On Real-World Experiments with Wireless Multihop Networks

—

Design, Realization, and Analysis

Inaugural-Dissertation

zur
Erlangung des Doktorgrades der
Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Wolfgang Kiess

aus Künzelsau

April 2008

Aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Referent:	Prof. Dr. Martin Mauve Heinrich-Heine-Universität Düsseldorf
Koreferent:	Prof. Dr. Stefan Conrad Heinrich-Heine-Universität Düsseldorf

Tag der mündlichen Prüfung: 03.06.2008

Abstract

In wireless multihop networks (WMN), nodes cooperate to forward data packets for each other. This forwarding works without infrastructure, being a huge advantage if no such infrastructure is available, e.g. because it has been destroyed by a disaster. Furthermore, this networking paradigm is also promising in the context of vehicular safety and traffic efficiency applications. After years of simulation-based research, the next step in the development of this paradigm is its evaluation under real-world conditions. However, due to the distributed nature of such a network in combination with the complex effects of electromagnetic wave propagation, it is extremely difficult to perform these experiments systematically. In this thesis, we tackle the fundamental problems of the control and analysis of such experiments.

Our first step is to develop a guidebook of existing wireless multihop network experimentation techniques. Furthermore, we present our initial experiments, among them the first large-scale real-world study of ring flooding which reveals that even this simple algorithm exhibits complex, unexpected behavior in realistic settings. The experiences made during these evaluations as well as those made by other researchers are condensed into a description of requirements to be fulfilled by an ideal WMN testbed. Repeatability, comprehension and correctness have been especially neglected so far and are crucial for systematic experiments.

With this knowledge, we develop the EXC testbed based on semi-automatic experiment control. This control approach automates most actions while the experimenter still can supervise and flexibly steer the experiment. EXC is a modular and highly portable software toolkit allowing other researchers to create their own testbed installation and thus test their protocols in the very environment for which they are designed.

Controlling and analyzing WMN experiments requires a timekeeping accuracy that exceeds the quality of normal computer clocks. The standard solution, using online clock synchronization protocols like NTP, cannot be applied as this requires a network connection to a reference clock which would interfere with the experiment traffic. To support

the control of the experiment, we exploit the capability of the NTP daemon to correct clock speed when disconnected from the reference clock. We have performed a study of the timekeeping quality achieved by this approach on devices typically used in WMN experiments. It demonstrates that this increases clock precision by two orders of magnitude, reaching millisecond precision. However, for experiment analysis this precision is not sufficient. Therefore we created a post-experiment timestamp synchronization algorithm by means of a maximum likelihood estimator (MLE) that is suited for all networks with local broadcast media. It estimates the clock deviations based on the recorded event log files of the single nodes and synthesizes globally consistent timestamps for these events. In our experimental evaluation, it exhibits an error in microsecond range. The MLE approach is integrated in `pcapsync`, a tool to synchronize packet trace files in standard libpcap format.

To cope with the need of flexible data analysis after an experiment, we have developed the modular data analysis tool EDAT. It follows a flow-based, visual programming approach and produces graphs directly usable in scientific publications, a large fraction of the graphs in this thesis have been created with this tool.

Combining EXC, `pcapsync`/MLE timestamp synchronization and EDAT, we perform the first systematic study on experimental repeatability in wireless multihop networks. Up to now, most often it was implicitly assumed that if all devices perform the same actions in two experiments, also the outcome will be somewhat similar and can therefore be compared or averaged. Due to the complex electromagnetic wave propagation effects, this is a risky assumption. Therefore, we propose to consider and verify repeatability on a topological level based on layer two information. We derive the AD metric to quantify the topological similarity of experiments and show that it is sensitive to both interference and changes in node movement. This metric is used to examine – in strictly controlled experiments – topology variance in real-world environments.

Zusammenfassung

In drahtlosen Multihop-Netzwerken (engl. WMN) kooperieren die beteiligten Knoten um füreinander gegenseitig Datenpakete weiterzuleiten. Die Weiterleitung erfolgt dabei ohne Infrastruktur, was einen großen Vorteil darstellt, wenn eine solche beispielsweise nach einer Naturkatastrophe nicht verfügbar ist. Daneben kann dieses Netzwerkparadigma auch im Kontext von fahrzeugbasierten Verkehrsicherheits- und Verkehrseffizienzanwendungen genutzt werden. Nach Jahren der simulationsbasierten Forschung ist der nächste Schritt in der Entwicklung dieses Paradigmas dessen Bewertung und Erforschung unter realistischen Bedingungen. Da es sich bei WMNs um verteilte Netzwerke handelt die zudem den komplexen Effekten der elektromagnetischen Signalausbreitung unterworfen sind, ist es äußerst schwierig solche Experimente systematisch durchzuführen. In dieser Arbeit werden Lösungen für die bei der Durchführung und Analyse solcher Experimente auftretenden fundamentalen Probleme untersucht und präsentiert.

