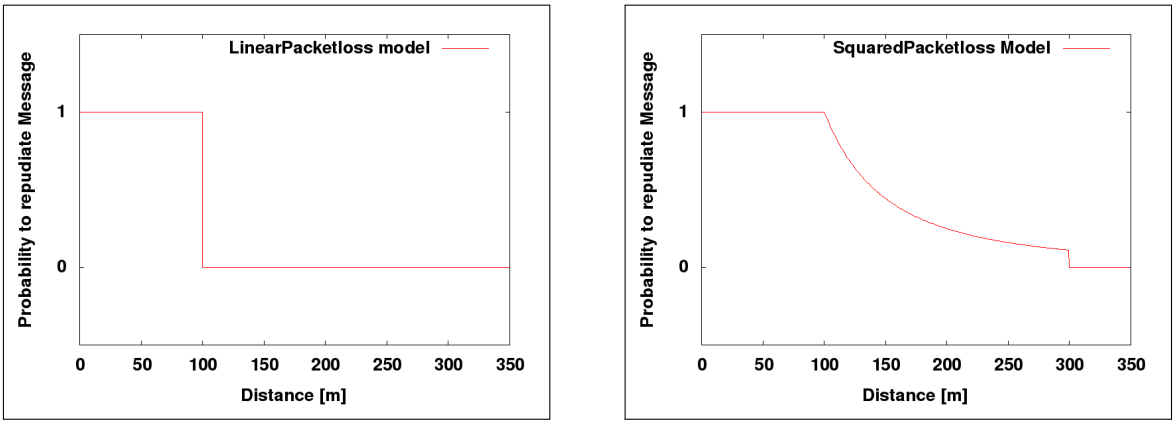


Figure 4.3: Inverse-square law

[1] cf.: http://en.wikipedia.org/wiki/Inverse-square_law

An example for *LinearPacketLoss* and *SquaredPacketLoss* model is given in figure 4.4.



(a) LinearPacketLoss

(b) SquaredPacketLoss

Figure 4.4: Different packetloss models with a “stop” distance at 100 m

The other sending models are the *NoLatency* and *MobilityLatencyModel* which implement the *LatencyModel*. The *MobilityLatencyModel* calculates the latency of the sending progress. The distance between sender and receiver divided by the light speed amounts the latency. This time value is added to the arrival-time.

The *MovementModel* is the last interface to be set up. Three implementations are given:

NoMovement
: This model returns the old position as the new one. Thus there is hosts’ movement.

RandomMovement
: By selecting this instance the host receives its new position within the world by random choice.

EvalMovement
: This model is prepared for the evaluation. It returns the next position 1000 mm away from the old one. All hosts with an odd *NetID* move towards the right side of the world. The others move leftwards. By reaching an edge they turn vice versa.

EvalMovement5
: Similar to the previous model its serves for evaluation. It was built for five hosts. One host moves around the center point. The other hosts move towards the edges. Two of them sideways, the other two in horizontal direction.

An example for host moving within the first 10 seconds with *EvalMovement* including two hosts is

given in listing 4.2 at the end of this section. Here the horizontal movement is obvious. Every second the host moves 1000 mm to the left or right side.

We run ten simulations with *RandomMovement*. It includes 10 hosts and a simulation time of 120 minutes. Table 4.1 presents the average number of movements of each and every host. The average number of movements of each host is 56,04 movements.

If all the interfaces listed in section 4.1 are chosen, the *DefaultHostBuilder* calls the function *createComponent* in the *MobilityFactory*. This creates a *DefaultMobilityHost* with the *PlacementModel*, *AccelerationModel* and *MovementModel* selected above. With the function call a host is transfered. It is assigned to the *DefaultMobilityHost*. Thereafter, the *MobilityHost* gets a position from its *PlacementModel* and is added to the world. Afterwards the function creates a new *MobilityNetID* and accordingly to this, a *MobilityNetLayer* will be prepared. The netlayer sets up the *PacketLossModel*. The *MobilityFactory* returns the *MobilityNetLayer* to the *DefaultHostBuilder*.
Further modules are essential for the implementation.
The class *PostionVector* is implemented in the folder:
`src\org\peerfact\impl\util\positioning.`
Functions are added to this file, such as:
The new *PositionVectorN* implements the *NetPosition*. Furthermore, we add a new constructor. It permits values of type *long*. Concerning the future work it already includes a third dimension e.g. to simulate mountains.

We have to add a new event to the class: *SimulationEvent*. We call this event *MOVE_EVENT*. It is handled by the *HardWorld*. It extends the class *SimulationEventHandler* as aforementioned.

