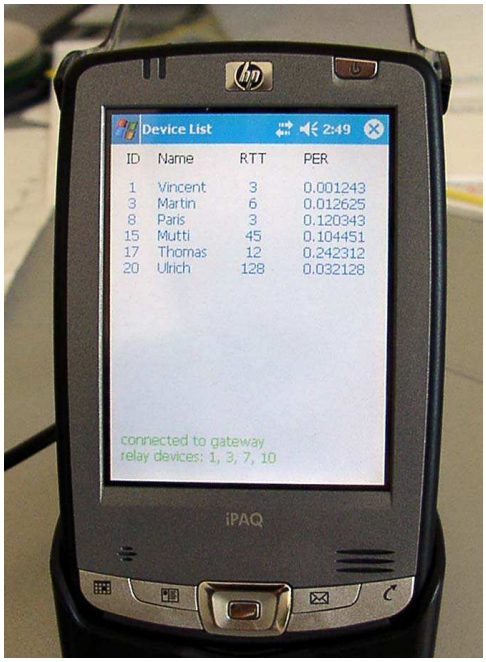
Analysis of Dynamics in Mobile Ad-Hoc Networks

Student Project



Student: Jörg Wagner
Advisors: Vincent Lenders, Martin May
Professor: Bernhard Plattner

31. July 2005

Task Formulation

Summary

Small and mobile devices such as handhelds (PDAs) with wireless communication capabilities (e.g. WLAN) are becoming more and more ubiquitous. Nowadays, the use of WLAN with such devices is mostly limited to downloading messages or surfing the Web via a fixed access point. Ad hoc networking is a promising communication paradigm where devices form a network in a dynamic and self-organizing manner without any fixed infrastructure support such as access points or dedicated routers. Therefore, communication is no more limited to fixed places controlled by network operators but becomes in principle possible anywhere as long as cooperative nodes are available in the neighborhood. In order to develop protocols for ad hoc networks, researchers have made a set of assumptions on the user mobility. However, very few people have validated these assumptions with real networks. The goal of this thesis is to empirically analyze the mobility behavior of users and the resulting network characteristics. For this purpose, the student should collect data from a testbed consisting of approximately 20 handhelds carried by mobile users . The first task of the student is to develop the required tools and programs to monitor the desired network characteristics. Then, the student should distribute the handhelds to test users (e.g. TIK members) and conduct the relevant measurements. When enough data is collected, the student should analyze the collected data and extract characteristics on relevant network metrics.

Tasks

- Make a survey of different developer platforms for Windows Mobile Edition 2003.
- Choose one platform and familiarize yourself with it
- Learn how to use the WLAN interface.
- Define a set of relevant network metrics that you want to assess.
- Implement the required tools to assess the metrics.
- Distribute the handhelds to test users and make the necessary measurements.
- Analyze the collected data and extract characteristics for the relevant network metrics.

Acknowledgement

I want to thank my supervisors Vincent Lenders and Martin May for their strong commitment, which compared to student fellows' experiences concerning supervision of student projects was out of the ordinary and posed additional motivation to me.

Jörg Wagner

Abstract

Performance of routing algorithms nowadays is mainly analyzed based on mobility models, such as the well-known random waypoint model. It is not clear, however, whether these models actually reflect real user behavior. Therefore data is collected from a real network and analyzed with respect to node degrees, path lengths, route stability under different routing metrics, and connectivity to a fixed gateway in this student project. The underlying testbed consists of 20 iPAQs connected through Wireless LAN in ad-hoc mode. The main contributions are two mathematical models on the issue of route stability enabling the separation of route interruptions due to user mobility and due to critical links. This separation also enables a comparison of data collected from the testbed with the outcome of simulations based on mobility models. It is demonstrated that with respect to connection time between nodes the random waypoint model shows good match with the empirical network data. Perfect match is achieved with the random reference point mobility model, which is one of the most general group mobility models.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless ad-hoc networking is a more and more emerging communication alternative to infrastructure networks and a major object of current research. Routing in such networks is of particular interest, as it is a much more sophisticated issue than in wired networks. A difficulty in this context is the degraded stability of routes, which is an immediate consequence of the mobility of the network nodes. Further the packet forwarding process itself causes much more costs than in wired networks, as the data cannot be directed to the next relay device by simply choosing the appropriate wire, but also constitutes traffic disturbing other nodes' channels.

So far experimental investigation of ad-hoc networks has mainly focused on static networks: In [1] different routing metrics for static ad hoc networks are investigated, [2] focuses on link level characteristics in such networks. The performance of routing protocols in mobile ad-hoc networks has mainly been investigated in terms of simulations based on mobility models, e.g. the popular random way point model. It is not clear, however, whether these models can actually yield reasonable matches with real networks.

In this thesis measurements are conducted in a mobile ad-hoc network consisting of 20 iPAQs equipped with 802.11b WLAN modules. The testbed is described in detail in the second chapter of the report. The measurements mainly aim to enable the reconstruction of the logical topology at discrete points of time as well as the according link levels in terms of packet error rates and round trip times. The received signal strength – although a value of interest in this context – could not be determined due to limitations of the WLAN module. In order to further determine connectivity to a backbone network the network is flooded by a virtual gateway node periodically.

The investigated network characteristics may be classified into two categories. First some general network properties such as the node degree distribution, the clustering coefficient and the average shortest path length, which actually completely characterize a static network (see [3]) are discussed. The second part of the evaluation is mainly about mobility specific characteristics afterwards, in particular about the issue of route stability. In this context two mathematical models are proposed which enable the separation of route interruptions due to user mobility and due to critical links. Further the empirical data is compared with the outcome of simu-

lations based on two common mobility models, namely the random waypoint model and the random reference point group mobility model.

The obtained results in general have to be evaluated against the background of the chosen experiment environment, which in this case is a research lab of ETH Zurich.

Chapter 2

Testbed

2.1 Measurement Application

2.1.1 Threads and Their Tasks

In this section the rough structure of the application running on the iPAQs for conducting the experiment is described. It consists – the graphical user interface not taken into account – of six threads, which are listed in the following.

The code for the key functions of these threads is shown in the appendix. The application essentially shall enable the reconstruction of the logical topology at discrete points of time as well as some link level measures, namely packet error rates and round-trip times. The received signal strength which also would be a parameter of interest is not measured due to technical limitations of the WLAN module. Actually signal strength values could be read from the network interface card, however, these turned out to be mixtures of several devices' signal contributions. In addition to the measurements of those parameters, which essentially are all measured by single-hop packet exchange between nodes, a single virtual gateway initiates packet flooding over the whole network periodically. If these packets are received by a certain node this is interpreted as the device having a connection to the gateway.

All network traffic generated by the application is handled by UDP sockets based on the *Winsock API*, which is supported by the eMbedded Visual C++ 4.0 IDE to a large extent.

In the following a short summary of the single threads and their tasks is listed:

- **Main-Thread**
  - Provides user interface for getting nickname.
  - Starts other threads.
  - Runs message loop for receiving messages from other threads and – based on these – serving the GUI threads.

- **Client-Thread**
  - Sends broadcast packets containing device ID, time stamp and packet index every 500 ms.
  - Scans for BSSIDs in order to reduce effect of BSSID partitioning (see section 2.1.3).
  - Asks the (possibly blocked) gateway thread, whether it has received any packets from the gateway within the last five seconds (necessary for being able to display whether a connection to gateway is currently available or not).
- **Server-Thread**
  - Receives broadcast packets from other devices' client-threads and writes their information to logfiles (new logfiles are created whenever the current one reaches a size of 1 MB).
  - Computes packet error rate.
  - Wakens the RTT-request-thread by informing it, which devices currently are in range and need to be addressed, every five seconds.
  - Sends messages to the main-thread every five seconds in order to update the display.
- **RTT-Request-Thread**
  - Receives message from server-thread with devices which are currently in range.
  - Sends (UDP) unicast packets (withouht RTS/CTS) with a time stamp to all devices listed by the server-thread (waits 200 ms between two such packets).
- **RTT-Acknowledge-Thread**
  - Receives request packets from other devices RTT-Request-Threads and immediately bounces them back.
  - Receives packets from other devices RTT-Acknowledge-Threads, gets current time, computes RTT and writes it to logfiles.
- **Gateway-Thread**
  - Receives packets from gateway or relays.
  - Adds the own device as relay in the packet, forwards (via broadcast) and logs it if it has not seen it yet.

The gateway device shares the first five threads with the applications running on the other devices. Its gateway thread has to be different of course. It just sends indexed broadcast packets every 2.5 seconds on the gateway.

The periodic broadcast packets send by the client-thread actually are redundant as all the information they contain could be retrieved from the MAC-layer beacon packets. Windows

Packet Type	Pkts per 5 sec
Client-Broadcasts	$10 \cdot D$
RTT-Request	D^2
RTT-Acknowledge	D^2
Gateway-Forward	$2 \cdot D$
Total	$2 \cdot D^2 + 12 \cdot D$

Table 2.1: Overall traffic generated by D devices within five seconds.

Mobile 2003, however, does not support raw sockets, which are necessary in order to have access to those beacon packets on transport layer.

2.1.2 Estimation of Caused Traffic

To estimate the overall traffic we have a look at an artifical scenario, where there are D devices all being in each other's ranges. In this case within five seconds the traffic in the air is as shown in Tab. (2.1.2).

Each packet contains 62 header bytes (24 802.11 MAC, 20 IP, 8 UDP), 48 bytes payload and thus 110 bytes in total. The transmission rate is 11 megabit per second. Assuming the worst case scenario that $D = 20$ this results in a time occupation of

$$\frac{1040 \frac{\text{packets}}{5\text{s}} \cdot 110 \frac{\text{bytes}}{\text{packet}} \cdot 8 \frac{\text{bits}}{\text{byte}}}{11 \cdot 10^6 \frac{\text{bits}}{\text{s}}} = 1.7\%. \tag{2.1}$$

Although beacon packets sent by the MAC layer are neglected in this rough estimation, there should not be a huge number of collisions even under that extreme condition.

2.1.3 BSSID Partitioning

When doing the first test runs for the handheld experiment the following problem emerged: Although two devices A and B were separated less than some meters they could not receive each other's packets. On the other hand each device A or B itself could receive packets from a couple of other nodes, which again were not seen by the other device B and A.

When sniffing packets with *tcpdump* on a notebook the problem became obvious: some packets, although being assigned to the same service set identifier (SSID), contained different basic service set identifiers (BSSID). The BSSID is a concept making lots of sense in infrastructure mode. If a wireless station is in the range of two access points with equal SSID, it clearly has to decide in favor of one of them. Therefore it only picks up packets containing the access point specific BSSID (which is related to its MAC-address) in their header.

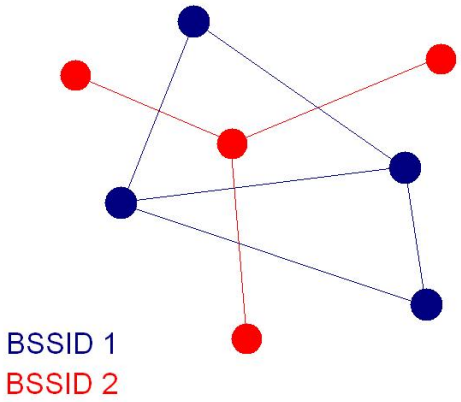


Figure 2.1: Illustration of BSSID partitioning.

Unfortunately this BSSID is of relevance even in WLAN networks driven in ad-hoc mode. Instead of the access points each network node itself is able to advertise a own BSSID to the other devices. Whenever two devices meet each other they agree on one common BSSID, which they again advertise to newly arriving devices in the following. If a single device now gets in range of this network it recognizes that there already exists an ad-hoc network and attaches to it without problems. If one wants to establish a static ad-hoc network BSSID partitioning therefore is not a major problem. As long as one just adds one device after another to the network, all nodes will finally share the same BSSID.

The actual problem emerges in mobile ad-hoc networks, whenever two single nodes meet outside the main network and finally get into its range simultaneously. Since the devices had met without any other network being available, they will have set up their own network specified by one of the BSSIDs they had advertised. In the moment they get in range of the main network therefore two independent networks collide. The problem emerging is that not only the devices in the 2-node-network are willing to attach to the larger network in that case, but even vice versa. Each node in both networks will recognize that there are two networks it could attach to, some will attach to the one, some to the other. Each device thus starts randomly hopping forth and back between the two nets. Without any handling it actually took up to half an hour until this process reached its steady state, when all devices finally were united in one network.

Although plugging a lot of time into handling that issue, I did not manage to completely get rid of it. However, I could reduce the time span until two colliding networks converge to a single one significantly by making use of the network device interface specification (NDIS) driver development library provided by Windows. With this utility I scanned for access points (which correspond to networks and their BSSIDs in ad-hoc mode) in parallel to the network card driver. Scanning for access points in this context means not only scanning, but when detecting

more than one BSSID with the desired SSID connecting to one of them (not necessarily the one with stronger signal strength as it turned out in some tests). If such a scan is followed by a change to a lower BSSID the application running on the device (unfortunately not the driver) stops scanning. As on the other devices, which have changed to or still are at the higher BSSID, still both, driver and application, are scanning, the probability that they switch to the lower BSSID is larger than the one that devices at the lower BSSID switch to the higher one.

Clearly this does not solve the BSSID partitioning problem. But in experiments it turned out, that the stabilization process that took up to half an hour previously was completed in less than two minutes in most cases afterwards.

Unfortunately this procedure does its job very well for two colliding networks only. For three or more colliding BSSIDs the convergence in one single network still takes a lot of time. This actually happened every morning, when people started to work at different ends of the floor and the network was unified by new devices arriving in the middle of them. I got rid of that problem, by starting evaluating the measured data not till that transient phase had been completed. However, in some cases there also were such periods up to 15 minutes, when groups of devices returned from the mensa after lunch, which actually influenced the measurements.

I actually wanted to register the number of networks with the same SSID, a device sees while scanning for access points. Unfortunately that plan failed due to a bug in my software.

When searching the internet I noticed, that there have been even other experiments conducted, where BSSID partitioning was an issue. When MIT people set up their "Roofnet", a wireless 802.11b multihop network (see [2]), they kept exchanging the network cards until they finally found some that do not suffer from that problem. Several cards from Senao finally turned out to work fine.

2.2 Experiment Setup

In this section the experiment setup is described in detail. The network being investigated in this thesis consisted of 20 nodes (HP iPAQs hx2200) – 19 of these nodes were mobile and carried by members of a research group of ETH Zurich and some students doing student projects with that group. The 20th one was fixed and considered as a virtual gateway to a backbone network, e.g. the internet. The network was based on 802.11b WLAN technology and organized in ad-hoc mode. The users spent most of the time at their desks. They mainly moved for going to the lavatory, having lunch, taking coffee breaks, picking up printouts or meeting each other for discussions. While some participants took along their devices wherever they went, some others now and then forgot the iPAQs in their office. Although those people where encouraged not to do that, I do not regard this as major falsification source of my measurements, since in a real world application this would be their natural behavior.

The experiment ran for five days within one week from 10 am to 5 pm in each case. While the virtual gateway was always active during that period, the mobile nodes where allowed to leave

the floor. In particular some participants started to work later than 10 am, finished earlier than 5 pm or left for meetings outside ETH. During lunch devices where taken along to the mensa, where they often formed a new independent network.

Fig. (2.2) shows a site plan of the floor the experiment was conducted on. The inserted numbers indicate the locations where the corresponding nodes lingered most of the time, i.e., the desks of their users. Four devices changed their user during the week, as some became sick, had two days off resp. where abroad over more than one day due to conferences. In these cases the devices where redistributed among other group members in order not to shrink the long-term network size. Two devices thereby had to be moved to different offices (depicted in orange in Fig. (2.2)).

As mentioned previously I did not manage to completely get rid of BSSID partitioning. The problem arising with that difficulty was grave in particular, when more than two independent networks met each other. This of course was the case in the morning, when people started to work at different ends of the floor and the network was unified by a new device arriving in the middle of both. I got rid of that problem, by starting evaluating the measured data not till that transient phase had been completed. However, in some cases there also were such periods up to 15 minutes, when groups of devices returned from the mensa after lunch, which actually influenced the measurements. Further there were some short adaption periods up to two minutes for unifying two meeting networks (e.g. when a pair of users returned from a coffee break) throughout the whole day.

2.3 Evaluation Methodology

Before actually being evaluated in Matlab the measured data somehow had to be converted from their raw version in the logfiles into a representation Matlab can deal with. Further I had to remove the time shifts which emerged due to the enormous clock drifts in the iPAQs (more than a second during one day). The corresponding tool I developed in C++ mainly consisted of three parts.

The first part managed to figure out and remove the time shifts between the device clocks. This is done by investigating a devices logfile and comparing the temporal order of packet receptions and the assigned origination time.

