Adaptive routing in opportunistic networks using global information

Master Thesis

by

Marek Holze

born in
Düsseldorf

submitted to

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

October 2017

Supervisor:
Jun.-Prof. Dr.-Ing. Kalman Graffi
Raphael Bialon

Abstract

The widespread use of mobile phones introduces a lot of scenarios, where conventional network structures and routing protocols are not useful anymore. Examples are file sharing at conferences or concerts is one example of this. Therefore, the importance of opportunistic networks is growing as well as the importance of Ad-Hoc networks. Opportunistic networks can be used in remote areas with no internet access. Previous work about routing in opportunistic networks were used without knowledge or local information of a peer. In this work, we look at multiple solutions. We decide to take a closer look at the solutions, where a peer has to make a decision every time one wants to either forward a message or not to send the message at all. This decision is based on a given or calculated probability. In this work, we aim to optimize this probability. We look at three approaches. We implement, validate and evaluate each of them. The first approach only looks for different probabilities of multiple scenarios and saves the best sending probability in a routing table for each scenario. The second approach uses simulated annealing to optimize a given mathematical structure. This mathematical structure tries to use different local and global information to generate a good probability. The last approach uses reinforcement learning to optimize the sending probability inside a simulation run. The routing table is built during a simulation run. For evaluation, we look at different output metrics. We see that the simulated annealing approach gives better results than the optimization approach. Though the optimization approach is faster in terms of calculation. We notice that the major computational complexity comes from generating the routing table. After we have generated the table, the both approaches are equally fast. One may say that obtaining the better results is more important. The reinforcement learning approach produces the best results. But it is the most computational complex approach because during every simulation a new routing table is generated. In future work a reinforcement learning based routing should be able to initialize the routing table, thus running the algorithm will be computationally less complex than our algorithm. We compare all approaches of this work to epidemic routing. Therefore, this work presents alternative solution for every case one would use epidemic routing. This implies the approaches are possibly helpful for every routing algorithm in opportunistic networks, like the use case of file sharing on events. This thesis is able to get improvements in all of the three approaches compared to epidemic routing.

Acknowledgments

I would first like to thank my thesis advisor Raphael Bialon of the Technology of Social Networks Lab at Heinrich Heine University in Düsseldorf. The door to office of Jun.-Prof. Dr. Graffi was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to acknowledge Dr. Thorben Jensen as the second reader of this thesis, and I am gratefully indebted to his for his very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without her. Thank you.

Contents

List of Figures

List of Tables

List of algorithms

Chapter 1.

Introduction

The widespread availability of mobile communication devices makes them an inseparable part of our daily lives. The number of mobile phone users worldwide is approximately 3.3 billion, [LB93] and [Rud76]. This number is close to half of the world's population. In the modern world, almost every mobile phone is installed with communication interfaces and sensory equipment. Furthermore, most of the modern automobiles are also equipped with Bluetooth, Wi-Fi, and numerous other components.

Conventionally, centralized networks are used. There are base stations and connected peers. Thus, every part of the network has its own routing protocol. For example, within a local area network (LAN) every host can send messages directly to the destination after building a routing table. There is no real decision to make when a host wants to send a message. The host can simply look up the destination within the routing table.

However, there are scenarios where there are no base stations or fully connected graphs of networks. Situations where such centralized networks are not fit include catastrophe scenarios [MP12], [SK12], or remote areas, like forests or oceans. There is simply no infrastructure to provide connectivity. Other examples where other network architectures are beneficial are:

- In the era of self-driving cars, the importance of car-to-car communication is growing. However, the car-to-car communication is not like the ordinary routing in local area networks. Each car follows its own path, which means that for a car the appearance of a connection to another car is random.

- The 'Internet of Things' is expected to prepare elevated connectivity of systems, services, and devices that use machine-to-machine communications. Additionally, 'Internet of Things' typically covers diversity of applications and protocols, [T$^+$03]. For many fields of automation, the interconnection of these devices is expected to be useful. Examples include applications

like a smart grid or on a large scale, smart cities. Like in the previous example, we have many devices with small or medium range wireless communication technologies that capture the mass-market.

- Finally, an interesting case of science are mobile sensor networks, which are also called MANETs, [Ipp15], or mobile Ad-Hoc networks. There are many examples, where scientists want to collect information from different locations within a territory. For example, they want to detect chemicals, motion or temperature. In conventional models these sensors are connected to a base station. But in the case of mobile sensors and in combination with their limited communication range, sensors cannot always be connected to a base station. However, we can assume that individual sensors periodically come into contact with each other through node mobility and due to the limitation of the territory. MANETs do not have to be opportunistic networks. In general, MANETs have a static or clarified structure. Opportunistic networks are networks, generally not structured. This means that the movement and position of peers are totally randomized. Sensors usually stay at the same place or move around a place. This can, for example, happen when a sensor is placed on a tree and the tree is shaken by the wind.

Ad-Hoc networks have one disadvantage: they use fully connected networks. In the given examples, a completely crosslinked network is unlikely and thus conventional infrastructure can become cost-ineffective. This is the reason for considering opportunistic networks.

Mobile communication devices are the key to the development of opportunistic mobile social networks [Tan89]. In opportunistic communications, the key factor is human mobility. There could be delays in message transfers when the movement has random factors, which means it is not calculable. What are the main similarities of the mentioned examples and thus the basis for opportunistic networks? To answer this question this thesis uses three assumptions for connectivity of the underlying network:

1. There is never a base station in range of any sender.

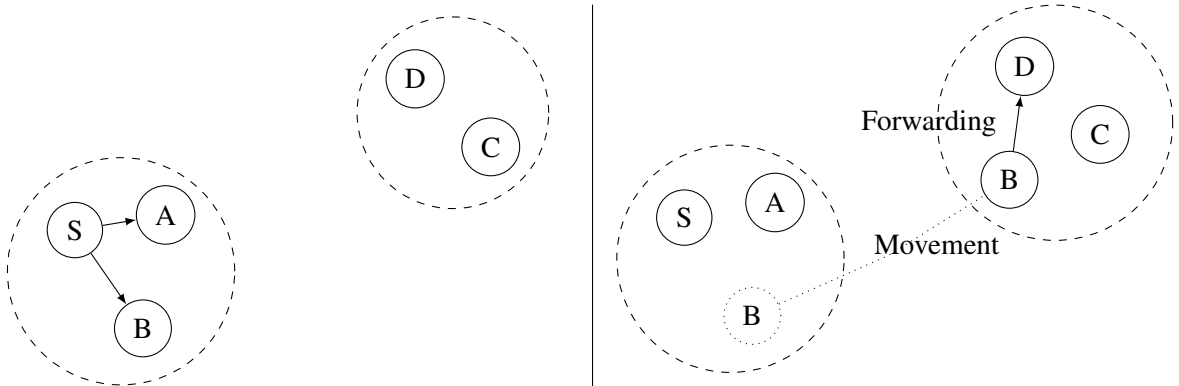
2. The sender does not know the best route to follow and where the receiver is currently located.

3. Through node mobility, pairs of hosts randomly and periodically come into communication range of each other.

In opportunistic networks, a path from the source to the destination does not have to exist at the time of sending. Networks have the characteristic that a peer can move randomly. Additionally, a peer can have no connection some time. Thus the peer has no chance to route a message if he want to. He has to wait until another peer comes into his connection range. This results in cases with no chance to route a message from a peer to the destination of the message. Other peers have to wait for an existing

connection with this peer. A routing protocol has to 'hope' the peers that contain the message meet the destination at some point.

Figure 1.1(a) shows an opportunistic network. A source S wishes to send a message to a destination D. But from S to D no connected path exists. S transmits its message to its neighbor B, which is in direct communication range. At a later point, as shown in fig. 1.1(b), B comes within direct range of D and sends the message to its destination.



(a) The source node sends its message to its neighborhood

(b) Some time later Node B comes in the transmission range of the receiver

Figure 1.1.: Example for an opportunistic network

As design issues for routing protocols we presuppose the uncertainty of the route. The sender has imprecise knowledge regarding the location of the nodes throughout the system. Therefore, there are two main issues. The first issue is to determine whether to transmit a message or not when a host comes into the range of potential carrier. The second issue is to determine which host of the neighborhood should send an appearing message. To solve the key issues this thesis compares three different attempts.

Opportunistic networks seem to fit to a large topic of real world problems on the one hand, on the other hand routing protocols have to find a routing path which exists probably only in future or never as a full connected path. State of the art solutions are generally working in two completely different routing ways. Regarding the first one, called epidemic routing, each peer gives everyone it meets all of his messages from the buffer. It is clear that this process is not complex and generates large overhead. This leads to the following question and the second method of routing protocols in opportunistic networks: Which neighbor should a peer pass the message to in an ideal scenario without knowing upcoming node constellation scenarios? One way to answer this question is to use historical data to compute the probability of meeting the destination node in the future. The protocol PRoPHET is based on this idea, [LDS04].
The usage of local data and its improvement against epidemic routing in PRoPHET+ leads to the

usage of global network information.

Since the routing using no information was able be improved by using local information, the strong suspicion develops that further improvements can be achieved through the use of global information. This work explores three ways to use global network information and make improvements over epidemic routing. Therefore algorithms are build for every approach. All of these approaches base on the basic idea to use a sending probability. The sending probability is looked up in a routing table every time a peer wants to send a message. A message is only being send if a random taken number is larger than the sending probability. The following three routing designs are explored and implemented in this work:

1. First, we evaluate and thus optimize the sending probability after every simulation run.

2. Second, we use simulated annealing and mathematical structures based on network attributes.

3. Finally, a machine learning algorithm is used to build a routing table, named reinforcement learning.

Design criteria of this work are efficiency and effectiveness. In terms of effectiveness we discuss the KIP of received messages by destination. For discussion of efficiency we will look at rate of forwarded messages, hop count and dropping rate of a message.

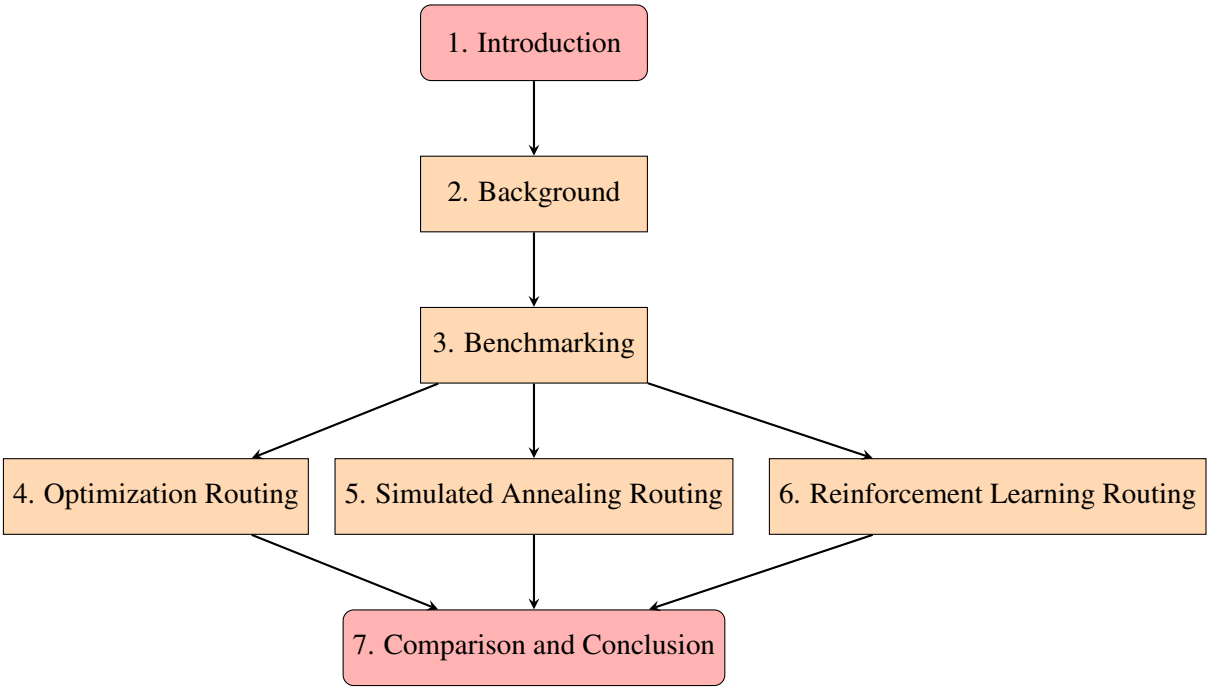


Figure 1.2.: Chapter of this thesis

This work is structured as shown in fig. 1.2. It starts by defining and explaining the problem in section 2.1. After that, we look at the current scientific research results from related works in section 2.2. In chapter 3 we discuss the evaluation environment. It contains all topics that are not part of later discussion, however, they are still important for understanding the simulation and the corresponding issues. In chapter 4 we try to find a good routing table using optimization based on global values such as network size. This is the easiest way to find a routing table. We discuss the routing algorithm and the evaluation of the found routing table as well. In chapter 5, we use simulated annealing to optimize the weight of a structure that we define. The structures give suggestions whether the routing algorithm should send a message or not. chapter 5, along with the simulated annealing algorithm itself, includes the implementation and evaluation of the algorithm. Chapter 6 discuss the complex task of finding the ideal routing table based on global information using reinforcement learning. The theoretical background of reinforcement learning, the routing algorithm and the evaluation of the found routing table are also discussed in that chapter. Finally, in chapter 7 we compare the presented approaches and give a final recommendation on in which case which approach should be used.

Chapter 2.

Background

In this chapter, we define and explain the problem considered in this work. After this we look at related work. Furthermore we discuss the used movement model and the used network architecture.

2.1. Problem definition

Opportunistic networks are based on assumptions. In terms of routing protocols, this means we assume that the probability of the peer we gave a message to is high enough to meet the destination of the message. In many scenarios, this assumption can be improved. Related works created routing protocols and tested them for a few scenarios. In general, the routing protocols performed better in one or two specific scenarios and generated worse values in other scenarios. Therefore, related works created routing protocols that generate good values in one or a few scenarios.

There are two approaches. First, different routing protocols can be used depending on global or local information at the time of sending. In order to implement this approach, characteristics generated from a monitoring are used. The goal of this approach is:

$$f(x_1, x_2, \ldots, x_n) \neq f(x_1) \cdot f(x_2) \cdot \ldots \cdot f(x_n)$$

where $x_1, x_2, \ldots, x_5$ are network attributes and n is the number of used information parts given by the network. This means that the new routing protocol would not be a concatenation of old routing protocols - instead, for each situation, the best case protocol to use should be found. The decision for the best case protocol is based on individual knowledge.

According to the second approach, a protocol could use the best sending probability for different network properties to make the decision whether or not to send a message. A message is only being

send if a random taken number is larger than the sending probability. A routing table contains for every possible network state a sending probability. A search problem can be defined: Find an optimal routing table *routingTable* with sending probabilities such that

$$fittness(routingTable) = \max\{fittnes(x)|x \in \Omega\}.$$

Here Ω is the set of all routing tables and $fittnes$ returns the achieved benchmark criteria of a simulation run in mean. A benchmark criteria could be the forwarded messages to destination hosts. An adaptive routing protocol results, which knows the best sending probability for the current network state. The following approaches could be considered:

1. First, we evaluate the sending probability after every simulation run. We run multiple simulations using different network properties and save the best probability for every network property. This is done in chapter 4.

2. Second, we use simulated annealing. We construct some structures, which should represent different information about the network or peer. A weighted sum of these values gives the sending probability for a specific moment in the network. Then search of an optimal weighting using simulated annealing takes place. We look at this in chapter 5.

3. Finally, this thesis implements an approach, which evaluates the sending probability after every routing step. For this purpose, a single simulation or slot of the simulation time is enough for creating a routing table. The only disadvantage is that every simulation run needs a lot of calculation time. This is done in chapter 6.

In all of this three solutions a search problem is defined: find the best sending probability for a network state.

The both approaches are different. The first one considers the use of multiple routing protocols, while the second approach uses a simple routing protocol based on a given probability to make the decision whether or not to send a message. The goal of this approach is to find the optimum for the probability based on the network state.

In this thesis, we focus on the second approach. We have decided to consider it because opportunistic networks are based on randomness. It is simple to look at a probability based routing protocol and optimize this probability. In future, both approaches could be compared.

2.2. Related Work

There are many characteristics of routing protocols. One of the most important characteristics while setting up a system is based on whether or not to create a replicate of a message. Routing protocols that never replicate a message are considered forwarding-based, whereas protocols that do replicate messages are considered replication-based.

This thesis studies the replication-based routing protocols. Epidemic routing and PRoPHET+, [HLC10], are represented in this category. They will be used to compare the adaptive routing protocol of this work to already existing solutions.

In Epidemic routing the sender node replicates packets and forwards them to all nodes that come in contact with it. The goals of Epidemic Routing are to minimize message latency, minimize the total resources consumed in message delivery and to maximize the message delivery rate. In [VB$^+$00] it is shown that Epidemic Routing achieves eventual delivery of 100% of messages with reasonable aggregate resource consumption in a number of interesting scenarios. On the other hand, epidemic routing results in flooding the network with identical data. It also produces unnecessary buffer and power overhead. The biggest disadvantage is the missing control over message dropping. In the majority of scenarios, almost all peers have a full buffer. Thus, every sending of a message initiates a dropping. Because epidemic routing sends a message whenever it is possible, this leads to a missing control over message droppings.

In opportunistic networks, the chance of having a direct or indirect connection path from source node to the destination is rare. Therefore, the PRoPHET algorithm tries to detect potential intermediate carriers. Forwarding data to couriers that rarely encounter the destination node will probably fail to deliver the data. While PRoHET only uses knowledge from historical meetings, PRoPHET+ makes use of a calculated predictability value. This value is calculated using the history of encounters between nodes, the remaining energy, the bandwidth of both peers and buffer to evaluate the packet forwarding preference.

