HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Location-oriented routing in opportunistic networks

Bachelor Thesis

by

Timo Lux

born in
Duisburg

submitted to

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

August 2017

Supervisor:
Raphael Bialon, M. Sc.

Abstract

This thesis focuses on location-oriented routing in opportunistic networks extending the popular probabilistic routing protocol PRoPHET. The goal is to provide a possibility to route packets to geographical locations instead of to other hosts. This way other information, like regular packets routed by PRoPHET, can be wrapped up into a location-oriented packet in order to send it to a much frequented place, e.g. train stations, from where the chances to find a route to the original receiver is much higher. The designed extension is implemented in the network simulator PeerfactSim.KOM and used in the simulation of several scenarios. The results of the scenarios are evaluated regarding the hop count, arrival ratio and time until arrival of packets.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Motivation

In today's world, the internet is a vital part of everyday life. It replaces activities which were traditionally done differently, like communicating with other people. Instant messaging services, e.g. WhatsApp and Telegram, have replaced other means of communication like SMS and shopping can easily be done online with the products arriving as fast as the same day in some places. Corporations are able to manage huge infrastructures and amounts of data much easier with than without the internet. While the whole connectivity has its advantages, it becomes more and more evident, that there are people and governments who want to control as much of the internet as possible. Governments might block communication within a country to make it harder for people to organize.

In such a case alternative networks like opportunistic networks, might be the key to keep communication up. In opportunistic networks no centralized infrastructure exists, in which communication could be blocked easily. Instead, devices capable of wireless communication with other devices, for example via WLAN or bluetooth, establish connections to each other as soon as they are within each others communication range. In order to be able to find routes for packets to the proper receivers, routing algorithms have to be developed. One large group of routing protocols for this purpose are probabilistic routing protocols, which depend on probabilities which are calculated opportunistically when two hosts meet. One of those is the Probabilistic Routing Protocol using History of Encounters and Transitivity, also called by its acronym PRoPHET ([LDDG12]). PRoPHET, as its name suggests, uses a history of encounters and transitivity to calculate probabilities to meet certain hosts. As [GDLD11] shows, it performs quite well.

However, depending on the situation it can require a long time until packets reach an area which enough other hosts frequent, so the probability is higher to find a host which will move into the direction of the packet receiver. It might be worthwhile to route a packet to a frequently visited location first, instead of directly routing it to another host. For example, one could wrap a usual PRoPHET packet into another packet, which is targeted at such a location, e.g. train stations, city centers or tourist attractions. As soon as this packet reaches its destination, it unwraps and continues as a packet to the receiving host.

In this thesis I focus on designing a location-oriented extension for PRoPHET, which just as the original uses probabilities to reach a certain destination. I implement this design in PeerfactSim.KOM ([Gra11]), a network simulator focusing on peer-to-peer and ad hoc networks.

1.2 Related Work

Globase.KOM ([KLS$^+$07]) is a peer-to-peer overlay which aims at providing an efficient way of location-based searching in large-scale peer-to-peer networks. It structures peers in a tree structure in which super peers have control over larger areas and are connected to other super peers. The overlay provides operations to search for all peers in a defined location, for a peer in a specific location or for the geographically closest peer. Those peers can respond with any information it wishes to, like the services it offers or an object it represents (e.g. restaurants, universities, etc.).

1.3 Outline

In Chapter 2 the fundamentals of PRoPHET and the location-oriented extension are explained. The implementation is described in Chapter 3 and the simulation and its result are shown in Chapter 4. Finally the thesis is summarized and concluded in Chapter 5 with some examples of how the work of this thesis can be followed up on.

Chapter 2

Fundamentals

In this chapter the fundamentals of this thesis will be explained. In Section 2.1 the used simulator will be introduced. Section 2.2 shortly explains the characteristics of opportunistic networks, which is the type of network simulated later. In Section 2.3 I will describe the central parts of PRoPHET, the routing protocol onto which the location-oriented parts will be built as described in Section 2.4.

2.1 PeerfactSim.KOM

PeerfactSim.KOM (hereafter called *Peerfact*) is a network simulator written in Java. It was originally written to simulate peer-to-peer networks, but has been enhanced to also be able to simulate ad hoc networks with or without opportunistic encounters. Peerfact was first developed at the TU Darmstadt and later extended by the University of Paderborn and the Heinrich-Heine-University in Düsseldorf. It is a good fit for this thesis, because it is structured modularly and thus it is easy to add new routing algorithms beside the existing ones. Peerfact uses XML configuration files to define simulation scenarios, which will be further elaborated on in Chapter 4.

2.2 Opportunistic Networks

Opportunistic networks are a type of network, in which nodes only sporadically connect to each other because of their mobility property. Because connections are intermittent and huge delays often occur, oppotunistic networks are also a form of a delay-tolerant network. These encounters can generally not be foreseen and so each node has to opportunistically decide which of its contemporary neighbors are best suited to forward the packets it is holding. Traditional routing protocols rely on a network topology which rarely changes and generally expect end-to-end paths to exist between any two nodes. In opportunistic networks they are not useful, because end-to-end paths generally do not exist and the topology changes frequently, which is why newer routing protocols have been developed in recent years to tackle this challenge. One of these protocols is the Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET). Its general workings are explained in Section 2.3. As for routing in opportunistic networks in general, refer to [PPC06].

2.3 PRoPHET

PRoPHET is a routing protocol used in delay-tolerant networks. It does not rely on a fixed topology of the network, but instead uses probabilities (called *delivery predictability* or simply *predictability*) to determine whether to send a packet to a connected host. These delivery predictabilities are based on the history of encounters of this host, but also transitively on the encounters the connected host had.

Throughout this section I will explain everything from the perspective of a host called *A*. The host it is currently communicating with will be called *B* and when refering to a third host, which is not close to the other two, it will be called *C*.