Furthermore, we have to add a new *Reason* called *MOBILITYSEND* to the *Monitor* - interface. It is required to evaluate the simulation and draws the graphs “Net-Messages_in” and “Net-Messages_out”. The *DefaultMonitor* handles the *Reason*. To every sent message an action message is added in the *NetAnalyzer*. The *MobilitySubnet* extends the *AbstractSubnet*. It represents the *Subnet* interface.

```
Mobility Model by Tobias Korfmacher
Time | NetID | X-Pos | Y-Pos | Speed | Acc
0 | 0 | 250000 | 250000 | 1000.0 | 1.0
0 | 1 | 250000 | 250000 | 1000.0 | 1.0
1000000 | 0 | 249000 | 250000 | 1000.0 | 1.0
1000000 | 1 | 251000 | 250000 | 1000.0 | 1.0
2000000 | 0 | 248000 | 250000 | 1000.0 | 1.0
2000000 | 1 | 252000 | 250000 | 1000.0 | 1.0
3000000 | 0 | 247000 | 250000 | 1000.0 | 1.0
3000000 | 1 | 253000 | 250000 | 1000.0 | 1.0
```

```
4000000 | 0 | 246000 | 250000 | 1000.0 | 1.0
4000000 | 1 | 254000 | 250000 | 1000.0 | 1.0
5000000 | 0 | 245000 | 250000 | 1000.0 | 1.0
5000000 | 1 | 255000 | 250000 | 1000.0 | 1.0
6000000 | 0 | 244000 | 250000 | 1000.0 | 1.0
6000000 | 1 | 256000 | 250000 | 1000.0 | 1.0
7000000 | 0 | 243000 | 250000 | 1000.0 | 1.0
7000000 | 1 | 257000 | 250000 | 1000.0 | 1.0
8000000 | 0 | 242000 | 250000 | 1000.0 | 1.0
8000000 | 1 | 258000 | 250000 | 1000.0 | 1.0
9000000 | 0 | 241000 | 250000 | 1000.0 | 1.0
9000000 | 1 | 259000 | 250000 | 1000.0 | 1.0
10000000 | 0 | 240000 | 250000 | 1000.0 | 1.0
10000000 | 1 | 260000 | 250000 | 1000.0 | 1.0
```

Listing 4.2: Host-Positions with *EvalMovement*

Simulation	Number of movements	Average number of movements per host
1.	2796	55,92
2.	2805	56,1
3.	2813	56,26
4.	2825	56,5
5.	2792	55,84
6.	2828	56,56
7.	2789	55,78
8.	2796	55,92
9.	2811	56,22
10.	2765	55,3
Average number of movement of all simulations	2802	56,04

Table 4.1: Average movement of 50 hosts during 120 minutes

4.3 Broadcast Messages

For future work it is proposed to implement routing algorithms. For this reason we implemented a function which allows the host to send broadcast messages. If the receiver's *NetID* in the message equals the *broadcastID* the host sends the message to all nodes within its radius. The *broadcastID* is the value of *Integer.MAX_VALUE* viz 2147483647. It is the highest number that type *Integer* is able

to display and apply. The value of *Integer.MAX_VALUE* is similar to the broadcast address. In general the broadcast address is the highest *IP* in the network.

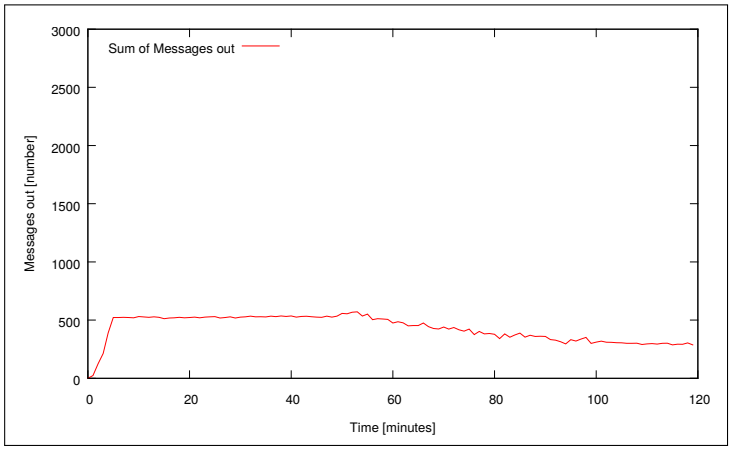
During the message process the host calls the function *updateNeighbors()*. It examines the distance between the sender host and every other node in the world. A potential host is added to the *Neighbor* list in the *MobilityNetLayer* in case the distance is verified by the *PacketLossModel*. If the distance between sender and potential host is too far the sender netlayer deletes the potential host from the neighbor list. Although the neighbor list has to contain the potential host. In case the list does not contain it the *MobilityNetLayer* moves on to the next potential host. The broadcast message sender repeats the procedure.

Figure 4.5 and figure 4.6 show the expected results of **NetMessages_out**(4.5) and **NetMessage_in**(4.6). The referring setup contains five hosts. The simulation time is configured at 120 min. All other modules are set up as in listing 2.1 except the *MovementModel*. For this simulation we use *EvalMovement5*.