[LDS04] describes the original PRoPHET protocol and evaluates if it's able to deliver more messages than Epidemic Routing with a lower communication overhead. Usage of network information improves the routing characteristics. Therefore, [HLC10] initiates PRoPHET+, a routing scheme for opportunistic networks designed to maximize successful data delivery rate and minimize transmission delay. It is based on local information. This means that information like buffer size, energy status etc. is being used. [HLC10] has proved that PRoPHET+ is able to achieve at least the performance of PRoPHET. [Hol16] discusses and proves the logical conclusion that PRoPHET+ also performs better than Epidemic routing.

All of this related work used local information for routing. Local information includes such information from the sending host as current empty buffer size or remaining energy of the host. In this work, we consider routing protocols based on global information.

2.3. Ad-Hoc networks

Since we are considering opportunistic networks in this work, we have to take a look at the network structure we are using for simulating. We use Ad-Hoc networks with full randomization of movement. So, our opportunistic networks are simulated.

Ad-Hoc networks are radio networks that connect multiple hosts to a meshed network. In a meshed network, every peer is at least connected to one other peer. Networks that build and configure themselves independently are special cases of Ad-Hoc networks. Ad-Hoc networks connect mobile devices such as mobile phones and notebooks without fixed infrastructure such as wireless access points. Data is passed from network nodes to network nodes until they have reached their receiver. Here the data load is more distributed than in networks with a central access point. Rare resources such as energy, computing power, and data rate call for effective network interconnection. Special routing procedures ensure that the network constantly adapts when nodes move, join or fail. Opportunistic networks are networks of wirelessly connected nodes. These nodes may either be mobile or fixed. Here we look at opportunistic networks with mobile nodes. [FTH16] also looked at multi-hop device-to-device networks. Therefore, it presents possible solutions to interconnect different groups to create multi-hop Ad-Hoc networks.

2.4. Random waypoint

In this work, we use the method of the random waypoint to simulate the movement of peers. The random waypoint is a generalization of the random walk model. In the random walk model, a peer makes a new decision after every step. This implies that the decision consists of points, which are directly next to the actual position. The stochastic process S_n of a random walk can be defined as follows:

$$S_n = S_0 + \sum_{j=1}^{n} Z_j, \qquad n \in \mathbb{N}_0 \tag{2.1}$$

Here $Z_1, Z_2, \ldots$ is a sequence of stochastic independent random variables with values in $\{-1,0,1\}^2$.

S_0 is set by the simulator and depends on the set seed.

The expectation $E(S_n)$ of S_n is zero. In our models this means every peer has a home area. The mean of all movements approaches zero as the number of movements increases. This follows by the finite additive property of expectation:

$$E(S_n) = \sum_{j=1}^{n} E(Z_j) = 0$$

An example of a random walk with two dimensions is shown in fig. 2.1. There are shown several peers represented by different colours. The scaling represents the world map coordinates.

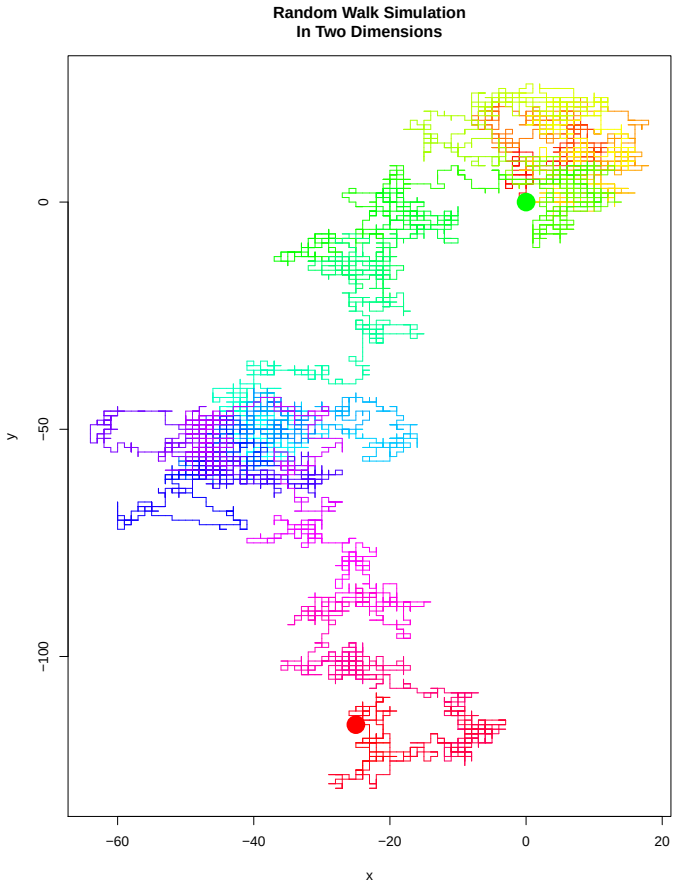


Figure 2.1.: Example for a movement of peers using random walk
Colors represent different hosts

The generalized model random waypoint is used in this work. It is a random model for the movement of mobile users. Movement models give a calculation for the location, velocity, and acceleration

change over time. This means that for the random movement model, the speed, destination, and direction of each host are chosen randomly and independently of other peers. The movement of nodes is managed by following steps:

1. For a fixed number of seconds every node begins with pausing.

2. The node then selects a random speed between 0 and some maximum speed and destination in the simulation area.

3. The node moves to this destination.

4. Go back to step one until the simulation is finished.

For eq. (2.1) this means $Z_1, Z_2, \ldots$ is a sequence of stochastic independent random variables with values in the set of *World*, defined in eq. (2.2). There first and second dimension refers to the two dimensions of $\mathbb{R}^2$.

$$\begin{aligned} World = \{(x,y) | (x,y) \in \mathbb{R}^2_+ \\ \text{and x is lower than maximum of first dimension of the worldmap} \\ \text{and y lower than maximum of second dimension of the worldmap}\}. \end{aligned} \tag{2.2}$$

In fig. 2.2 we can look at an example of a moving host using a random waypoint.

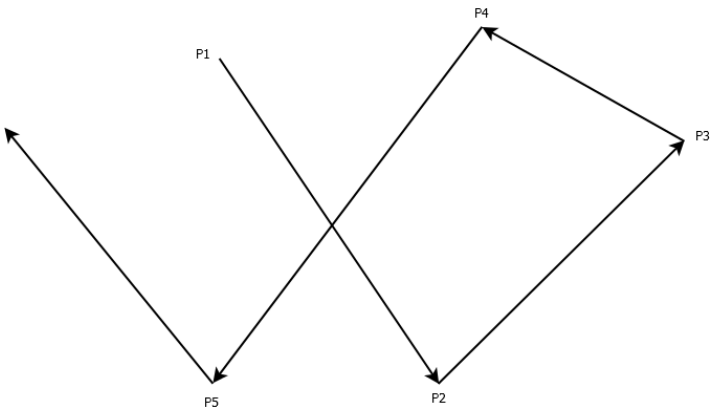


Figure 2.2.: Example for a movement of a peer using random waypoint

2.5. Monitoring in peer to peer networks

We discussed the simulation environment and movement model we use. Next, we have to clarify how we obtain the information on every host. In this work, we simulate opportunistic networks. We look at simplified models and vary specific parameters in order to build routing tables. If we would use real scenarios, we would not be able to build and compare routing tables for every scenario. Also, we are able to bypass the monitoring of the necessary network attributes. In order to obtain the required information, we use oracle-based optimization in this work. Here it means we use the analyzer in PeerfactSim.KOM to obtain the global information on every host. In general network monitoring are the supervision and regular control of networks, their services, and hardware. This means that we obtain (global) knowledge on the system statistics. One differentiates between external and internal monitoring. In the case of external monitoring, an additional monitoring device is connected to the network, but not for internal monitoring. In this work, we consider internal monitoring because external monitoring would be impractical in case of opportunistic networks. This would contradict to the assumption that there is never a base station in range of any host. External monitoring would prepare a device that can be used as a base station. A further characterization is made with the terms active and passive. In the case of active monitoring, additional packets are sent to the network, while the passive is only "monitored". In this work, we use passive monitoring. In future work, both methods can be used.

2.6. Conclusion

In this chapter, we have looked at the problem definition, related work and some background information. We introduced two approaches and decided to consider the second one in this thesis. Previous work looked at routing protocols considering local information. The main difference between related work and this work is the use of global information. Furthermore we looked at concepts to improve the the understanding of this thesis. To simulate opportunistic networks we use Ad-Hoc networks and a random movement model of the hosts.

Chapter 3.

Benchmarking

In the following chapter, we discuss the used simulation environment, permanent and variable parameters.

3.1. Benchmark environment

We use in our implementation the PeerfactSim.KOM, [pee17], simulation framework.

On simulated networks, every node has its own state, including the current load, capacities, strategies etc. Additionally a set of possible actions exist, which are triggered by workload or autonomously. For incoming messages, there is a defined reaction. We look at the event-based simulator PeerfactSim.KOM. In such cases, every event is scheduled for a time point and is only passed to the receiver when the time is due. An event can initiate new events. In general, the event-based model initiates a strict order for every event. This is more realistic, however, it is also more work to implement.

The simulator PeerfactSim.KOM was started in 2005 as an evaluation tool. The simulator was first developed at the TU Darmstadt at the Multimedia Communications Lab (KOM) under the guidance of Prof. Dr.-Ing. Ralf Steinmetz. It has been further extended and maintained at the University of Paderborn (UPB) and the University of Düsseldorf (HHU). The Community Edition of the PeerfactSim.KOM simulator has been initiated in July 2011 to support timely dissemination of scholarly and technical work, which is created by the users of PeerfactSim.KOM. Now it is used and heavily extended in the following P2P projects:

- DFG - research group 733 - QualP2P
- Improvement of the quality of Peer-to-Peer systems by systematically researching quality fea-

tures and their independencies

PeerfactSim.KOM is a simulator for large scale distributed P2P systems aiming at the thorough evaluation of interdependencies in multi-layered P2P systems. It is an event-based simulator, written in Java. Simulations up to 100 000 peers are possible. Its focus is on simulation of P2P systems in various layers. (Compare to [pee17])

In this work, we look at the network layer. There are taken the routing decisions. The simulator works by assigns a random point on the world map for every host. Then it defines the route for a host from the current position to this point and computes all meetings with other hosts. The simulator creates an event for every meeting. At the network layer following simple latency models are implemented:

- Static latency
- Distance-based latency

In this work, we use the static latency model. Packet loss depends on the geographical positions. The simulator has a layer structure based on the ISO/OSI model of communication networks. This structure is shown in fig. 3.1.

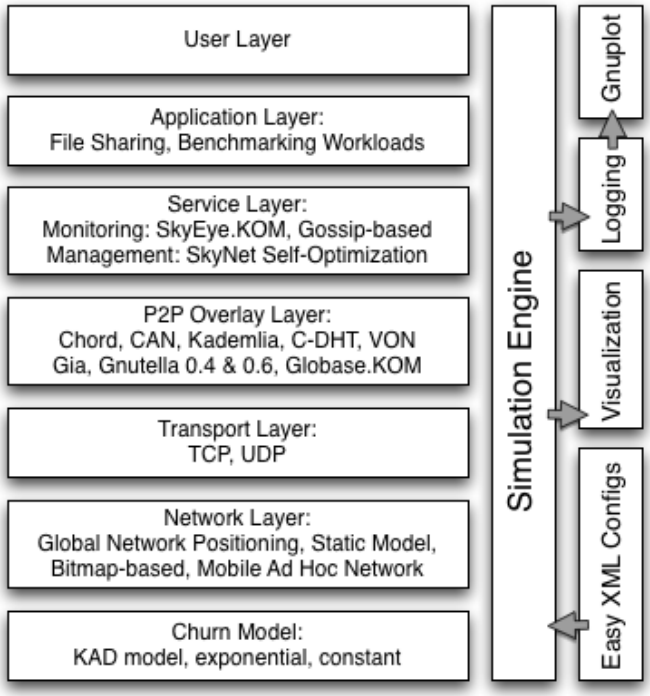


Figure 3.1.: Layer architecture of PeerfactSim.KOM, compare to [pee17]

For evaluation, we take from 2 to 65 peers, which send messages over the entire time and map with a size of a world of 65 for the x-axis and y-axis.

The buffer size is limited to 1 900 000 B. The message size is calculated by adding the payload size and the ip header. In general this means a size of 600 820 B bytes. Thus a peer can handle three messages.

For creating messages, we use the following logic: every time a host gets a message and if he is the final destination of the message, then he creates a new message. A message is not sent to the target if the target already knows the message, which means that every time there is the amount of messages in the network, then replications are excluded.

3.2. Optimization parameter set

In the following section, we discuss the attributes of a network simulated by PeerfactSim, which we later use for our routing protocols.

- First, we take the number of neighbors to which we are able to send a given message.
- Second, we use the count of messages that are already forwarded for a given minute.
- We want to use the potential messages that currently exist in the network.
- We use the mean of messages that are send without forwarding to a destination.
- Finally, we use the count of messages that are already sent for a given minute.

The most complex attribute of that is the count of potential messages in the network for a given minute. We take a closer look at it. The number is increased every time a connection is built between two hosts. Then both hosts look at their buffer and the sum of the count of messages in the buffer from both hosts is added to the number of potential messages. At this step, we additionally check which message from the neighbor hosts the hosts already know. For example, host A connects to host B and A has five messages at the buffer. But B knows already three of them. Then we add two to the number of potential messages. Potential messages also mean which messages should be forwarded ideally before the hosts are disconnected. Similarly, we subtract something if two hosts are disconnected, since the potential messages of the both hosts are no longer relevant. A host counts the messages of his buffer that have not been seen yet by the corresponding host. This count is subtracted from the count of potential messages.

If a host receives a message he counts the neighbors which have not seen this message yet. Then he adds this number to the count of potential messages. Similarly, if a message is sent to a neighbor then one is subtracted from the count. In cases when a message has to be dropped from a host then we subtract one for every neighbor who has not seen this message yet.

3.3. Benchmark criteria

Before starting with evaluation, we discuss the considered metrics. For random movement, first, we look at the count of forwarded messages to the destination with time in minutes on the x-axis. Messages that are sent directly from the source node to the destination are excluded. In terms of opportunistic networks, this metric is interesting because it can never be guaranteed that a message always reaches its destination. After that, we look at the hop count per minute. This gives a feeling how long a message exists. Delivering a message to its destination or dropping a message causes the end of the process. In addition, we look at the count of dropped messages. Thus, we get a feeling about how many messages there are at peers as well as when the peer itself does not forward the message to its destination. But the message could still be important to the peer. In cases, when the peer is part of a hop list of messages forwarded to its destination, a dropping is good. This is because the message is already delivered and there are no more copies needed. Finally, we look at the count of unfinished forwards. This is important when we examine movement models. For example, forwarding is stopped, because the host moves too much far away. In random movement, they may still be moving in the same direction.

3.4. Conclusion

In this chapter, we have discussed several basic topics that are required to understand this work. First, we looked at the simulation framework PeerfactSim.KOM that we use. The ad-hoc network structure and random movement model have already been implemented there. Therefore, we presuppose them in the following chapters. We looked at the network attributes we use in this work. The corresponding monitoring mechanism for obtaining these values has not been implemented in PeerfactSim.KOM yet. Therefore, we implement an oracle based monitoring. The implementation can be found on the attached CD and is not discussed in this work anymore.

Chapter 4.

Optimization using global attributes

In this chapter, we discuss the optimization of sending probabilities. We define a sending probability inside the routing algorithm and run the simulator for multiple probabilities. This way we are able to take the best probability for a given scenario. We vary these scenarios by changing multiple simulation attributes like TTL or network size. We remember that we want to find a better routing protocol than epidemic routing. For this we try to use global information. In a global view of this work, optimization using global information is the easiest way to build a routing table. We do not need to look closely at the way the networks work. The routing table is built by varying all attributes and gives the best probability to use for routing. We see in further chapters that it is also helpful to take a closer look at structures and the way the network is working, like in chapter 5.

4.1. Routing protocol

For routing inside of the simulator, every host has his own routing layer. This routing layer calls a routing function in three cases:

- For every minute of the simulation, the routing layer tries to send all the messages from the host's buffer to all of his neighbors. We set the interval of one minute in this work. For future evaluations, this interval can be changed.

- For a new neighbor, the routing layer tries to send all the messages from the host's buffer to all of his neighbors. In general, the other neighbors have already received the messages.

- For a new message from the link layer or transport layer, the routing layer tries to send all the messages from the host's buffer to all of his neighbors. In general, the neighbors have already received the previous messages.

In the following section, we take a look at the routing function. The function is called for every message separately. Parameters of the function are the message to send, the complete neighborhood of the current host and the current host. First, we route the message to its final destination if it is in the neighborhood. For this purpose, we are iterating over each host in the neighborhood checking whether the neighbor has not seen the message yet and if the neighbor is the final destination. In case the final destination is in the neighborhood, then there is nothing to do except sending the message and forgetting about it.

Algorithm 1 Routing optimize delivery rate

```
Inputs
    Message
    Neighborhood
    Actual Host
End
for all hosts in neighborhood do
    if Check if the message was already routed to neighbor then
        if The neighbor is the final destination then
            Send message
            Drop message at buffer of current host
            return
        end if
    end if
end for
for all hosts in neighborhood do
    if Check if the message was already routed to neighbor then
        potentialNeighborhood → potentialNeighborhood + 1
    end if
end for
for all potential hosts do
    generate a random number
    if Check if the number is higher than the given probability then
        numberForwards = numberForwards + 1
    end if
end for
for all numbers in range of 1 to potentialNeighborhood do
    while Got a new neighbor, use list of already routed neighbors do
        Fetch a random neighbor
    end while
    Add the neighbor to list of already routed neighbors
    Send message to the neighbor
end for
```

Next, we obtain the number of neighbors, which are interested in receiving the message. This means that we count the number of neighbors which have not seen the message yet.