The second part translated the pure logfiles in actual network topologies $G(V, E)$ with vertices V and edges E. These are represented in $N \times N$ adjacency matrices A, with N the number of iPAQs denoted as vertices $u_1 \dots u_N$ in use, where

$$a_{ij} = \begin{cases} 1 & \text{if } u_i u_j \in E \\ 0 & \text{else.} \end{cases} \tag{2.2}$$

Note that this is not an adjacency matrix in a mathematically strict sense, since it is not

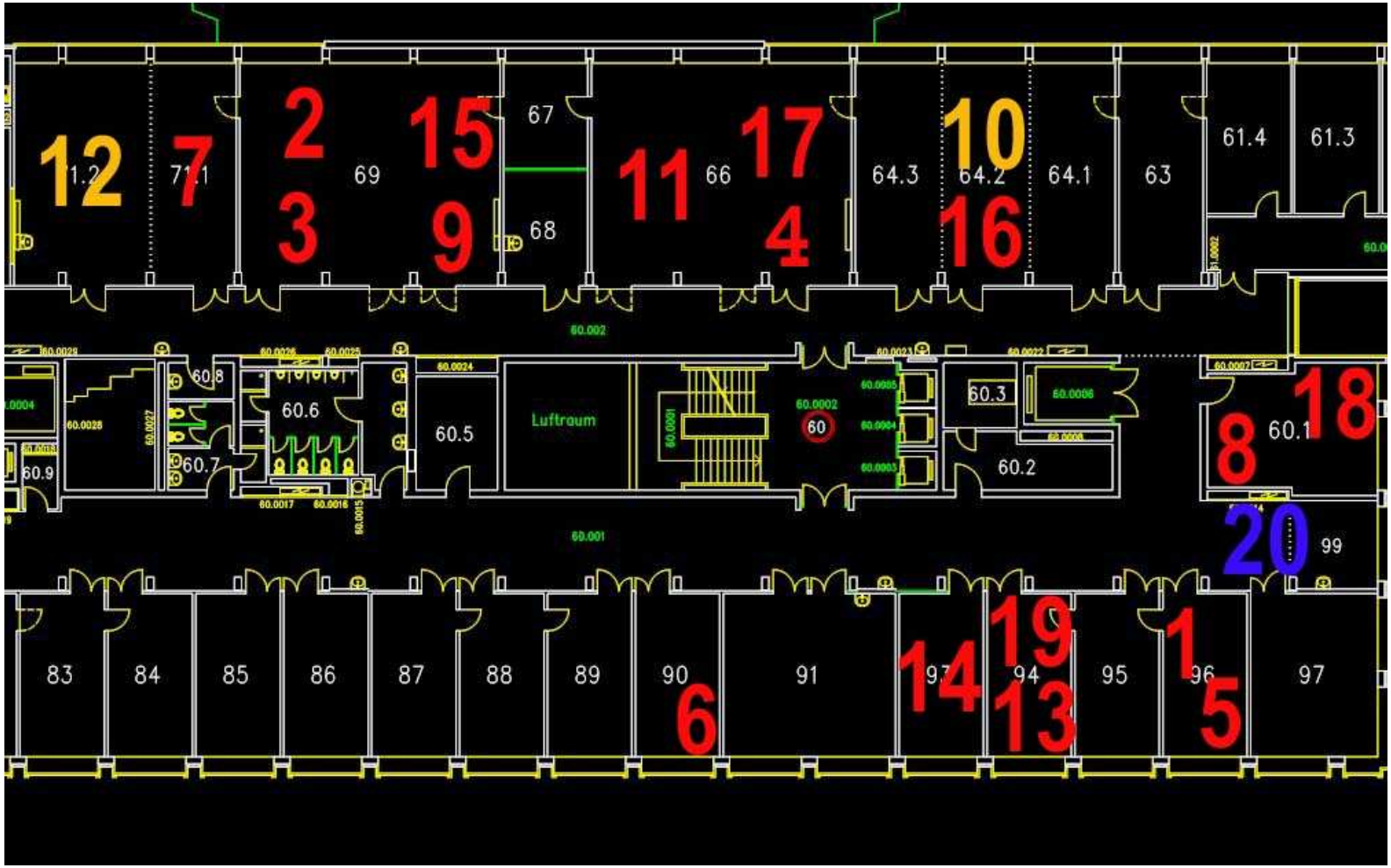


Figure 2.2: Floor plan of G-floor in the ETZ building. Numbers indicate the single users' offices. Virtual gateway is shown in blue, devices with changing users are shown in orange.

symmetric necessarily. This is as there might occur scenarios, where one device sees another one, but not vice versa, e.g. if one of them has lots of hidden nodes around it and the other one has not.

The topology is sampled every seven seconds. This is to make sure that at least one RTT scan has been completed within that period. Every sample is represented in an element of a linked list. Each such element contains

- The according topology in an adjacency matrix A.
- An identically organized matrix P containing the packet error rates between each pair of nodes.
- Another such matrix R containing the RTT between each pair of nodes, as well as a matrix that indicates whether a round trip time was actually measured or copied from the last sample due to a current packet loss (necessary in order not to copy RTTs more than once).

For detailed descriptions on how a link between two devices is defined, and how packet error rate and round trip time are computed see section (3.3).

The third task of the analysis tool finally is to extract some information out of the topology, that can be interpreted in Matlab afterwards. In particular the tool extracts

- The distribution of the number of single-hop-neighbors each single device has over all samples. For device i the number of single-hop-neighbors $n_{\mathrm{s},i}$ in the current sample can be found as
$$n_{\mathrm{s},i} = \sum_{j=1}^{N} a_{ij}. \tag{2.3}$$
- *The distribution of the number of multi-hop-neighbors each single device has over all samples:* This is done by constructing spanning trees (one per subnet) over the network by applying the breadth-first-algorithm. A slight inaccuracy emerges in this context as A is assumed symmetric for that purpose.
- *The contact and inter-contact times between the devices:* This is done by considering the difference between two successive topology samples $\delta A = A_i - A_{i+1}$. If a $\delta a_{ij} = -1$ the end of a contact is detected and logged and the according inter-contact time counter reset to 0, if a $\delta a_{ij} = 1$ a new contact is detected, the inter-contact time logged and the according contact time counter reset to 0. If finally a $\delta a_{ij} = 0$ both contact and inter-contact time counter for that device pair are incremented by one.
- *The path lengths between all device pairs under different routing metrics:* These can be found by applying Dijkstra's Algorithm, where edges in the network graph are assigned a certain weight depending on the metric of interest.

- *The time spans over which the routes are stable under these metrics depending on their lengths:* This is done by taking a found route and traversing the linked list as long as all links the route consists of are stable. Further also the time a route is actually the optimal one with respect to a particular metric is registered.

The extracted data resp. the matrices and vectors finally are transferred to Matlab. This is done by establishing a pipe to the Matlab process via the Matlab Engine API.

Chapter 3

Results

In this chapter the various results – both extracted from the measurements and from mathematical models – are discussed. The chapter is subdivided in three sections. First some general network characteristics, such as node degree distribution, path lengths and gateway connectivity are evaluated. This section shall mainly give some intuition on the network structure. The second section contains the key contributions of this thesis, namely two mathematical models which enable the separation of route interruptions due to user mobility and critical links. The proposed procedures are applied to the data collected from the testbed. The outcome of one of them afterwards is compared to two mobility models. Finally the performances of three routing metrics with respect to stability are compared in the third section.

3.1 General Network Characteristics

Complex (static) networks are often assumed to be reasonably characterized by three parameters, namely the degree distribution of their nodes, the clustering coefficient and the average shortest-path length [4]. These and some additional characteristics are discussed in the following subsections.

3.1.1 Degree Distribution and Clustering Coefficient

The degree of a node in a network graph is equal to the number of its immediate neighbors k, i.e., those nodes which are reachable within one hop. We look at the node degree distribution not in terms of the distribution within a particular realization of a topology, but average over all realized topologies.

In random graphs all nodes are assumed to follow the same distribution law. In real mobile networks this is not the case of course, but strongly depending on the behavior of the corresponding user. This is confirmed in Fig. (3.1), where degree distributions for all 20 nodes are shown. By considering centrally located nodes such as device no. 1 (refer to Fig. (2.2)

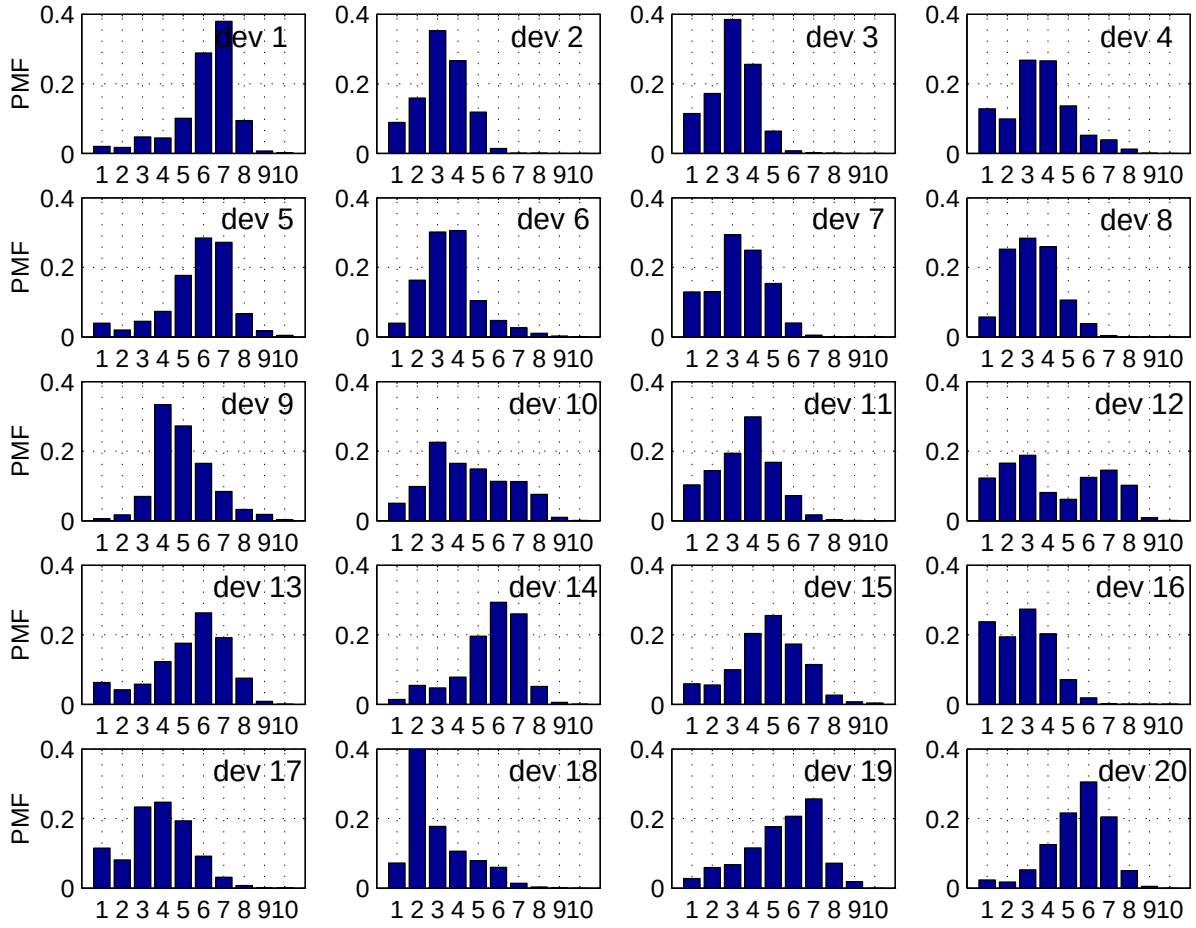


Figure 3.1: PMF of number of single-hop-neighbors for each single device.

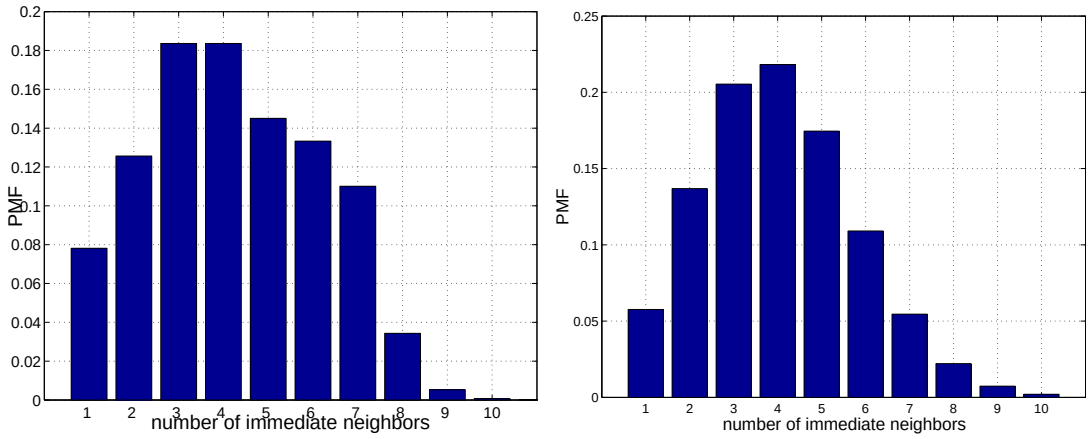


Figure 3.2: PMF of number of single-hop-neighbors empirical (averaged over all devices) (left) and for a random network with connection probability 0.2 (right).

for comparison) one recognizes that these have more neighbors than nodes often located at the network boundary such as device no. 2. If one nevertheless wants to consider the degree distribution averaged over all nodes one ends up with the distribution shown in Fig. (3.2). The peak of the distribution is somewhere between three and four neighbors, its mean is 4.25. In a random graph with N nodes and connection probability p the degree k_i of node i would follow a binomial distribution [4] according to

$$\Pr(K_i = k) = \binom{N-1}{k} \cdot p^k \cdot (1-p)^{N-1-k}; \tag{3.1}$$

The PMF of a corresponding distribution with $p = \mathsf{E}[K]/N = 4.25/20 = 0.213$ is plotted for comparison and shows a surprisingly good match with the empirical data.

As the clustering coefficient is defined as $C = \mathsf{E}[K]/N$ this p is also equal to the clustering coefficient.

3.1.2 Multihop-Neighbors and Shortest Path Routes

Let's consider the distributions of multi-hop neighbors l_i the single devices have next. The according PMFs are shown in Fig. (3.3). One can observe that several nodes (1,5,6,8,12,13,14,18,19 and 20) have a very strong peak at $l = 9$. This is as the link between devices 8 resp. 18 and 16 (refer to Fig. (2.2) for orientation) turned out to be very lossy. Therefore during a lot of time there was a separation of the network at this transition. As the network was very dense in the region from device 8 to device 6, this subnet usually was not further subdivided and consequently all devices being contained have had the same number of multi-hop neighbors at a time. One might ask, why there is no such characteristic peak in the distributions of the remaining nodes. This is as the second large subnet by far was not that dense on the one hand

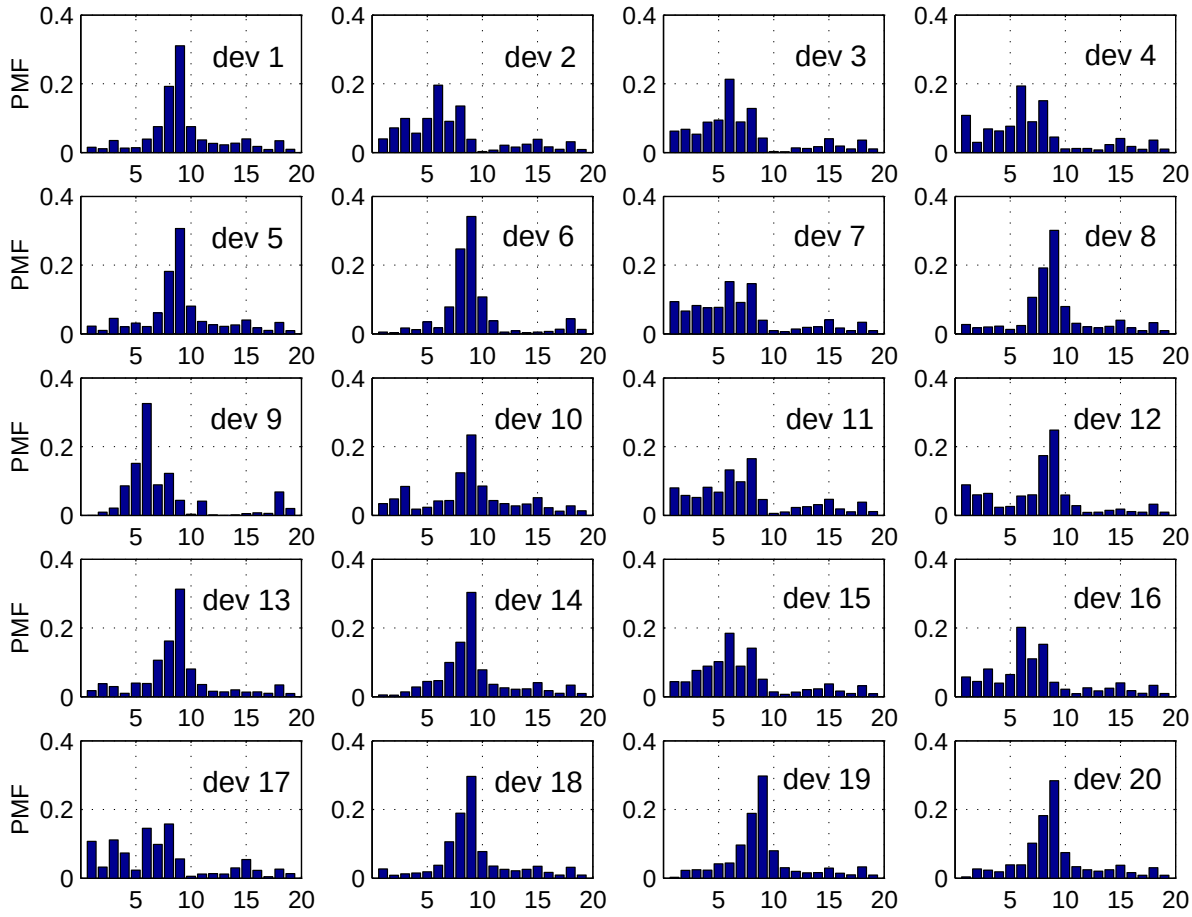


Figure 3.3: PMF of number of multi-hop-neighbors for each single device.

and secondly the users on this floor site (students and support group) where much more mobile than the ones on the other side of the building (PhD students).