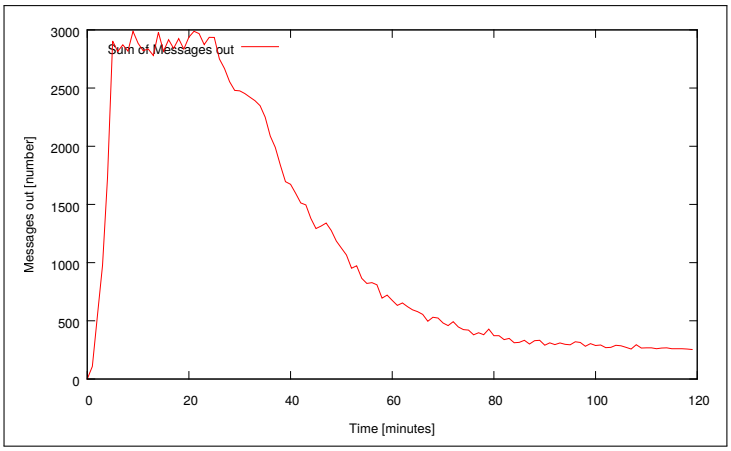
Broadcast message(figure 4.5b) provides five times more **NetMessages_out** compared to direct communication presented in figure 4.5a.

Accordingly the number of **NetMessages_in** is higher, too. Just to remind us:
If a host sends a ping and the message reaches the receiver it sends a pong back to the sender.

The subsequent chapter is about the scenario of evaluating and gives the results.

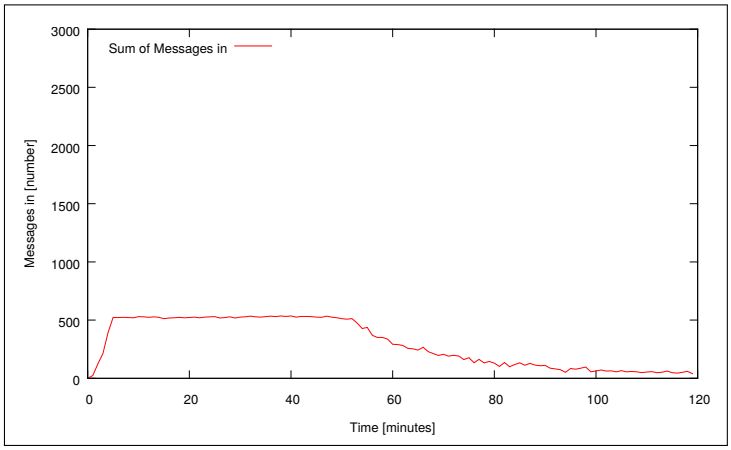


(a) Single send *NetMessages* out

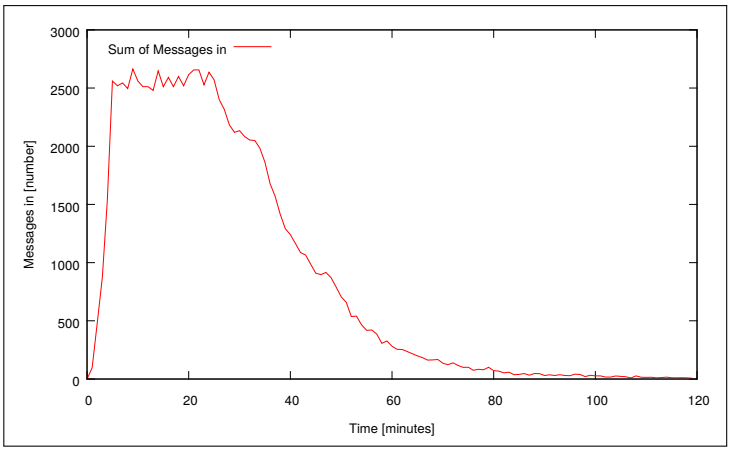


(b) Broadcast *NetMessages* out

Figure 4.5: Comparison between single and broadcast send messages out



(a) Single send *NetMessages* in



(b) Broadcast *NetMessages* in

Figure 4.6: Comparison between single and broadcast send messages in

Chapter 5

Evaluation

In order to evaluate the *MobilityModel* we build a special implementation. It is presented in section 5.1. In subsection 5.1.1 we briefly explain the *Star - Overlay* used for the evaluation. The results are introduced in section 5.2.

5.1 The Eval - Scenario

For testing the implementation we build the following scenario.