After that, we obtain the number of neighbors we send the message to. In order to achieve this, we use the given probability, which we aim to optimize in this chapter. We iterate over the number of potential neighbors and take a random number between 0 and 100. In case the number is higher than the given probability, we do not increase the number of hosts we want to forward the message to. Otherwise, we increase.

Next, we route the given routing package to random neighbors maximum up to the given number of previous steps. We iterate over the given number of neighbors and take a random neighbor. We check if we have already routed the message to this neighbor. In case we have, then we take another random number. We repeat this until we get a new host. Thus, we are creating a list of hosts that have already been selected randomly and those that have not seen the message yet. In case of failed forwarding, we do not try this again. Then we add the new host to a list of already routed hosts. Finally, we send the message to the neighbor.

A pseudocode of the described algorithm is given in section 4.1. The principle of first in first out (FiFo) is used as the buffer policy and time to live of messages (TTL) is used as the dropping policy.

4.2. Optimization

We take a short look at the way we perform the optimization. Our method consists of trying several probabilities of multiple scenarios and finding the maximum. For running the large numbers of simulations, we need a script. There we manipulate the XML-files. The different scenarios are composed by varying the time to live of messages, size of the world map and number of hosts. For each of these scenarios, we run additionally the epidemic routing protocol once. This gives us a comparison to the found optimal. After this, we iterate over the probabilities we want to test. The steps which are used for iterating can be looked up in section 4.2. The limits are taken with the criteria that the whole simulation is done at least after two days.

In the following part, we discuss the optimization method. We have performed the optimization in R and here discuss the logic and formulas. The created table, table A.1, can be used as a routing table and is attached. We describe its use later.

After running the simulation, we calculate the mean of received messages by their destination from a forwarder for every simulation we ran. Next, we search for the highest mean of delivery rate depending on the sending probability. In order to find the optimized sending probability and the corresponding mean of delivery rate, we use the logic of eq. (4.1) and eq. (4.2). Here SP defines the sending probability, SR defines the matrix of simulation results and the parameter P defines the corresponding used

Algorithm 2 Script for optimization of sending percentage

```
for ttl in range from 2 to 20, using steps of 4 do
    for world size in range from 70 to 125, using steps of 10 do
        for host number in range from 5 to 55, using steps of 10 do
            Set ttl in config file
            Set world size in config file
            Set host number in config file
            SIMULATOR(EpidemicRouting)
            for percentage in range from 9 to 100, using steps of 10 do
                Set percentage in config file
                SIMULATOR(OptimizationRouting)
            end for
        end for
    end for
end for
```

percentage in the simulation run. In the second equation, the mean delivery is defined by MD.

$$RoutingTable(SP) := SR\left[i + \underset{x\in\{0,\ldots,9\}}{argmax}\left(\{SR[i+x](P)\}\right)\right](SP) \tag{4.1}$$

$$\text{with } i \in \{x | 1 \leq x \leq 2511 \text{and } x \equiv 1 \ mod\ 10\}$$

$$RoutingTable(MD) := SR\left[i + \underset{x\in\{0,\ldots,9\}}{argmax}\left(\{SR[i+x](P)\}\right)\right](MD) \tag{4.2}$$

$$\text{with } i \in \{x | 1 \leq x \leq 2511 \text{and } x \equiv 1 \ mod\ 10\}$$

Here i iterates over all the simulation results. Because we run ten simulations to find the optimal sending percentage, we add ten to i for every iteration over the time to live of a message, its size, and the number of hosts. This results in a maximum of $2511 = 2501 + 10$.

The resulting routing table is shown in table A.1. We notice that the optimization routing has a better performance than epidemic routing in terms of delivery rate. We discuss the reason for this in the evaluation section. In order to use the created routing table, a host in an opportunistic network has to know the network attributes. In general, this could be done via a monitoring procedure. After a host

has obtained all the values, he only has to look up the corresponding sending probability and use it as discussed.

4.3. Validation and verification

Because this work is rich in programming, verification is an important point. In further chapters, it will become of more importance. However, we perform some validations of optimization routing as well. It is possible to validate the script by looking directly at what happens while running the complete optimization. Every single simulation requires about one minute.

In fig. 4.1 we can look at all the sent messages of a five-minute simulation with five hosts in form of a graph. All the messages that are sent from source to destination directly are excluded.

There are a lot of forwarding between host two and three. The hosts two and three are always either the source or destination of a message. We take a closer look at one interesting message, which is forwarded through several hosts. The message with source host four and destination host three is forwarded the first time from host four to two. After this, host two forwards the message to host one. Finally, host one forwards the message to its destination, which is host three. Because two forwards the message also directly to three, we know that this connection is built some time later. Before a message is forwarded to the neighbors, we check if the final destination is in range.

On the other hand, there are messages forwarded to a host, which will never be forwarded again to another host or the destination. One example of this is the message with source host one and destination two, which is forwarded from one to four.

Time is a critical key for validation. We look at table 4.1 for examples of time in our validation. In the previous example, we were not able to look at the messages that were not sent. However, in this work, they are important. For example, host five decides not to send the message with source host four and destination four to host two during the third minute. During the fourth minute, every host repeats the decision whether or not to send a message. In this case, host five decides to send the message $(4, 3)$ to host two.

4.4. Evaluation

At this point of this work, we designed an easy to implement algorithm based on global information. In the following section, we discuss differences between optimization routing and epidemic routing

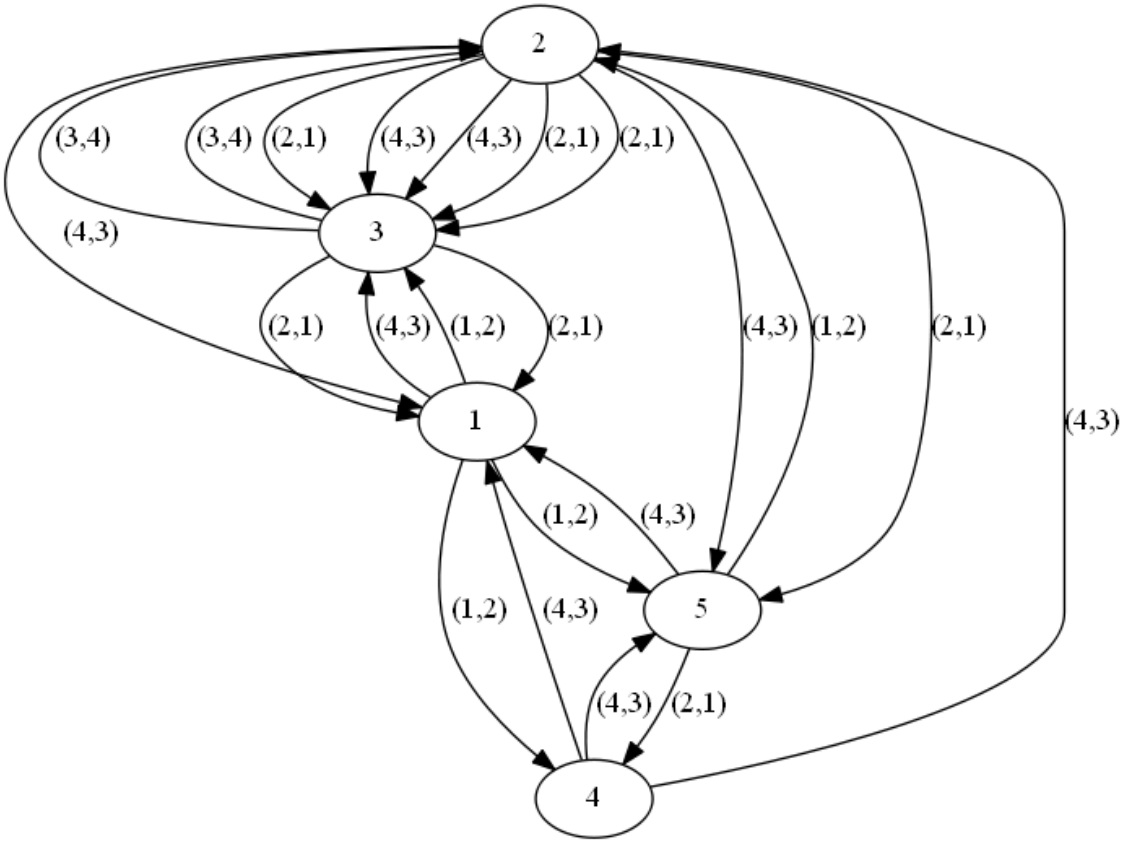


Figure 4.1.: Example graph of send messages using a sending probability of 50%
Hosts represented by nodes

for the first case of table 4.2. In this way we are able to proof that the design algorithm results in better outcomes than epidemic routing. We start by looking at the metrics as discussed in section 3.3.

For evaluation, we run simulations for several attributes and sending probabilities as discussed in section section 4.2. Depending on these attributes we run the simulation for optimization routing and epidemic routing. Thus, we are able to compare optimization routing and epidemic routing while creating a table for every attribute. Later we compare the delivery rate of messages. This means we count the messages forwarded from a host to their final destination. Thereby the sending host has to be not equal to the original sender.

In the following part, we run the generated routing table and look at the outputs. For that we take a short look at two cases, using the logic of eq. (4.3) and eq. (4.4).

Sended	Minute	Current Host → Receiver	Message (Sender,Destination)
Not sended	3	3→1	(3,4)
Not sended	3	3→2	(3,4)
Not sended	3	3→1	(3,4)
Sended	3	1→3	(1,2)
Not sended	3	1→3	(1,2)
Not sended	3	1→3	(3,4)
Not sended	3	3→1	(3,4)
Not sended	3	5→4	(4,3)
Not sended	3	5→2	(4,3)
Not sended	3	2→5	(3,4)
Not sended	3	1→3	(3,4)
Not sended	3	1→2	(3,4)
Not sended	3	4→5	(4,3)
Not sended	3	4→2	(4,3)
Not sended	3	1→2	(1,2)
Not sended	4	5→4	(4,3)
Sended	4	5→2	(4,3)
Not sended	4	5→4	(4,3)
Not sended	4	5→2	(4,3)
Not sended	4	4→5	(4,3)
Sended	4	4→2	(4,3)

Table 4.1.: Example table of send messages using probability of 50%

Case	htl	size	hosts	percent	optimization	flooding
Equation (4.3)	14	75	45	9	21.50	1.84
Equation (4.4)	14	115	5	69	1.34	1.34

Table 4.2.: Two cases using optimization routing

$$case = Routingtable[\underset{x\in\{1,\ldots,125\}}{argmax}\ (Routingtable[x](OptimizationRouting) - Routingtable[x](EpidemicRouting))] \tag{4.3}$$

$$case = Routingtable[\underset{x\in\{1,\ldots,125\}}{argmin}\ (Routingtable[x](OptimizationRouting) - Routingtable[x](EpidemicRouting))] \tag{4.4}$$

In table table 4.2 we can look at the two results of eq. (4.3) and eq. (4.4).

Case	htl	size	hosts	percent	optimization	flooding
Case 1	10	105	25	9	10	2.19

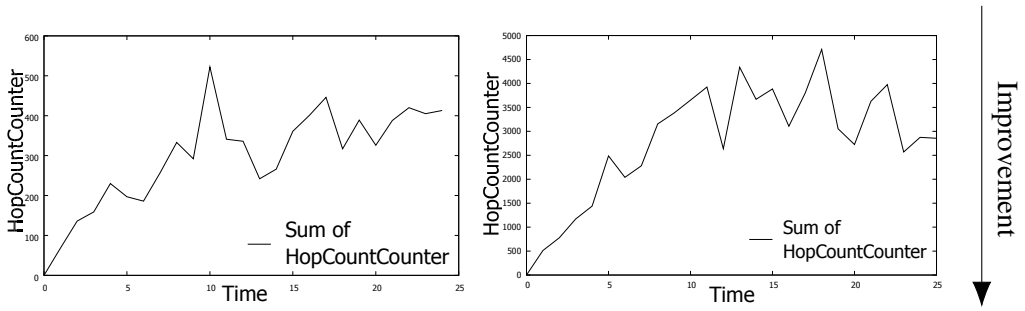
Table 4.3.: Random case of routing table using optimization routing

At next we take one random cases of the routing table and discuss it. The case is shown in table 4.3.

Every graph in the following part represents the changing over time on the horizontal axis like discuss in chapter 4. The vertical axis represents the count/ sum of the respective attribute.

Hopcount

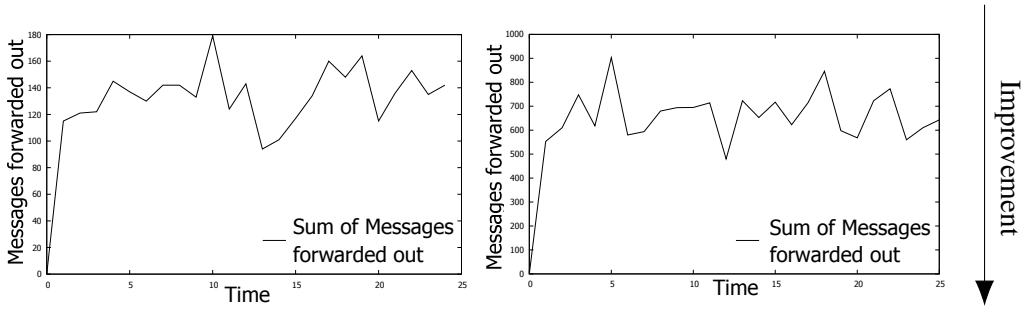
First, we compare the hop count.



(a) Hopcount of optimization routing (b) Hopcount of epidemic routing

Figure 4.2.: Hopcount of Case 1, scale on y-axes vary,
arrow on right indicates direction of improvement

We notice that the values in the both graphs in fig. 4.2 are growing over time and remain high with massive outliers. There is a difference in the vertical value range. The maximum value in epidemic routing is at about 4900 and is reached after about 19 minutes. The maximum hop count of optimization routing is about 520, which is taken after 10 minutes. The time required to reach the maximum value is probably caused by a random outlier. But the massive difference on the y-axis is serious. All in all, the hop counts from the graphs look similar in terms of structure, but there is a large difference in the count. These graphics represent the sum of hop counts and are based on the count of forwarded messages per minute. We look at it next.



(a) Forwarded messages using optimization routing

(b) Forwarded messages using epidemic routing

Figure 4.3.: Forwarded messages of Case 1, scale on y-axes vary, arrow on right indicates direction of improvement

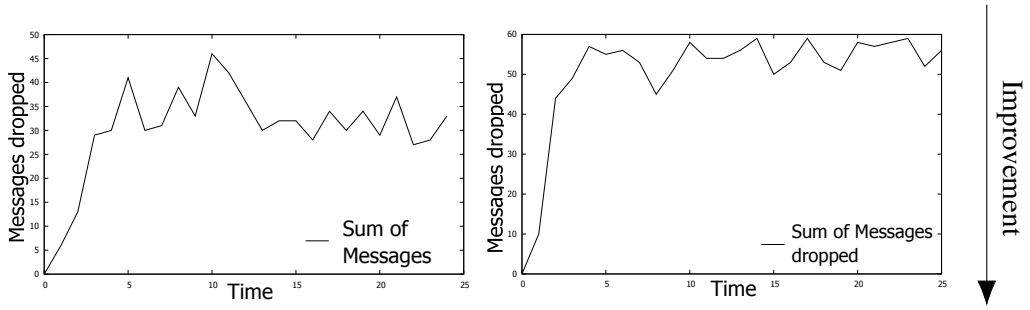
Forwarded messages

We are discussing the graphs from fig. 4.3, which show the count of forwarded messages per minute. Here too the structure of the graphs looks similar. The main difference is that optimized routing forwards on average 140 messages per minute, while epidemic routing forwards on average 650 messages per minute. On both graphs, there are many outliers. Once again, the maximum values are captured during different minutes, which is probably caused by the random outliers. We come again to the conclusion that the both graphs look similar except the used range on the y-axis. This is due to the reason that we are only sending with the probability of 19 percent in optimization routing, like shown in eq. (4.5).

$$forwards = 650\ messages \cdot 19\% = 123.5 \tag{4.5}$$

Dropping rate of messages

The dropping rate is shown in fig. 4.4. Here for the first time, the range of the counts is nearly the same. Maximum value from optimization routing is 45 and the one from epidemic routing is 60. There are once again many outliers on the both graphs. While optimization routing in the majority of cases is around 30 and has some outliers at the beginning up to 45, epidemic routing mainly fluctuates around value 55 and has some outliers below 43. All in all, optimization routing has fewer droppings than epidemic routing.