Again we consider the distribution averaged over all devices. This of course is dominated by the peak of the devices of the large subnet. The course to the left and the right follows no particular shape and is rather uniform. When regarding the probability of all devices being united in one subnet one has to take into account that most of the time at least one device was not around due to absence of its user.

When regarding the distribution of multi-hop neighbors another point of interest is the distribution of the shortest paths between the devices. It is shown in Fig. (3.4). The average shortest path length is 2.03 hops. While single-hop pathes are most likely, there are nevertheless (shortest) pathes with up to eight hops between source and destination node. Such a shortest path might have existed between device 6 and device 12, e.g.. Keep this distribution

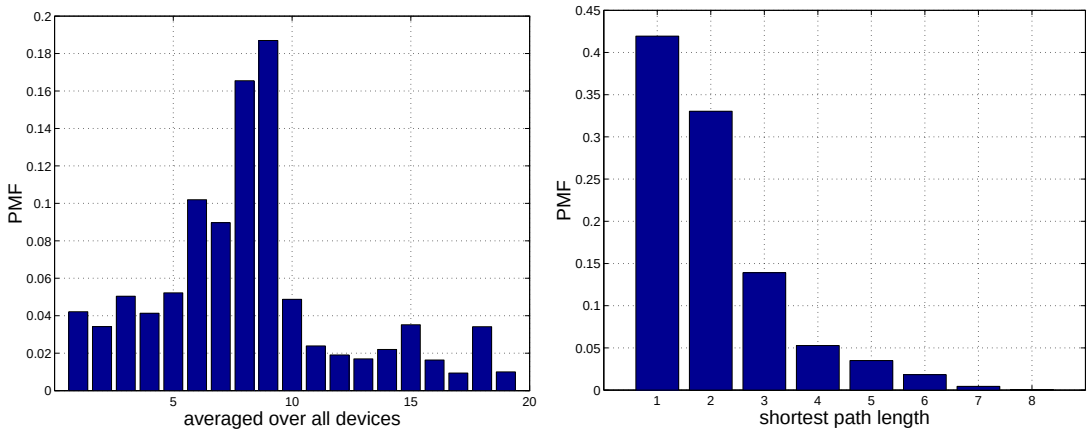


Figure 3.4: PMF of number of multi-hop-neighbors averaged over all devices and distribution of shortest path between any two devices.

in mind when later regarding the stability of routes over several hops. It will turn out namely, that the stability worsens drastically for increasing numbers of hops. The (in this case rather short) radius of the network consequently is very important measure for the design of routing algorithms.

3.1.3 Gateway Connectivity

As mentioned in chapter (2.1) device no. 20, which was considered as a virtual gateway sent broadcast packets every 2.5 seconds, which then spread over the whole network by being flooded. This experiment is subject of investigation in this subsection.

Clearly the number of hops it takes a packet to arrive at a particular destination device again strongly depends on the location of the users's office. This can be seen from Fig. (3.5). While devices 1, 5, 8 and 18, whose users's offices are next to the gateway are reached with one hop throughout most of the time, devices mainly located at the other end of the floor such as 2, 3 and 7 often are reached over four, five or six hops. Devices 5 and 19 originally were located at the upper floor side, however, spent a considerable amount of time on the lower side as well. Therefore their characteristics show two peaks – one at one resp. two hops and one at four hops.

Let's have a look at the number packets the single nodes have received and further the number of packets received by other devices, which have been relayed by these nodes. Note that the latter number can be larger than the first one, since a single relayed packet, which is received by more than one node in the following is counted as being relayed more than once only. By considering Fig. (3.6) one can see that the devices on the lower floor side are connected to the gateway much more often than the ones on the other side. This is – as discussed in previous

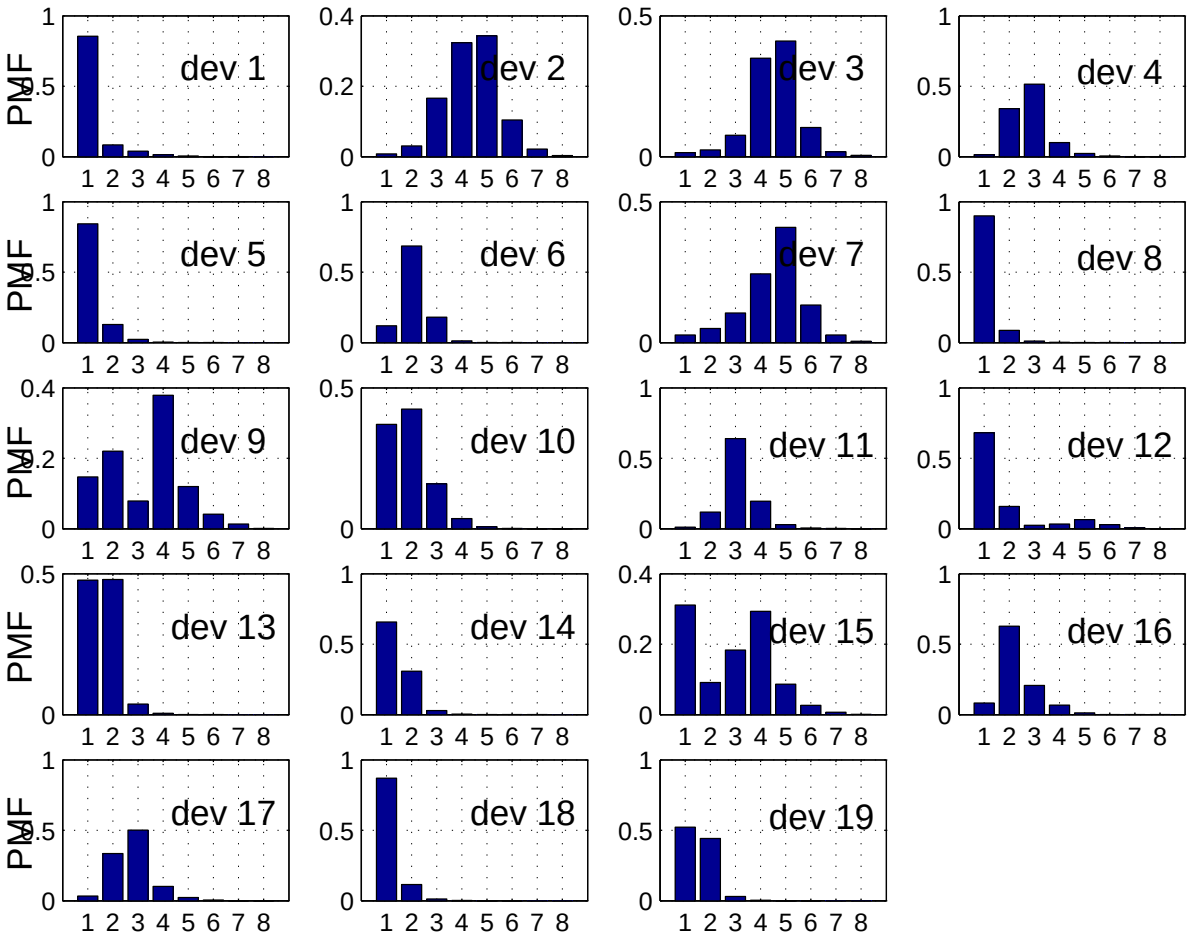


Figure 3.5: Distribution of distance to gateway (in hops) for each single device.

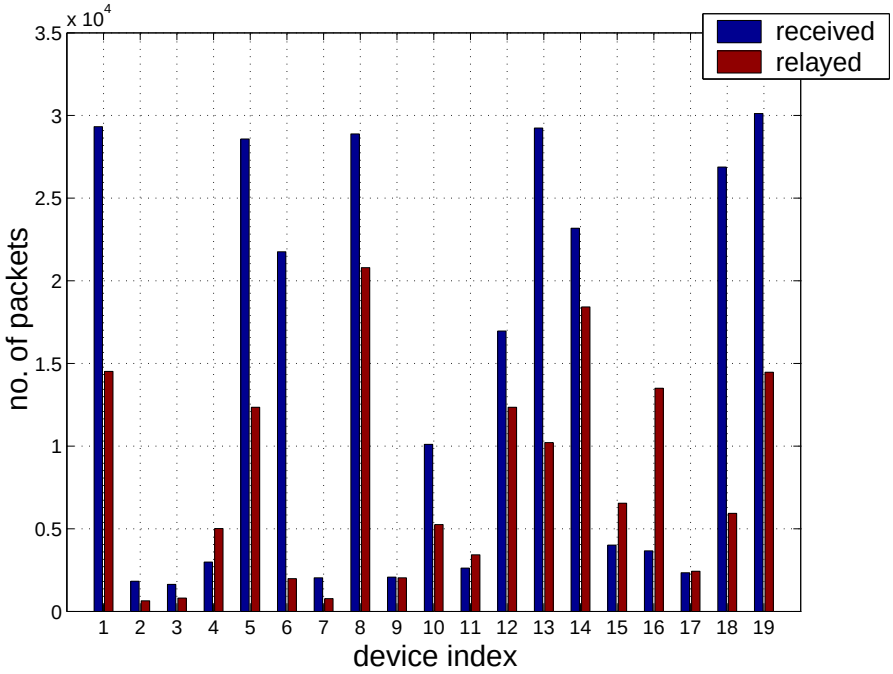


Figure 3.6: Number of received (blue) and forwarded (red) gateway packets.

sections – due to the lossy link between device 8 and 16 and due to the larger user mobility on the upper floor side.

The importance of the single nodes for the spanning of the whole network becomes obvious by comparing the number of packets a device has received and the number of packets which reach other devices after having been relayed by the device. Device 16 seems to be a key device in this context. Each received packet is forwarded to more than one further device on average. This is as the node can be considered as kind of an entry point to the subnet on the upper side of the floor. Devices 4, 11 and 15, which show similar characteristics, are also very important for spanning a large network as they serve as major relays for covering the not so dense upper floor side. A device whose importance cannot be seen from this representation is device 8, which serves as a "bridge" between gateway and upper floor side. This is as it receives much more packets than it can actually forward to device 16 due to the lossy link. Its large absolute value of forwarded packets indicates its importance nevertheless.

Indeed one can find a better representation based on which the importance of the single devices for the overall range coverage can be estimated. Therefore one considers the probability that device x_i has served as a relay, if device x_j receives a packet from the gateway along the route $\mathcal{X}_j$. The proposed importance measure consequently is

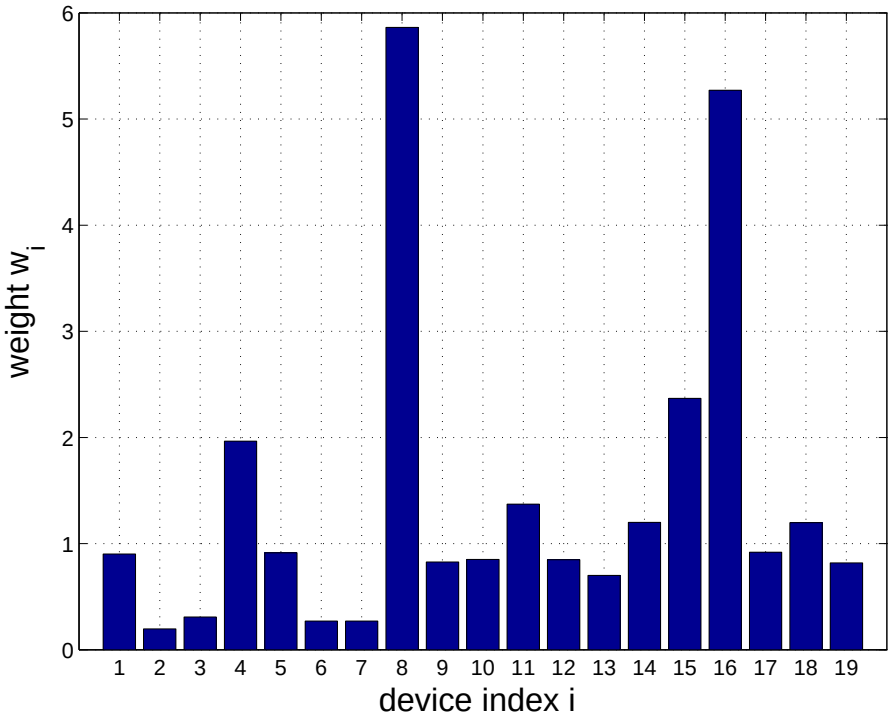


Figure 3.7: Importance of single devices for network covering.

$$w_i = \sum_{j=1}^{N-1} \Pr(\mathcal{X}_j \ni x_i). \tag{3.2}$$

The corresponding plot for all devices is shown in Fig. (3.7). Here the devices 8 and 16 are the most important ones, what finally matches the expectation.

3.2 Breaking Routes – Two Fundamental Causes and Their Separation

This section forms the core of this thesis and contains the key contributions. It is subdivided in several subsections for reasons of clearness. Each subsection is dedicated to one major step in the chain of thoughts.

3.2.1 Outline

There are two reasons causing an interruption of a route in a wireless mobile ad-hoc network. The first one is a link error due to a heavily loaded, deeply faded or strongly noisy channel between two source, relay or destination devices (denoted as interruption event I in the following). A second interruption event is immediately related to the mobility in a mobile ad-hoc network, namely a route interruption due to one or more moving nodes (denoted as interruption event II in the following). These two interruption events have different implications on the design of routing protocols. Just one example: While it might be promising to reuse a route that temporarily was not available due to lots of traffic or a deep fade, a route which was broken due to movement of its user might hardly be recovered in near future. If it was possible to reasonably estimate, whether a route was interrupted by event I or II one could exploit this information in the design of a routing protocol.

In this section a statistical model for the stability of routes in dependence of their length is introduced first. Based on this model a procedure is proposed to reconstruct two separated density functions of residual link lifetimes for interruption event I or II from empirically measured residual route lifetimes. Applied to our data set it will yield two clearly separated regions of residual link lifetime – each one strongly dominated by one of the two interruption events. The extraction of the influence of the user mobility will further enable a fair comparison of the so gathered data with the outcome of simulations based on various mobility models. In this context the random waypoint model and the random reference point group mobility model are considered. Finally the two types of route interruptions are investigated from a different angle based on temporal correlation in the topology, which might be rather suitable for online estimations.

3.2.2 A Word about Notation

To follow the subsequent derivations it is important to be aware of slight, but very important differences in notions.

When talking about route stability one has to differentiate whether this is done in terms of *lifetime* of the route or its *residual lifetime* at a certain time instance when the route already has reached a certain *age*. While the last two terms are taken one to one from renewal theory, lifetime actually relates to the inter-arrival time. As this notion is not very intuitive in the context of route stability it has been replaced here.

A *link* denotes a connection between two immediate neighbors, i.e., two devices that are separated by no more than one hop. A *route* consists of one or several links and denotes a connection between an arbitrary pair of nodes.

With this in mind a list of all random variables considered in the following discussions is given as a reference in Tab. (3.1). All random variables are meant to describe the (residual) lifetime of a route picked at a random instance of time. This means that we consider the routes from the angle of a reactive routing protocol, when it discovers a new route.

RV	Description
$T_{\text{rlt,r}}$	residual lifetime of a route
$T_{\text{lt,r}}$	lifetime of a route
$T_{\text{rlt,l}}$	residual lifetime of a link
$T_{\text{lt,l}}$	lifetime of a link
$T_{\text{rlt,l,e}}$	residual lifetime of a link in absence of mobility
$T_{\text{lt,l,e}}$	lifetime of a link in absence of mobility
$T_{\text{rlt,l,o}}$	residual lifetime of a link in absence of noise/fades/collisions
$T_{\text{lt,l,o}}$	lifetime of a link in absence of noise/fades/collisions

Table 3.1: Summary of relevant random variables.

3.2.3 Statistical Model

The probability of a link interruption between two nodes within the time interval t caused by errors due to collisions, fading and noise is denoted by $p_{\text{e}}(t)$. The probability that a node moves within the interval in a way that it causes a link interruption is denoted by $p_{\text{o}}(t)$. The according probability for a link interruption then is $2 \cdot p_{\text{o}}(t) - p_{\text{o}}(t)^2$. Consequently we can state the CDFs of $T_{\text{rlt,l,e}}$ and $T_{\text{rlt,l,o}}$ for links between two mobile nodes as follows:

$$\Pr(T_{\text{rlt,l,e}} < t) = p_{\text{e}}(t) \tag{3.3}$$

$$\Pr(T_{\text{rlt,l,o}} < t) = 2 \cdot p_o(t) - p_o(t)^2. \tag{3.4}$$

These are the quantities of interest that shall be derived in the course of this section, as they form CDFs for $T_{\text{rlt,l,e}}$ and $T_{\text{rlt,l,o}}$. The task of estimating those CDFs will be accessed through first estimating $p_{\text{e}}(t)$ and $p_{\text{o}}(t)$. The underlying mathematical model, which describes the stability of a route in dependence of its length, is introduced in the following.