2.3.1 Encountering Hosts

When two hosts come into vicinity of each other, so that a connection can be established, the first part of data exchange begins. If it is the first time A meets B, a defined predictability,

$P_{\text{encounter}_{\text{first}}}$, will be assigned to the encounter entry for B. If an entry, and thus a predictability, for B already exists, the new predictability is calculated according to Equation (2.1).

$$P(A,B) = P(A,B)_{\text{old}} + (1 - \delta - P(A,B)_{\text{old}}) \cdot P_{\text{encounter}}(\text{intvl}) \tag{2.1}$$

Here, δ is a very small number, e.g. 0.01, which sets an upper limit to the predictability value. This is only needed, because a value of 1.0 is reserved for $P(X,X)$ for any host X. $P_{\text{encounter}}$ is a function defined and explained below.

$$P_{\text{encounter}}(\text{intvl}) = \begin{cases} P_{\text{encounter}_{\text{max}}} \cdot (\text{intvl}/I_{\text{typ}}) & \text{for } 0 \leq \text{intvl} \leq I_{\text{typ}} \\ P_{\text{encounter}_{\text{max}}} & \text{for intvl} > I_{\text{typ}} \end{cases}$$

Hosts A and B might technically encounter each other multiple times, when we would not actually count it as multiple encounters. For example, the connection could be interrupted and reestablished several times in a short period of time. To prevent this from causing unrepresentative predictability values, $P_{\text{encounter}}$ is applied. This function uses I_{typ}, which is the pre-defined amount of time two encounters should be apart to be counted as two separate encounters. When A encounters B a second time before this amount of time has passed, the function takes care of applying only a fraction of the change to the predictability value, so that multiple encounters within this time should roughly have the same effect on the predictability as have two encounters I_{typ} apart. $P_{\text{encounter}_{\text{max}}}$ determines the rate at which the predictability value increases for each distinct encounter.

The delivery predictabilities cannot always just increase. If host A does not meet host B for a longer time, the probability to meet it has to decrease somehow. For this purpose the predictability values will be aged regularly according to Equation (2.2).

$$P(A,B) = P(A,B)_{\text{old}} \cdot \gamma^{K} \tag{2.2}$$

This equation uses an aging constant γ and the number of time units passed since the last aging update, K. This function is called everytime another host is met.

Finally, the transitivity has to be taken care of. Host A receives all of B's encounters and recalculates its predictability values for each host C, B has encountered. For this, equation Equation (2.3) is used.

$$P(A,C) = \max(P(A,C)_{\text{old}}, P(A,B) \cdot P(B,C) \cdot \beta) \tag{2.3}$$

If $P(A,C)_{\text{old}}$ does not exist, 0.0 is assumed. β is a scaling constant which affects the impact of transitivity on the predictability value.

After all this is done, the exchange of packets can begin.

2.3.2 Routing

Everytime an encounter takes place and the initial exchange of encounter data has been finished, the two hosts have to determine which of their packets should be sent to the other host. As can be seen in Listing 2.1, the Time To Live (TTL) is checked first. If the packet has reached the maximum number of hops, it will be dropped. Next, the predictability of all neighbors for every packet is queried and compared to the own predictability. If the neighbor's predictability is higher than the own, the packet is sent, unless the neighbor is already in the hop list and thus already has this packet.

```
if not TTL reached:
    for each connected neighbor:
        if neighbor did not see the packet yet:
            if neighbor predictability > own predictability:
                sendPacket()
else:
    dropPacket()
```

Listing 2.1: Pseudocode of the PRoPHET Routing Algorithm

2.4 Location-oriented PRoPHET

In order to add location-oriented routing to PRoPHET, some changes have to be made to the protocol, which will be explained in this section.

2.4.1 Way Points

Each host has to keep track of the locations it has visited. To do this, it saves its position along with a predictability value in a given interval $\mathrm{WP}_{\mathrm{intvl}}$. Everytime a position is about to be logged, it is first checked whether another way point close enough to the new position has already been stored. If there is none, a new way point entry is made with an initial predictability value of $P_{\mathrm{WP}_{\mathrm{first}}}$. A way point is close enough to another one, if the distance between their x-values is not greater than $\mathrm{WP}_{\mathrm{MinXDistance}}$ and the distance of their y-values is not greater than $\mathrm{WP}_{\mathrm{MinYDistance}}$.

Analogous to Equation (2.1), an equation to increase the predictability of a way point is needed.

$$P(A,W) = P(A,W)_{\mathrm{old}} + (1 - P(A,W)_{\mathrm{old}} \cdot P_{\mathrm{WP}_{\mathrm{max}}} \tag{2.4}$$

Equation (2.4) shows how the predictability host A has for a way point W is calculated. In contrast to the original equation, no δ is needed here, because a value of 1.0 is not reserved for anything. Also, we do not need a function similar to $P_{\mathrm{encounter}}$, because way points are already only saved and updated in a fixed interval $\mathrm{WP}_{\mathrm{intvl}}$, which should represent a time for which it is reasonable to increase the predictability when the location has not been changed.

Before a way point is saved, all stored way points have to be aged. The function used for this is essentially the same as the aging function described in Equation (2.2) and can be seen in Equation (2.5). It also uses an aging constant γ_{WP} and the amount of time units passed K.

$$P(A,W) = P(A,W)_{\mathrm{old}} \cdot \gamma_{\mathrm{WP}}^{K} \tag{2.5}$$

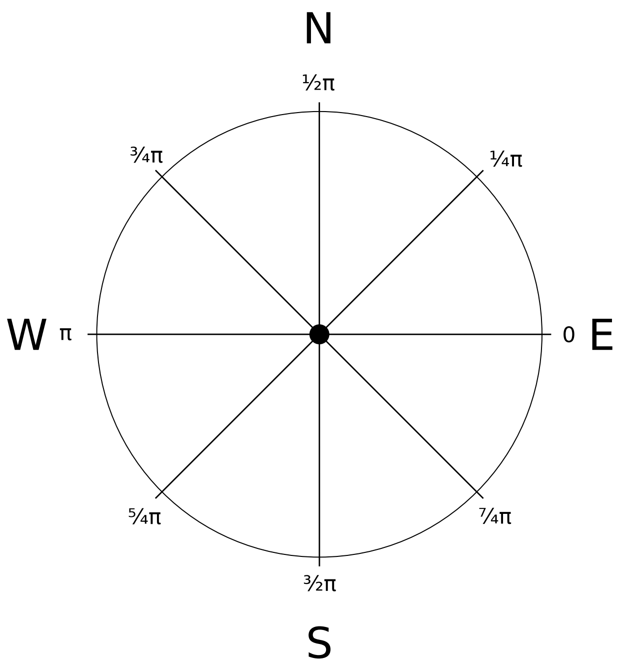

Figure 2.1: Angles mapped to the directions

2.4.2 Direction Vectors

Exchanging way points between hosts would raise privacy concerns, because every host would know exactly which locations any encountered host visits. By looking at the predictability value, one would also know if a location is visited regularly or infrequently. For this reason way points are converted to direction vectors before sending them to the connected neighbors. This way only the directions a host supports for routing can be seen, but not the distance, hence not the exact locations visited.