(a) Dropped messages on optimization routing (b) Dropped messages on epidemic routing

Figure 4.4.: Dropped messages at Case 1, scale on y-axes vary, arrow on right indicates direction of improvement

Received messages

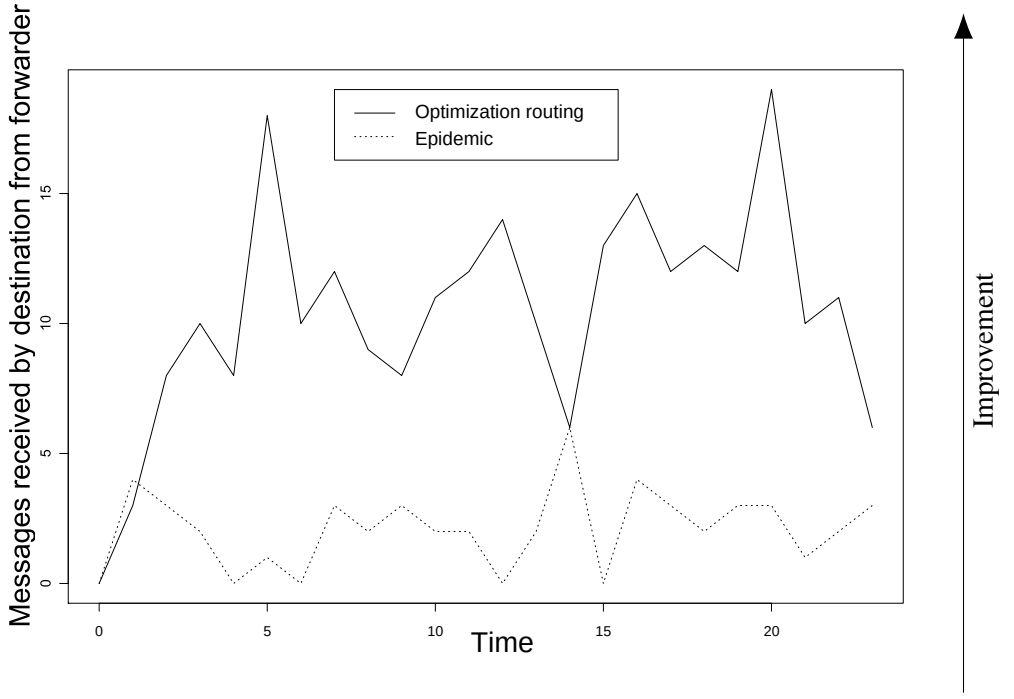


Figure 4.5.: Received messages at case 1, arrow on right indicates direction of improvement

Here we discuss the delivery rate. We can look at it in fig. 4.5. The general impression is that optimization routing delivers many more messages to a destination than epidemic routing. The maximum amount of received messages per minute for optimization routing is 20 and for epidemic routing it is 7. The maximum value of epidemic routing is achieved once. Optimization routing is able to reach its maximum value twice. During the first minute, epidemic routing delivers one more message to a destination in comparison to optimization routing. After this epidemic routing is never able to deliver

more messages. During minute 14, both protocols deliver the same number of messages. However, graphs have many outliers.

All in all, optimization routing performs better than epidemic routing. We notice that optimization routing sends fewer messages per minute, but forwards more messages to their destination. We compare the total improvement of optimization. In eq. (4.6) and eq. (4.7) using the metric of delivery rate to a destination we are able to look at the total number of cases, where optimization routing is better than epidemic routing and vice versa. The number of forwarded messages is stored for an x in $Routingtable_{OptimizationRouting}[x]$ or $Routingtable_{EpidemicRouting}[x]$. In numbers optimization routing always has a higher delivery rate than epidemic routing . Here FM defines the metric for the count of forwarded messages.

$$\sum_{x=1}^{100} 1_{\mathbb{R}_{\geq 0}}(Routingtable^{FM}_{OptimizationRouting}[x] - Routingtable^{FM}_{EpidemicRouting}[x]) = 100 \tag{4.6}$$

$$\sum_{x=1}^{100} 1_{\mathbb{R}_{\leq 0}}(Routingtable^{FM}_{OptimizationRouting}[x] - Routingtable^{FM}_{EpidemicRouting}[x]) = 0 \tag{4.7}$$

Now we try to understand the improvements of delivery rate. Like in section 4.4 the supposed reason for the better performance of optimization is that messages are not sent directly. Thus, every single host is less overloaded. If a message is not sent, the algorithm tries to send it again during the next action of the host. In general, this means that all hosts do not lose many messages while routing. In fig. 4.6 we can look at the unfinished forwarded messages. We notice that epidemic routing forwards more unfinished messages than optimization routing. The corresponding graph is shown in fig. 4.6.

In order to prove this adoption, we need to know in how many cases optimization routing has fewer unfinished messages. We calculate this in eq. (4.8) and eq. (4.9) if we save the count of unfinished messages at the RoutingTable. Here UM defines the metric for the count of unfinished forwarded messages.

$$\sum_{x=1}^{100} 1_{\mathbb{R}_{\geq 0}}(Routingtable^{UM}_{OptimizationRouting}[x] - Routingtable^{UM}_{EpidemicRouting}[x]) = 8 \tag{4.8}$$

$$\sum_{x=1}^{100} 1_{\mathbb{R}_{\leq 0}}(Routingtable^{UM}_{OptimizationRouting}[x] - Routingtable^{UM}_{EpidemicRouting}[x]) = 92 \tag{4.9}$$

Here it is clear that optimization routing sends fewer messages that are unfinished forwarded. In 92

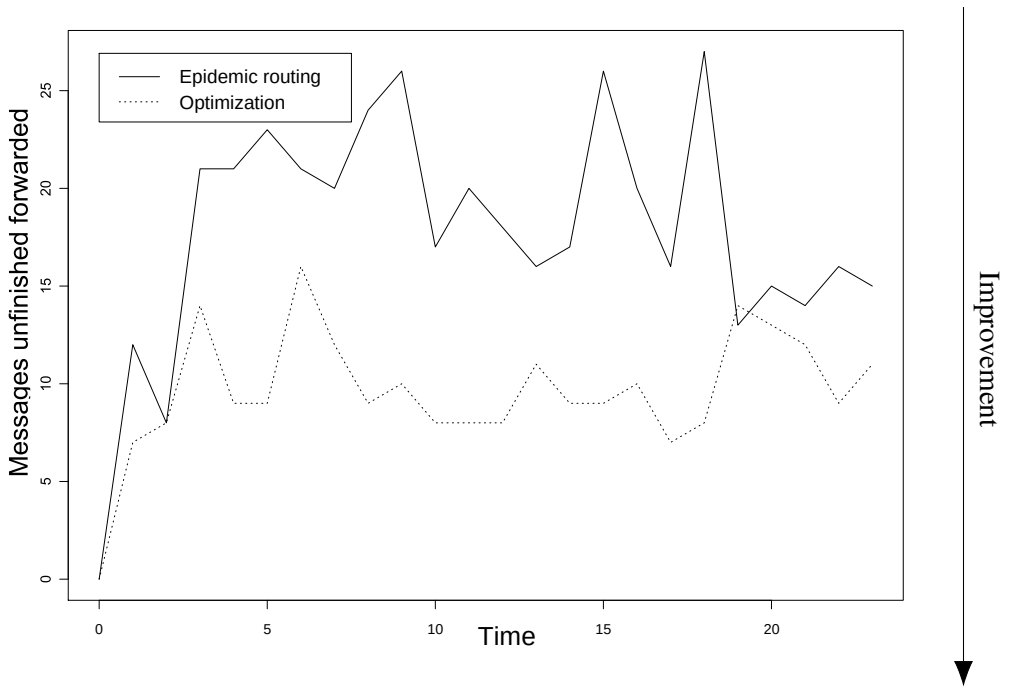


Figure 4.6.: Unfinished messages at Case 1, scale on y-axes vary,
arrow on right indicates direction of improvement

cases epidemic routing has more unfinished forwarded messages than optimization routing.

Up to now, we have evaluated a single simulation run. In following we look at the variations between 1000 simulation runs for the network attributes of case one in table 4.2 using multiple seeds.

In fig. 4.7 we can look at the boxplots of 1000 simulation runs using a logarithmic y-axis scaling. For hop count, there is a small range of median. Also, the proposition between the lower/ upper quartile and the maximum/ minimum looks symmetric. There are a few outliers above the maximum, however, they are close to the maximum. The general range is about 500 +/- 50. The median of the count of forwarded messages is about 210. We notice that the lower and upper quartiles are close to the median. The maximum and minimum lie at the distance of about ten to the median. There are outliers above and below the median, but again they are close to the maximum/minimum. The boxplot of dropped messages looks like the most focused metric. Again, median and upper and lower quartiles are close to each other and are around 40. The maximum and minimum values are at the distance of about one from the median. There are outliers that are very close to the maximum and minimum. Finally, there is the metric for messages received by the destination from the forwarder. The median is close to the upper quartile and has a value of about ten. The lower quartile has value nine. Maximum and minimum are about two points above or below the upper or lower quartiles. There are outliers next to

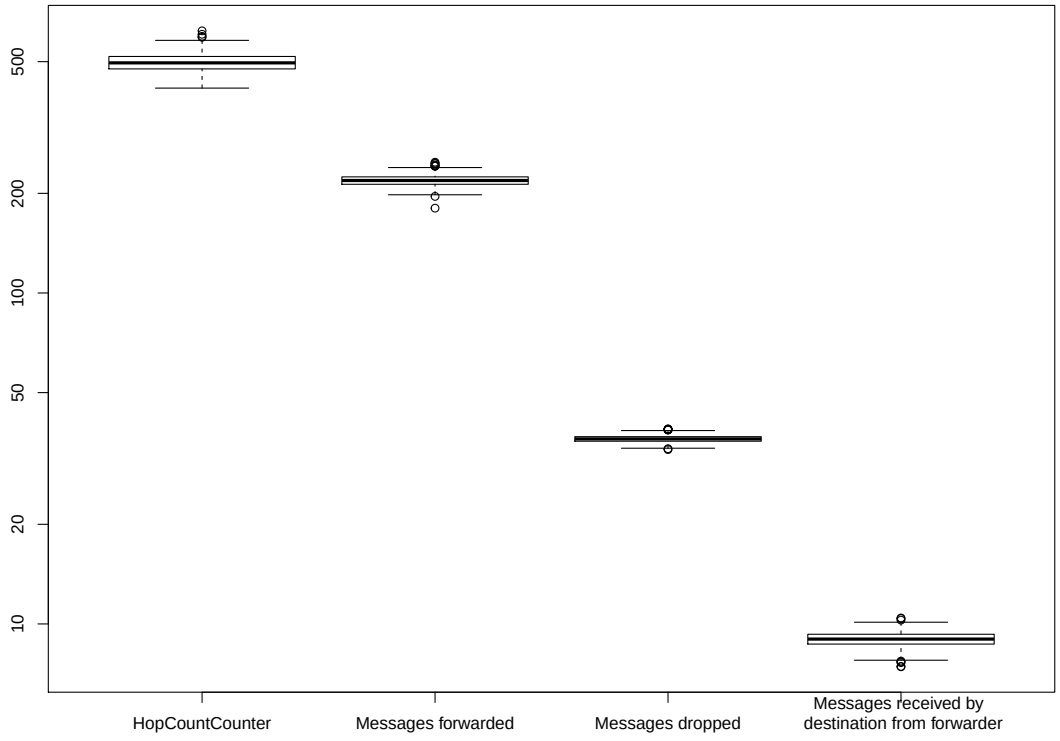


Figure 4.7.: Evaluation of Optimization Routing for 1000 simulation runs using Case 1 in table 4.3

the maximum and minimum.

4.5. Conclusion

As discussed in beginning of this chapter, we tried to fiend an easy to implement routing protocol which is better than epidemic routing. This performance increased should be raised by using global information. All in all, we have seen that optimization routing performs better than epidemic routing in terms of the maximizing delivery rate and reducing unfinished forwarded messages. We noticed that optimization routing sends fewer messages per minute, but forwards more messages to a destination. Thus, we know that optimization routing does not send more messages. Optimization routing is able to generate a knowledge of the network and performs better with same resources. We have seen that in every case optimization routing is able to deliver more messages and in 92 cases there are more unfinished messages at epidemic routing. In terms of complexity, this means that the routing algorithm is $O(1)$. The routing table can be sorted, so a peer only has to look it up. But depending on

the monitoring procedure additional messages are necessary. Their complexity is discussed in the in chapter 3.

In global view of this work, we already reached our target. In next chapters we still try to improve the performance using other algorithms. The algorithms still use global information, however, they are more complex.

Chapter 5.

Simulated annealing

In the previous chapter, we discussed an approach, in which global information is used. This information is constant in one simulation run (chapter 4). In the following chapter, we discuss the second approach: We want to use constant information in one simulation run and information of a peer or network at the time of sending. Based on this information we build some mathematical structures to make a decision. We optimize this decision and take a closer look at the way opportunistic networks work. Because the algorithm is complex, we run it only for one case. Thus, no routing table is generated in this chapter.

5.1. Simulated annealing algorithm

The simulated annealing method is an optimization algorithm introduced by Kirkpatrick in 1983 [KGV$^+$83] and by Cerny in 1985 [Cer85]. It is a heuristic optimization procedure. The procedure is used to find an approximated solution for optimization problems and excludes listing all possibilities or mathematical solutions due to their high complexity.

Given is the value range D, a fittness function $f : \mathbb{R} \to \mathbb{R}$, an environment term $U(x)$ and a termination criterion. We are looking for an approximate solution of the global minimum of f over D.

We select a starting solution $x \in D$. Next, we choose a monotonically decreasing sequence to zero $T_t, t \in \mathbb{N}$ and a sequence $N_i, i \in \mathbb{N}$ which indicates how many steps the evaluation of the simulator contains. We start with $t = 0$ and $i = 0$.

If $i \leq N_t$ we select a neighbor $y \in D$ from the environment $U(x)$, otherwise we set $i = 1$ and $t = t + 1$ and search again for a neighbor.

We have following decisions:

- In case of $f(y) \leq f(x)$ we set $x = y$ and if $f(y) \leq f(x_{\text{approx}})$ we set $x_{approx} = y$
- In case of $f(y) > f(x)$ we set $x = y$ using the probability $\exp\left(-\frac{f(y)-f(x)}{T_t}\right)$

Section 5.1 presents the simulated annealing heuristic as described above. It starts from a state x and continues while the *temp* is greater than one. At every iteration, *temp* is multiplied by $step = 1 - rate$. When we initialize *temp* at the value 10 and *rate* with 1e-3, then we have 2302 iterations. In the process, a dimension is randomly chosen to be optimized. We have three dimensions. Next, a new point is selected for the chosen dimension using the probability from above. Here the function f executes a simulation run and returns the delivery rate of messages.

Algorithm 1 The simulated annealing pseudo code

Inputs
  f
  x
  $temp \leftarrow 10000$
  $rate \leftarrow 1e-03$
End
$step \leftarrow 1 - rate$
$n \leftarrow length(x)$
$xbest \leftarrow xcurr \leftarrow xnext \leftarrow x$
$ybest \leftarrow ycurr \leftarrow ynext \leftarrow f(x)$
for $temp > 1$ **do**
  $temp \leftarrow temp \cdot step$
  $i \leftarrow \lfloor RandomNumber(min = 0, max = n) \rfloor$
  $xnext[i] \leftarrow NormalDistribution(n = 1, mean = xcurr[i], sd = temp)$
  $ynext \leftarrow f(xnext)$
  $accept \leftarrow exp(-(ynext - ycurr)/temp)$
  if $ynext < ycurr || RandomNumber(n = 1) < accept$ **then**
    $xcurr \leftarrow xnext$
    $ycurr \leftarrow ynext$
  end if
  if $ynext < ybest$ **then**
    $xbest \leftarrow xcurr$
    $ybest \leftarrow ycurr$
  end if
end for
return $xbest$

In fig. 5.1 we can look at a graphic for which it might be helpful to use simulated annealing. If this graphic is already generated, it is easy to go through all points on the x-axis. But in the case of continuous functions and when the calculation for a given value is complex, then it might be more reasonable to use simulated annealing. For example, in the shown graphic we start at zero. It is a local maximum. For the majority of following iterations, we stay at zero. However, some random time

later, we try the value 44 and notice it is better than zero. Then we move continuously to 45 and stay there. After we have tried enough iterations we accept that this is the global maximum.

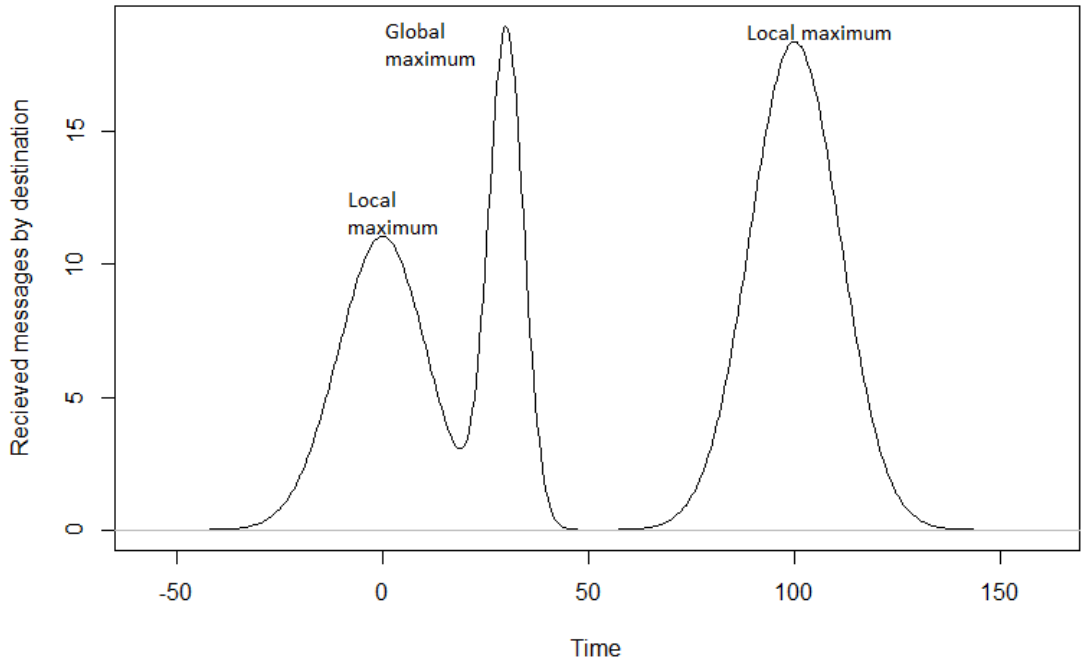


Figure 5.1.: Example for a environment using simulated annealing finding global maximum

5.2. Implementation in PeerfactSim.KOM

The most important part of this section is to find a "good" structure for manipulating the decision whether to send or not to send. We use the following three attributes of the network:

- Number of hosts at network.
- Potential messages in the network for sending at a given minute.
- Messages at network, which are already send at a given minute.