Basic Model

The overall probability that a route is stable over a particular range in time in a mobile wireless ad-hoc network assuming

- $T_{\text{rlt,l,e}} \perp T_{\text{rlt,l,o}}$
- IID moving nodes
- IID distributed link qualities

can be written as

$$\Pr(T_{\text{rlt,r}} > t) = (1 - p_{\text{e}}(t))^N \cdot (1 - p_{\text{o}}(t))^{N+1}, \tag{3.5}$$

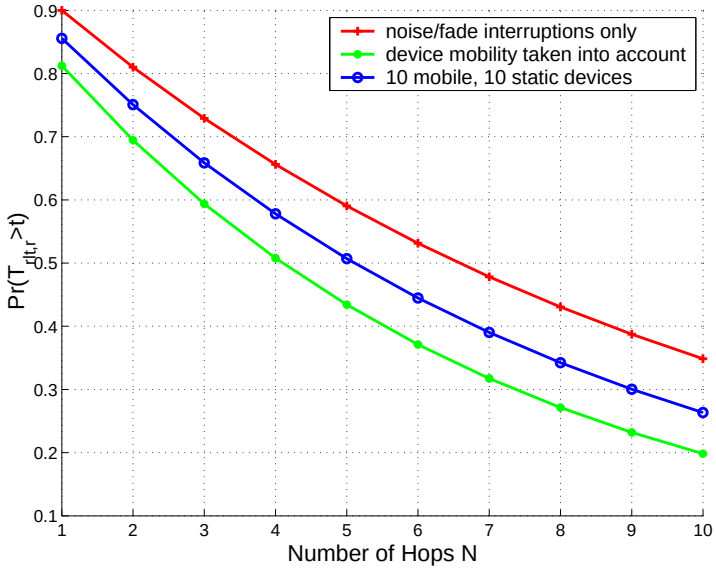


Figure 3.8: Probability that the residual lifetime of a route is larger than t vs. the number of hops for static (red), mobile (green) and partially mobile (blue) ad-hoc networks if $p_\mathrm{e} = 0.1$ and $p_\mathrm{o} = 0.05$.

where N denotes the number of hops between source and destination node and $N+1$ therefore is the number of nodes involved into the transmission. The first factor represents the necessity of N non-disturbed links, the second one accounts for the fact that none of the nodes may be moved out of the ranges of its two neighbors in the path.

When considering this expression one recognizes that the route stability vs. hops characteristic experiences a significant modification compared to the case of a static network, which is only influenced by interruption event I, at this point. In the latter case it was simply given by $\Pr(T_{\mathrm{rlt,r}} > t) = (1 - p_\mathrm{e}(t))^N$. The fact that p_e and p_o are contained in bases of different exponents will be the starting point for the density separation procedure in the next subsection.

The impact of the second route interruption reason is illustrated by the red and green curves in Fig. (3.8). On the one hand there is an offset – as an additional route interruption event clearly has to result in worth route stability performance, on the other hand the exponential decay does not go with exponent N any longer.

Extension for Fixed Nodes

So far we have assumed equally behaving nodes in the network. This is an assumption hardly satisfied in our experiment, since some of the users left there devices on their desk even when they were moving. Further also the virtual gateway constituted a static node. We therefore categorize nodes in moving and fixed nodes. Eq. (3.5) consequently has to be generalized for

a network containing both mobile and fixed nodes. For simplicity it is assumed that the nodes are spatially distributed in a fashion that the probability of a fixed/mobile node being the next hop destination in the route is equal to the number of remaining fixed/mobile stations divided by the total number of remaining devices.

We first state that the overall probability of a route being stable over the interval t is equal to the sum of all conditioned probabilities given that M of the $N+1$ devices within the route are mobile:

$$\Pr(T_{\mathrm{rlt,r}} > t) = \sum_{m=0}^{N+1} \Pr(T_{\mathrm{rlt,r}} > t | M = m) \cdot \Pr(M = m). \tag{3.6}$$

According to equation (3.5) the conditioned probability expression is given by

$$\Pr(T_{\mathrm{rlt,r}} > t | M = m) = (1 - p_{\mathrm{e}}(t))^N \cdot (1 - p_{\mathrm{o}}(t))^m. \tag{3.7}$$

The probability that a particular route contains M mobile nodes due to our assumption regarding the spatial arrangement of the nodes is hypergeometric distributed. If $\mathcal{M}$ is the set of mobile nodes and $\mathcal{F}$ the set of fixed nodes in the network this results in

$$\Pr(M = m) = \frac{\binom{|\mathcal{M}|}{m} \cdot \binom{|\mathcal{F}|}{N+1-m}}{\binom{|\mathcal{M}|+|\mathcal{F}|}{N+1}}. \tag{3.8}$$

Substituting (3.7) and (3.8) into (3.6) finally yields

$$\Pr(T_{\mathrm{rlt,r}} > t) = \sum_{m=0}^{N+1} (1 - p_{\mathrm{e}}(t))^N \cdot (1 - p_{\mathrm{o}}(t))^m \cdot \frac{\binom{|\mathcal{M}|}{m} \cdot \binom{|\mathcal{F}|}{N+1-m}}{\binom{|\mathcal{M}|+|\mathcal{F}|}{N+1}} \tag{3.9}$$

A sample curve for $|\mathcal{M}| = |\mathcal{F}| = 10$ is shown in blue in Fig. (3.8) for comparison.

3.2.4 Estimation Procedure

Based on the model proposed above the densities of link lifetime and residual link lifetime between two mobile nodes – both for interruption event I and II – shall be estimated from empirically measured overall residual route lifetimes. This will be done in two major steps:

- Estimation of $p_{\mathrm{e}}(t)$ and $p_{\mathrm{o}}(t)$ by fitting Eq. (3.9) to the empirical data in a least square sense for several residual lifetimes t. With (3.4) and (3.4) this yields empirical CDFs of the desired distributions.
- Approximation of the CDFs by an adequate analytical distribution and derivation of density functions – two for lifetimes and two for residual lifetimes.

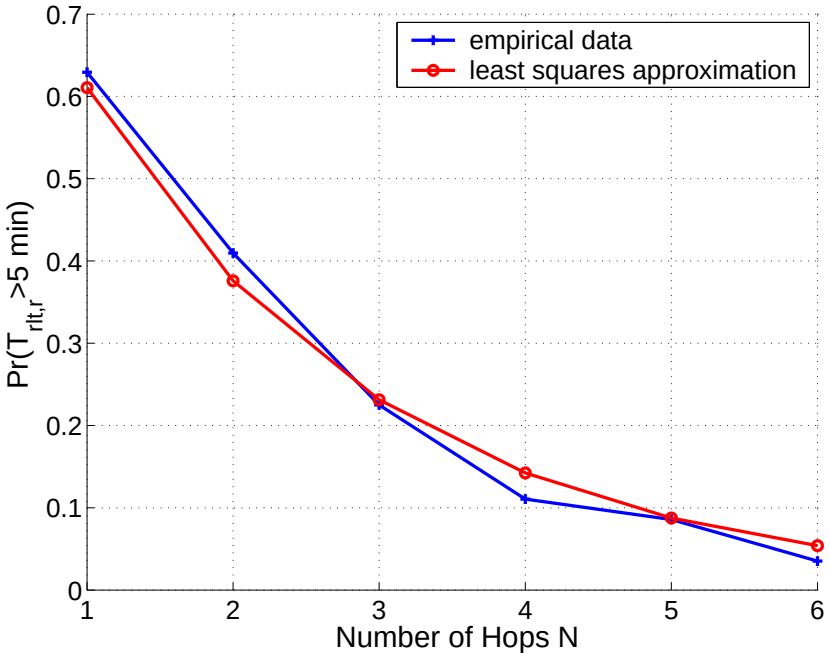


Figure 3.9: Empirical route stability and its least squares approximation according to (3.9). Resulting interrupt probabilities are $p_{\mathrm{e}} = 0.33$ and $p_{\mathrm{o}} = 0.04$, 15 mobile and 5 fixed nodes are assumed.

t [min]	5	10	15	20	25	30	35	40	45	50	55
p_e	0.33	0.40	0.44	0.46	0.50	0.54	0.55	0.57	0.60	0.61	0.65
p_o	0.04	0.18	0.30	0.40	0.47	0.54	0.63	0.69	0.74	0.79	0.82

Table 3.2: Values for p_e and p_o for different time intervals t.

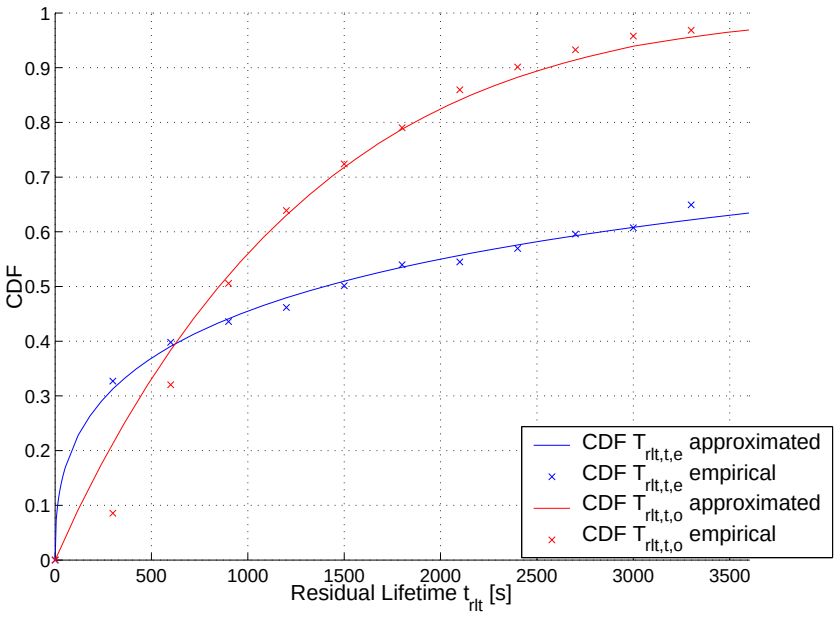


Figure 3.10: CDFs for $T_{\text{rtl,l,e}}$ and $T_{\text{rtl,l,o}}$ approximated based on $p_e(t)$ and $p_o(t)$.

Estimation of Empirical CDFs

A sample of a least squares approximation is shown in Fig. (3.9). The maximum number of hops taken into account is $\max((N : f(N) > 0) - 2, 2)$ in order not to get in the region where hardly any routes have been registered in the experiment. The routes we consider have been discovered according to a HOP metric. The estimated values of $p_e(t)$ and $p_o(t)$ are shown in Tab. (3.2). The consequent CDFs according to Eq. (3.4) and (3.4) and also their analytical approximation as derived in the subsequent subsection are plotted in Fig. (3.10).

At this point we can already state that the resulting CDFs basically make a lot of sense. One could have expected that $p_e(t) \gg 2 \cdot p_o(t) - p_o(t)^2$ for short intervals t, i.e., that interruptions due to collision/fading/noise occur much more often than due to node location changes (see Tab. (1)) in the immediate future. For very long intervals t vice versa it is not surprising that $2 \cdot p_o(t) - p_o(t)^2$ becomes larger than $p_e(t)$ at a certain point. Stated differently this means

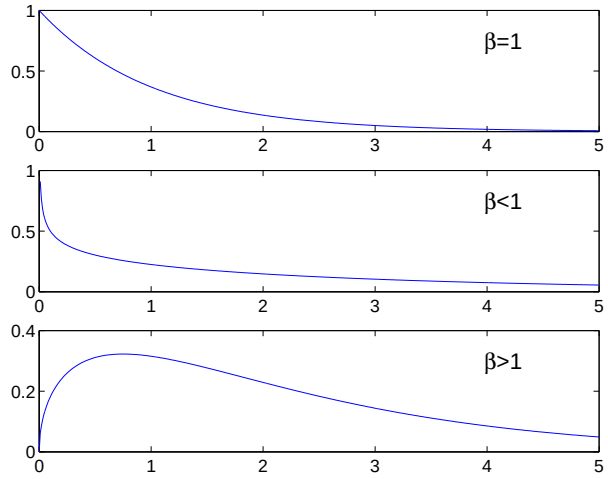


Figure 3.11: Plots of the PDF of the Weibull distribution for different β.

that very stable links tend to be terminated by node movements. For intuition consider a link that has not been corrupted by collisions, fades or noise in a way that the link is declared broken for a very long time. It therefore can be expected that the packet error rate is far below the corresponding threshold (50 percent in our case), what again implies very low interruption probability for the future. This is a significant difference to a scenario where link interruptions occur memoryless over time, i.e., the corresponding lifetimes are distributed exponentially.

Fitting to Analytical Distributions

Next we have to choose a distribution for the approximation of the empirical CDFs. In reliability theory lifetimes are modelled by Weibull distributions. Their density functions with parameters α and β are given by

$$f_T(t) = \begin{cases} 0, & t < 0, \\ \alpha \cdot \beta \cdot t^{\beta-1} \cdot \exp(-\alpha \cdot t^{\beta}), & t \geq 0. \end{cases} \tag{3.10}$$

The Weibull distribution is a generalization of several well known distributions, such as the exponential distribution or the Rayleigh distribution. It is capable of describing monotonically increasing ($\beta > 1$), constant ($\beta = 1$) and monotonically decreasing ($\beta < 1$) failure rates (see [6]). Some characteristic sample courses are shown in Fig. (3.11).

$\Pr[T_{rlt,l,e} < t]$ and $\Pr[T_{rlt,l,o} < t]$, however, are CDFs of residual link lifetimes – not of lifetimes. We therefore have to somehow relate these quantities. This can be achieved by using the law of total probability:

$$f_{T_{\mathrm{rlt,l}}}(t) = \int_{-\infty}^{\infty} f_{T_{\mathrm{rlt,l}}|T_{\mathrm{lt,l}}}(t|\tau) \cdot f_{T_{\mathrm{lt,l}}}(\tau) \cdot d\tau. \tag{3.11}$$

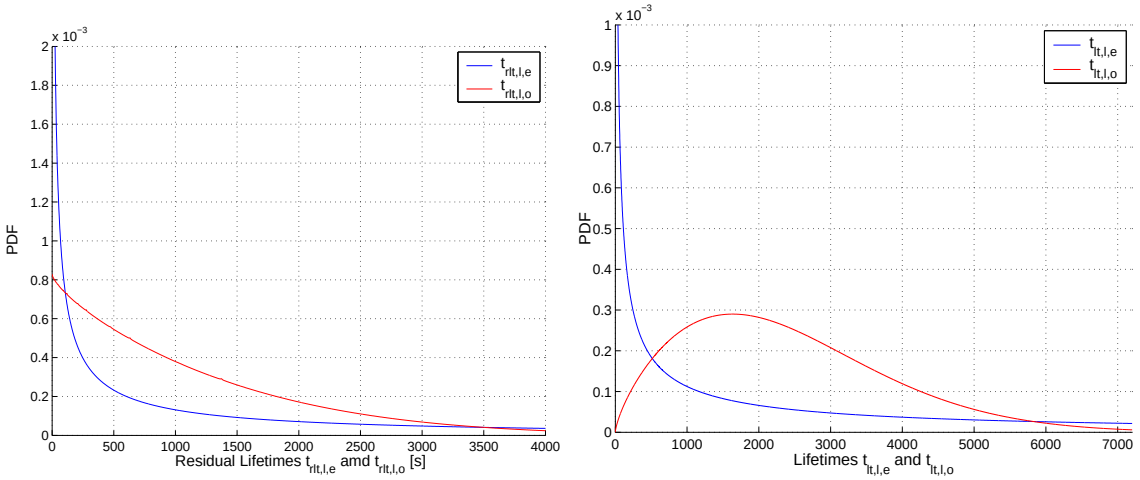


Figure 3.12: PDFs for residual lifetimes and lifetimes corresponding to CDFs in Fig. (3.10)

As the distribution of $T_{\mathrm{rlt,l}}$ given $T_{\mathrm{lt,l}} = \tau$ is uniform between 0 and τ, we get

$$f_{T_{\mathrm{rlt,l}}}(t) = \begin{cases} \int_0^\infty \frac{1}{\tau} \cdot I\{0 < t < \tau\} \cdot \alpha \cdot \beta \cdot \tau^{\beta-1} \cdot \exp(-\alpha \cdot \tau^\beta) \cdot d\tau, & t \geq 0, \\ 0, & t < 0. \end{cases} \tag{3.12}$$

$$= \begin{cases} \int_t^\infty \alpha \cdot \beta \cdot \tau^{\beta-2} \cdot \exp(-\alpha \cdot \tau^\beta) \cdot d\tau, & t \geq 0, \\ 0, & t < 0. \end{cases} \tag{3.13}$$

The desired CDF now can be found by integration

$$F_{T_{\mathrm{rlt,l}}}(t) = \begin{cases} \int_0^t \int_\eta^\infty \alpha \cdot \beta \cdot \tau^{\beta-2} \cdot \exp(-\alpha \cdot \tau^\beta) \cdot d\tau \cdot d\eta, & t \geq 0, \\ 0, & t < 0. \end{cases} \tag{3.14}$$

Non of the appearing integrals can be solved analytically.

Eq. (3.14) now can be used to approximate the empirical CDFs in Fig. (3.10). This is done by using the *Matlab optimization toolbox and the the Matlab ode45 integrator*. The results are already indicated by the solid lines in Fig. (3.10). The corresponding estimated parameters are $\hat{\alpha} = 1.35 \cdot 10^{-6}$, $\hat{\beta} = 1.725$ for $F_{T_{\mathrm{rlt,l,o}}}(\cdot)$ and $\hat{\alpha} = 0.021$, $\hat{\beta} = 0.43$ for $F_{T_{\mathrm{rlt,l,e}}}(\cdot)$.

We are now also able to reconstruct the densities of lifetime and residual lifetime. They are shown in Fig. (3.12). Mean and standard deviations can be evaluated numerically. They are shown in Tab. (3.3).

The shown densities further point up what was already visible from the according CDFs: When regarding the densities for $T_{\mathrm{rlt,l,e}}$ we see that interruption event I – exaggerated stated – occurs

RV	Mean [min]	Std. Dev. [min]
$T_{\mathrm{rlt,l,o}}$	19	17
$T_{\mathrm{rlt,l,e}}$	216	724
$T_{\mathrm{lt,l,o}}$	38	22
$T_{\mathrm{lt,l,e}}$	432	1233

Table 3.3: Mean and standard deviation of PDFs.

either very soon after the link discovery or never. To see that also the results for $T_{\mathrm{rlt,l,o}}$ make sense we consider the according density of the lifetime. Here we see that the majority of links break at an age of 10 to 50 minutes. There are hardly links breaking either immediately after their emergence or more than two hours later.