There exist two hosts. They are placed in the middle of the world. In the "eval-config" the position is $[250000, 250000]$. We remind ourselves that the world is created in mm. Both hosts move rectilinear at a speed of $35\,{}^{\text{mm}}/_{\text{s}}$. We run two simulations. The first simulation endures 120 minutes, the second runs 240 minutes. At a speed of $35\,{}^{\text{mm}}/_{\text{sec}}$ both hosts reach the edges of the world within the first two hours. As the second simulation continues two more hours the hosts, as they reach the edges, return to the middle of the world. Just as a reminder: The world size is defined by a rectangle of $[0, 0]$ and $[500000, 500000]$. The first host moves towards the left edge, the other vice versa. During the movement the nodes send messages between each other as long as the distance between them is below the limit configured in the *SquaredPacketLoss* model. Increasing the *pingInterval* in the *Star - Overlay* to $1\,{}^{\text{Ping - Message}}/_{\text{second}}$ we achieve significant results. In addition we modify the *StarNode*. It sends a message only if it is not the receiver. If both hosts are located in the middle of the world, their distance is 0 mm. Thus the ping messages reaches the receiver immediately. As the movement starts the distance increases. It follows that the probability to repudiate the message is rising, too.

Beyond the value of "stop" + 200000 defined in the *XML - config* the message is dropped. Those kinds of simulations display the behavior of mobility nodes using wireless communication.

5.1.1 Star - Overlay

The *Star - Overlay* has already been implemented by Mr. Tobias Amft (HHU [Hei15]) in the **Community Edition** of the simulator *PeerfactSim.Kom* [Pee15].

A node asks the *StarBootstrapManager* for the *CenterNode*. The *StarBootstrapManager* registers this node as the *CenterNode*, in case it has not yet registered one. The *StarBootstrapManager* conveys this *CenterNode* to every node which joins the network. Every new node saves the *CenterNode* and sends a ping message to the *CenterNode* periodically. In general the period takes one minute time. The periodic interval can be varied in the class *PingPongOperation*. The variable is *pingInterval*. If the *CenterNode* receives a ping message it sends back a pong message. Otherwise, the nodes keep sending ping messages.

Here, the difference compared to other overlays are revealed.

5.2 Results

The following figures display the results of the “Eval-scenario” described in section 5.1.

5.2.1 Simulation Time: 120 min

In figure 5.1 two hosts are placed in the middle of the world. The distance between them is 0 mm. At minute three they start sending messages, each one ping message per second. Plot 5.1a shows the sum of **NetMessages_out**. The host with the *NetID* “0” sends a ping message. During the sending process the distance between them stays lower than calculated in the *SquaredPacketLoss*. Therefore, the receiver sends back a pong message. The hosts’ motion speed is set at $35\,{}^{\text{mm}}/_{\text{sec}}$. The distance towards the edges is 250000 mm. It takes the host

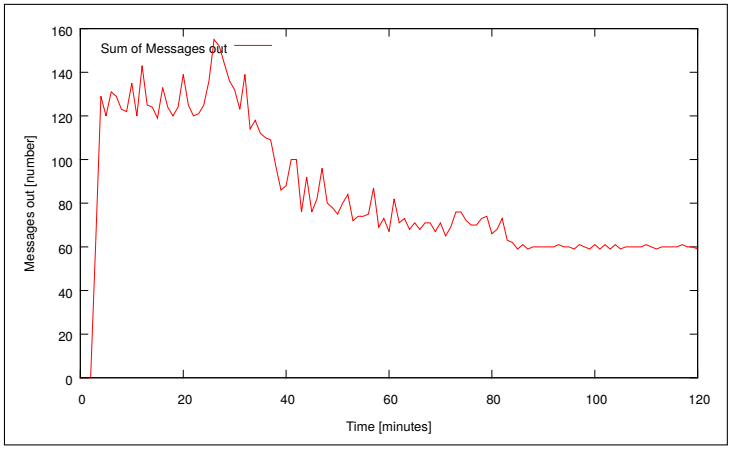
$$\frac{250000\,\text{mm}}{35\,{}^{\text{mm}}/_{\text{sec}}} = 7142,86sec \Rightarrow \frac{7142,86\,\text{sec}}{60} \approx 119,3min$$

to reach the edges.

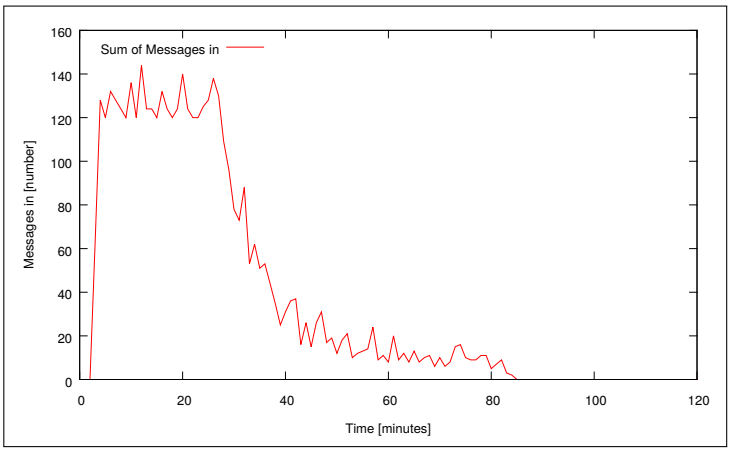
The packet loss distance is assigned to 100000 mm. The packet loss calculation starts at 50000 mm movement as the hosts move in opposite directions.