Direction vectors consist of a point of origin, which is simply the location the encounter took place at, an angle representing the direction and again, a predictability value. The angle is a value in the half-open interval $[0, 2\pi)$. As shown in Figure 2.1, 0 corresponds to east, $\frac{1}{2}\pi$ corresponds to north, π corresponds to west and $\frac{3}{2}\pi$ corresponds to south. To calculate this angle, the function `atan2` ([Gli11], [ata]), a variation of arctangent and present in many programming languages, can be used. It calculates the angle between the (shifted) x-axis and a given vector, with a positive result for the upper half-plane (above and on our x-axis) and a negative result for the lower half-plane (below our x-axis). Since this is not exactly how the angle is needed, 2π has to be added to a negative result as shown in Equation (2.6).

$$\alpha = \begin{cases} \arctan2(\Delta y, \Delta x) & \text{if } \arctan2(\Delta y, \Delta x) \geq 0 \\ \arctan2(\Delta y, \Delta x) + 2\pi & \text{else} \end{cases} \tag{2.6}$$

When A receives a direction vector from B, then A multiplies the vector's predictability value by the predictability it has for B as can be seen in Equation (2.7).

$$P(A,D) = P(A,B) \cdot P_D \tag{2.7}$$

If a direction vector with exactly the same origin and angle already exists, it's predictability will simply be overwritten, if the new value is higher than the stored one.

Direction vectors also have to age. An equation similar to Equation (2.2) is used, as illustrated in Equation (2.8). Like before, γ_{DV} is an aging constant and K is the amount of time units passed.

$$P(A,D) = P(A,D)_{\text{old}} \cdot \gamma_{\text{DV}}^{K} \tag{2.8}$$

2.4.3 Encountering hosts

Just like the standard PRoPHET implementation, the location-oriented adaption has to take care of some data exchange before the actual exchange of packets takes place. First, the same procedure as described in Section 2.3.1 is executed. The next step is to age all stored direction vectors according to Equation (2.8). After that, the connected hosts exchange direction vectors and save them with the predictability calculated as stated in Equation (2.7).

2.4.4 Routing

After the initial transfer of data, the algorithm in Listing 2.2 is executed for every packet. The general workings are the same as in Listing 2.1, but the predictability is based on the location the packet is targeted at. Hosts scan their own way points and stored direction vectors to return the best predictability value they can find. If it is higher than the best predictability value the host with the packet has, then it is transfered.

```
if not TTL reached:
    if packet is location-targeted:
        for each connected neighbor:
            if neighbor did not see the packet yet:
                if neighbor location predictability > own location
    ↪ predictability:
                    sendPacket()
    else:
        let standard PRoPHET take care of this packet
else:
    dropPacket()
```

Listing 2.2: Pseudocode of the location-oriented PRoPHET Routing Algorithm

Chapter 3

Implementation

After covering the fundamentals in Chapter 2, in this chapter I will describe the implementation within the simulator Peerfact. First, I will go into the challenges I have faced during implementation in Section 3.1. Then, I will outline the workings of the implemented classes in Sections 3.2 to 3.7 and the database model in Section 3.4.

3.1 Challenges

3.1.1 PRoPHET+

The implementation of PRoPHET+, as described in [HLC10], which had already been done in Peerfact, was not adequate to build location-oriented routing upon. While it provided some of the basic equations and mechanisms (i.e. encounters) needed, it lacked the transitivity aspect of PRoPHET. Furthermore it depended on several more parameters when deciding whether to forward a packet. While that is not bad per se, those would be additional factors which might interfere with the simulation and its results. Therefore I decided to implement a standard version of PRoPHET according to Section 2.3 instead of building onto PRoPHET+.

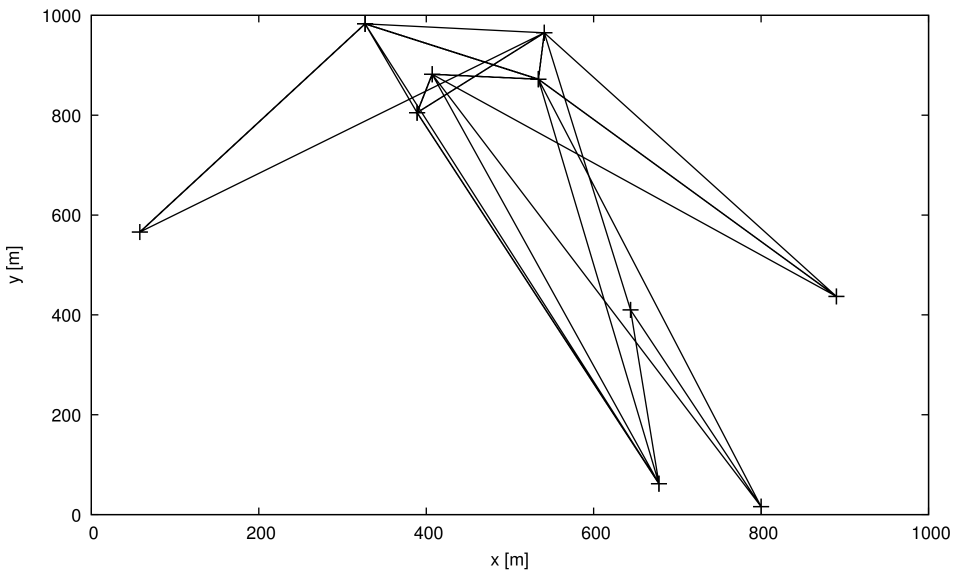

Figure 3.1: Example of Host Movement

3.1.2 Movement Model

Peerfact uses movement models, which implement an interface by which the next movement points can be retrieved. First, I used the random movement model while writing and testing the rest of the implementation. But for the simulation I wanted something less random, because PRoPHET relies on the non-randomness of node movement. I tried to use other already implemented movement models, which seemed like they would suffice (i.e. line movement, rectangle movement), but none of them worked. Finally I decided to implement my own movement model, in which for each host a fixed number of random points is chosen at the start of the simulaton. Hosts can only move directly between these points. Everytime a point is reached, a new point is randomly chosen. Now hosts do not move through the whole map, eventually reaching the packet's target location on their own, but instead rely on meeting other hosts. Two examples of host movement can be seen in Figures 3.1 and 3.2.