3.2.5 Online Estimation of Route Interruption Reasons

From Fig. (3.10) one can see that there are two clearly separated regions – each one dominated by one of both interruption events. This might be exploited by a reactive routing protocol in order to estimate the actual route interruption reason based on the duration the route has existed.

The according optimal decision rule is a simple threshold decision. This can be seen by recognizing that given a route interruption after t seconds the probability of a correct decision is maximized by choosing the event with higher a priori probability. The threshold is therefore given by the intersection point of the densities for $T_{\mathrm{rlt,l,e}}$ and $T_{\mathrm{rlt,l,o}}$ in the case of a 1-hop route, in our case at $t = 110$ s.

In order to find the decision thresholds for routes larger than 1 hop, we have to derive the appropriate probability densities. These can be found by considering the CDF for the residual lifetime of a route with N hops first, which is given by

$$F_{\mathrm{rlt,r}}(t) = 1 - \Pr(T_{\mathrm{rlt,l}} > t)^N = 1 - (1 - F(rlt, l)(t))^N. \tag{3.15}$$

The PDF now is found by taking the derivative

$$f_{\mathrm{rlt,r}}(t) = N \cdot f(rlt, l)(t) \cdot (1 - F(rlt, l)(t))^N. \tag{3.16}$$

Comparing $f_{\mathrm{rlt,r,e}}(t)$ and $f_{\mathrm{rlt,r,o}}(t)$ yields the following decision thresholds:

N	1	2	3	4	5	6
t_{th} [s]	106	83	66	53	44	38

In the following we compute the probability of a wrong decision under the optimal decision rule. In order to do this we have to consider two events resulting in a wrong decisions. The first

one is the probability of a link interruption through node movements earlier than t_{th} seconds, the secondly a wrong decision occurs, if a link breaks later than t_{th} through a link error.

We compute the first probability

$$\begin{aligned} \Pr[\text{error}|T_{\text{rlt,r}} < t_{\text{th}}] &= \Pr[T_{\text{rlt,r,o}} < T_{\text{rlt,r,e}}|T_{\text{rlt,r}} < t_{\text{th}}] && (3.17) \\ &= \frac{\Pr[T_{\text{rlt,r,o}} < T_{\text{rlt,r,e}}, T_{\text{rlt,r}} < t_{\text{th}}]}{\Pr[T_{\text{rlt,r}} < t_{\text{th}}]}. && (3.18) \end{aligned}$$

Analogously

$$\begin{aligned} \Pr[\text{error}|T_{\text{rlt,r}} > t_{\text{th}}] &= \Pr[T_{\text{rlt,r,o}} > T_{\text{rlt,r,e}}|T_{\text{rlt,r}} > t_{\text{th}}] && (3.19) \\ &= \frac{\Pr[T_{\text{rlt,r,o}} > T_{\text{rlt,r,e}}, T_{\text{rlt,r}} > t_{\text{th}}]}{\Pr[T_{\text{rlt,r}} > t_{\text{th}}]}. && (3.20) \end{aligned}$$

The overall probability of error now is given by

$$\begin{aligned} \Pr(\text{error}) &= \Pr(\text{error}|T_{rlt,r} < t_{\text{th}}) \cdot \Pr(T_{rlt,r} < t_{\text{th}}) + \Pr(\text{error}|T_{rlt,r} > t_{\text{th}}) \cdot \Pr(T_{rlt,r} > t_{\text{th}}) && (3.21) \\ &= \Pr[T_{\text{rlt,r,o}} < T_{\text{rlt,r,e}}, T_{\text{rlt,r}} < t_{\text{th}}] + \Pr[T_{\text{rlt,r,o}} > T_{\text{rlt,r,e}}, T_{\text{rlt,r}} > t_{\text{th}}] && (3.22) \\ &= \int_0^{t_{\text{th}}} \left(f_{T_{\text{rlt,r,o}}}(t) \cdot (1 - F_{T_{\text{rlt,r,e}}}(t))\right) \cdot dt + \int_{t_{\text{th}}}^{\infty} \left(f_{T_{\text{rlt,r,e}}}(t) \cdot (1 - F_{T_{\text{rlt,r,o}}}(t))\right) dt && (3.23) \end{aligned}$$

These expressions again can be analyzed numerically. The error probabilities for different numbers of hops are

N	1	2	3	4	5	6
Pr(error) [%]	27.6	30.2	29.8	28.7	27.4	25.9

3.2.6 Comparison with Mobility Models

In subsection (3.2.4) separated distributions for route interruptions due to critical links and due to user mobility have been derived. By taking the latter distribution one has extracted the real influence of user mobility on the route stability. Therefore we are now able to compare our data set with simulation results from various mobility models. In this subsection we compare link stabilities in the test network with those in the random waypoint model and the random reference point model. These models and their properties are described in [9] in detail.

Random Waypoint Model

The implementation of the random waypoint model in *Matlab* is shown in the appendix. It is used in its purest version in this thesis, i.e., each node

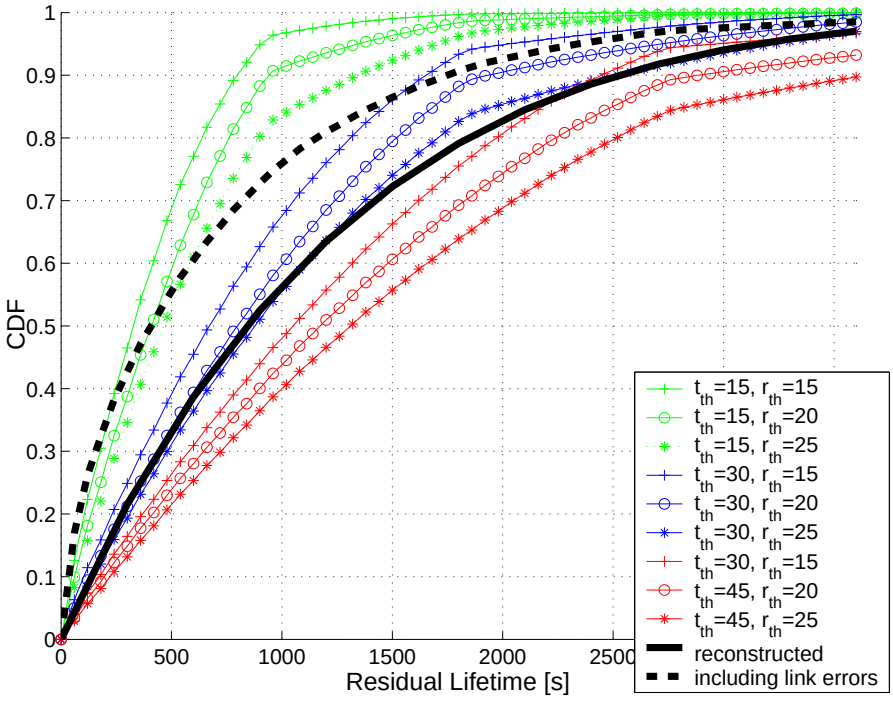


Figure 3.13: CDFs of residual link lifetimes as resulting from the random waypoint model under different ranges and thinking times. The CDFs for $T_{rlt,l,o}$ (solid) and $T_{rlt,l}$ (dashed) as measured during the experiment for comparison.

- chooses its destination randomly in a plane according to a uniform distribution,
- moves with deterministic velocity,
- waits for a deterministic period after arriving its target.

The plane size is chosen to be 100 x 20 meters. This roughly corresponds to the effective size of the floor, as nodes on the upper and nodes on the lower floor side had no connection to each other. The user velocity is chosen to be 1 m/s. While plane size and user velocity are fixed, thinking time and range are varied in order to achieve match with the empirical data.

In Fig. (3.13) a set of curves for the CDF of $T_{\mathrm{rlt,l,o}}$ as found by simulating according to the random waypoint model is shown. It covers the range of reasonable values for range and thinking time. The solid line shows the CDF as extracted from the experiment data. It can be seen that indeed good match is achieved with the curve for a range of 25 meters and a thinking time of 30 minutes. On the other hand one can see that the dashed line representing the overall residual link lifetime, i.e., including link errors, cannot be matched by the model, as it is much more concave than the set of reasonable curves.

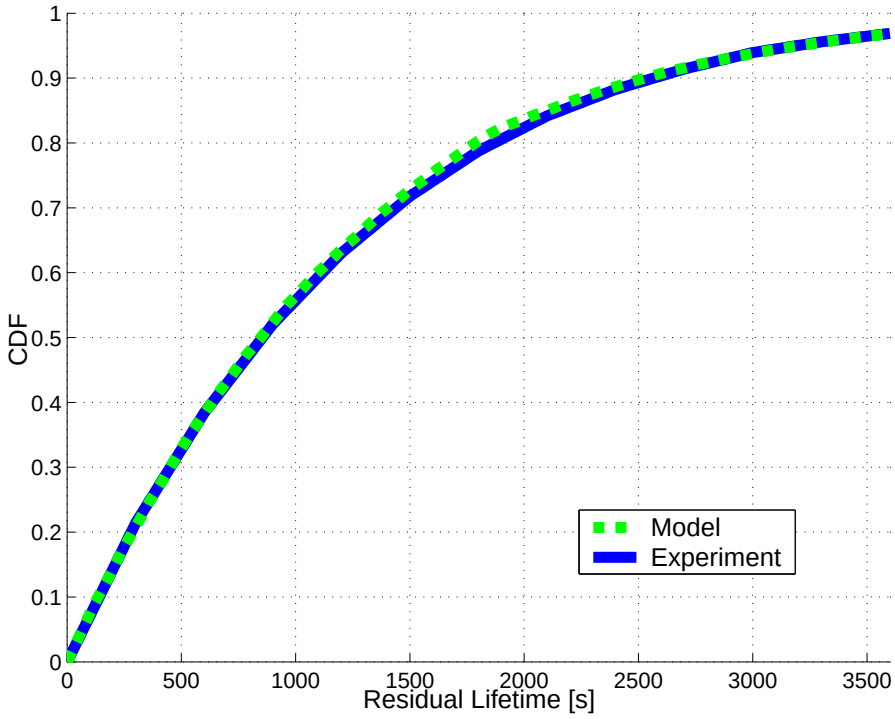


Figure 3.14: CDFs of residual link lifetimes as resulting from the random reference point group mobility model and as extracted from the measured data from the experiment.

Random Reference Point Group Mobility Model

The random reference point group mobility model is implemented as two encapsulated random waypoint entities (again see appendix for Matlab code). On the one hand members of a group move completely randomly within a rectangular sub-plane around the group center. The group center again moves randomly according to the random waypoint model, such that the sub-planes do not exceed the boundaries of the large plane.

The model has more degrees of freedom than the random waypoint model. Again plane size (100 x 20 m) and sub-plane sizes (33 x 20 m) are fixed. Further 10 groups of sizes 6, 3,2, 2, 2, 1, 1, 1, 1 and 1 are assumed. Both group centers and users within the group move with velocity 1 m/s. The course of the CDF is matched to the one from the experiment by varying thinking times of users and group centers as well as the range. For thinking times of 63 minutes for the group and 32 minutes for the users and a range of 15 meters the curve is matched perfectly to the one resulting from the measurements. Both are shown in Fig. (3.14).

3.2.7 Dynamic Characterization Employing Auto-Covariance Functions

In section (3.2.5) decision thresholds for the estimation of a route interruption reason have been derived. These thresholds, however, can be assumed to strongly depend on the user behavior in a particular network. Therefore each node has to figure out the dynamic network properties online in order to choose the thresholds appropriately. What is needed is another dynamic model of significantly reduced complexity, which – instead of determining thresholds based on detailed distributions – only needs some key parameters for a reasonable estimation. Besides the low complexity of the model the threshold estimation procedure will be based on another demand is the practicability on a single node without extensive data exchange in the network, i.e., the estimation should mainly be based on data that can be collected locally by each single device. One way of characterizing the dynamic in the network from the angle of a particular node contained in it, is to determine the auto-covariance function of its node degree. This function generally is a measure for the correlation between two topologies and satisfies our requirements of being computed both with low complexity and based on locally available data.

In order to determine decision thresholds one is particularly interested in how the two discussed types of link interruptions influence the covariance function. In order to solve this question analytically, we assume the following:

- There are two kinds of links: first there are stable links, secondly there are critical links. When regarding the number of neighbors of a particular node analogously we differentiate neighbors connected by a stable link, whose number is denoted by $S(t)$, and neighbors connected by a critical link, whose number is denoted by $C(t)$. The total neighbor number of a node consequently is $N(t) = S(t) + C(t)$.
- We assume stable links to be stable (i.e. free of link errors) forever, critical links are assumed to be present with probability p at time t independently of $C(t - t_0)$, i.e., $C(t) \perp C(t - t_0) \quad \forall t_0 \in \mathbb{R} \setminus \{0\}$.
- The number of stable neighbors and the number of critical neighbors are independent, i.e., $S(t_1) \perp C(t_2) \forall \quad t_1, t_2$.
- Both $S(t)$ and $C(t)$ are wide sense stationary random processes, i.e., $\mathsf{K}_{\mathrm{NN}}(t, \tau) = \mathsf{K}_{\mathrm{NN}}(\tau)$.

Denoting the mean of the processes $N(t)$, $S(t)$ and $C(t)$ m_{N}, m_{S} and m_{C}, we consider the covariance function $\mathsf{K}_{\mathrm{NN}}(\tau)$ of the total number of neighbors of a particular node:

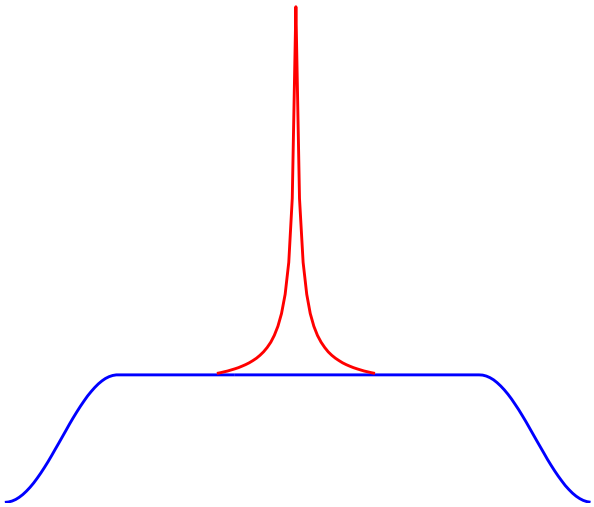


Figure 3.15: Sketch of auto-covariance functions with contributions from stable (blue) and critical (red) nodes.

$$
\begin{aligned}
\mathsf{K}_{\mathrm{NN}}(\tau) &= \mathsf{E}\left[(N(t+\tau)-m_N)\cdot(N(t)-m_N)\right] && (3.24)\\
&= \mathsf{E}\left[(S(t+\tau)+C(t+\tau)-m_S-m_C)\cdot(S(t)+C(t)-m_S-m_C)\right] && (3.25)\\
&= \mathsf{E}[S(t)S(t+\tau)]+2\mathsf{E}[S(t)C(t+\tau)]+\mathsf{E}[C(t)C(t+\tau)]-(m_S+m_C)^2 && (3.26)\\
&= \mathsf{E}[S(t)S(t+\tau)]-m_S^2+2\mathsf{E}[S(t)C(t+\tau)]-2m_Sm_C+\mathsf{E}[C(t)C(t+\tau)]-m_C^2 && (3.27)\\
&= \mathsf{K}_{\mathrm{SS}}(\tau)+2\cdot\mathsf{K}_{\mathrm{SC}}(\tau)+\mathsf{K}_{\mathrm{CC}}(\tau) && (3.28)\\
&= \mathsf{K}_{\mathrm{SS}}+\sigma_{\mathrm{C}}^2\delta(0). && (3.29)
\end{aligned}
$$

The last line follows from our assumptions that there is no cross-correlation between $S(t)$ and $C(t)$ and that $C(t)$ is spectrally white. One recognizes that the only contribution from critical neighbors appears at $\tau = 0$ s.

When considering Fig. (3.16), which shows the auto-covariance functions as determined by the nodes in the experiment, one notices that assuming $C(t)$ being a white process was too optimistic. Instead the peak is quite washed out. Nevertheless, the two contributions – one from $\mathsf{K}_{\mathrm{SS}}(\tau)$ and one from $\mathsf{K}_{\mathrm{CC}}(\tau)$ – can be clearly distinguished for most of the nodes, e.g. 4, 12, 13, 17, in Fig. (3.16). A sketch of how the contributions of stable and critical neighbors might look like is shown in Fig. (3.15).

In order to actually find an optimal way of determining decision thresholds from auto-covariance functions, measurements from networks with different dynamic behavior would have to be collected and analyzed with respect to the relation between both quantities. Possible approaches are threshold estimations based on

- the ratio of the peak level at $\tau = 0$ to the level of the flat base,
- the time shift with a certain steepness in the curve,
- the ratio of time shifts where full resp. 50 % de-correlation is reached.

The first approach is the nearest at hand as it reflects the ratio of "power" in the processes $S(t)$ and $C(t)$. However, it might be difficult in some cases to determine the base level (see curve for device no. 1 e.g.).