Thus $\frac{50000\,\text{mm}}{35\,{}^{\text{mm}}/_{\text{sec}}} = 1428,57sec$ the set distance exceeded. Hence the packet loss model *SquaredPacketLoss* generates packet loss using the following formula:

$$\frac{1}{(distance)^2} < randomvalue \in [0,1] \Rightarrow \text{packet loss.}$$



(a) NetMessage Out



(b) NetMessage In

Figure 5.1: Duration time 120 min

This formula is valid up to a distance of 350000 mm between the hosts. They go beyond after 5001 seconds of simulated time.

Calculation:

$$\frac{\frac{1}{2} \cdot 350000\,\text{mm}}{35\,\text{mm/sec}} = \frac{175000\,\text{mm}}{35\,\text{mm/sec}} = 5000\,\text{sec} \Rightarrow \frac{5000}{60}\,\text{sec} \approx 83\,\text{min}$$

After this time, the hosts still send a ping message every second but do not receive a pong message, see figure 5.1b. It displays the **sum of Messages_in**.

The sending method is realized by the following algorithm.

Note: In order to raise the readability of this thesis we name the sender - *NetLayer* ‘sender’ and the receiver - *NetLayer* ‘receiver’.

1. The sender receives the sending call from the overlay-layer. It includes the message, the receiver

- *NetID* and the *NetProtocol*.

2. If the protocol is supported (currently it is *IPv4*), the sender creates a *MobilityNetMessage* including the sender - *NetID* and the contents mentioned above.

3. The sender checks its network-state. In the case of being online, it calculates the distance towards the receiver.

   a) In case the hosts do not move, the sender calls the function *getDistance(DefaultMobilityHost receiver)*. The function returns the euclidean distance between sender and receiver:
   $distance = \sqrt{(|sender_X - receiver_X|)^2 + (|sender_Y - receiver_Y|)^2}$

   b) If the sender or the receiver moves, they call the function *getPositionAt(long time)*. It returns the position at the time of "time". The new position is computed with following trigonometric operations:
   $\varphi = \arctan(\frac{newPosition_Y - oldPosition_Y}{newPosition_X - oldPosition_X})$
   $CurrentPosition_X = \cos\varphi \cdot \frac{|CurrentTime - MovementStartTime| \cdot speed}{1000000} + oldPosition_X;$
   $CurrentPosition_Y = \sin\varphi \cdot \frac{|CurrentTime - MovementStartTime| \cdot speed}{1000000} + oldPosition_Y;$
   The events in the simulator are schedule in nanosec. Therefore it is necessary to divide by 1000000 to get the correct position.

4. The resulting distance is divided by the velocity of light, which is $299792458000\,{}^{mm}/_{sec}$.

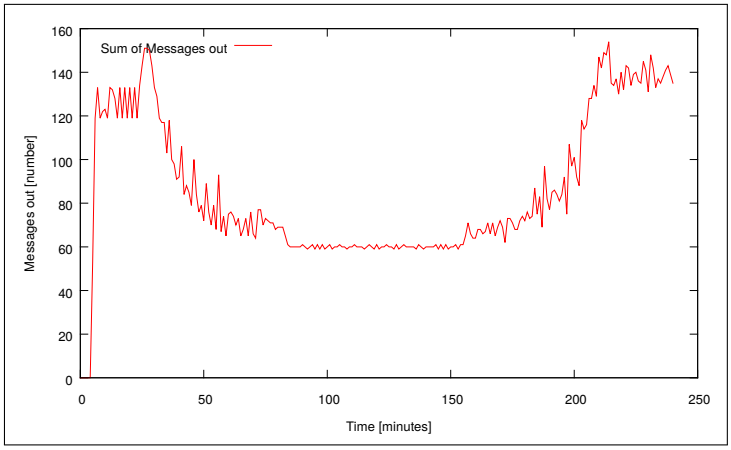
5. Now the sender calls the function *repudiateMsg* from his packetloss model to receive the decision whether the message should be sent or dropped.

5.2.2 Simulation Time: 240 min

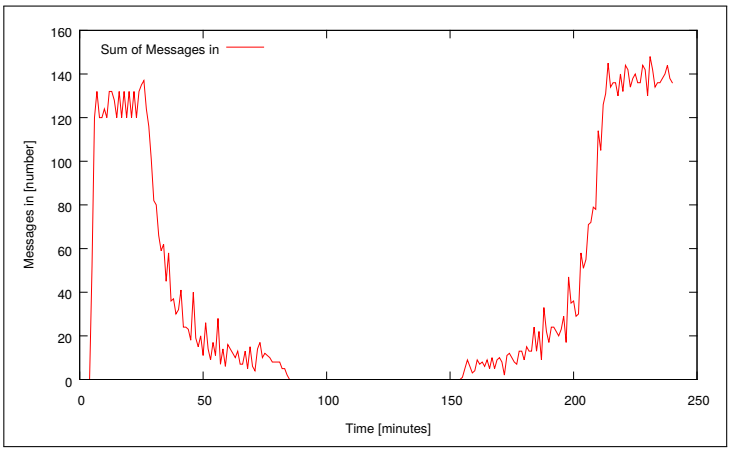
The sum of in and outgoing message during the 4 hours - simulation is shown in figure 5.2.