Now we come up with some formulas to create v_1, v_2 and v_3. Later we find an optimal weighting v_1, v_2 and v_3 for every w_i, $i \in \{1,2,3\}$. Every w_i represents a structure mentioned above and returns a probability for sending. Here, we describe the formulas:

$$w_1 = 1/potMsgs \tag{5.1}$$

$$w_2 = \frac{\min\left\{CountMsgsWantToSend, \frac{MeanSendMsgs}{CountHosts}\right\}}{MsgsWantToSend} \tag{5.2}$$

$$w_3 = \frac{\min\{CountMsgsWantToSend, MeanSendMsgs - CountMsgsAlreadySend\}}{MsgsWantToSend} \tag{5.3}$$

The implementation in PeerfactSim.KOM returns the sending probability of eq. (5.4).

$$sendingProbability = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 \tag{5.4}$$

5.3. Validation

In this chapter, we take a closer look at the generated values using the formulas from eq. (5.1), eq. (5.2) and eq. (5.3). This is part of the validation and we are able to get a better feeling of what is happening. Running the simulated annealing returns only a vector of the global maximum. In the following, we try to print the relationship between W_1, W_2 and W_3. For visualization, we use the programming language R. We are using the function "peerfact(x)" defined in the previous chapter. We are able to generate 3-dimensional plots. We print W_i on the x- and W_j on the y-axis with $i, j \in \{1,2,3\}$ and $i \neq j$. The third dimension for each combination is the mean of forwarded messages to a destination.

When we take a look at fig. 5.2, we notice that there are not so many changes for W_1 over time, excluding the outliers. W_2 is represented on the y-axis and has some changes. It looks as if it reaches the maximum at about 50 percent.

While study fig. 5.3 we notice that W_1 is not changing lot if we hold W_3 on zero. But once W_3 reaches circa 40 percent we are able to notice strong deviations for W_1. There the deliverability rate performes

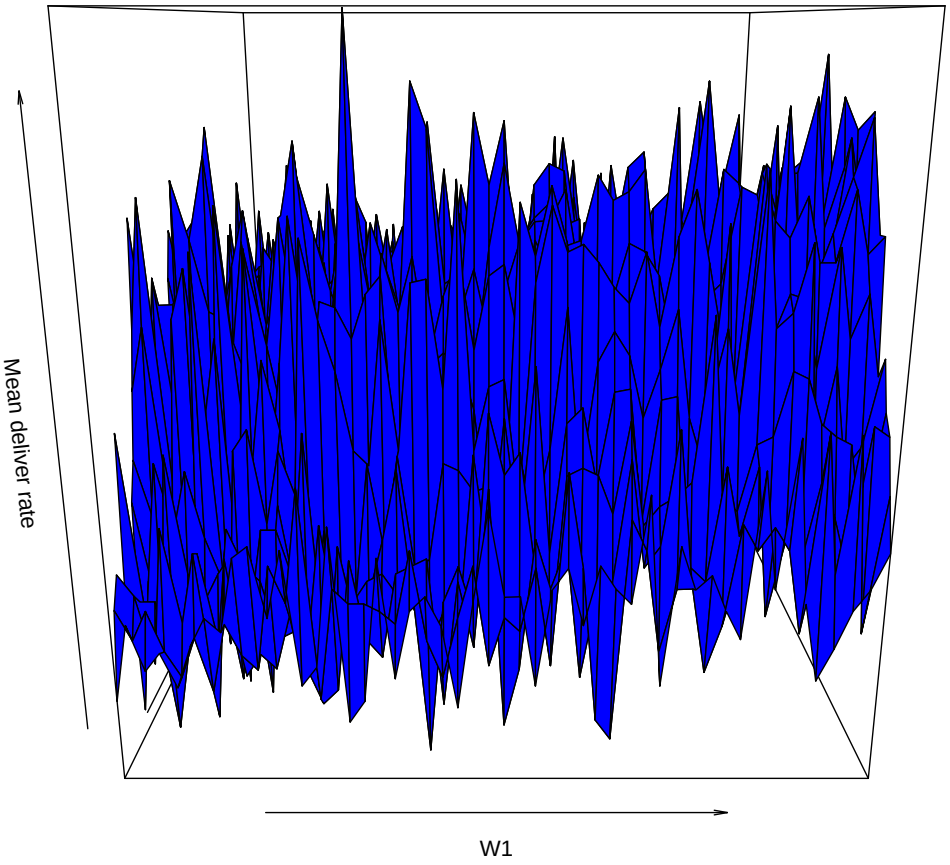


Figure 5.2.: Comparison between W_1 and W_2

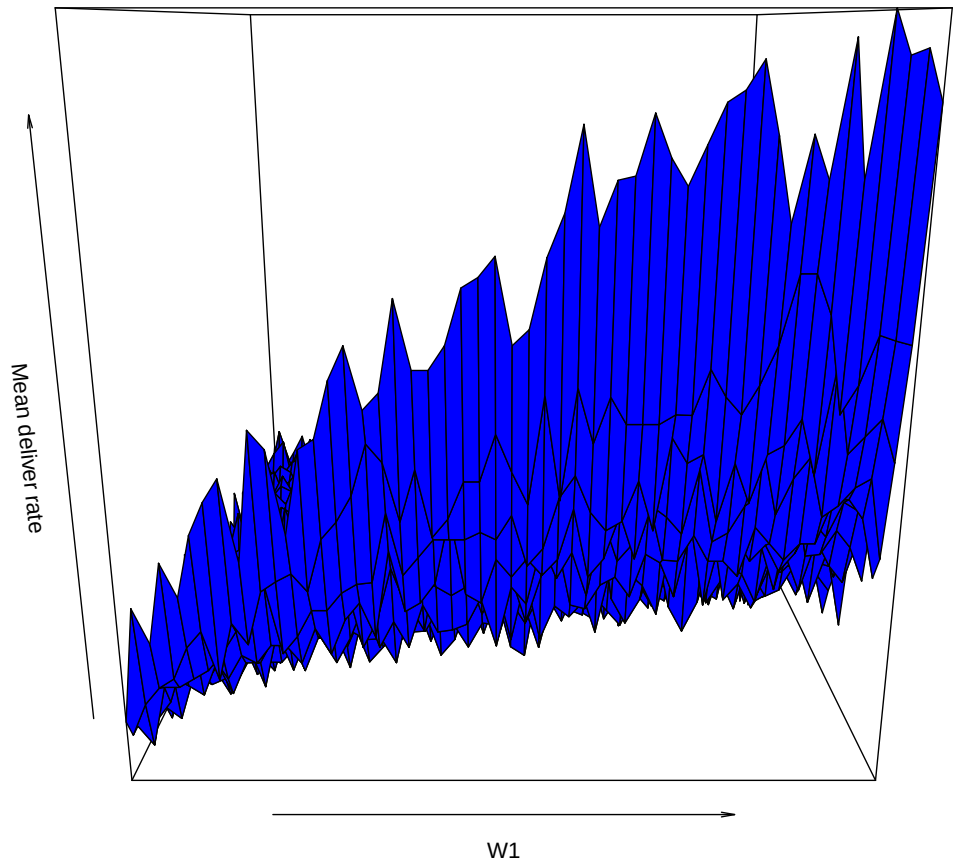


Figure 5.3.: Comparison between W_1 and W_3

much more better while increasing W_1. The meaning of this is the following one: If we give V_1 circa fifty percent of the power to decide if we send a message to a neighbor wether not, we are able to improve the results by increasing the power of V_1. So it looks like there is a connection between W_1 and W_3. In eq. (5.3) we are using the information about how much messages are already send at the given minute. This information was not used in eq. (5.2).

While studying fig. 5.3 we notice that W_1 is once again not changing much while setting W_3 to zero. But once W_3 is around 40 percent we notice strong deviations for W_1. The rate of delivery performs much better when increasing W_1. This has the following meaning: Giving V_1 about fifty percent of the power to decide whether or not to send a message to a neighbor improves our results. Then we can deduce that we are able to improve the results by increasing power of V_1. It seems as if there is a relationship between W_1 and W_3. In eq. (5.3) we use the information about how many messages are already sent at the given minute. This information was not used in eq. (5.2).

If we run simulated annealing using the parameter W_2 and W_3 it looks additional like there could be a connection between the both parameters. Here it looks like W_2 is able to increase the result if W_3 has circa 30 percent power. We notice that the connection here is another one than in fig. 5.3. On both graphs there is a wall with maximums in direction of x-axis. But W_2 changed from y-axis to x-axis.

When we run simulated annealing using parameters W_2 and W_3, then we notice that there could be a connection between these two parameters as well. Here it seems as if W_2 is able to improve the result when W_3 has about 30 percent the most power. We also notice that the connection is different from the one in fig. 5.3. On both graphs, there is a maximum in direction of x-axis. But W_2 was changed from y-axis to x-axis.

5.4. Evaluation

We designed in chapter 4 a routing protocol based on global information which performed better than epidemic routing. The global target of this work is finding routing protocols which performs better than epidemic routing. Therefore we will do this again for the routing protocol designed in this chapter. Later we compare the designed protocols to each other. In this section, we run the simulated annealing algorithm. This takes some time and returns the following vector:

$$optima = (98.10185, 52.69242, 1.39716)$$

The mean of forwarded messages to destination is 5.5625, which we use later when looking at the simulation outputs. We use the metrics as discussed in section 3.3.

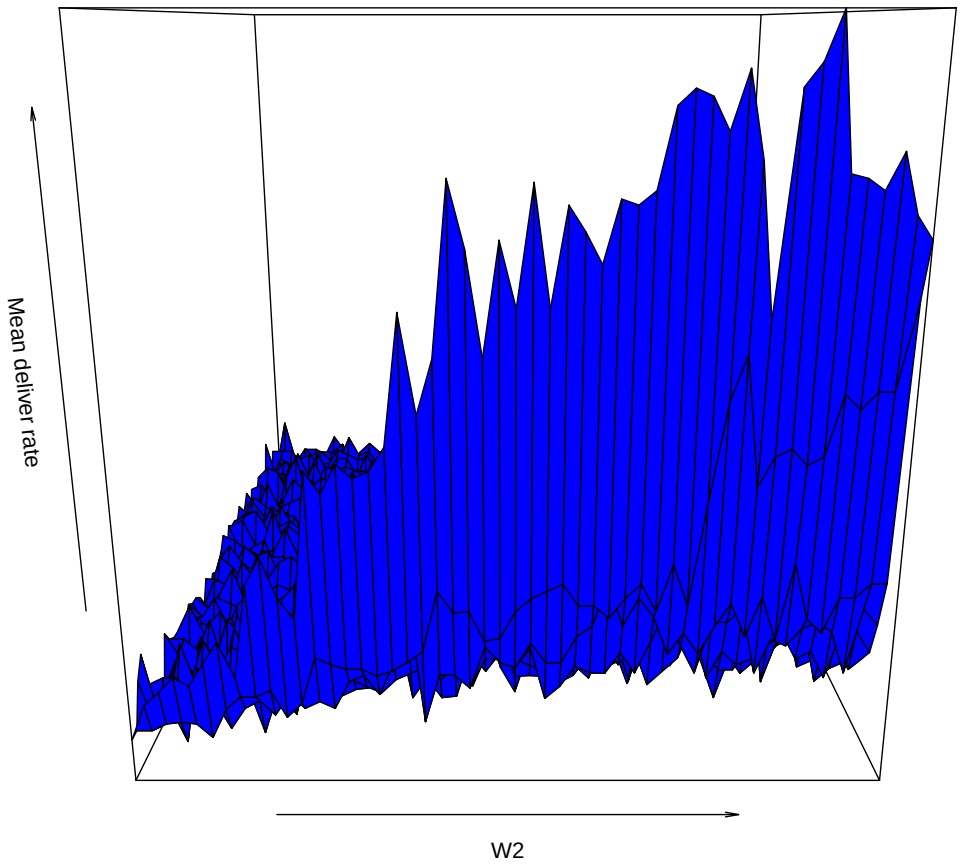
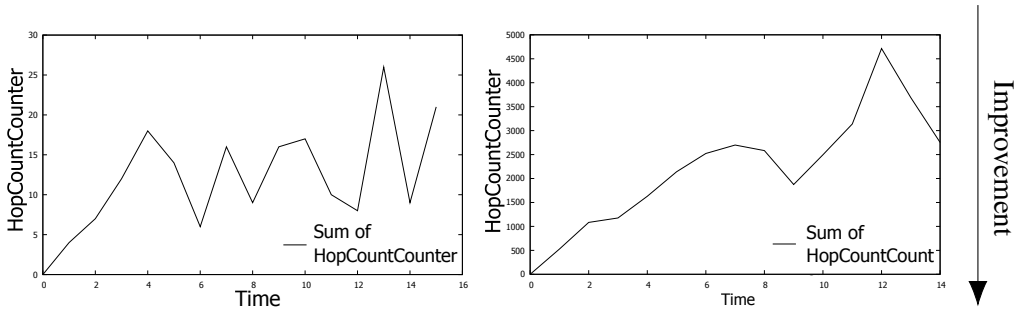


Figure 5.4.: Comparison between W_2 and W_3

In the following part, we evaluate the simulated annealing routing. We focus on the scenario, which is generated from the previous simulated annealing run. We supply the algorithm with the initial vector $(50, 50, 50)$. Since this algorithm is computationally intensive, this is the only run we try. For evaluation, we compare the outputs of the optimization with outputs of epidemic routing.

Hopcount

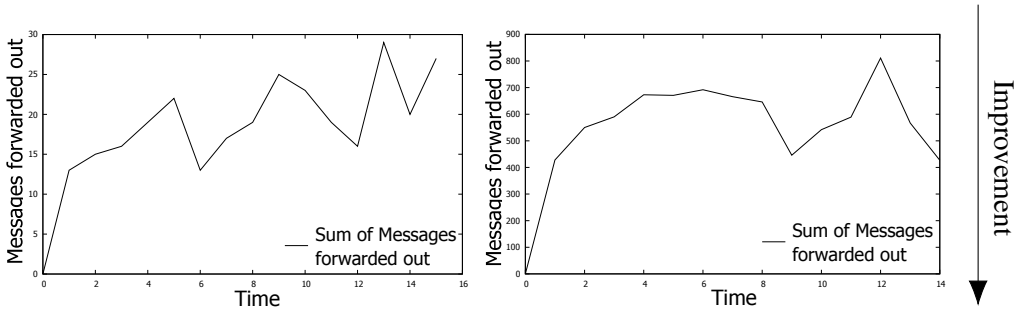
First, we compare the hopcount. It is shown at fig. 5.5.



(a) Hopcount of simulated annealing routing (b) Hopcount of epidemic routing

Figure 5.5.: Hopcount comparing simulated annealing and epidemic routing, scale on y-axes vary, arrow on right indicates direction of improvement

We notice that the hopcounts in both graphs in fig. 5.5 are growing over time and reach their maximum after about 13 minutes. While the value of epidemic routing is constantly growing over time, it looks as if simulated annealing is growing until the third minute up to value 20 and then fluctuates around this value. It is also possible that the maximum at minute 13 is only randomly taken. There is a significant difference in the value range of the vertical axis of the graphs. The maximum value of the sums of the hopcounts for simulated annealing routing is 25, while for epidemic routing it is 4500. We notice that the simulated annealing routing has way more outliers. Epidemic routing is also dropping and growing, however, it does so over time. The outliers from simulated annealing are much more spontaneous. All in all, the hopcount looks different. The smaller hopcount of simulated annealing routing may indicate a higher rate of delivered messages to a destination or a higher dropping rate. Both of these cases might be the reason for having fewer messages with long life time. The sum of hopcounts depends on the count of sent messages per minute. We look at it next.



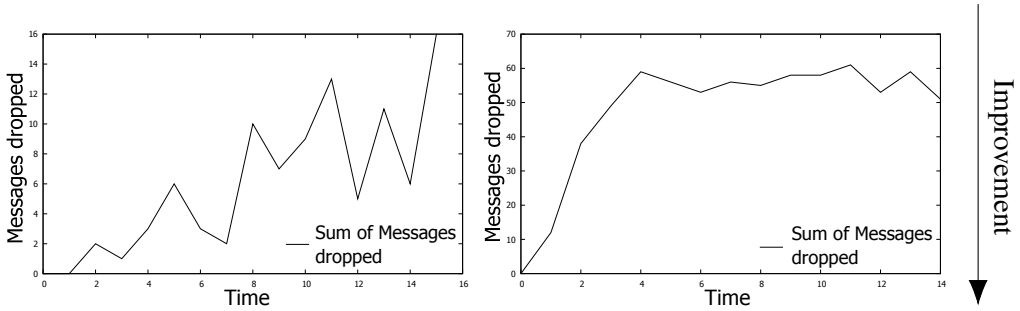
(a) Forwarded messages using simulated annealing routing (b) Forwarded messages using epidemic routing

Figure 5.6.: Forwarded messages comparing simulated annealing and epidemic routing, scale on y-axes vary, arrow on right indicates direction of improvement

Forwarded messages

Here we discuss fig. 5.6. The graphs once again look different. Simulated annealing routing has more outliers than epidemic routing. We assume that this is caused by the smaller count of forwarded messages from simulated annealing. The same outliers are then visible much better. The second difference is that the simulated annealing routing sends 25 messages per minute. In contrast to this, epidemic routing sends approximately 600 messages per minute. Like in optimization routing and reinforcement learning, this could be due to the reason that not every message is sent directly. The receiver is not overloaded and has more time to forward it. When a sender sends all of his messages directly, the receiver can only handle a few of them. But the sender marks it as already sent.