3.3 Comparison of Route Stabilities under HOP, PER and RTT Metrics

In this section differences in the stability of routes established under different metrics, in particular under HOP-, PER- and RTT-metric are compared. Originally also signal strength based routing should be investigate. The NIC interface of the iPAQs, however, did not assign the measured values to the according source devices. Before actually discussing the differences with respect to route stability the metrics shall be specified in the way they are used in this context:

- **Hop Count (HOP):** This metric aims to minimize the number of hops a packet traverses from source to destination. A link is declared as broken, if the current packet error rate (see next item on how this is determined) drops below 50 percent. The shortest path in a network can be computed by applying the Dijkstra algorithm, where each edge in the network graph has weight 1.
- **Packet Error Rate (PER):** The metric aims to minimize the probability of a packet loss on the way from source to destination. The current packet error rate is estimated based on indexed broadcast packets each device distributes among its neighbors each 500 ms. This is done by averaging over the last 15 received packets. If the PER drops below 50 percent or a device has not received any packet for more than 7.5 seconds, the PER is set to 1. The most reliable path in a network can be computed by applying the Dijkstra algorithm, where each edge in the network graph has weight $-\log p_{ij}$.
- **Round Trip Time (RTT):** The metric aims to minimize the round trip time between source and destination. The current round trip time is estimated by pinging all devices that are known to be immediate neighbors in the moment every five seconds and waiting for their acknowledgment. Should a packet get lost the round trip time of the previous measurement is used. After two subsequent packet losses the link is declared as broken. The path with the smallest overall round trip time in a network can be computed by applying the Dijkstra algorithm, where each edge in the network graph has weight $t_{RT,ij}$.

In order to asses the route stabilities under theses metrics, we consider the 50th percentiles of the residual route lifetimes for different numbers of hops. The corresponding plots are shown in Fig. (3.17).

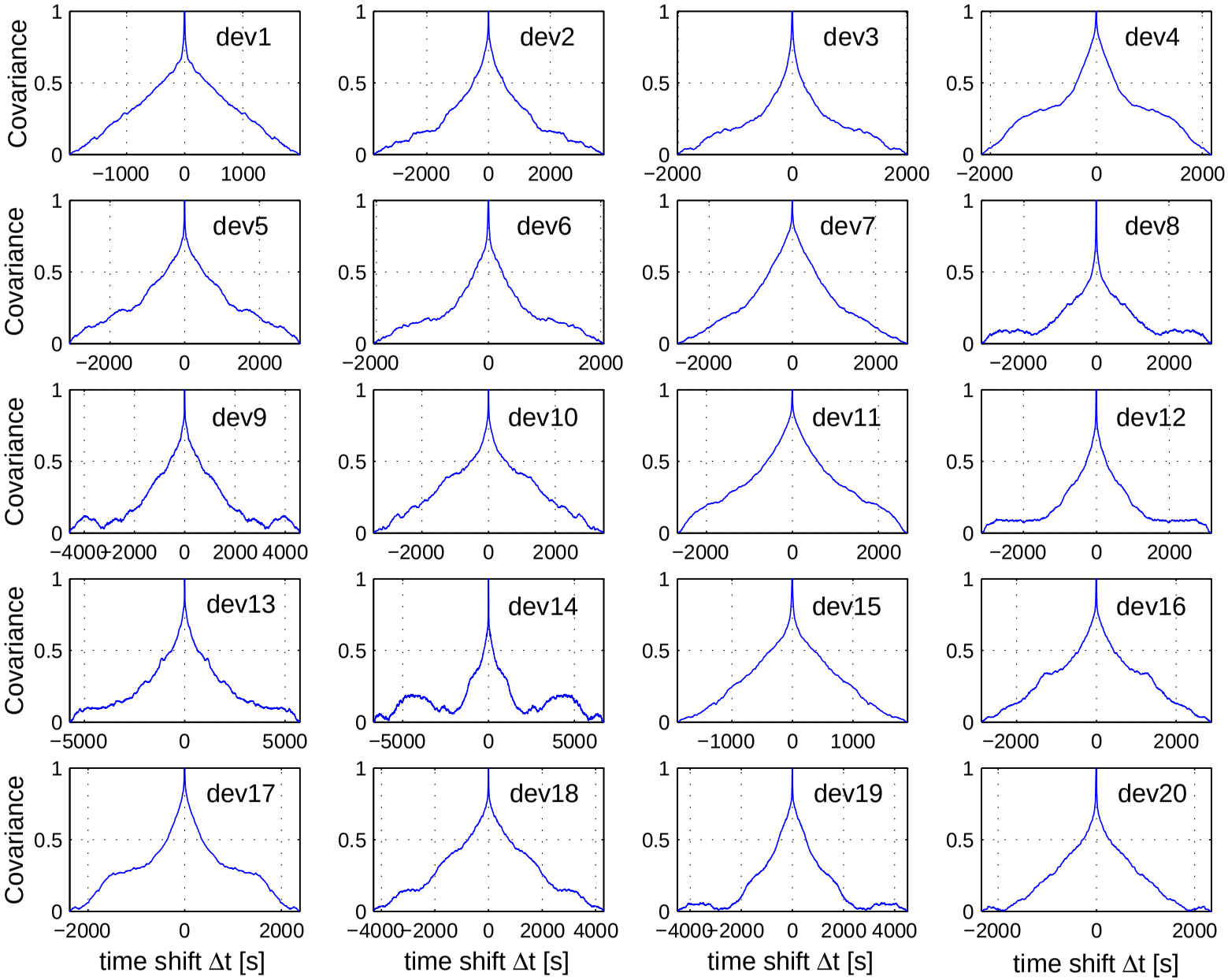


Figure 3.16: Autocovariance functions of node degree courses.

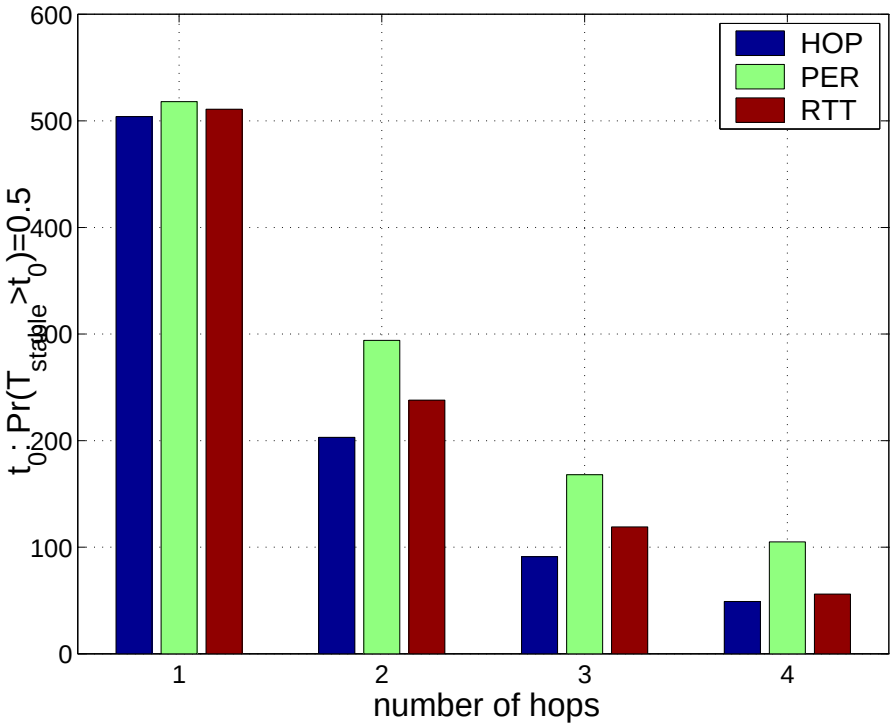


Figure 3.17: 50th percentile of remaining route durations under HOP, PER and RTT metric.

In order to discuss the route stabilities we first recall the two different route interruption events. First there might be a link interruption due to collisions, fading or noise (denoted interruption event I in the following). Secondly a link will be interrupted whenever a device is moved out of the range of its neighbor (denoted interruption event II in the following).

The problem coming with these sources of an interruption is that the minimization of their occurrence probability p_{e} resp. p_{o} follows different principles. While it is clear that it is recommendable to guide a packet via several reliable hops instead of one very lossy hop in order to minimize the probability of interruption event I, this contradicts the minimization strategy for the occurrence probability of interruption event II, namely to use as few hops as possible.

There probably might be both scenarios, where one metric outperforms the other or vice versa. A case where the HOP metric is superior to the PER metric might occur in networks with very high user mobility. As this is not the case in our experiment setup it is not very surprising, that the PER yields more robust routes.

The route stability achieved by the RTT metric finally lies somewhere in the middle between the metrics just discussed. As the round trip time is a measure for the amount of traffic oneself and a neighbor device (queuing and contention delay) is loaded with, rather than for the distance in indoor applications it is strongly related to the number of packet collisions that device experiences. Therefore it accounts for packet losses due to collisions on the one hand, but ignores packet losses due to low SNR. This means in some cases it decides similar to the PER metric, in the other case similar to the HOP metric. This is in correspondence with the empirical result in Fig. (3.17), where it performs better than HOP, but worse than PER. Therefore it might be the metric that is most robust in both high and low mobility networks, but never the optimal one.

Having a closer look at Fig. (3.17) one recognizes that the PER metric outperforms the other metrics in particular for increasing numbers of hops. This can be understood intuitively quite well. Consider the scenario, in which the HOP metric yields a route over two hops. This will only be the case if there is no straight link to the destination device. Therefore one can be sure that the average error probability over all such two hops links will be higher than the average error probability over the one hop links to its immediate neighbors. Under the PER metric this is not the case necessarily, since a two hop route might also be established in order to improve the route reliability. With this in mind one expects, that the stability gap between one and two hop routes is larger under the HOP-metric than under the PER-metric – and this is exactly what is observed in Fig.(3.17). The same argument holds for the transition from 2 to 3 and from 3 to 4 hops. At four hops the PER metric already yields links that are twice as stable as those generated by the HOP metric.

Fig. (3.18) finally shows the PMFs of the number of hops under the three discussed metrics. Their courses are as they are expected. While the PER metric yields links with up to 14 hops, the HOP metric of course chooses shorter routes. As discussed previously the RTT metric shares some commonness with both other metrics and therefore its PMF extends to more hops

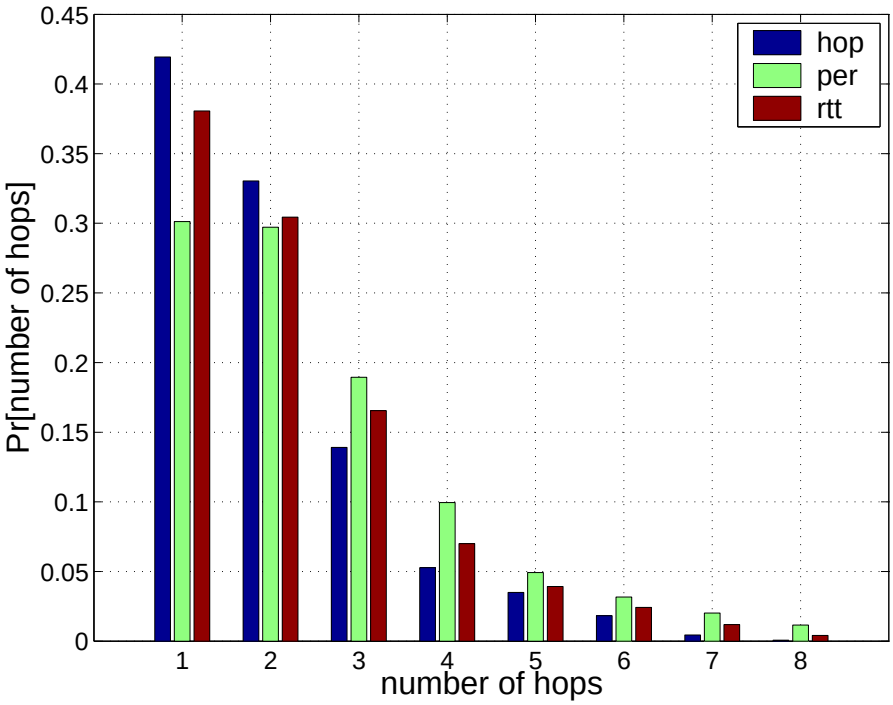


Figure 3.18: PMF of no. of hops in routes established based on HOP, PER and RTT metric.

than the one of corresponding to HOP, but to less than the one corresponding to PER.

Chapter 4

Conclusion

In this thesis data from a real mobile ad-hoc network was collected, analyzed with respect to node degrees, path lengths, clustering, connectivity to gateway, as well as route stability under different routing metrics, and compared with two mobility models. Key contributions are two mathematical models on the topic of route stability which provided a method for extracting the influence of user mobility on the stability of routes and thus also enabled a fair comparison between empirical data and simulation results from mobility models. It was shown that a mobile node can estimate the reason of a route interruption based on the time the route was in use until the interruption. Nodes might exploit this information for deciding on how to react on such an interruption. It was further demonstrated that even simple mobility models, such as the random waypoint and the random reference point group mobility model, yield good match with the empirical data from the real network with respect to route stability. When comparing three different routing metrics it turned out that a metric aiming to minimize the packet error rate yielded the stablest routes for our particular experiment network.

Bibliography

[1] R. Draves, J. Padhye and B. Zill, ,,Comparison of Routing Metrics for Static Multi-Hop Wireless Networks“.

[2] D. Aguayo, J. Bicket, S. Biswas, G. Judd and R. Morris, ,,Link Level Measurements from an 802.11b Mesh Network.“

[3] L. Li, D. Alderson, R. Tanaka, J.C. Doyle and W. Wilinger, ,,Towards a Theory of Scale-Free Graphs: Definition, Properties, and Implications (Extended Version)“.

[4] R. Albert and A.-L. Barabasi, ,,Statistical Mechanics of Complex Networks“.

[5] A. Chaintreua, P. Hui, J. Crowsoft, C. Diot, R. Gass adn J. Scott, ,, Pocket Switched Networks: Real-World Mobility and its Consequences for Opportunistic Networking“.

[6] H. Weber, ,,Einfuehrung in die Wahrscheinlihckeitsrechnung und Statistik fuer Ingenieure“.

[7] P. Bremaud, ,,An Introduction to Probabilistic Modelling“.

[8] A. Tanenbaum, ,,Computer Networks“.

[9] , T. Camp, J. Boleng and V. Davies, ,,A Survey of Mobility Models for Ad Hoc Network Research“.

Appendix A

Measurement Application

Client-Thread

```
int client() {

    ////////////////////////////////////////
    // DECLARATION AND INITIALIZATION STUFF //
    ////////////////////////////////////////

    // general variables
    long cycles=0, lItems = 0, lNumItems = 0, rc;
    int dur=0, ipaqnets = 0, i, day, hour, min, sec, startday, wait=-1;
    char buf[256], ref[20],comp[20];
    MAC_ADDR my_mac_address, my_old_mac_address;
    transobj to;
    SOCKET s;
    SOCKADDR_IN addr, loopback;
    CTime theTime;
    UCHAR Ssid[32] = {0}, BSsid[32] = {0}, refSsid[32]={0}, compSsid[32]={0};
    FILE *in, *bssidlog;

    // NDIS related variables
    WRAPI_NDIS_DEVICE   *pDeviceList = NULL;
    HRESULT             hRes;
    AP_DATA             *pAP_list = NULL;
    CHAR                *pSSId = "UCSD";
    ULONG               lSSIdLength = strlen(pSSId);

    // get my user's name
    in=fopen("myname.txt","r");
    fscanf(in,"%s",&myname[0]);
    fclose(in);
    DeleteFile(TEXT("myname.txt"));

    // open file for logging bssids
    bssidlog=fopen("\\CF Card\\bssidlog.txt","a");

    // initialize NDIS
    Ssid[0]='\0';
    CWRAPIApp theApp;
    if(!theApp.InitInstance())
        printf("Initialisation failed!\n");
    hRes=theApp.EnumerateDevices(&pDeviceList, &lItems);
    while(lItems==0)
    {
        // get the list of Devices in the System
        PlaySound (TEXT("Alert"), NULL, SND_SYNC);
        MessageBox(NULL,TEXT("the wlan module is switched off! please enable it again!"),NULL,MB_SETFOREGROUND);
        hRes=theApp.EnumerateDevices(&pDeviceList, &lItems);
    }
    hRes=theApp.GetBSSId(my_mac_address, pDeviceList[0].pDeviceName);
    hRes=theApp.GetBSSId(my_old_mac_address, pDeviceList[0].pDeviceName);

    // print list of devices obtained
    for (i = 0; i < lItems; i++)
    {
        printf("==========================\nDecive no: %d\n",i+1);
        printf("Desciption: %S\nName: %S\n==========================\n", pDeviceList[i].pDeviceDescription, pDeviceList[i].pDeviceName);
```

```
}

// create and initialize client-socket
rc=startWinsock();
if(rc!=0)
{
    printf("Fehler: startWinsock, fehler code: %d\n",rc);
    getchar();
    return 1;
}
else
{
    PlaySound (TEXT("Alert"), NULL, SND_SYNC);
    MessageBox(NULL,TEXT("client is active!"),TEXT("Notification"),MB_SETFOREGROUND);
}
s=socket(AF_INET,SOCK_DGRAM,0);
if(s==INVALID_SOCKET)
{
    printf("Fehler: Der Socket konnte nicht erstellt werden, fehler code: %d\n",WSAGetLastError());
    getchar();
    return 1;
}
addr.sin_family=AF_INET;
addr.sin_port=htons(1234);
addr.sin_addr.s_addr=inet_addr("192.168.0.255");
loopback.sin_family=AF_INET;
loopback.sin_port=htons(4321);
loopback.sin_addr.s_addr=inet_addr("127.0.0.1");

// get current time
theTime=CTime::GetCurrentTime();
startday=theTime.GetDay();



//////////////////
// THE MAIN LOOP //
//////////////////

while(1)
{
    // update packet index and time and write it to transmit struct
    cycles++;
    theTime=CTime::GetCurrentTime();
    day=theTime.GetDay();
    hour=theTime.GetHour();
    min=theTime.GetMinute();
    sec=theTime.GetSecond();
    sprintf(to.time,"%d.%d.%d.%d", day, hour, min, sec);
    sprintf(to.name,"%s", myname);
    to.index=cycles;
    memcpy(&buf, &to, sizeof(transobj));

    // transmission via client-socket
    rc=sendto (s,buf,sizeof(transobj)+1,0,(SOCKADDR*)&addr,sizeof(SOCKADDR_IN));
    if(rc==SOCKET_ERROR)
    {
        PlaySound (TEXT("Alert"), NULL, SND_SYNC);
        MessageBox(NULL,TEXT("unable to send!\nensure that WLAN is activated!"),NULL,MB_SETFOREGROUND);
        printf("Fehler: sendto, fehler code: %d\n",WSAGetLastError());
        getchar();
        return 1;
    }

    // bssid partitioning check every APSCAN cycles
    if(!(cycles%APSCAN))
    {
        if(wait>0)
        {
            wait--;
            printf("wait: %d\n", wait);
        }
        else if(wait==0)
        {
            hRes=theApp.GetBSSId(my_mac_address, pDeviceList[0].pDeviceName);
            hRes=theApp.GetBSSId(my_old_mac_address, pDeviceList[0].pDeviceName);
            wait--;
        }
        else
        {
            hRes=theApp.GetBSSId(my_mac_address, pDeviceList[0].pDeviceName);
            sprintf(ref,"%02x%02x%02x%02x%02x%02x", my_mac_address[0],my_mac_address[1],
            my_mac_address[2],my_mac_address[3],my_mac_address[4],my_mac_address[5]);
```

```
            sprintf(comp,"%02x%02x%02x%02x%02x%02x", my_old_mac_address[0],my_old_mac_address[1],
            my_old_mac_address[2],my_old_mac_address[3],my_old_mac_address[4],my_old_mac_address[5]);
            if(ref[11] < comp[11])
            {
                wait=WAITSCANCYCLES;
            }
            else
            {
                hRes=theApp.GetBSSId(my_old_mac_address, pDeviceList[0].pDeviceName);
                for (i = 0; i < DEVICES; i++)
                    apscanflag[i]=INTERLEAVE;
                hRes=theApp.GetAPList(&pAP_list, &lNumItems ,pDeviceList[0].pDeviceName);
            }
        }
        fprintf(bssidlog,"%02x%02x%02x%02x%02x%02x %d.%d.%d.%d\n",my_mac_address[0],my_mac_address[1],
        my_mac_address[2],my_mac_address[3],my_mac_address[4],my_mac_address[5] , day,hour,min,sec);

    }

    // ask gateway thread, whether there is a connection
    if(!(cycles%ALERTGWCHECK))
    {
        to.index=-1;
        memcpy(&buf, &to, sizeof(transobj));
        rc=sendto (s,buf,sizeof(transobj)+1,0,(SOCKADDR*)&loopback,sizeof(SOCKADDR_IN));
        if(rc==SOCKET_ERROR)
        {
            PlaySound (TEXT("Alert"), NULL, SND_SYNC);
            MessageBox(NULL,TEXT("unable to send!\nensure that WLAN is activated!"),NULL,MB_SETFOREGROUND);
            printf("Fehler: sendto, fehler code: %d\n",WSAGetLastError());
            getchar();
            return 1;
        }
    }

    // wait
    Sleep(SLPER);
  }
  return 0;
}
```