Im ersten Schritt entwickeln wir dazu ein Handbuch das existierende Techniken zur Durchführung und Bewertung solcher Experimente behandelt. Daneben präsentieren wir eigene Experimente, darunter die erste großflächige experimentelle Studie über das Verhalten von Ringfluten. Diese Studie demonstriert, dass selbst dieser einfache Algorithmus unter realistischen Bedingungen ein komplexes, unerwartetes Verhalten zeigt. Die dabei gewonnen Erfahrungen werden mit denen anderer Wissenschaftler zu einem Anforderungskatalog für ein WMN Testbett verdichtet. Dabei zeigt sich, dass besonders Wiederholbarkeit, Verständnis und Korrektheit bisher vernachlässigt wurden und einen integralen Bestandteil von systematischen Experimenten bilden.

Basierend auf diesem Wissen wurde das EXC-Testbett entwickelt, welches auf einer halbautomatischen Kontrolle von Experimenten beruht. Dieser Ansatz für die Experimentdurchführung automatisiert die meisten Aktionen der beteiligten Geräte und erlaubt es dennoch, das Experiment zu überwachen und flexibel zu steuern. EXC ist ein modulares, hochportierbares Software-Werkzeug das es anderen Wissenschaftlern ermöglicht ein eigenes Testbett aufzubauen und neue Algorithmen in genau der Umgebung zu testen für die diese entwickelt wurden.

Die Durchführung und Analyse von WMN-Experimenten erfordert Uhrgenauigkeiten, die die von normalen Computeruhren weit überschreiten. Der Standardansatz, die Synchronisation der Uhren über eine Netzwerkverbindung mittels des NTP-Protokolls, ist hierbei nicht anwendbar da die dabei ausgetauschten Datenpakete das Experiment stören können. Um die Durchführung von Experimenten zu unterstützen nutzen wir deshalb die Fähigkeit des NTP-Deamons zur Korrektur der Uhren ohne bestehende Netzwerkverbindung. In Messungen mit bei WMN Experimenten oft eingesetzter Hardware zeigt sich, dass die Uhrgenauigkeit damit um zwei Größenordnungen verbessert werden kann, im aktuellen Fall betragen die Unterschiede nur noch wenige Millisekunden. Dennoch ist diese Genauigkeit für die Analyse von Experimenten nicht ausreichend. Deswegen wurde von uns ein auf der Maximum-Likelihood-Methode (engl. MLE) basierendes Verfahren zur nachträglichen Synchronisation von Zeitstempeln entwickelt, das für alle Netzwerke mit lokalen Broadcasteigenschaften eingesetzt werden kann. Dieses Verfahren schätzt die Uhrenfehler mittels der aufgezeichneten Logdateien und erzeugt basierend auf dieser Schätzung global konsistente Zeitstempel für die aufgetretenen Ereignisse. In einer experimentellen Auswertung hat dieses Verfahren einen Fehler im Mikrosekundenbereich. Dieses Verfahren ist auch in `pcapsync` integriert, einem Werkzeug zur Synchronisation von Paketlogdateien im weit verbreiteten libpcap-Format.

Um ein Experiment nach dessen Ende einfach und gleichzeitig flexibel analysieren zu können, wurde im Rahmen dieser Arbeit das modulare Datenanalysewerkzeug EDAT entwickelt. Es nutzt einen datenflußbasierten, visuellen Ansatz und kann direkt in wissenschaftlichen Publikationen verwendbare Diagramme erzeugen. Dies wird auch durch die Tatsache unterstrichen, dass ein Großteil der in dieser Arbeit gezeigten Diagramme mit diesem Werkzeug erstellt wurden.