3.2 The LocationRoutingPacket

A packet routed by a location-oriented routing algorithm has to store information about the area, to which it has to be forwarded. The existing *RoutingPacket* already provides basic

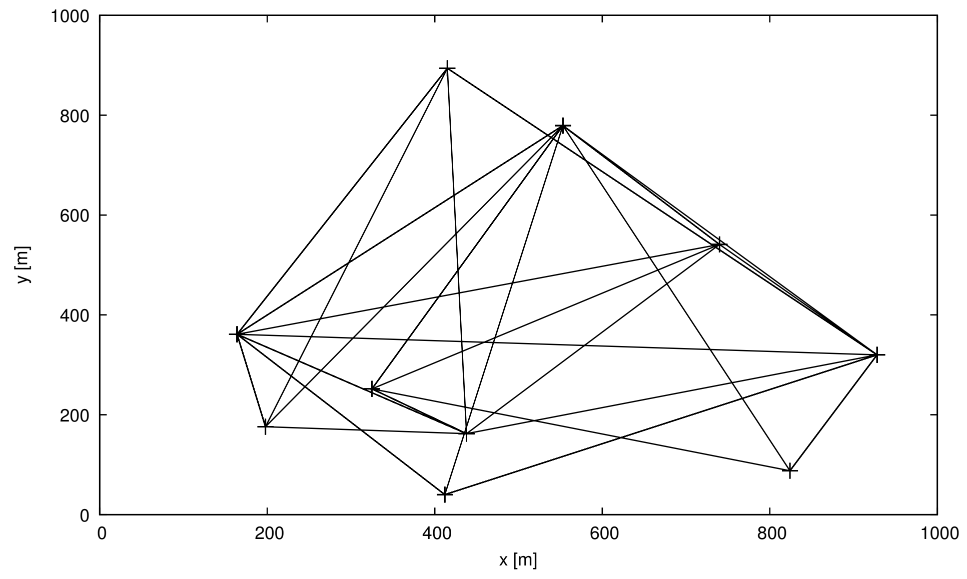

Figure 3.2: Another Example of Host Movement

features such as a hop counter, TTL and some more features not used in this thesis. The *LocationRoutingPacket* extends the RoutingPacket and adds coordinates and a radius to its information. Furthermore it adds a list of locations, so one can track the path of each packet. Everytime the simulator calculates a new location for a host, it adds it to all LocationRoutingPackets this host holds. An example of the movement of a packet and its target area can be seen in Figure 3.3.

3.3 Configuration Classes

Both, standard PRoPHET and location-oriented PRoPHET rely on a configuration class to feed them with the values of variables and exact working of the methods described in Sections 2.3 and 2.4. *ProphetConfig* stores the information relevant to standard PRoPHET, while *LocationProphetConfig* extends that class and adds information relevant to the location-oriented PRoPHET. By using these classes, or more precise instances of them, it is easy to pass objects with host specific configurations to other parts of the implementation, which need to access it.

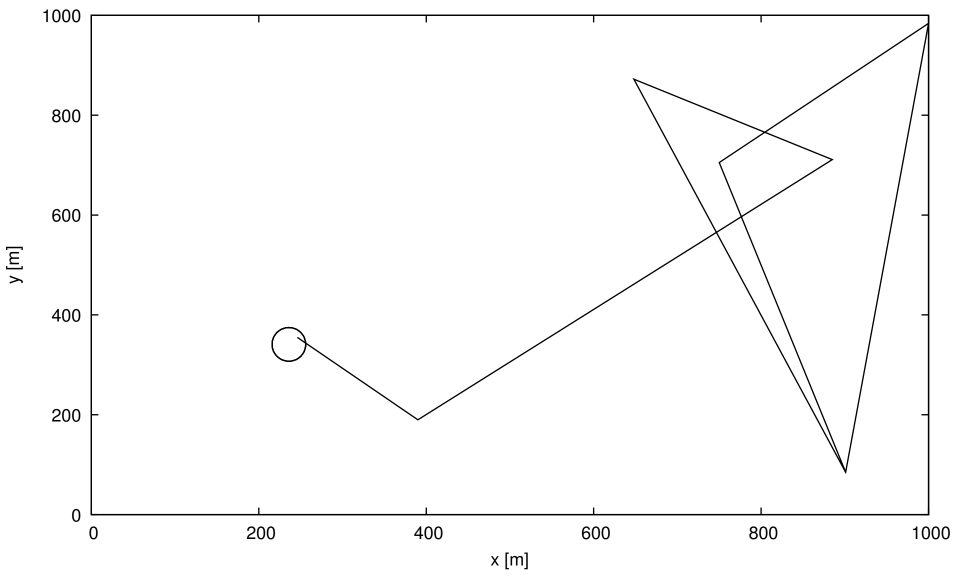

Figure 3.3: Example of Packet Movement and its Target Location

3.4 Database

To store all the information every node aquires over time, I have chosen to use an in-memory SQLite database. The reasoning behind this is, that it works reliable and fast even with large amounts of data. To minimize the overhead, only one database exists for all nodes, as opposed to one database per node.

3.4.1 Entity-Relationship Model

As can be seen in Figure 3.4, the database consists of the three following tables: `encounters`, `way_points` and `direction_vectors`. All three save their respective predictability value in `predictability` and a time stamp of the last time the predictability value has been aged in `lastUpdate`. They also all use `id` as primary key for easy reference to the entries.

In addition to that, `encounters` stores the owner of the entry in `host1`, the encountered host in `host2` and a time stamp of the last time the two have encountered each other in `lastEncounter`. There can only be one entry for each (host1, host2) combination.

`way_points` stores the owner of the entry in `host` and its coordinates in `x` and `y`. There can only be one entry for each combination of host, x and y.