Dropping rate of messages



(a) Dropped messages from simulated annealing routing (b) Dropped messages from epidemic routing

Figure 5.7.: Dropped messages comparing simulated annealing and epidemic routing, scale on y-axes vary, arrow on right indicates direction of improvement

The dropping rate, shown in fig. 5.7, looks different. The only similarity is that the range is a little bit closer than in the previous evaluations. However, while the dropping rate of simulated annealing is growing over time and reaches its maximum during minute 15, epidemic routing is growing much faster at the beginning up to value 60 and then remains around that value. The maximum value from simulated annealing is 16, while the maximum value from epidemic routing is about 60. We notice that simulated annealing has a higher rate of outliers, which might be caused by the smaller range.

Received messages at destination

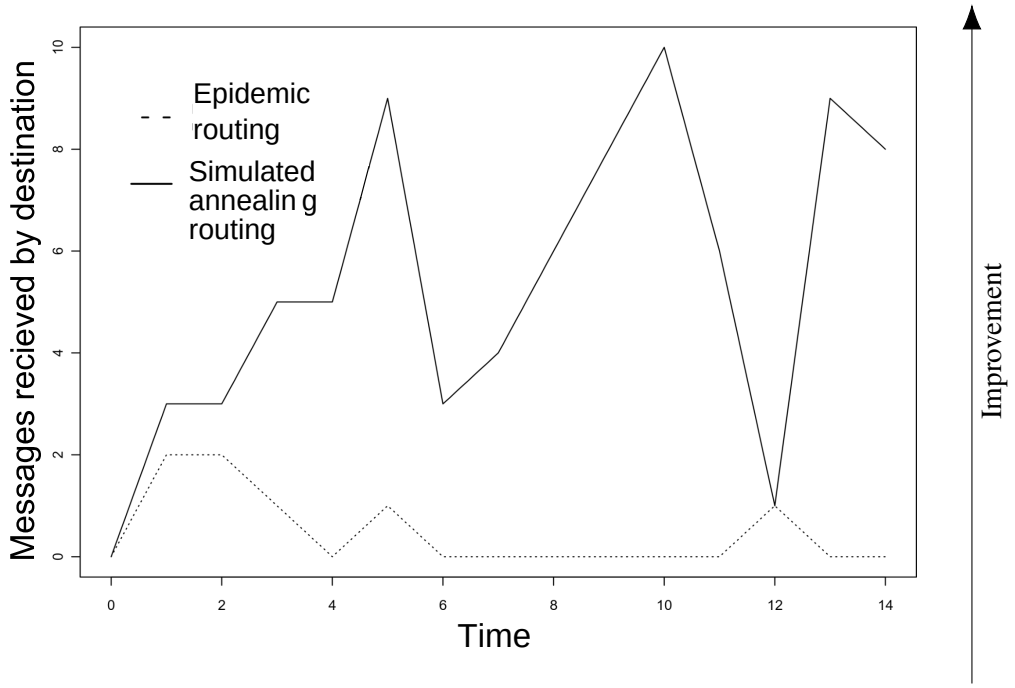


Figure 5.8.: Received messages comparing simulated annealing and epidemic routing, arrow on right indicates direction of improvement

Finally, we discuss the delivery rate from fig. 5.8. Our impression is that each time simulated annealing routing delivers more messages to the destination than epidemic routing. The maximum value of received messages per minute for simulated annealing routing is 15. We have already discussed the reason for sending more messages. For epidemic routing, the maximum value is two.

Up to now, we have evaluated a single simulation run. In the following section, we look at the variations among 1000 simulation runs for the network attributes of case one from table 4.2 using multiple seeds.

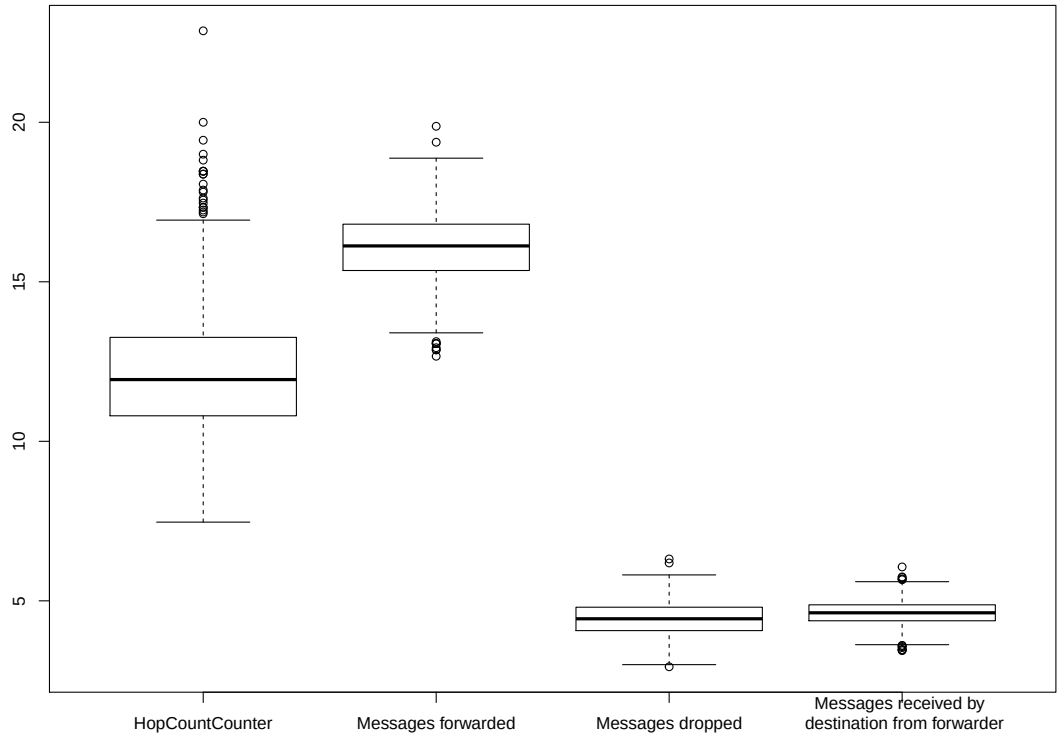


Figure 5.9.: Evaluation of Simulated Annealing Routing for 1000 simulation runs

In fig. 5.9 we look at the boxplots of 1000 simulation runs using a logarithmic y-axis scaling. The median of the hop count is around twelve. The lower quartile is around eleven and the minimum value is around seven. Furthermore, the upper quartile is around 13 and the maximum value is around 17. There are a few outliers above the maximum. The median for the count of forwarded messages is approximately 17. We notice that the lower and upper quartiles are at the distance of one from the median. The maximum and minimum values are about four points away from the median. There are also outliers above and below the median, but they are close to the maximum and minimum values. On the boxplot of dropped messages, median as well as upper and lower quartiles are close to each other and are around value four. The maximum and minimum values are three points away from the median. There are outliers that are close to the maximum and minimum. Finally, there is the metric of messages received by destination from a forwarder. This boxplot looks like the most focused one. The median has a value of four and is close to the upper and lower quartiles. The maximum and minimum values are two points above and below the upper and lower quartiles respectively. There are outliers next to the maximum and minimum values.

5.5. Conclusion

In conclusion, the generated results are surprising and show that there are benefits of using simulated annealing. It has a slow dropping and hopcount rates. On the other hand, it always delivers more messages to the destination than epidemic routing. In addition, the running time is the same for the both approaches, because all the attributes were generated in the previous part. Thus, there are no arguments for not using simulated annealing after we have generated the optimum one time. On the other hand, we have to come up with a good logic for the attributes W_1, W_2, and W_3. After this, generating the maximum takes a lot of time. Here it has taken two days to run only the simulated annealing algorithm.

For global point of this thesis we found a second routing protocol based on global information which is better than epidemic routing. In next chapter, we try to find a third one. At the end of this work, we compare each of them.

Chapter 6.

Reinforcement-learning

Over the last couple of years, the field of reinforcement learning has grown significantly. The paper of [BL94] demonstrates that the practical task of routing packets through a communication network is a natural application for reinforcement learning algorithms. Their "Q-routing" algorithm is related to certain distributed packet routing algorithms as in [Rud76] and [Tan89]. It learns a routing policy that minimizes the number of "hops", which a packet will take. It does so by experimenting with different routing policies and gathering statistics about decisions that minimize total delivery time. The learning is continuous and online, and it uses only network connection patterns and load.

The experiments in [BL94] were carried out using a discrete event simulator to model the transmission of packets through a local area network and are described in detail in [LB93].

In the previous chapters, we always have run multiple simulations to find an optimized routing policy. In this way we designed two algorithms to improve results against epidemic routing. Now we perform optimization inside a simulation run. A learned model is only specified for one simulation run. In the following chapter, we try to maximize the forwarded messages to their destinations. Again, the goal of this section is the finding of an algorithm which performs better than epidemic routing. For implementation, we use the package tensorflow, [ten17], developed by Google.

6.1. Routing

The routing algorithm is similar to the one from chapter 4. The only difference is the way we are choosing the number of neighbors we want to send the message to. This is discussed in section 6.2.4.

6.2. Reinforcement learning algorithm

In this section, we discuss reinforcement learning and the used algorithm.

6.2.1. The learning models

We have three models of the reinforcement learning algorithm. Each of them has its own complexity level. The easiest one is shown in fig. 6.1. It saves only the reward for every action. There is no link between states of the network and actions/rewards. This model is good for scenarios, if there is no connection between input states and the parameter for optimization. The input status can be for example the count of dropped messages. The parameter for optimization can be for example the variance between the count of forwarded messages from a specific minute and the mean of forwarded messages from all minutes. If we already know that there is no connection between state and reward, it is better to use no state, like in fig. 6.1. Our action is the number of neighbors we want to send a message to, like in chapter 4. The reward is the amount of the messages sent to destination for the minute when the action takes place. It is one divided by the difference between the mean of forwarded messages and the count of forwarded messages from the minute the action took place.

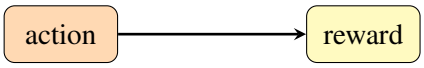


Figure 6.1.: Multi-action network

The next model is the one for using states: section 6.2.1. We want to use this model for our scenario of maximization forwarding messages. Our state is composed of two variables:

- The count of neighbors.
- Secondly, we give the algorithm the knowledge about the count of the already sent messages for the current minute.
- The count of all potential messages in network.

If there is no connection between this state and the equalization of forwarding messages, then the algorithm should give results that we would receive by using a random count of neighbors a message should be forwarded to.

The most complex model is shown in figure section section 6.2.1. A new state will be calculated in this model using an action. In our scenario, this means we have to return the next state from the

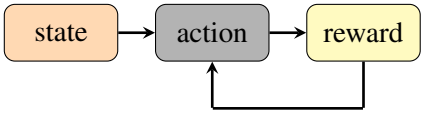


Figure 6.2.: Contextual Agents

simulator. Then the reinforcement algorithm calculates the reward on its own. Because the returning of the next state is more complex, we start with the second model. If it does not return good results, we have to try the first model. If it returns good results, then we can try the third model.

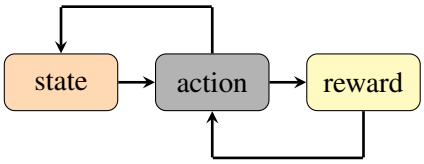


Figure 6.3.: Full RL Problem

6.2.2. The Contextual Routing

Here we define our contextual routing. In this scenario, we are using network states. This means that each network state consists of the following attributes:

- First, we take the number of neighbors that we can send the message to.
- Second, we use the count of messages which are already forwarded for a given minute.
- Finally, we want to use the potential messages that currently exist in the network.

Each state has different success probabilities for each number of neighbors to send a message, and as such requires different actions to obtain the best result. We want our agent to learn to always choose the number of neighbors that will most often give a positive reward, depending on the network state presented. Therefore we create a list *networkStates* of probabilities for all states. To match a given sensor to a probability of *networkStates* we create an array *mappingStates*, which contains all the network states itself. For doing the matching we write a function. It is given in section 6.2.2. The function returns the position of the given state in *networkStates*.

Each state has different success probabilities when sending a message to each neighbor, and as such requires different actions for obtaining the best result. We want our agent to learn to always choose the number of neighbors that will most often give a positive reward, depending on the present network state. For this purpose, we create a list of *networkStates* with probabilities for all states. In order to

match a given sensor to a probability of *networkStates*, we create an array of *mappingStates*, which contains all the network states. For doing the matching we write a function. It is given in section section 6.2.2. The function returns the position of the given state in *networkStates*.

Algorithm 1 Mapping a state to probability: *getNetworkStateIndex*()

```
for x in range {0,...,length(mappingStates)} do
    checkSensor = False
    for y in range {0,...,length(State)} do
        if mappingStates[x,y] ≠ sensor[y] then
            checkSensor = True
        end if
    end for
    if checkSensor = True then return x
    end if
end for
```

6.2.3. The Policy-Based Agent

We set up a simple neural network. Its input is the current network states and it returns an action for every request. In our case, it is the number of neighbors we want to give the message to. We are using a single set of weights. The policy gradient method is used by our agent. Thus, for a selected action the values are moved towards the received reward. This enables the agent to learn which actions are the best at the current state. This is the important step of our reinforcement learning problem. The code can be found in listing 1.

6.2.4. Training the Agent

The training of our agent is split in three steps:

- First, the agent obtains a state from the environment.
- Then the agent takes an action.
- Finally, the agent receives a reward and updates the model.

Using these three parts, we are able to update the networking with the following condition: given States, the agent chooses actions that will return the highest rewards over time. The pseudo code is given in section 6.2.4. We want to iterate over our code infinitely. After that, we iterate over inputs as long as we give only actions. When we receive a reward, we stop the inner loop. After this, the

algorithm expects an input from the simulator. Then we split the input. On the left side, we expect the type of the input, reward, sensor or printing of the routing table. In case of printing, we export all states to an excel file except the ones that were never used. This means that we check whether the rewards are not equal to zero. Next, we check if the given input is a reward. If yes, we extract the *action* and *reward* from the input and break the inner loop. Before starting a new loop and taking the next input, we run the reinforcement algorithm on *action* and *reward* and update the neural network. If the input is a sensor, we take from the agent the action for the given sensor and give it to the simulator.

Algorithm 2 Training the Agent

```
for True do
    reward = 0
    for True do
        input = get input from simulator
        if Check if "print" is in input then
            for Iterate the built table of rewards do
                if Check if line of table is not equal to zero then
                    Export line to excel
                end if
            end for
        else
            sensor = [ get type of input,
                get first parameter of input,
                get second parameter of input]
            s = getNetworkStateIndex(sensor))
            if Check if "reward" is in input then
                action = get third parameter of input
                reward = get fourth parameter of input
                break inner loop
            else if Check if "sensor" is in Input then
                action = take action from agent given the sensor
                give networkStates[s][action]) to simulator
            end if
        end if
        run reinforcement algorithm on action and reward and update the neuronal network
    end for
end for
```

6.3. Implementation

In this section, we discuss the implementation of reinforcement routing. This means that we show the way a network state is defined and the implementation of rewards.

We use a threshold to define the number of cases where we use the routing table of the reinforcement routing or we use a random number. The routing table or random number returns the number of neighbors we want to give a specific message to. If we take a result from the reinforcement routing table, we should not give a reward. Otherwise specific results are unjustifiably promoted. Therefore, we need the random generated sending probabilities.

In case we want to take a number from the routing table, then we create the network state using the following attributes:

- First, we take the number of neighbors that we can possibly send the message to.
- Second, we use the count of messages that are already forwarded for a given minute.
- Finally, we want to use the potential messages that currently exist in the network.

In the following part, we look closer at the implementation. Like discussed above we consider two cases. The first case is a probabilistic generated result, which indicates a learning for the reinforcement algorithm. For this purpose, a list containing all sensors of randomly generated actions and the generated action is generated. We call this list *sensorList*. Otherwise, we consider the case of a given result by the reinforcement algorithm. The logic is shown in section 6.3.

Algorithm 3 RL-returns

```
random = get random number in {0,...,100}
if random > randomThreshold and currentTime > timeThreshold then
    return Get a reward from reinforcement algorithm, given the sensor
else
    Action = random number in {0,...,countOfNeighbors}
    sensorList append action
    return action
end if
```

Next, we look at sensors and actions that have to be given as a reward to the reinforcement learning algorithm. Therefore, we iterate over all items in *sensorList*. If the difference between the current simulator time and the time of a generated sensor + action in *sensorList* is greater or equal to one, a reward for the entry is given to the reinforcement algorithm. We create a list named *alreadyLearnedList* containing all sensors that have already been evaluated. We check if the new sensor has already been evaluated for the simulation time during which the sensor is generated. If not, we give the sensor + action to the reinforcement learning algorithm, remove the sensor from *sensorList* and add it to *alreadyLearnedList*. The algorithm is shown in section 6.3.

First, we check if the attributes have already been evaluated for the minute. If it has not already been

Algorithm 4 RL-learning

Inputs

Given is the sensor built by number of neighbors, already send messages and count of potential messages in network.