The graphics demonstrate the hosts movement as they first move towards the edges and then return to the center. This behavior is express by the sum of sent messages.

After the simulation is completed the *MobilityAnalyzer* generates a file called "Host-Movements.txt". The file includes the number of movements. See listing 5.1.



(a) NetMessage_Out



(b) NetMessage_In

Figure 5.2: Duration time 240 min

```
Mobility Model by Tobias Korfmacher
Host: 0 had 505 movements.
Host: 1 had 505 movements.
```

Listing 5.1: Host-Movements.txt

The next chapter summarizes the bachelor thesis and gives several ideas for future work.

Chapter 6

Conclusion

After summarizing the results in section 6.1, some ideas for possible future work with the simulator *PeerfactSim.KOM* and this *MobilityModel* follow in section 6.2.

6.1 Summary

The main aim of this bachelor thesis was to implement a mobility model in a peer-to-peer simulator successfully. After having done the work, it becomes obvious that choosing the *PeerfactSim.KOM* simulator was the right decision. In this simulator we built up a new *NetLayer*. We began the thesis describing the mobile movement model of Carsten Snider and reasoning the decision why it was better to create a new version instead of extending the old one. The next chapter contains the evaluation of the network layer and settings that have been built only for the purpose of this thesis. The results confirm that the *MobilityModel* was implemented correctly and worked as expected. They clearly express that the step of creating a new environment was right as it supports an easy adding and testing of new models. Furthermore, it is able to evaluate routing protocols or to analyze further topics in connection with moving hosts.

6.2 Future Work

Now, we discuss the ideas for future work with this *MobilityModel* and the peer-to-peer simulator *PeerfactSim.KOM*.

6.2.1 Linklayer

One possibility for the future would be the creation of a new layer representing a *LinkLayer*. This step enables to choose a host via mac address instead of its *NetID*. Hence routing in mobile network would be possible. Corresponding models are described in section 3.3. A data storage for messages that did not reach the receiver would be imaginable, too.

6.2.2 OpenStreetMap

OpenStreetMap [osm15] is a licence-free project with the aim to create a free world map. It either presents the data unwrought or in calculated maps and can be used on the *Java ME platform* [Ora15]. *OSM* allocates the so called *OSM*-ways, -nodes and –tags in a *XML-file*. For the future of *Peerfact-Sim.KOM* the creation of waypoints would be possible. During the simulation the hosts can choose their next destination in correspondence with these waypoints. Thereof, a movement along a predefined route is conceivable.

6.2.3 Visualization

Another future application could be the visualization of movements by using the created "Host-Positions.txt" that permanently lists the position of every hosts in one file. This scenario was shown in section 4.2. Maybe it would be possible to create a database with all positions or save the position of each host in a separate file. Indeed, this would arise the question how to show the positions in parallel to the simulation. The faster and therefore more effective method would be a visualization subsequent to the simulation. It could be handled like the visualization of the chord overlay in a new *GUI* as presented in figure 6.1.

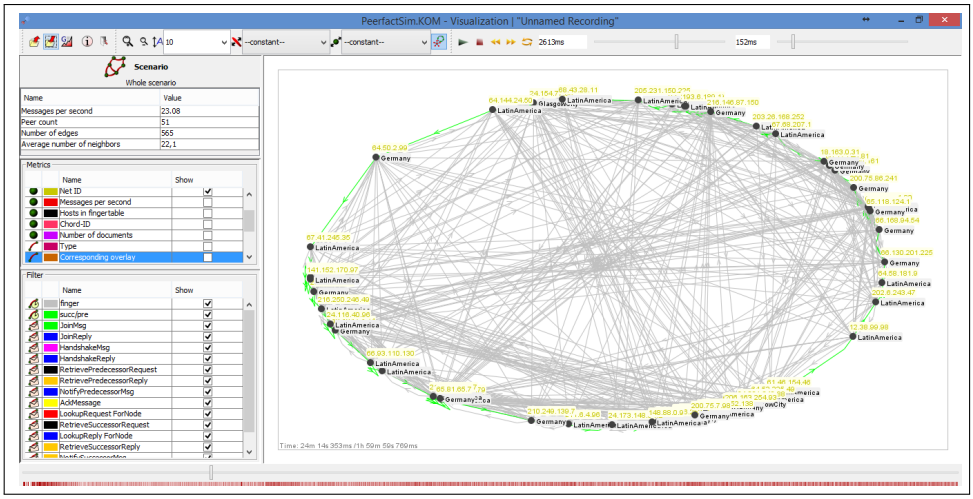


Figure 6.1: Visualization of the chord overlay with 51 hosts

Finally, the *MobilityModel* provides a better basis for simulating and testing peer-to-peer overlays on moving hosts with a multi-hop network in the *PeerfactSim.KOM* simulator than the Snider model. Applying the ideas for future work described beforehand, it would be possible to investigate the behavior of different peer-to-peer overlays in association with a mobility network and to compare the liability of each overlay.