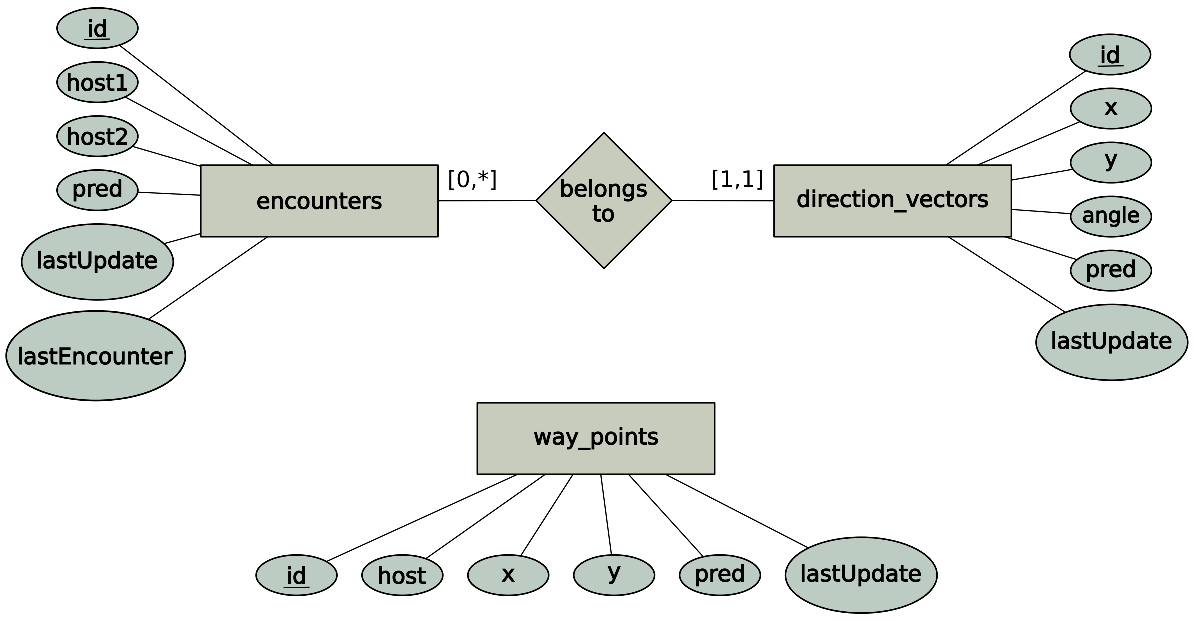

Figure 3.4: Entity-Relationship Model of the Database

`direction_vectors` stores its point of origin in `x` and `y`, its direction as an angle in `angle`. It also stores `encounter_id` as a foreign key to the `encounter`'s primary key, and thus each entry belongs to one encounter. There can only be one entry for each combination of (x, y, angle, encounter_id).

3.4.2 Class Interface

```
// Standard PRoPHET
void addOrUpdateEncounter(ProphetConfig pConfig, NetID
    ↪ encounteringHost, NetID encounteredHost, long currentTime);
void ageAllPredictabilities(ProphetConfig pConfig, NetID host, long
    ↪ currentTime);
void addEncountersTransitively(ProphetConfig pConfig, NetID
    ↪ encounteringHost, NetID encounteredHost, List<Encounter>
    ↪ encounterList, long currentTime);
double getPredictability(NetID encounteringHost, NetID encounteredHost
    ↪ );
List<Encounter> getAllEncounters(NetID host);

// Location-oriented PRoPHET
void addOrUpdateWayPoint(LocationProphetConfig pConfig, NetID host,
    ↪ long x, long y, long currentTime);
```

```
List<WayPoint> getAllWayPoints(NetID host);
void ageAllWayPoints(LocationProphetConfig pConfig, NetID host, long
    ↪ currentTime);
List<DirectionVector> getDirectionVectorsFromWayPoints(NetID host,
    ↪ long originX, long originY);
void addDirectionVectors(LocationProphetConfig pConfig, NetID
    ↪ encounteringHost, NetID encounteredHost, List<DirectionVector>
    ↪ dvList, long currentTime);
List<DirectionVector> getAllDirectionVectors(NetID host);
void ageAllDirectionVectors(LocationProphetConfig pConfig, NetID host,
    ↪  long currentTime);
```

3.5 Helper Classes

Because all hosts use one big database, it would make sense to let each of them access the other hosts' information instead of letting them actually transfer any of it. But I want to portray interactions between hosts as realistically as possible, so some helper classes are needed to transfer information about the encounters a host had, its way points and its stored direction vectors. These are `Encounter`, `WayPoint` and `DirectionVector` respectively.

3.6 Routing Algorithm Classes

The actual routing algorithm classes are the core of this implementation. They provide methods to process encounters and route packages to other hosts. *StandardProphetRouting* concerns itself with delivery predictabilities of encountered hosts and routing of packets targeted at hosts instead of locations. *LocationProphetRouting* is an extension to the aforementioned class and adds algorithms to take care of way points and direction vectors. Obviously, this class is also responsible to route packets targeted at locations.

3.7 Application

The last part of the implementation is a small application, which generates LocationRouting-Packets and lets hosts route them. It chooses random locations which serve as the target area the packets have to be routed to. The radius of the circular area can be defined in the scenario configuration file (see Section 4.1) by setting a lower and an upper limit. Between those, the radius is also randomly chosen. Because of the structure of the simulator the packets cannot be empty, so I have created a dummy UDP datagram, which is used for all created routing packets.

Chapter 4

Simulation and Results

In this chapter the implementation described in Chapter 3 will be used to simulate several scenarios. First, the way Peerfact scenarios are set up will be explained in Section 4.1. Then, the configuration of the nodes which implement location-oriented PRoPHET will be shown and explained in Section 4.2. After that, I will describe the different scenarios in Section 4.3 and present their results in Section 4.4 in regard to the metrics *hop count*, which is the number of nodes which carried the packet at the time it reached its destination, the *arrival ratio*, which is the percentage of total initial packets which arrived at its destination and the *time until arrival*, which is the time difference between when a packet was first created and the time it reached its destination.