Server-Thread

```
int server()
{
    ////////////////////////////////////////
    // DECLARATION AND INITIALIZATION STUFF //
    ////////////////////////////////////////

    // variables
    long lastseen[DEVICES], probestart[DEVICES], rc, cnt=0, idx=1;
    int recvcount[DEVICES], remoteAddrLen=sizeof(SOCKADDR_IN), seen[DEVICES], i;
    double per[DEVICES], packets=PACKETS;
    char namebuf[DEVICES][20], *id, filename[100], dum[4], buf[sizeof(transobj)+1];
    bool flag=true;
    SOCKET s;
    SOCKADDR_IN addr;
    SOCKADDR_IN remoteAddr;
    CTime curtime, oldtime, dtime;
    CTimeSpan tspan=CTimeSpan(0,0,0,DEVICEPERIOD);
    FILE *out;


    // find current logfile
    while(flag)
    {
        strcpy(filename,"\\CF Card\\logfile");
        _itoa(idx,dum,10);
        strcat(filename,dum);
        strcat(filename,".txt");
        out=fopen(filename,"r");
        if(out==NULL)
            flag=false;
        else
        {
            idx++;
        }
        fclose(out);
    }

    //some (important!!!) initializations
    for(i=0;i<DEVICES; i++)
```

```
{
    namebuf[i][0]='\0';
    lastseen[i]=-PACKETS-1;
    probestart[i]=-PACKETS-1;
}
for(i=0;i<DEVICES;i++)
    seen[i]=0;



// create, initialize and bind socket
rc=startWinsock();
if(rc!=0)
{
    printf("Fehler: startWinsock, fehler code: %d\n",rc);
    return 1;
}
s=socket(AF_INET,SOCK_DGRAM,0);
if(s==INVALID_SOCKET)
{
    printf("Fehler: Der Socket konnte nicht erstellt werden, fehler code: %d\n",WSAGetLastError());
    getchar();
    return 1;
}
addr.sin_family=AF_INET;
addr.sin_port=htons(1234);
addr.sin_addr.s_addr=ADDR_ANY;
rc=bind(s,(SOCKADDR*)&addr,sizeof(SOCKADDR_IN));
if(rc==SOCKET_ERROR)
{
    printf("Fehler: bind, fehler code: %d\n",WSAGetLastError());
    getchar();
    return 1;
}


// get current time
curtime=CTime::GetCurrentTime();
oldtime=CTime::GetCurrentTime();
dtime=CTime::GetCurrentTime();


////////////////////
// THE MAIN LOOP //
////////////////////

while(1)
{
    // receive incoming packets
    rc=recvfrom(s,buf,sizeof(transobj)+1,0,(SOCKADDR*)&remoteAddr,&remoteAddrLen);
    if(rc==SOCKET_ERROR)
    {
        printf("Fehler: recvfrom, fehler code: %d\n",WSAGetLastError());
        getchar();
        return 1;
    }
    else
    {
        id=inet_ntoa(remoteAddr.sin_addr);
        id=id+10;
        if(seen[atoi(id)-1]==0)
        {
            seen[atoi(id)-1]=1;
            sprintf(namebuf[atoi(id)-1],"%s", ((transobj*)buf)->name);
        }

        // check whether system was rebooted
        if(((transobj*)buf)->index < probestart[atoi(id)-1])
        {
            probestart[atoi(id)-1]=((transobj*)buf)->index;
            lastseen[atoi(id)-1]=((transobj*)buf)->index;
            per[atoi(id)-1]=0.0;
        }

        // check whether there was an AP scan (was not sure whether this would yield packet losses)
        if(apscanflag[atoi(id)-1]>0)
        {
            // AP scan exception handling
            if(((transobj*)buf)->index-lastseen[atoi(id)-1]<PACKETS)
            {
                for(i=lastseen[atoi(id)-1]+1;i<=((transobj*)buf)->index;i++)
                {
                    //log possibly lacking entries here!!!
```

```
            out=fopen(filename,"a");
            fprintf(out,"%s %d *\n",id, i); // * instead of time stamp
            fclose(out);
        }
        lastseen[atoi(id)-1]=((transobj*)buf)->index;
        apscanflag[atoi(id)-1]--;
    }
    else
    {
        apscanflag[atoi(id)-1]=0;
    }
}
else
{
    // new device has just come (back) in range *
    if(((transobj*)buf)->index-lastseen[atoi(id)-1]>PACKETS)
    {
        per[atoi(id)-1]=0.0000000;
        lastseen[atoi(id)-1]=((transobj*)buf)->index;
        probestart[atoi(id)-1]=((transobj*)buf)->index;
    }
    // not enough packets received by a newly arrived device yet
    else if((((transobj*)buf)->index-lastseen[atoi(id)-1]<PACKETS) && (((transobj*)buf)->index-probestart[atoi(id)-1]<PACKETS))
    {
        if(((transobj*)buf)->index != lastseen[atoi(id)-1])
        {
            per[atoi(id)-1]=per[atoi(id)-1]+(((transobj*)buf)->index-lastseen[atoi(id)-1]-1)/packets;
            lastseen[atoi(id)-1]=((transobj*)buf)->index;
        }
    }
    // per computation in steady state
    else if((((transobj*)buf)->index-lastseen[atoi(id)-1]<PACKETS) && (((transobj*)buf)->index-probestart[atoi(id)-1]>=PACKETS))
    {
        if(((transobj*)buf)->index != lastseen[atoi(id)-1])
        {
            per[atoi(id)-1]=per[atoi(id)-1]-(((transobj*)buf)->index - lastseen[atoi(id)-1]) * per[atoi(id)-1] / packets;
            per[atoi(id)-1]=per[atoi(id)-1]+(((transobj*)buf)->index - lastseen[atoi(id)-1]-1)/packets;
            lastseen[atoi(id)-1]=((transobj*)buf)->index;
        }
    }
    else
        printf("error\n");
}

// check, whether it is time to waken rttsend-thread
curtime=CTime::GetCurrentTime();
dtime=curtime-tspan;

// if so, do it
if(dtime>oldtime)
{
    ids[0]='\0';
    names[0]='\0';
    frames[0]='\0';
    rtt[0]='\0';

    for(int i=0;i<DEVICES; i++)
    {
        if(seen[i]==1)
        {
            rttcandidate[i]=1;
            sprintf(ids,"%s%i\n",ids,i+1);
            sprintf(names,"%s%s\n",names,namebuf[i]);
            sprintf(frames,"%s%f\n",frames,per[i]);
            sprintf(rtt,"%s%i\n",rtt,rtts[i]);
            rtts[i]=0;
            //printf("%i\n",rtts[i]);
            //recvcount[i]=0;
            seen[i]=0;
        }
    }
    // ask main thread to initiate window update
    InvalidateRect(hwndMain,&rectId,true);
    InvalidateRect(hwndMain,&rectName,true);
    InvalidateRect(hwndMain,&rectFrames,true);
    InvalidateRect(hwndMain,&rectRtt,true);
    InvalidateRect(hwndMain,&rectGw,true);
    oldtime=curtime;
    PostThreadMessage(rttsendThreadId,WM_NOTIFY,NULL,NULL);
}

//check current logfile size, create new one if too large
cnt++;
```

```
            if(cnt > MAXFILESIZE)
            {
                idx++;
                cnt=0;
                strcpy(filename,"\\CF Card\\logfile");
                _itoa(idx,dum,10);
                strcat(filename,dum);
                strcat(filename,".txt");
            }

            // log received pacekt
            out=fopen(filename,"a");
            fprintf(out,"%s %i %s\n",id,((transobj*)buf)->index,buf);
            fclose(out);
        }
    }
    return 0;
}
```

RTT-Request-Thread

```
int rttrequest() {
    ////////////////////////////////////////
    // DECLARATION AND INITIALIZATION STUFF //
    ////////////////////////////////////////

    // variables
    long rc;
    int i;
    SOCKET s;
    SOCKADDR_IN addr;
    char addrdum[13]={'1','9','2','.','1','6','8','.','0','.','\0','\0','\0'}, id[3], buf1[20], buf2[20];
    MSG msg;

    // all sent packeets start with 'r' for request
    buf2[0]='r'; // request flag
    buf2[1]='\0';

    // socket creation and initialization
    rc=startWinsock();
    if(rc!=0)
    {
        printf("Fehler: startWinsock, fehler code: %d\n",rc);
        getchar();
        return 1;
    }
    s=socket(AF_INET,SOCK_DGRAM,0);
    if(s==INVALID_SOCKET)
    {
        printf("Fehler: Der Socket konnte nicht erstellt werden, fehler code: %d\n",WSAGetLastError());
        getchar();
        return 1;
    }
    addr.sin_family=AF_INET;
    addr.sin_port=htons(2345);



    ////////////////////
    // THE MAIN LOOP //
    ////////////////////
    while(true)
    {
        // wait for "wake up" message and rtt candidates from server-thread
        GetMessage(&msg,NULL,WM_NOTIFY,WM_NOTIFY);

        // send requests to all neighbors
        for(i=0;i<DEVICES;i++)
        {
            if(rttcandidate[i]==1)
            {
                sprintf(id, "%d", i+1);
                strcat(addrdum,id);
                addr.sin_addr.s_addr=inet_addr(addrdum);
                _itoa(GetTickCount(),buf1,10);
                strcat(buf2,buf1);
                rc=sendto (s,buf2,strlen(buf2)+1,0,(SOCKADDR*)&addr,sizeof(SOCKADDR_IN));
                if(rc==SOCKET_ERROR)
```

```
                {
                    PlaySound (TEXT("Alert"), NULL, SND_SYNC);
                    MessageBox(NULL,TEXT("unable to send!\nensure that WLAN is activated!"),NULL,MB_SETFOREGROUND);
                    printf("Fehler: sendto, fehler code: %d\n",WSAGetLastError());
                    getchar();
                    return 1;
                }
                addrdum[10]='\0';
                buf2[1]='\0';
                rttcandidate[i]=0;
                Sleep(SLPERRT2);
            }
        }
    }
    return 0;
}
```

RTT-Acknowledge-Thread

```
int rttacknowledge() {
    ////////////////////////////////////////
    // DECLARATION AND INITIALIZATION STUFF //
    ////////////////////////////////////////

    // variables
    FILE *out;
    char filename[100], dum[4],buf[20];
    long rc, tics, idx=1;
    int remoteAddrLen=sizeof(SOCKADDR_IN), day, hour, min, sec, cnt=0;
    CTime curTime;
    SOCKET recvs, sends;
    SOCKADDR_IN addr;
    SOCKADDR_IN remoteAddr;
    bool flag=true;

    // find current logfile
    while(flag)
    {
        strcpy(filename,"\\CF Card\\rttlog");
        _itoa(idx,dum,10);
        strcat(filename,dum);
        strcat(filename,".txt");
        out=fopen(filename,"r");
        if(out==NULL)
            flag=false;
        else
        {
            idx++;
        }
        fclose(out);
    }

    // create and initialize sockets
    rc=startWinsock();
    if(rc!=0)
    {
        printf("Fehler: startWinsock, fehler code: %d\n",rc);
        return 1;
    }
    recvs=socket(AF_INET,SOCK_DGRAM,0);
    if(recvs==INVALID_SOCKET)
    {
        printf("Fehler: Der Socket konnte nicht erstellt werden, fehler code: %d\n",WSAGetLastError());
        getchar();
        return 1;
    }
    sends=socket(AF_INET,SOCK_DGRAM,0);
    if(sends==INVALID_SOCKET)
    {
        printf("Fehler: Der Socket konnte nicht erstellt werden, fehler code: %d\n",WSAGetLastError());
        getchar();
    }
    addr.sin_family=AF_INET;
    addr.sin_port=htons(2345);
    addr.sin_addr.s_addr=ADDR_ANY;
    rc=bind(recvs,(SOCKADDR*)&addr,sizeof(SOCKADDR_IN));
    if(rc==SOCKET_ERROR)
    {
        printf("Fehler: bind, fehler code: %d\n",WSAGetLastError());
        getchar();
    }
```

```
    ////////////////////
    // THE MAIN LOOP //
    ////////////////////
    while(true)
    {
        // wait for incoming pacekts
        rc=recvfrom(recvs,buf,20,0,(SOCKADDR*)&remoteAddr,&remoteAddrLen);
        if(rc==SOCKET_ERROR)
        {
            printf("Fehler: recvfrom, fehler code: %d\n",WSAGetLastError());
            getchar();
        }
        // check, whether it is a req or ack
        else if(buf[0]=='r')
        {
            // req => reply
            buf[0]='a';
            remoteAddr.sin_port=htons(2345);
            rc=sendto (sends,buf,20,0,(SOCKADDR*)&remoteAddr,sizeof(SOCKADDR_IN));// reply
            if(rc==SOCKET_ERROR)
            {
                PlaySound (TEXT("Alert"), NULL, SND_SYNC);
                MessageBox(NULL,TEXT("unable to send!\nensure that WLAN is activated!"),NULL,MB_SETFOREGROUND);
                printf("Fehler: sendto, fehler code: %d\n",WSAGetLastError());
                getchar();
            }
        }
        else if(buf[0]=='a') // acknowledge
        {
            // ack => compute rtt, log it
            tics=GetTickCount();
            rtts[atoi(inet_ntoa(remoteAddr.sin_addr)+10)-1]=tics-atoi(buf+1);


            curTime=CTime::GetCurrentTime();

            day=curTime.GetDay();
            hour=curTime.GetHour();
            min=curTime.GetMinute();
            sec=curTime.GetSecond();

            // if logfile larger than 1 MB create new one
            cnt++;
            if(cnt > MAXFILESIZE)
            {
                idx++;
                cnt=0;
                strcpy(filename,"\\CF Card\\rttlog");
                _itoa(idx,dum,10);
                strcat(filename,dum);
                strcat(filename,".txt");
            }

            // log rtt
            out=fopen(filename,"a");
            fprintf(out,"%s %d %d.%d.%d.%d\n", inet_ntoa(remoteAddr.sin_addr)+10,rtts[atoi(inet_ntoa(remoteAddr.sin_addr)+10)-1],day, hour, min, sec);
            fclose(out);
        }
        else
        {
            printf("received rtt buffer truncated\n");
            printf("\n**********\n%c\n*********\n",buf[0]);
        }
    }

    return 0;
}
```