Bibliography

[Ka15] KALMAN GRAFFI UND MATTHIAS FELDOTTO: *PeerfactSim.KOM - The Peer-to-Peer System Simulator - Community Edition Getting Started*. `http://peerfact.com/wp-content/uploads/2015/04/PeerfactSim.KOM__Getting-Started.pdf`, 2015

[AJS03] AMIT JARDOSH, Kevin C. A. Elizabeth M. Belding-Royer ; SURI, Subhash: Towards Realistic Mobility Models for Mobile Ad Hoc Networks. In: *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, 2003 (MobiCom '03)

[Bet01] BETTSTETTER, Christian: Smooth is Better than Sharp: A Random Mobility Model for Simulation of Wireless Networks, 2001

[BS00] BEICHL, Isabel ; SULLIVAN, Francis: The Metropolis Algorithm, 2000

[Car09] CARSTEN SNIDER: *Mobility Aware Peer-to-Peer Networking*. 2009

[Dav01] DAVID B. JOHNSON, DAVID A. MALTZ AND JOSH BROCH: Ad Hoc Networking. 2001, Kapitel DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks

[Dio10] DIONYSIOS EFSTATHIOU, ANDREAS KOUTSOPOULOS AND SOTIRIS NIKOLETSEAS: Analysis and Simulation for Parameterizing the Energy-latency Trade-off for Routing in Sensor Networks. In: *Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, 2010 (MSWIM '10)

[Eli] ELIZABETH M. ROYER AND CHAI-KEONG TOH: A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks .

[Hei15] HEINRICH-HEINE-UNIVERSITÄT DÜSSELDORF: *Universität Düsseldorf: Startseite*. `http://www.uni-duesseldorf.de/`, 2015

[Hid03] HIDEAKI TAKAGI AND LEONARD KLEINROCK: *Optimal Transmission Ranges for.Randomly Distributed Packet Radio Terminals*. 2003

[JYN] JUNGKEUN YOON, Mingyan L. ; NOBLE, Brian: Random Waypoint Considered Harmful. In: *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Franciso, CA, USA, March 30 - April 3, 2003*

[Kha10] KHALED AHMED ABOODOMER: Analytical Study of MFR Routing Algorithm for Mobile Ad Hoc Networks. In: *J. King Saud Univ. Comput. Inf. Sci.* 22 (2010), Januar

[LH03] LIANG, Ben ; HAAS, Zygmunt J.: Predictive Distance-based Mobility Management for Multidimensional PCS Networks. In: *IEEE/ACM Trans. Netw.* 11 (2003), Oktober, Nr. 5

[Mar01] MARTIN MAUVE AND JÖRG WIDMER AND HANNES HARTENSTEIN: A Survey on Position-Based Routing in Mobile Ad-Hoc Networks. In: *IEEE Network Magazine* 15 (2001), November, Nr. 6, S. 30–39

[Mur96] MURTHY, SHREE AND GARCIA-LUNA-ACEVES, J. J.: An Efficient Routing Protocol for Wireless Networks. In: *Mob. Netw. Appl.* 1 (1996), Oktober, Nr. 2

[Ora15] ORACLE CORPORATION: *JAVA PLATFORM, MICRO EDITION (JAVA ME)*. `http://www.oracle.com/technetwork/java/embedded/javame/index.html`, 2015

[osm15] *OpenStreetMap - Deutschland*. `http://www.openstreetmap.de/`, 2015

[Par97] PARK, VINCENT D. AND CORSON, M. SCOTT: A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In: *Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, 1997 (INFOCOM '97)

[PB94] PERKINS, Charles E. ; BHAGWAT, Pravin: Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In: *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, 1994 (SIGCOMM '94)

[Pee15] PEERFACTSIM.KOM: *PeerfactSIM.KOM*. `https://sites.google.com/site/peerfactsimkom/`, 2015

[Per03] PERKINS, C. AND BELDING-ROYER, E. AND DAS, S.: *Ad Hoc On-Demand Distance Vector (AODV) Routing*. 2003

[TCD02] TRACY CAMP, Jeff B. ; DAVIES, Vanessa: A Survey of Mobility Models for Ad Hoc Network Research. In: *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications* 2 (2002), Nr. 5

[Tec15] TECHNISCHE UNIVERSITÄT DARMSTADT: *Home – Technische Universität Darmstadt*. `http://www.tu-darmstadt.de/`, 2015