4.1 Configuration file

Peerfact allows to define scenarios in XML files. Here, the behaviour of the different networking layers can be adjusted to one's needs. An example of a configuration file can be seen in Listing 4.1. In lines 2 to 12 some variables are defined, which can be used throughout the file. Lines 14 to 16 set up the simulator with a seed and the length of the simulation. The routing algorithm and algorithms for insertion of packets into the buffer and dropping packets are defined in lines 18 to 22. The application, as explained in Section 3.7, is set up with the radius the packet's target area should have in lines 24 to 26. The `GeoLayer`, which takes care of generating the world and placing and moving hosts, is configured in lines 28 to 33. Specifically, in line 32 the movement model described in Section 3.1.2 is set. In lines

35 to 40 the `HostBuilder` is constructed, which distributes nodes into groups, so different actions can be defined for each of them. Those actions are defined in another file, which is loaded by the `Scenario` class, configured in lines 42 to 44.

```
 1  <Configuration>
 2    <Default>
 3      <Variable name="seed" value="0" />
 4      <Variable name="finishTime" value="300m" />
 5      <Variable name="actions" value="LocationProphetScripts/
     ↪ locationProphet.dat" />
 6      <Variable name="world_X" value="1000" />
 7      <Variable name="world_Y" value="1000" />
 8      <Variable name="size" value ="100" />
 9          <Variable name="sizeSenders" value="50" />
10          <Variable name="sizeOthers" value="50" />
11          <Variable name="targetRadius" value="20" />
12    </Default>
13
14    <SimulatorCore class="org.peerfact.impl.simengine.Simulator"
15      static="getInstance" seed="$seed" finishAt="$finishTime">
16    </SimulatorCore>
17
18    <RoutingLayer class="org.peerfact.impl.routing.RoutingFactory" >
19      <BufferInsertAlghorithm class="org.peerfact.impl.routing.
     ↪ bufferPolicies.FiFo" />
20      <DroppingAlghorithm class="org.peerfact.impl.routing.
     ↪ droppingpolicies.TimeToLive"/>
21      <AbstractRoutingAlgorithm class="org.peerfact.impl.routing.
     ↪ algorithm.LocationProphetRouting" />
22      </RoutingLayer>
23
24    <Overlay class="org.peerfact.impl.application.
     ↪ locationpacketgenerator.LocationPacketGeneratorFactory"
25          radius="$targetRadius">
26      </Overlay>
27
28    <GeoLayer class="org.peerfact.impl.geo.modular.factory.
     ↪ DefaultGeoFactory">
29      <World class="org.peerfact.impl.geo.modular.world.DefaultWorld"
30        worldX = "$world_X" worldY ="$world_Y"/>
31      <PlacementModel class="org.peerfact.impl.geo.modular.placement.
     ↪ RandomPlacement"/>
32      <MovementModel class="org.peerfact.impl.geo.modular.movement.
```

```
      ↪ FixedPointMovement"/>
  </GeoLayer>

  <HostBuilder class="org.peerfact.impl.scenario.DefaultHostBuilder"
    experimentSize="$size">

    <Group groupID="Senders" size="$sizeSenders" />
    <Group groupID="Others" size="$sizeOthers" />
  </HostBuilder>

  <Scenario class="org.peerfact.impl.scenario.CSVScenarioFactory"
    actionsFile="$actions"
    componentClass="org.peerfact.impl.application.
      ↪ locationpacketgenerator.LocationPacketGenerator" />
</Configuration>
```

Listing 4.1: XML Configuration File Example

4.2 PRoPHET configuration

PRoPHET's configuration must be adjusted according to the situation it is used in. While the request for comments in which PRoPHET is defined suggests some initial default values for its parameters, some had to be slightly modified so that predictabilities do not reach extreme values. The relative differences are what matter here, and they cannot be taken advantage of if it is easy for the delivery predictabilities to reach 0.99 or 1.0 for encounter predictabilities and way point predictabilities respectively.

In Table 4.1 the used configuration for the nodes in this simulation is shown. The first part of the table consist of the variables used by the standard PRoPHET implementation, while the second part is used by the location-oriented part. The variables in the first part which differ from the suggested default values are $P_{\text{encounter}_{\text{max}}}$ (default 0.7) and γ (0.999). As for the second part of the variables, I have used default variables for similar variables and adjusted them from there. As for I_{typ}, I have run some of the scenarios and calculated the average of the time passed between two encounters of any two hosts. The calculated value is 5 468 793 497 µs, which is about 91.146 55 min in the simulator. For the application of the settings, refer to Chapter 2.

Setting	Value
$P_{\text{encounter}_{\text{max}}}$	0.75
$P_{\text{encounter}_{\text{first}}}$	0.5
$P_{\text{firstThreshold}}$	0.1
β	0.9
γ	0.99
δ	0.01
I_{typ}	ca. 91 min
$P_{\text{WP}_{\text{max}}}$	0.7
$P_{\text{WP}_{\text{first}}}$	0.5
$P_{\text{WP}_{\text{firstThreshold}}}$	0.1
γ_{WP}	0.99
WP_{intvl}	15 min
$\text{WP}_{\text{MinXDistance}}$	10
$\text{WP}_{\text{MinYDistance}}$	10
$P_{\text{DV}_{\text{threshold}}}$	0.01
γ_{DV}	0.9

Table 4.1: Settings of PRoPHET

4.3 Scenarios

In this section I will describe the different simulated scenarios. Some basic configuration is common to all scenarios. The simulated time for all scenarios is 300 min and will take place in a world with the dimensions of 1000 m times 1000 m. These settings can be seen in line 4 and lines 6 to 7 of Listing 4.1 respectively.

The scenarios differ in the number of nodes, number of created packets and the radii of the target locations of the packets. As shown in Table 4.2, I have defined 12 scenarios. In the first four scenarios, A1 to A4, 100 nodes participate, sending out 50 packets at minute 20. The radius of the packets is 20, 40, 60 and 80 meters respectively. The next four scenarios, B1 to B4, define 300 participating nodes with 150 created packets. The time when they are created and the radii are the same as in the A scenarios. Lastly, C1 to C4 use the same node and packet count as the B scenarios, as well as the same radii. The packets are created much later here, so that the nodes have more time to learn about their way points and meet other nodes.

Each scenario has been run multiple times with different seeds and their results have been

Scenario	Nodes	Packets	Send time	Radius
A1	100	50	20 min	20
A2	100	50	20 min	40
A3	100	50	20 min	60
A4	100	50	20 min	80
B1	300	150	20 min	20
B2	300	150	20 min	40
B3	300	150	20 min	60
B4	300	150	20 min	80
C1	300	150	120 min	20
C2	300	150	120 min	40
C3	300	150	120 min	60
C4	300	150	120 min	80

Table 4.2: Scenario configurations

combined into averages, to ensure representative results and not only those of a possible outlier. Scenarios A1 to A4 have been run 20 times, while scenarios B1 to B4 and C1 to C4 have been run 10 times each.