End

for *sensor* in *sensorList* **do**
  if *currentSimulatorTime* − *sensorTime* >= 1 **then**
    alreadyExist = *false*
    for *alreadylearned* in *alreadyLearnedList* **do**
      if *alreadylearned*(*Time*) = *sensor*(*Time*)
        and *alreadylearned*(*CountOfNeighbors*) = *sensor*(*CountOfNeighbors*)
        and *AlreadyLearned*(*CountOfAlreadyForwardeMessages*) = *sensor*(*CountOf AlreadyForwardeMessages*)
        and *AlreadyLearned*(*CountOfPotentialMessages*) = *sensor*(*CountOfPotentialMessages*) **then**
        alreadyExist = *true*
      end if
    end for
    if not *alreadyExist* **then**
      reward = get count of forwarded messages to destination for *sensor*(*Time*)
      alreadyLearnedList append [*sensorTime*, *sensor*(*CountOfNeighbors*), *sensor*(*CountOfAlreadyForwardeMessages*), *sensor*(*CountOfPotentialMessages*)]
      give reward to reinforcement learning algorithm (*sensor*, *reward*)
      remove sensor from *sensorList*
    else
      add sensor to list *toRemove*
    end if
  end if
end for

evaluated, then we calculate the reward by using the count of messages that were forwarded to the destination during the minute the entry was created at *sensorList*. After that, the reward is given to the reinforcement learning algorithm and the entry is added to the list of already learned sensors.

6.4. Validation

In the following chapter, we evaluate the reinforcement algorithm. For this purpose, we look at two subjects. First, we make some validations of the reinforcement algorithm. Second, we look at some outputs of the simulator.

In the following, we add some inputs to the reinforcement learning algorithm and discuss the happenings. In table 6.1 the first column represents the minute the line was printed.

Time (minute)	comment	Messages in	Forwards without to destination	potential messages	return	reward
13	Random	4	203	400	4	
13	get result	3	203	400	0	
13	get result	3	206	397	0	
13	Random	2	209	394	1	
13	give reward for minute 12	1	232	358	1	6
13	get result	3	210	396	0	
13	get result	3	213	393	0	
13	give reward for minute 12	3	237	359	3	6

Table 6.1.: Comparison optimization routing and epidemic routing

We look at some action from minute three. The second column contains three cases:

- Random: During the first minute in 10 percent of cases a random number is generated to make a decision about how many messages should be sent.
- Give reward: If a random number is used for making the decision, this number can be evaluated a minute later. The minute later the number of forwarded messages to destination is given to

the reinforcement algorithm.

- Get result: In general, after some time the routing table of the reinforcement algorithm should be filled more and more. Then the simulator can take results of this routing table.

Here we can see that the random coincidentally generated numbers are four, one and three. Two coincidentally generated numbers are evaluated in the reinforcement algorithm. They are one and three from minute twelve. The total count of forwarded messages to a destination is six at the 12th minute. The result of the routing table here is always zero.

Next, we take a look at the generated routing table.

Messages in	Forwards without to destination	potential messages	return
1 3	232 237	358 359	1 3

Table 6.2.: Comparison optimization routing and epidemic routing

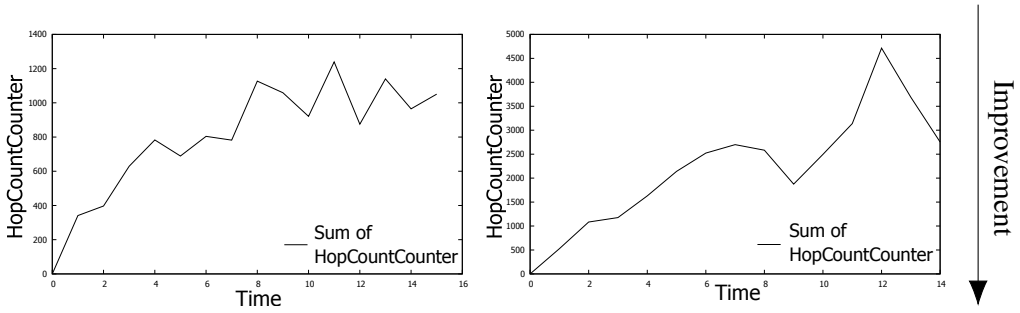
In table 6.2 we can look at an abstract of the routing table. The cases are evaluated at the cases of table 6.1. We notice that the evaluated results are marked as the best ones in the routing table. An additional check gives us feedback that there are no other cases that are evaluated with a different return. The complete routing table can be found in the attached CD and its extractions, in table A.4.

6.5. Evaluation

Since the goal of this work is finding better algorithms than epidemic, we evaluate the designed reinforcement algorithm in the following section. We focus on the scenario that we have already focused during the evaluation of optimization routing. We take the random scenario from table 4.3. Accordingly, we can compare both protocols in the concluding chapter of this work. The simulation is carried out for 15 minutes. Because the reinforcement algorithm is computationally intensive, this is the maximum we can afford with the available computing power. We use the metrics as discussed in section 3.3.

Hopcount

We notice that the hopcount values in the both graphs in fig. 6.4 are growing over time and remain high with some outliers. The vertical value range of the graphs is different. For reinforcement routing,

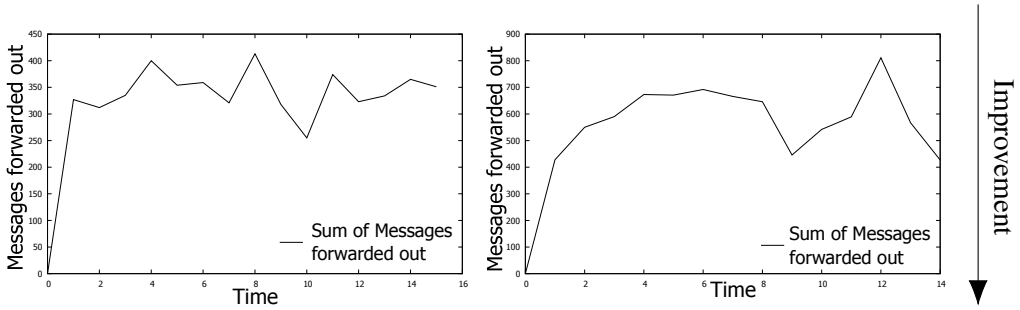


(a) Hopcount of reinforcement routing (b) Hopcount of epidemic routing

Figure 6.4.: Hopcount of Case 1, scale on y-axes vary,
arrow on right indicates direction of improvement

the maximum value of the sum of hop counts is 1200, while for epidemic routing it is 4500. In both graphs, the maximum value is reached after about twelve minutes. The hop counts look similar, except the fact that the hop count values for epidemic routing are generally higher than for reinforcement routing. The smaller hop count of reinforcement routing might indicate a higher rate of delivered messages to a destination or a higher dropping rate. Either of these cases can be the reason there are fewer messages with a long lifetime. The sum of the hop counts depends on the count of sent messages per minute. Next, we look at the metric of forwarded messages.

Forwarded messages



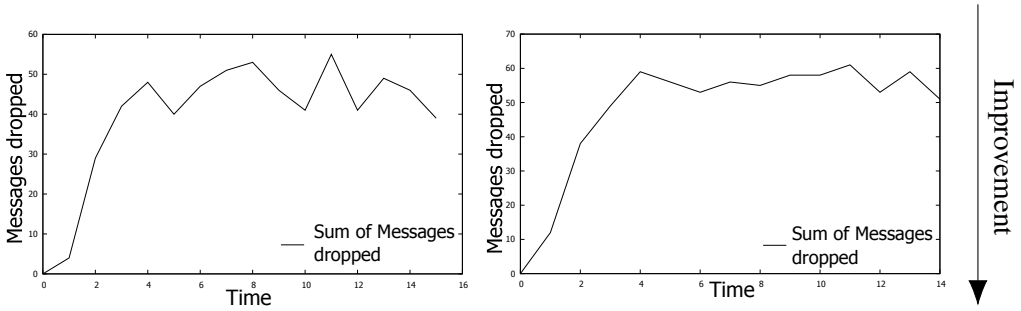
(a) Forwarded messages using reinforcement routing (b) Forwarded messages using epidemic routing

Figure 6.5.: Forwarded messages of Case 1, scale on y-axes vary,
arrow on right indicates direction of improvement

Here, we discuss the graphs from fig. 6.5. These graphs look similar. Reinforcement learning has more outliers than epidemic routing. The difference in outliers might be caused by the randomness factor used in reinforcement routing. Ten percent of the decision of how many neighbors should be recipients of the message is generated coincidentally. Another difference is that reinforcement

routing is capable of sending 300 messages per minute, while epidemic routing sends approximately 600 messages per minute. Like in optimization routing this might be due to the reason that not every message is sent directly. The receiver is not overloaded and has a little bit more time to forward it. When a sender sends all of his messages directly, the receiver can only handle a few of them. But the sender marks them as already sent.

Dropping rate of messages



(a) Dropped messages on reinforcement routing (b) Dropped messages on epidemic routing

Figure 6.6.: Dropped messages at Case 1, scale on y-axes vary, arrow on right indicates direction of improvement

The dropping rate is shown in fig. 6.6. Here once again the graphs look similar. We notice again that reinforcement learning has a higher rate of outliers, which might be caused by the randomness factor of the algorithm. In addition, epidemic routing has about ten drops more per minute.

Received messages at destination

Finally, we discuss the delivery rate. We can look at it in fig. 6.7. The general impression is that reinforcement routing always delivers more messages to a destination than epidemic routing. The maximum amount of received messages in a minute for reinforcement routing is 15, which occurs during the first minute. Epidemic routing forwards no message to a destination during the first minute. This could be due to the reason that reinforcement routing is not sending every message immediately. There are many situations, where reinforcement routing goes through the buffer and tries to send the messages to the destination. We have already discussed the reason for sending more messages in reinforcement learning. For epidemic routing, the maximum is five.

In this section, we do not make an evaluation of multiple simulation runs, since reinforcement learning requires a lot of running time.

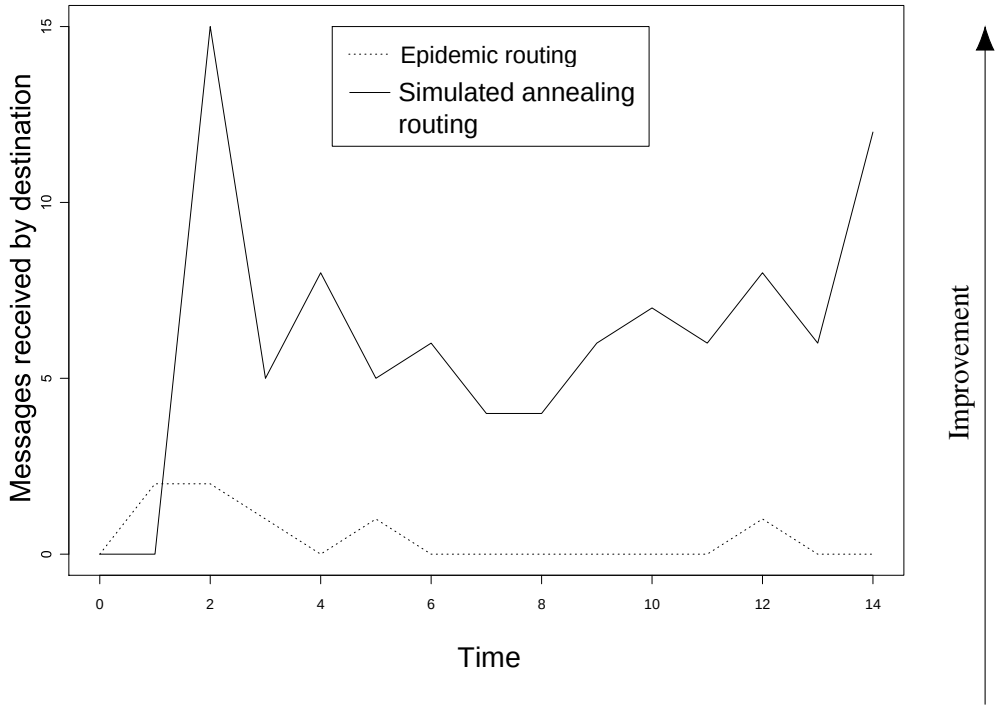


Figure 6.7.: Received messages at Case 1, scale on y-axes vary, arrow on right indicates direction of improvement

6.6. Conclusion

In this chapter, we have discussed the implementation of a reinforcement learning algorithm for creating a routing table. Because of this as well as for preparation of a prototype more introduction was necessary. We were able to achieve good results with this prototype. The number of received messages per minute is about 5 more than in the other methods. We did not analyze using boxplots because the required computing power is too high. In conclusion, reinforcement learning constantly performs much better regarding all considered benchmark criteria.

Finally we found a third algorithm that use global information and performs better than epidemic routing.

Chapter 7.

Conclusion and future work

Since opportunistic networks seem to fit to a large topic of real world problems on the one hand, on the other hand routing protocols have to solve the difficult issue of finding a routing path which exists probably only in future or never as full connected path. Since the routing using no information was able to be improved by using local information, the strong suspicion develops that further improvements can be achieved through the use of global information. We designed three algorithms to create routing tables based on a sending probability. These algorithms are using global network information. In this chapter, we discuss the results of this work. We will also look at future work in this field.

7.1. Conclusion

We already set the design options optimization routing, simulated annealing routing and reinforcement routing against outputs of epidemic routing in this thesis. The comparison against each other is done in this chapter.

We began with optimization routing. The comparison to epidemic routing can be improved in all cases in terms of every discussed metric, especially forwarded messages to a destination. However, the improvement was in the range of 0-10 received messages by destinations. This improvement was only caused by the fact that we did not send every message immediately when it was possible. No information or attributes were used from the network. Thus, it was comparatively simple to set this routing up. We can recommend this routing for the cases when the setup needs to be fast and the routing should be a little better than flooding.

In chapter 5, in which the simulated annealing algorithm was explored, the outputs were significantly better than the ones from epidemic routing. But the reason for that was not the simulated annealing algorithm. For the routing the construction of W_1, W_2 and W_3 like in equations 5.1, 5.2 and 5.3,

was much more important. Simulated annealing is like the optimization from chapter 4. Simulated annealing is good for giving more exact optima. Thus, it was helpful to use it here. We have shown that the structures of W_1, W_2, and W_3 are helpful to increase the delivery rate.

Reinforcement routing was able to achieve much better results than epidemic routing. In the case discussed in this work and in all other test cases reinforcement learning always generated better results than epidemic routing. Thus, reinforcement routing is better than optimization routing in terms of the considered metrics. The disadvantage is that reinforcement routing needs a lot of time to set it up. Time of setting up is made of implementing the reinforcement learning algorithm and training the learning model, and therefore the available routing table. For the purposes of this work we set up a contextual network and used three discreet parameters.

In case one has to choose between these two approaches we point out the general difference between reinforcement routing and simulated annealing or optimization routing. We have seen that the simulated annealing approach should be preferred to optimization routing if constructing the routing table can be done without time pressure or limited computing capacity. Otherwise, optimization routing should be used. After the routing table is done, both routing protocols are equally fast. Reinforcement learning routing should be used if a good delivery rate is needed and computing capacity is high.

Routing algorithm	**Effectiveness**	**Efficiency**	**Time**	
			Set up	Routing
Optimization routing	**+**	**-**	**+**	**+**
Simulated annealing routing	**+**	**+**	**-**	**+**
Reinforcement learning routing	**+**	**+**	**-**	**-**

Table 7.1.: Comparison design options of this thesis

In 7.1 the comparison between design options is summarized. Design criteria of this work are efficiency and effectiveness. In terms of effectiveness we discuss the criteria of received messages by destination. For discussion of efficiency we look at rate of forwarded messages, hop count and dropping rate of a message. All of the three design options reached nearly the same effectiveness which is better than the one of epidemic routing. The efficiency of optimization routing exceeds the one of epidemic routing, however it still has potential for improvement. Simulated annealing routing and reinforcement learning routing achieve similar efficiency which is much better than the one of optimization routing. The time factor is divided in time of finding the optimized routing table and time of execution of adaptive routing. Optimization routing is set up in a short time and the execution is in $O(1)$. The set up for simulated annealing takes a long time, however it is fast in execution ($O(1)$). The reinforcement learning algorithm is time consuming in implementation and execution.

We have seen several approaches for increasing the delivery rate. For our next work, we would focus more on one approach. Here it was not clear which approach had enough potential for a whole work. Retrospective reinforcement learning would have the most potential. The expansion of the defined structure in chapter 5 would also have enough potential.

7.2. Future Work

In the following chapter are future works for the three design options of this thesis presented. In in future project a larger reinforcement learning model could be set up and a comparison between different software libraries can be done. Next new structures like W_1, W_2 and W_3 can be defined. At least simulated annealing can be used without structures like W_1, W_2 and W_3.

The most prioritized future work is an implementation of a full reinforcement learning problem, as indicated in chapter 6. This would improve the learning process. On the other hand, the network state could be significantly expanded. In this work, the network state was only represented by three types of information. In general, all network and host attributes should be represented in a network state. This includes, for example, the location of all hosts and the buffer status. Then the learning algorithm would have much more possibilities to learn the best routing strategies. Thus, in our opinion, the section of reinforcement routing has the highest potential for future work. The reason for the high prioritization is the large disadvantage of reinforcement learning in this work: Large computational complexity. But this disadvantage can be avoid by set a initialization of the learning or routing table. Therefore an initial simulation run has to be run, like in optimization and simulated annealing routing. The resulting routing table is saved and can be used for every routing next time. In reinforcement routing this routing table is continuously updated. The frequency of update can be manipulated by setting variable *randomThreshold*. A random number and thus also a computational complex updating of the routing table is done often if *randomThreshold* is large. Goal of future work could be to combine the advantages of low computational complexity of optimization routing and simulated annealing at running time and the good results, in terms of count of forwarded messages, of the reinforcement learning implementation of this work.

Next we look at future work regarding the simulation annealing routing, explored in chapter 5. We found some structures for calculating a sending probability. In future works, other structures could be found as well. For example, for decreasing the standard division of deviation rate. The network is much more calculable, which can be helpful for increasing the performance of applications. Another example could be decreasing power usage, with second prioritization. The found structures in this work achieve second best results of this work and there are no direct disadvantages to fix in future work. Therefore finding new structures is prioritized with the second highest significance. The goal

of future work would be to adapt the mechanism to improve other attributes like delivery rate in this work.