Gateway-Thread

```
int gatewayconnect() {
    ////////////////////////////////////////
    // DECLARATION AND INITIALIZATION STUFF //
    ////////////////////////////////////////

    //variables
    char *id=NULL, dummy[1000], filename[100], buf[sizeof(transobj)+1];
    long gwstamp=0, recvgwstamp=0, dum=0, hops=0;
    int remoteAddrLen=sizeof(SOCKADDR_IN), gateway=0,i, day, hour, min, sec;
    long rc, cnt=0, idx=1;
```

```
bool flag=true;
FILE *out;
SOCKET sends, recvs;
SOCKADDR_IN addr;
SOCKADDR_IN remoteAddr;
CTime curTime, oldTime;
CTimeSpan span=CTimeSpan(0,0,0,GATEWAYTIME);

// find current logfile
while(flag)
{
    strcpy(filename,"\\CF Card\\gatewaylog");
    _itoa(idx,dummy,10);
    strcat(filename,dummy);
    strcat(filename,".txt");
    out=fopen(filename,"r");
    if(out==NULL)
        flag=false;
    else
    {
        idx++;
    }
    fclose(out);
}

// update display by calling main thread
sprintf(gws,"looking for gateway\n");
InvalidateRect(hwndMain,&rectGw,true);

// creating, initialzing and binding sockets
rc=startWinsock();
if(rc!=0)
{
    printf("Fehler: startWinsock, fehler code: %d\n",rc);
    return 1;
}
recvs=socket(AF_INET,SOCK_DGRAM,0);
if(recvs==INVALID_SOCKET)
{
    printf("Fehler: Der Socket konnte nicht erstellt werden, fehler code: %d\n",WSAGetLastError());
    getchar();
    return 1;
}
sends=socket(AF_INET,SOCK_DGRAM,0);
if(sends==INVALID_SOCKET)
{
    printf("Fehler: Der Socket konnte nicht erstellt werden, fehler code: %d\n",WSAGetLastError());
    getchar();
    return 1;
}
addr.sin_family=AF_INET;    // my address
addr.sin_port=htons(4321); // different port chosen!!!
addr.sin_addr.s_addr=ADDR_ANY;
rc=bind(recvs,(SOCKADDR*)&addr,sizeof(SOCKADDR_IN));
if(rc==SOCKET_ERROR)
{
    printf("Fehler: bind, fehler code: %d\n",WSAGetLastError());
    getchar();
    return 1;
}
addr.sin_family=AF_INET; //destination address
addr.sin_port=htons(4321);
addr.sin_addr.s_addr=inet_addr("192.168.0.255");

// get current time
curTime=CTime::GetCurrentTime();
oldTime=curTime;



////////////////////
// THE MAIN LOOP //
////////////////////

while(true)
{
    // wait for incoming packets
    rc=recvfrom(recvs,buf,sizeof(transobj)+1,0,(SOCKADDR*)&remoteAddr,&remoteAddrLen);
    if(rc==SOCKET_ERROR)
    {
        printf("Fehler: recvfrom, fehler code: %d\n",WSAGetLastError());
        getchar();
        return 1;
```

```
        }
        else
        {
            gwstamp=((transobj*)buf)->index; // gwindex in index field
            hops=atoi(((transobj*)buf)->time); // no of hops in time field

            // if message from own device update display
            if(gwstamp==-1) // message from own device
            {
                curTime=CTime::GetCurrentTime();
                if(curTime-span>oldTime) // check whether received message from gateway within last 20 sec
                {
                    gateway=0; // if not set gateway to 0
                    sprintf(gws,"no connection to gateway\n");
                    InvalidateRect(hwndMain,&rectGw,true);   //update rect in window
                }
            }
            else
            {
                // check whether there was a reboot
                if(gwstamp<recvgwstamp-100)
                {
                    recvgwstamp=gwstamp;
                }

                // if packet has not been seen yet, log and forward it
                if(gwstamp>recvgwstamp)
                {
                    // forwarding
                    id=inet_ntoa(remoteAddr.sin_addr);
                    id=id+10;
                    hops++;
                    _itoa(hops,dummy,10);
                    sprintf(((transobj*)buf)->time,"%s",dummy); // hops updated
                    if(strcmp(id,"20"))
                    {
                        sprintf(((transobj*)buf)->name,"%s %s",((transobj*)buf)->name,id); // relays updated
                    }
                    rc=sendto (sends,buf,sizeof(transobj)+1,0,(SOCKADDR*)&addr,sizeof(SOCKADDR_IN)); // forward message
                    if(rc==SOCKET_ERROR)
                    {
                        PlaySound (TEXT("Alert"), NULL, SND_SYNC);
                        MessageBox(NULL,TEXT("unable to send!\nensure that WLAN is activated!"),NULL,MB_SETFOREGROUND);
                        printf("Fehler: sendto, fehler code: %d\n",WSAGetLastError());
                        getchar();
                        return 1;
                    }

                    // register packet as received and update display string
                    recvgwstamp=gwstamp;
                    sprintf(gws,"gateway reached - %d hop(s)\nrelays: %s", hops,((transobj*)buf)->name);

                    // logging
                    curTime=CTime::GetCurrentTime();
                    oldTime=curTime;
                    day=curTime.GetDay();
                    hour=curTime.GetHour();
                    min=curTime.GetMinute();
                    sec=curTime.GetSecond();

                    // check whether log file has reached maximum size
                    cnt++;
                    if(cnt > MAXFILESIZE)
                    {
                        idx++;
                        cnt=0;
                        strcpy(filename,"\\CF Card\\gatewaylog");
                        _itoa(idx,dummy,10);
                        strcat(filename,dummy);
                        strcat(filename,".txt");
                    }

                    // log packet
                    out=fopen(filename,"a");
                    fprintf(out,"%d %d %s %d.%d.%d.%d\n", hops,gwstamp,((transobj*)buf)->name,day, hour, min, sec);
                    fclose(out);
                }
            }
        }
    }
    return 0;
}
```

Appendix B

Implementation of Mobility Models

B.1 Random Waypoint Model

```
% rectangular floor
a=100; b=20;

%number of devices
DEVICES=20;

% initial positions and targets
xt=a*rand(DEVICES,1);
yt=b*rand(DEVICES,1);
x=a*rand(DEVICES,1);
y=b*rand(DEVICES,1);

% number of considered samples
SAMPLES=5000;
% time between to samples in s
st=7;

% velocity in m/s
v=1.0;
d=v*st%*ones(DEVICES,1);

% thinking time in s (has to be a multiple of sample time)
t0=floor(.5*1806*rand(DEVICES,1)/7)*7; %3*1750
t=.5*1806*ones(DEVICES,1);

% threshhold for contact
rth=25;

% open file
fid=fopen('C:\\Documents and
Settings\\tikadmin\\Desktop\\AnalyseTool\\netlist.txt','w');

for i=1:SAMPLES
    i

    % set new target if thinking time has elapsed, otherwise sustain old target
    xt=a*rand(DEVICES,1).*(t0==0)+xt.*(t0~=0);
    yt=b*rand(DEVICES,1).*(t0==0)+yt.*(t0~=0);

    % determine distance to walk (either maximum or till destination)
    d0=(d < sqrt((yt-y).^2+(xt-x).^2)).*d + (d >= sqrt((yt-y).^2+(xt-x).^2)).*sqrt((yt-y).^2+(xt-x).^2);
    % check whether target is arrived with this step
    arrived=(sqrt((yt-y).^2+(xt-x).^2) == d0);

    % walk if t0<=0, wait otherwise
    x0=x;
    xt0=xt;
    y0=y;
    yt0=yt;
    x=(x0+d0.*(xt0-x0)./sqrt((yt0-y0).^2+(xt0-x0).^2)).*(t0<=zeros(DEVICES,1))+x0.*(t0>0);
    y=(y0+d0.*(yt0-y0)./sqrt((yt0-y0).^2+(xt0-x0).^2)).*(t0<=zeros(DEVICES,1))+y0.*(t0>0);

    % decremeant thinking time
```

```
    t0=t0-st;
    t0=t.*arrived+t0.*(1-arrived);

    %  reset after having arrived at target
    xt=-100*arrived+xt.*(1-arrived); % ensure that arrived flag is
    yt=-100*arrived+yt.*(1-arrived); % not set again in next round
    arrived=zeros(DEVICES,1);

    % get radii
    Xq=(kron(ones(1,DEVICES),x)-kron(x',ones(DEVICES,1))).^2;
    Yq=(kron(ones(1,DEVICES),y)-kron(y',ones(DEVICES,1))).^2;
    R=sqrt(Xq+Yq);

    % convert R to adjadcence matrix
    A= R < (rth * ones(DEVICES,DEVICES));


    % write netlist to logfile
    for aa=1:DEVICES
        for b=1:DEVICES
            fprintf(fid,'%d',A(aa,b));
        end
    end
    fprintf(fid,'b');



end


fclose(fid);
```

B.2 Random Reference Point Mobility Model

```
% number of devices
DEVICES=20;


% open file
fid=fopen('C:\\Documents and
Settings\\tikadmin\\Desktop\\AnalyseTool\\netlist.txt','w');



% rectangular floor
a=100; b=20;

% groups and their size
groups=[1,6;7,9;10,11;12,13;14,15;16,16;17,17;18,18;19,19;20,20];
groupsize=groups(:,2)-groups(:,1)+1;


% threshhold for contact
rth=15;

% number of considered samples
SAMPLES=5000;
% time between to samples in s
st=7;

% velocity in m/s
v=1.0; d=v*st;

%thinking time in s (has to be a multiple of sample time)
t0c=round(1.05*3605*rand(length(groupsize),1)/7)*7;
tc=round(1.05*3605*ones(length(groupsize),1)/7)*7;
t0e=round(1.05*1806*rand(DEVICES,1)/7)*7; %3*1750
te=round(1.05*1806*ones(DEVICES,1)/7)*7;

% initial positions and targets
xte=.33*a*rand(DEVICES,1); yte=b*rand(DEVICES,1);
xe=.33*a*rand(DEVICES,1); ye=b*rand(DEVICES,1);
xtc=.66*a*rand(length(groups),1); ytc=zeros(length(groups),1);
xc=.66*a*rand(length(groups),1); yc=zeros(length(groups),1);
```

```
for i=1:SAMPLES
    i

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % RANDOM WALK OF CENTER POINTS %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % set new target if thinking time has elapsed, otherwise sustain old target
    xtc=.66*a*rand(length(groupsize),1).*(t0c==0) + xtc.*(t0c~=0);
    ytc=0*b*rand(length(groupsize),1).*(t0c==0) + ytc.*(t0c~=0);%b*rand(length(groupsize),1).*(t0c==0)+ytc.*(t0c~=0);


    % determine distance to walk (either maximum or till destination)
    d0c=(d < sqrt((ytc-yc).^2+(xtc-xc).^2)).*d + (d >= sqrt((ytc-yc).^2+(xtc-xc).^2)).*sqrt((ytc-yc).^2+(xtc-xc).^2);
    % check whether target is arrived with this step
    arrivedc=(d >= sqrt((ytc-yc).^2+(xtc-xc).^2));


    % walk if t0<=0, wait otherwise
    xtc0=xtc;
    xc0=xc;
    ytc0=ytc;
    yc0=yc;
    xc=(xc0+d0c.*(xtc0-xc0)./sqrt((ytc0-yc0).^2+(xtc0-xc0).^2)).*(t0c<=0)+xc0.*(t0c>0);
    yc=(yc0+d0c.*(ytc0-yc0)./sqrt((ytc0-yc0).^2+(xtc0-xc0).^2)).*(t0c<=0)+yc0.*(t0c>0);

    % decrement thinking time
    t0c=t0c-st;
    t0c=tc.*arrivedc+t0c.*(1-arrivedc);

    %  reset after having arrived at target
    xtc=(-rand(1,1)-100)*arrivedc+xtc.*(1-arrivedc); % ensure that arrived flag is
    ytc=(-rand(1,1)-100)*arrivedc+ytc.*(1-arrivedc); % not set again in next round
    arrivedc=zeros(length(groupsize),1);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % RANDOM WALK OF MOBILE NODES %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    xte=.33*a*rand(DEVICES,1).*(t0e==0)+xte.*(t0e~=0);
    yte=b*rand(DEVICES,1).*(t0e==0)+yte.*(t0e~=0);%b*rand(length(groupsize),1).*(t0c==0)+ytc.*(t0c~=0);


    % determine distance to walk (either maximum or till destination)
    d0e=(d < sqrt((yte-ye).^2+(xte-xe).^2)).*d + (d >= sqrt((yte-ye).^2+(xte-xe).^2)).*sqrt((yte-ye).^2+(xte-xe).^2);
    % check whether target is arrived with this step
    arrivede=(sqrt((yte-ye).^2+(xte-xe).^2) == d0e);


    % walk if t0<=0, wait otherwise
    xte0=xte;
    xe0=xe;
    yte0=yte;
    ye0=ye;
    xe=(xe+d0e.*(xte-xe)./sqrt((yte-ye).^2+(xte-xe).^2)).*(t0e<=zeros(DEVICES,1))+xe.*(t0e>0);
    ye=(ye+d0e.*(yte-ye)./sqrt((yte-ye).^2+(xte-xe).^2)).*(t0e<=zeros(DEVICES,1))+ye.*(t0e>0);

    % decremeant thinking time
    t0e=t0e-st;
    t0e=te.*arrivede+t0e.*(1-arrivede);

    %  reset after having arrived at target
    xte=-100*arrivede+xte.*(1-arrivede); % ensure that arrived flag is
    yte=-100*arrivede+yte.*(1-arrivede); % not set again in next round
    arrivede=zeros(DEVICES,1);


    % overlay movements
    tmp=[];
    for k=1:length(groupsize)
        tmp=[tmp;kron([xc(k),yc(k)],ones(groupsize(k),1))];
    end
    x=tmp(:,1)+xe;
    y=tmp(:,2)+ye;



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % TOPOLOGY RECONSTRUCTION %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % get radii
    Xq=(kron(ones(1,DEVICES),x)-kron(x',ones(DEVICES,1))).^2;
    Yq=(kron(ones(1,DEVICES),y)-kron(y',ones(DEVICES,1))).^2;
```

```
    R=sqrt(Xq+Yq);

    % convert R to adjadcence matrix
    A= R < (rth * ones(DEVICES,DEVICES));


    % write netlist to logfile
    for aa=1:DEVICES
        for b=1:DEVICES
            fprintf(fid,'%d',A(aa,b));
        end
    end
    fprintf(fid,'b');

end

fclose(fid);
```

Appendix C

Windows CE Developer Platforms

This chapter shall provide a survey of available development environments for Windows CE based operating systems such as "Windows Mobile 2003", which is run by the handhelds used in this project. Most environments are provided by Microsoft itself. However there are also some Java based environments.

C.1 Microsoft Development Tools

C.1.1 eMbedded Visual C++ 4.0

Short Description: Microsoft eMbedded Visual C++ 4.0 is a stand-alone integrated development environment, i.e., it does not require any additional development environments, such as Microsoft Visual Studio. It allows to create applications and system components for Windows CE 4.2 based devices. In order to generate code for a specific device it requires a Software Development Kit (SDK) depending on the platform to target. eMbedded Visual C++ 4.0 includes the native code C and C++ compilers and is therefore well suited for the development of drivers or any other device applications that run natively on the device. It provides a just-in-time debugger for diagnosing unhandled exceptions, C++ structured exception handling and integration with the Pocket PC emulator provided by Microsoft. eMbedded Visual C++ code is based on the Win32 API, the Microsoft Foundation Classes and the ATL APIs. Generally speaking appearance and handling are very similar to the classical Visual C++ in the Visual Studio environment for desktop machines. In contrast to Visual Studio .NET (see next section) the standard Visual Studio environment does not support development for Windows CE based systems.

Purchasing: eMbedded Visual C++ 4.0 and several SDKs are available for free download on the Microsoft web pages.

eMbedded Visual C++ 4.0 is the development tool chosen for the accomplishment of this student project, since it is the only tool that allows immediate access to the network interface card. Further it is the only environment supporting C or C++ respectively. Using C/C++

is of particular importance as the network device interface specification (NDIS) provided by Microsoft for accessing network cards can be easily used by these languages.

C.1.2 Visual Studio .NET

Short Description: The Visual Studio .NET environment known from desktop programming is the second widely spread IDE provided by Microsoft suitable for development for Windows CE. As embedded Visual C++ it requires a specific SDK and toolkit in order to target the device. Unlike eMbedded Visual C++ it supports the languages C# and Visual Basic .NET and therefore builds managed instead of native code. Consequently it is easily portable between different devices running an appropriate interface, but on the other hand not capable of accessing any devices immediately. Instead of Win32 API, MFC and ATL it uses the .NET Compact Framework interface, which has to be installed on the handheld device. The .NET Compact framework is a smaller and slightly different version of the full .NET framework for desktop devices. The extensive class library available through the .NET Compact Framework allows applications to be written much faster than with eMbedded Visual C++ 4.0.

Purchasing: Depending on the edition Visual Studio. NET costs between 800 and $2500.

C.2 Java Based Development Tools

Short Overview: There are several JVMs for the Pocket PC (J2ME) on the market. Most of them, however, suffer from more or less severe problems: Sun's PeronalJava is not supported anymore, SuperWaba and EWE are not fully Java compatible, some others are on sale to OEMs only or out of date. Essentially IBM's J9 JVM and NSICOM's CrEme JVM are the only ones that are both fully Java compatible and supported.
vspace0.3cm Since J2ME/CDC APIs are close to regular desktop Java (J2SE) APIs, in the majority of cases it is possible to use regular Java IDEs such as Eclipse, JBuilder, or Sun Java Studio.

The advantages and disadvantages of Java based development tools are similar to those of Visual Studio .NET. Of course the hardware abstraction done by the JVM does not allow immediate access to devices or memory any more. However applications written in Java are portable to any device running a JVM.

Purchasing: The first one can be purchased from www.handango.com for $5.99, NSICOM's CrEme JVM is available as a free trial download under www.nsdicom.com, developer licenses can be purchased for $25.