4.4 Results

In this section the results of the simulations described in Section 4.3 are presented.

4.4.1 Hop Count

The results of the simulation regarding the hop count can be seen in Figure 4.1. We can see, that the average hop count of the scenarios with less nodes is significantly lower than in those with more nodes. Because packets do not have a lot of possible paths to their target location, they tend to stick with their hosts longer until a better host is found. In the second and third group of scenarios a downward tendency of the average hop count with increasing radii of the target locations becomes evident, which cannot be said about the A scenarios. While the average hop count here only drops from 4.93 to 4.25, it drops from 10.85 to 8.36 in the B scenarios and from 8.01 to 6.48 in the C scenarios. The average hop count in the C group

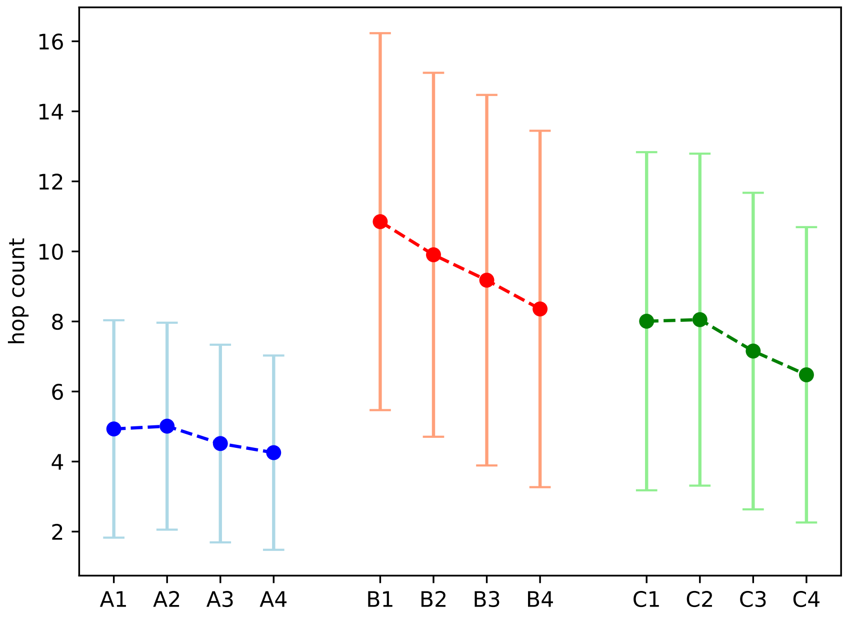

Figure 4.1: Hop Count

is generally lower than in the B group, because nodes have more time to learn about their environment and other nodes before having to deal with routing packets.

4.4.2 Arrival Ratio

The averages of the arrival ratio can be seen in Figure 4.2. With an increasing radius of the target location, the arrival ratio also considerably increases. While only about 11 % of packets arrive in the scenarios with the smallest radius, up to 44 % of packets reach their destination in the scenarios with the largest radius. Nodes have a much higher chance to go through a larger area, so more nodes are actually able to deliver a packet to its destination. The graphs of the different scenario groups are very similar, with slightly lower numbers in the B and C group. In all cases the average arrival ratio does not cross the 50 % mark.