[THB+02] TIAN, Jing ; HÄHNER, Jörg ; BECKER, Christian ; STEPANOV, Illya ; ROTHERMEL, Kurt: Graph-Based Mobility Model for Mobile Ad Hoc Network Simulation. In: *Proceedings 35th Annual Simulation Symposium (ANSS-35 2002), San Diego, California, USA, 14-18 April 2002*, 2002, S. 337–344

[Tho15] THOMAS WILLIAMS, COLIN KELLEY, RUSSELL LANG, DAVE KOTZ, JOHN CAMPBELL, GERSHON ELBER, ALEXANDER WOO ET AL.: *gnuplot homepage*. `http://www.gnuplot.info/`, 2015

[Umt] UMTS: Selection procedures for the choice of radio transmission technologies of the UMTS (UMTS 30.03 version 3.1.0) / UMTS. – Forschungsbericht

[Uni15] UNIVERSITÄT PADERBORN: *Universität Paderborn*. `http://www.uni-paderborn.de/`, 2015

[WKM04] WOLFGANG KIESS, Jörg W. Holger Füßler F. Holger Füßler ; MAUVE, Martin: Hierarchical Location Service for Mobile Ad-hoc Networks. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 8 (2004), Oktober, Nr. 4

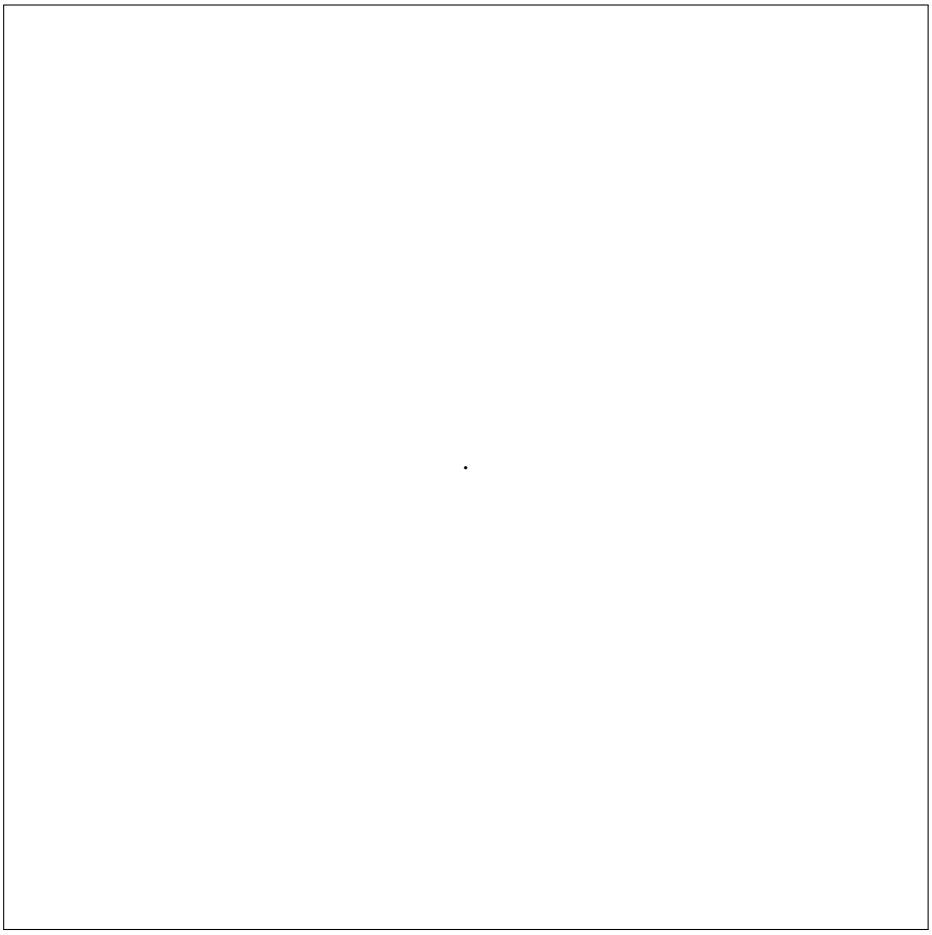
[XHC99] XIAOYAN HONG, Guangyu P. Mario Gerla G. Mario Gerla ; CHIANG, Ching-Chuan: A Group Mobility Model for Ad Hoc Wireless Networks. In: *Proceedings of the 2Nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 1999 (MSWiM '99)

[YN03] YOON, Liu M. Jungkeun ; NOBLE, Brian: Sound Mobility Models. In: *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, 2003 (MobiCom '03)

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 18.Mai 2015 Tobias Korfmacher



This CD contains:

- A *pdf* Version of this bachelor thesis
- All LaTeX and graphic files that have been used, as well as the corresponding scripts
- The source code of the software that was created during the bachelor thesis
- The measurement data that was created during the evaluation
- The referenced websites and papers