The optimization, which was done in chapter 4, could also be done using simulated annealing. There can be found a better optimum for sending probability. This could result in another performance optimization based on delivery rate. Additionally, optimization can be run for more cases, like host numbers up to 1000. Because of limited computing capacity, we only ran it for a few scenarios. Therefore future work build larger routing tables. Additionally, a goal of feature work could be to remove the structures W_1, W_2 and W_3 and run the simulated annealing algorithm again.

All in all the work has shown there are a lot of improvements possible in terms of opportunistic networks. Since the computational complex part can be done in previous for optimization and simulated annealing routing, the protocols can be easy adapted to real world scenarios. This is also valid for the reinforcement learning routing, since the training can be done remote and updates are send after doing the model training or hard coded time units. Therefore I expect some future work in which the algorithms of this work are adapted to some projects used in real world.

Appendices

Appendix A.

Routing tables

htl	size	hosts	percent	optimization	flooding
2	75	5	69	1.88	1.38
2	75	15	9	5.92	1.72
2	75	25	39	9.04	1.64
2	75	35	89	12.15	1.96
2	75	45	9	16.20	1.72
2	85	5	89	1.69	1.00
2	85	15	29	5.68	2.16
2	85	25	79	8.52	2.08
2	85	35	9	11.36	1.77
2	85	45	9	14.60	1.96
2	95	5	79	1.64	0.92
2	95	15	29	5.56	1.76
2	95	25	9	8.35	1.85
2	95	35	9	10.64	2.20
2	95	45	9	13.80	1.69
2	105	5	79	1.28	1.12
2	105	15	99	4.38	1.28
2	105	25	9	7.60	1.88
2	105	35	9	10.00	1.31
2	105	45	9	11.58	1.84
2	115	5	49	1.12	1.00
2	115	15	19	4.80	1.62
2	115	25	9	6.96	1.72
2	115	35	9	9.88	1.88
2	115	45	9	11.58	1.84
6	75	5	49	2.40	1.48
6	75	15	9	7.92	2.32
6	75	25	9	12.65	2.12
6	75	35	9	16.08	2.12
6	75	45	39	20.24	2.12
6	85	5	89	2.23	1.44
6	85	15	39	7.24	2.60
6	85	25	9	11.64	1.76
6	85	35	19	14.92	2.08
6	85	45	19	19.00	2.12
6	95	5	99	1.88	1.27
6	95	15	59	6.04	2.46
6	95	25	9	10.85	2.65
6	95	35	9	14.44	1.60
6	95	45	9	17.16	2.32
6	105	5	99	1.84	0.96
6	105	15	9	6.64	2.24
6	105	25	9	9.32	2.00
6	105	35	9	12.08	1.92
6	105	45	9	15.76	1.72
6	115	5	79	1.35	0.96
6	115	15	29	5.28	2.50
6	115	25	9	8.88	1.81
6	115	35	19	11.19	2.62
6	115	45	9	14.32	2.40

Table A.1.: Comparison optimization routing and epidemic routing - Part 1

htl	size	hosts	percent	optimization	flooding
10	75	5	49	2.00	1.27
10	75	15	49	8.16	2.12
10	75	25	19	12.52	2.40
10	75	35	19	16.77	2.15
10	75	45	9	21.12	2.32
10	85	5	69	2.12	1.31
10	85	15	19	7.92	2.40
10	85	25	9	11.32	1.92
10	85	35	9	16.00	2.92
10	85	45	9	18.88	1.88
10	95	5	59	1.84	1.12
10	95	15	49	6.52	2.04
10	95	25	9	11.72	2.12
10	95	35	9	14.23	2.20
10	95	45	9	16.46	2.04
10	105	5	79	1.62	1.15
10	105	15	29	5.92	1.56
10	105	25	9	10.32	2.19
10	105	35	9	12.36	1.73
10	105	45	9	16.32	2.00
10	115	5	79	1.56	0.92
10	115	15	9	5.36	2.16
10	115	25	9	9.04	2.20
10	115	35	9	11.44	2.19
10	115	45	9	14.88	1.96
14	75	5	89	2.23	1.50
14	75	15	9	7.88	1.76
14	75	25	19	13.20	1.84
14	75	35	9	17.62	2.68
14	75	45	9	21.50	1.84
14	85	5	89	2.00	1.32
14	85	15	39	7.68	1.62
14	85	25	9	11.80	2.32
14	85	35	9	15.20	2.56
14	85	45	9	18.32	2.24
14	95	5	99	1.92	1.19
14	95	15	29	6.40	1.52
14	95	25	9	10.52	1.92
14	95	35	9	14.76	2.16
14	95	45	9	16.84	2.36
14	105	5	89	1.56	1.27
14	105	15	19	6.16	2.04
14	105	25	9	10.24	2.23
14	105	35	9	13.84	2.08
14	105	45	9	16.92	1.73
14	115	5	69	1.35	1.35
14	115	15	9	5.80	1.69
14	115	25	9	9.48	2.23
14	115	35	9	11.52	2.44
14	115	45	19	13.48	2.12

Table A.2.: Comparison optimization routing and epidemic routing - Part 2

htl	size	hosts	percent	optimization	flooding
18	75	5	49	2.54	1.24
18	75	15	9	8.88	2.12
18	75	25	9	12.68	2.19
18	75	35	9	17.08	2.04
18	75	45	9	21.44	1.92
18	85	5	49	2.08	1.27
18	85	15	19	7.72	2.32
18	85	25	9	12.16	2.20
18	85	35	59	14.56	1.88
18	85	45	9	18.16	2.00
18	95	5	89	1.92	0.77
18	95	15	49	6.48	1.84
18	95	25	9	10.64	2.08
18	95	35	9	15.00	2.08
18	95	45	9	17.80	2.40
18	105	5	89	1.48	1.12
18	105	15	9	6.32	1.88
18	105	25	9	9.85	2.27
18	105	35	9	13.00	1.80
18	105	45	9	15.54	2.58
18	115	5	79	1.42	0.76
18	115	15	29	5.56	2.23
18	115	25	9	8.72	1.84
18	115	35	19	11.36	2.04
18	115	45	9	14.08	2.31

Table A.3.: Comparison optimization routing and epidemic routing - Part 3

Count of neighbors	Count of already forwarded messages	Count of potential messages	action
1	0	1	1
1	0	2	0
1	1	2	0
1	2	5	1
1	2	6	0
1	3	7	0
1	4	7	1
1	4	10	0
1	5	11	1
1	8	12	1
1	5	13	1
2	7	13	1
2	5	14	0
1	8	14	0
2	9	14	0
1	11	16	1
2	11	19	0
1	26	26	1
1	28	27	0
1	29	27	1
1	24	28	1
3	24	28	1
2	27	28	1
3	26	29	2
1	29	29	0
2	20	31	2
4	21	31	1
5	20	32	4
1	32	33	1
2	30	34	0
1	36	34	1
2	51	34	0
4	16	35	0
1	36	36	0
1	37	36	1
3	49	36	3
4	49	36	2
1	56	37	1
7	13	38	4
5	32	38	1
3	43	38	3
5	47	38	3
2	54	38	0
3	53	39	2
5	41	40	3
2	43	40	2
1	46	40	0
3	56	40	3
5	39	42	3

Table A.4.: Routing table created by reinforcement learning - Part 1

Count of neighbors	Count of already forwarded messages	Count of potential messages	action
3	60	42	3
2	43	43	2
4	43	43	1
3	56	43	2
4	59	43	3
7	37	44	5
2	60	47	2
1	60	49	0
1	60	50	1
2	63	50	2
1	65	50	0
2	63	51	2
3	63	51	3
2	64	51	1
2	96	51	1
3	63	52	2
2	66	52	2
1	95	52	0
5	61	53	3
2	66	55	0
5	92	55	2
8	81	57	3
1	87	57	0
2	86	58	1
5	88	59	1
3	68	62	3
1	71	64	1
1	109	64	1
1	70	65	0
2	71	65	0
8	73	65	0
2	108	65	1
3	68	67	1
4	102	67	0
4	106	67	2
2	100	69	0
6	98	71	4
2	130	71	2
3	97	72	2
1	127	72	1
4	109	73	0
5	127	74	2
1	141	76	0
7	120	79	0
5	137	80	1
6	136	81	5
4	117	82	1
1	145	82	0
1	146	83	1

Table A.5.: Routing table created by reinforcement learning - Part 2

Count of neighbors	Count of already forwarded messages	Count of potential messages	action
4	142	85	1
8	113	86	4
4	132	87	0
3	151	90	2
2	147	91	2
3	146	92	2
4	147	94	0
4	159	97	4
1	154	99	1
2	152	101	0
4	155	101	0
1	159	101	1
2	154	102	1
6	162	105	2
1	161	106	0
3	159	108	1
1	166	110	1
10	166	122	10
1	168	125	0
2	166	127	0
3	169	129	3
6	30	131	6
2	176	136	1
1	24	137	0
2	169	137	0
1	175	137	0
3	174	138	2
5	22	139	3
4	26	139	0
2	173	139	1
4	172	140	3
8	173	140	6
1	316	140	1
3	171	141	2
2	12	142	0
2	171	142	2
3	314	142	1
1	11	143	0
4	18	143	0
3	170	143	0
1	10	144	0
1	242	144	0
7	253	144	0
2	241	145	1
2	5	146	2
1	25	146	0
11	167	146	8
2	277	146	1
5	7	147	2

Table A.6.: Routing table created by reinforcement learning - Part 3

Count of neighbors	Count of already forwarded messages	Count of potential messages	action
1	177	147	0
5	250	147	2
1	313	147	0
2	3	148	0
2	6	148	1
1	176	148	0
2	241	148	2
2	310	148	2
1	312	148	0
2	2	149	1
2	5	149	1
1	30	149	1
1	175	149	0
1	205	149	0
2	241	149	2
1	245	149	1
6	274	149	3
1	300	149	1
1	310	149	1
1	311	149	0
1	1	150	0
3	182	150	0
4	241	150	4
3	244	150	2
3	278	150	3
2	310	150	1
1	0	151	0
1	36	151	1
1	39	151	1
2	39	151	2
2	203	151	0
3	228	151	0
1	243	151	0
2	272	151	0
5	245	152	0
3	271	152	2
5	297	152	2
1	30	153	1
5	185	153	3
1	206	153	1
4	238	153	1
9	290	153	2
4	303	153	4
5	36	154	2
11	159	154	3
4	178	154	0
1	241	154	1
4	280	154	0
4	306	154	0

Table A.7.: Routing table created by reinforcement learning - Part 4

Appendix B.

Algorithm

Listing 1: The Policy-Based Agent

```
class agent():
   def __init__(self, lr, s_size,a_size):
      self.state_in= tf.placeholder(shape=[1],dtype=tf.int32)
      state_in_OH = slim.one_hot_encoding(self.state_in,s_size)
      output = slim.fully_connected(state_in_OH,a_size,
         biases_initializer=None,
         activation_fn=tf.nn.sigmoid,
         weights_initializer=tf.ones_initializer())
      self.output = tf.reshape(output,[-1])
      self.chosen_action = tf.argmin(self.output,0)
      self.reward_holder = tf.placeholder(shape=[1],dtype=tf.float32)
      self.action_holder = tf.placeholder(shape=[1],dtype=tf.int32)
      self.responsible_weight =
         tf.slice(self.output,self.action_holder,[1])
      self.loss = -(tf.log(self.responsible_weight)*self.reward_holder)
      optimizer = tf.train.GradientDescentOptimizer(learning_rate=lr)
      self.update = optimizer.minimize(self.loss)
```

Bibliography

[BL94] BOYAN, Justin A.; LITTMAN, Michael L.: Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In: *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, 1994, S. 671–678.

[Cer85] CERN'Y, V.: Thermodynamical approach to the traveling salesma problem: An efficient simulation algorithm. In: *Journal of optimization theory and applications* (1985).

[FTH16] FUNAI, Colin; TAPPARELLO, Cristiano; HEINZELMAN, Wendi B.: Supporting Multi-hop Device-to-Device Networks Through WiFi Direct Multi-group Networking. In: *CoRR* abs/1601.00028 (2016). `http://arxiv.org/abs/1601.00028`.

[HLC10] HUANG, T. K.; LEE, C. K.; CHEN, L. J.: PRoPHET+: An Adaptive PRoPHET-Based Routing Protocol for Opportunistic Network. In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010. ISSN 1550–445X, S. 112–119.

[Hol16] HOLZE, Marek: Implementation and Evaluation of an opportunistic Routing protocol in the simulation environment Peerfact Sim.Kom. (2016).

[Ipp15] IPPISCH, Andre: *A fully distributed Multilayer Framework for Opportunistic Networks as an Android Application*, Department of Computer Science, Heinrich Heine University Düsseldorf, Diplomarbeit, M^b arz 2015

[KGV+83] KIRKPATRICK, Scott; GELATT, C D.; VECCHI, Mario P. u. a.: Optimization by simulated annealing. In: *science* 220 (1983), Nr. 4598, S. 671–680.

[LB93] LITTMAN, Michael; BOYAN, Justin: A distributed reinforcement learning scheme for network routing. In: *Proceedings of the international workshop on applications of neural networks to telecommunications* Psychology Press, 1993, S. 45–51.

[LDS04] LINDGREN, Anders; DORIA, Avri; SCHELEN, Olov: Probabilistic routing in intermit-

tently connected networks. In: *Service assurance with partial and intermittent resources* (2004), S. 239–254.

[MP12] MITRA, Pramita; POELLABAUER, Christian: Emergency response in smartphone-based mobile ad-hoc networks. In: *Communications (ICC), 2012 IEEE International Conference on* IEEE, 2012, S. 6091–6095.

[pee17] *PEERFACTSIM.KOM - COMMUNITY EDITION - Simulator Details*. `http://peerfact.com/simulator-details/`. Version: June 2017.

[Rud76] RUDIN, Harry: On routing and "delta routing": A taxonomy and performance comparison of techniques for packet-switched networks. In: *IEEE Transactions on Communications* 24 (1976), Nr. 1, S. 43–59.

[SK12] SRIKRISHNA, Devabhaktuni; KRISHNAMOORTHY, Rajeev: SocialMesh: Can networks of meshed smartphones ensure public access to twitter during an attack? In: *IEEE Communications Magazine* 50 (2012), Nr. 6.

[T+03] TANENBAUM, Andrew S. u. a.: Computer networks, 4-th edition. In: *ed: Prentice Hall* (2003).

[Tan89] TANENBAUM, A.: Computer Networks. In: *Prentice-Hall, second edition* (1989).

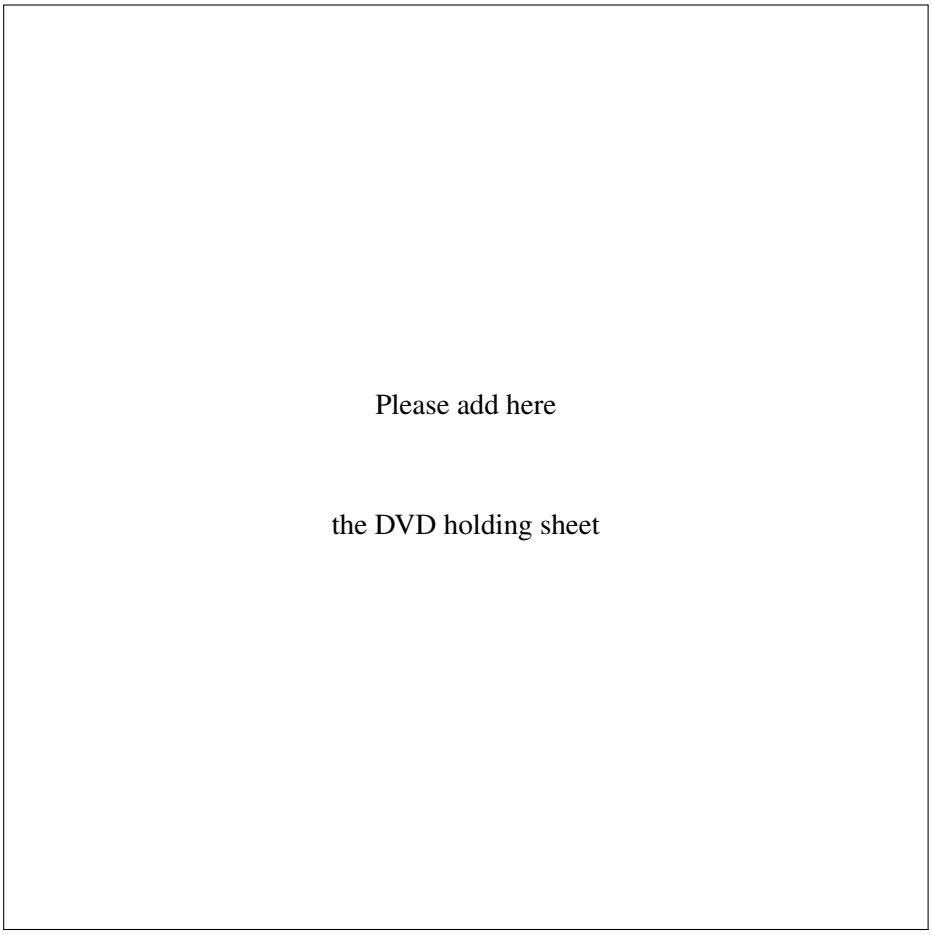
[ten17] *Tensorflow - official page*. `https://www.tensorflow.org/`. Version: October 2017.

[VB+00] VAHDAT, Amin; BECKER, David u. a.: Epidemic routing for partially connected ad hoc networks. (2000).

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit 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, 24.October 2017 Marek Holze



This DVD contains:

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