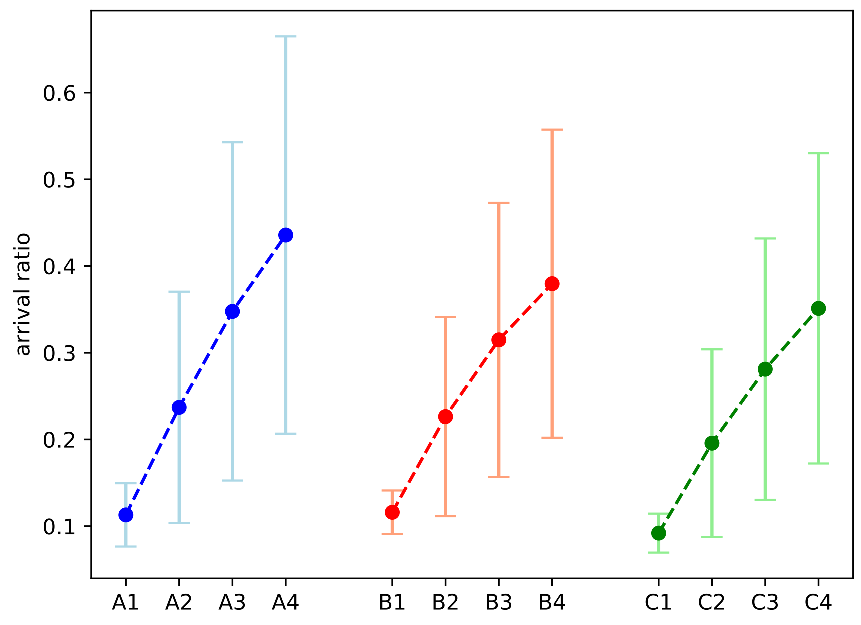

Figure 4.2: Arrival Ratio

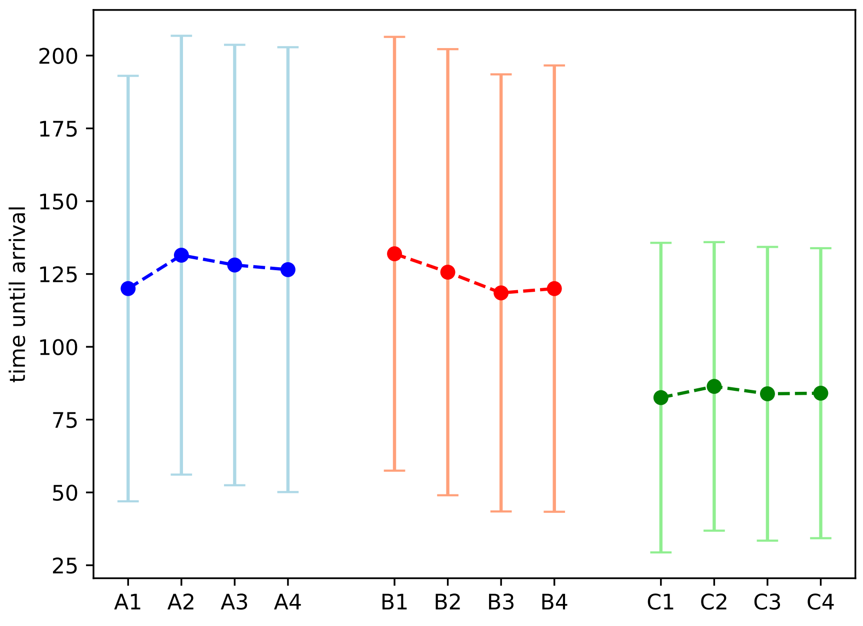

Figure 4.3: Time Until Arrival

4.4.3 Time Until Arrival

As can be seen in Figure 4.3, the radius of the target location does not seem to influence the average time until a packet arrives at its destination. Within each scenario group, the averages are always similar. The standard deviation is large in all cases, especially in the A and B groups. The reasoning behind this is, that there are always cases in which packets begin close to their destination or need only a few quickly found hops and cases in which the opposite is true. In the C group, packets need significantly less time to reach their destinations, because the nodes in this group have a lot more time to get accustomed to their surroundings and know much better if they are eligible to deliver a packet to its destination. While the packets in the A and B scenarios need about 125 minutes, those in the C scenarios only need about 85 minutes.

Chapter 5

Conclusion

The goal of this thesis was to extend the original PRoPHET protocol in such a way, that routing of packets is not only possible from one node to another, but instead packets can also be routed to a geographical location. For this purpose the location-oriented PRoPHET approach has been designed with way points and direction vectors, which much like the encounters of the original PRoPHET store a probabilistic value. On those values' basis it is decided opportunistically, whether encountered hosts are qualified to deliver packets to their destinations. The location-oriented implementation in Peerfact works hand in hand with the standard implementation, by letting it manage the encounters itself.

The implementation has been used to simulate several scenarios with different node counts, packet counts, radii of the target locations of packets and times at which the packets are created. The results of the hop count and time until arrival metrics show, that PRoPHET works at its best, when nodes had enough time to accumulate data about other nodes and in this case about their own way points and the direction vectors others send to them. This was to be expected, since the history of encounters and transitivity are what makes PRoPHET efficient and that carries over to the history of way points and exchanged direction vectors, which provide transitivity. The arrival ratio is generally very low, which can probably be attributed to the part of the implementation, where the location of the packet is added. Here, only a few points along the line of the host movement are calculated, with gaps inbetween. When hosts move through the target location of a packet they are carrying, but it is completely in such a gap, the packet is not considered to have reached the destination. This can for example be remedied by checking for an intersection between the line spanned over two movement points and the target location. Perhaps the location checking can also be implemented elsewhere in

Peerfact. Unfortunately, there was not enough time to correct this and let the simulations run through again.

Nonetheless, this thesis shows that a location-oriented approach to PRoPHET works and can be used as a basis for further development in the direction of location-oriented routing in opportunistic networks.

5.1 Future Work

The implementation in this thesis can still be improved. First of all, the aforementioned problem with checking whether a packet has arrived at its target location should be remedied.

Further, the cartesian coordinate system has been used, which would not be well applicable in the real world. Instead, it should be considered to use a geographic coordinate system with latitudes and longitudes, like the WGS84[1], which is the reference coordinate system of the Global Positioning System (GPS).

As the area this location-oriented approach is used in becomes larger, it might be worth considering clustering larger areas together, so that direction vectors can be used on a larger scale if needed.

Another feature for the target locations of packets might be a second radius, greater than the first one. Inbetween those two circles, a distance factor could be used to let the predictability of direction vectors decline, depending on its distance to the inner circle.

Momentarily, packets which reached their target location do not behave different than if they did not yet reach their destination. It might be useful to implement another kind of TTL, which is time-based instead of hop-based and defines the time it should remain in the target area.

[1]World Geodetic System, 1984

Bibliography

[ata] *Math (Java Platform SE 8).* `http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html#atan2-double-double-`.

[GDLD11] GRASIC, Samo; DAVIES, Elwyn; LINDGREN, Anders; DORIA, Avri: The evolution of a DTN routing protocol-PRoPHETv2. In: *Proceedings of the 6th ACM workshop on Challenged networks* ACM, 2011, S. 27–30.

[Gli11] GLISSON, T.H.: *Introduction to Circuit Analysis and Design.* Springer Netherlands, 2011. 348 S. `https://books.google.de/books?id=7nNjaH9B0_0C`. ISBN 9789048194438

[Gra11] GRAFFI, Kalman: PeerfactSim. KOM: A P2P system simulator—Experiences and lessons learned. In: *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on* IEEE, 2011, S. 154–155.

[HLC10] HUANG, Ting-Kai; LEE, Chia-Keng; CHEN, Ling-Jyh: Prophet+: An adaptive prophet-based routing protocol for opportunistic network. In: *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on* IEEE, 2010, S. 112–119.

[KLS$^+$07] KOVA, Aleksandra; LIEBAU, Nicolas; STEINMETZ, Ralf u. a.: Globase. kom-a p2p overlay for fully retrievable location-based search. In: *Peer-to-Peer Computing, 2007. P2P 2007. Seventh IEEE International Conference on* IEEE, 2007, S. 87–96.

[LDDG12] LINDGREN, A.; DORIA, A.; DAVIES, E.; GRASIC, S.: *Probabilistic Routing Protocol for Intermittently Connected Networks.* `http://tools.ietf.org/rfc/rfc6693.txt`. Version: August 2012. RFC6693

[PPC06] PELUSI, Luciana; PASSARELLA, Andrea; CONTI, Marco: Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. In: *IEEE communications Magazine* 44 (2006), Nr. 11.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelor Thesis 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, 03.August 2017 Timo Lux

Hier die Hülle

mit der CD/DVD einkleben

Diese CD enthält:

- eine *pdf*-Version der vorliegenden Bachelor Thesis
- die LaTeX- und Grafik-Quelldateien der vorliegenden Bachelor Thesis samt aller verwendeten Skripte
- **[anpassen]** die Quelldateien der im Rahmen der Bachelor Thesis erstellten Software XYZ
- **[anpassen]** den zur Auswertung verwendeten Datensatz
- die Websites der verwendeten Internetquellen