Durch die Kombination von EXC, `pcapsync`/MLE-Zeitstempel-Synchronisation und EDAT konnten wir die erste systematische Studie zur Wiederholbarkeit von WMN Experimenten durchführen. Bisher wurde meist implizit davon ausgegangen, dass identisches Knotenverhalten in zwei Experimenten auch zu identischen Ergebnissen führt. Aufgrund der komplexen Effekte elektromagnetischer Signalausbreitung ist dies jedoch eine riskante Annahme. Deswegen betrachten wir Wiederholbarkeit auf der Ebene der Netzwerktopologie. Mittels der neu entwickelten AD-Metrik ist es möglich, die Ähnlichkeit zweier Topologien quantitativ zu bestimmen. Wir zeigen, dass diese Metrik sowohl mit Interferenzen als auch mit Änderungen in den Knotenbewegungen umgehen kann. In streng kontrollierten Experimenten wird untersucht wie groß die tatsächlich auftretenden Topologieänderungen in realistischen Umgebungen sind.

Acknowledgments

The basic motivation for this thesis can be traced back to the days of my diploma thesis in May 2003. Quite satisfied with the simulation results, I attended a presentation about the experimental evaluation of the Fleetnet Router that took place some weeks thereafter. It was quite a shock to see how difficult it was to conduct meaningful experiments with only four nodes, while in simulations, networks with more than a thousand nodes were studied. This led to the insight that it must be possible to systematically conduct and examine real-world experiments if this networking paradigm should ever be useful in realistic environments. The efforts undertaken to solve this problem are documented in this thesis, and I am deeply thankful to all my colleagues, friends and my family for assisting and encouraging me.

First of all, I would like to thank my advisor Martin Mauve who invaluably supported me throughout the last years and finally brought me to the point of finishing this thesis. He guided me through the painful experience of writing my first research paper and set me back on track whenever my ideas took me off the way thereafter. Furthermore, I would like to thank Stefan Conrad who agreed to be referee for this thesis.

This thesis would not exists without the lively discussions, the invaluable feedback, and all the other contributions of Björn Scheuermann, Christian Lochert, Jedrzej Rybicki, and Michael Stini and all my other colleagues at the Computer Networks and Communication Systems group of the University of Düsseldorf. The fruitful atmosphere I was allowed to work in is best documented by the various papers we have co-authored. A special "thank you" goes (again) to Björn Scheuermann and Florian Jarre from the Mathematics Department of the University of Düsseldorf for making the time synchronization papers possible and to Ryan Plocher for proof-reading the thesis.

Furthermore, I owe gratitude to Holger Füßler and Jörg Widmer from the University of Mannheim. They supervised my diploma thesis and supported me in writing the HLS paper.

Implementing the numerous tools created during this thesis, conducting the experiments, and developing many of the ideas presented in this thesis required countless hours of programming, thinking and walking around the campus. A lot of this work has been conducted during students' thesis, master-level projects and by my student helpers. I am in depth to all those who worked with me during the last years on this project. Among them are Stephan Zalewski, Andreas Tarp, Thomas Ogilvie, Markus Kerper, Magnus Roos, Nadine Chmill, and Ulrich Wittelsbürger.

Marga Potthoff and Christian Knieling prevented me from getting lost in the complexity of computer- as well as university-system. While Marga guided me through the bureaucracy of the university and handled all the day-to-day tasks with impressive patience and knowledge, Christian was the go-to-guy, the one to ask when I needed a certain piece of software installed on one of the experiment computers or a special, exotic configuration for a certain network card.

Life would be nothing without friends and I am deeply thankful for my friends in Düsseldorf, Mannheim, Köln, München, and all over the rest of the world. You inspire me, are very different and at the same time very similar, open up my mind to music, sports, and art, show me what is important and what is not, and with this teach me what life really is about.

This thesis is dedicated to my family, my sister Carolin, my brothers Christian and Johannes, and especially to my parents, Gerhard and Marianne Kiess. They always supported and encouraged me and allowed me to become whatever I wanted. By opening all doors, they also opened that special one that I stride through with this thesis, thus it is their merit.

And finally: thanks to old god Neptune for providing the waves!

Contents

List of Figures

List of Tables

List of Abbreviations

AODV	Ad-hoc On-demand Distance Vector routing
APE	Ad-hoc Protocol Evaluation testbed
API	Application Programming Interface
APRL	Any Path Routing without Loops
ARM	Advanced RISC Machine
ARP	Address Resolution Protocol
CD	Compact Disk
COTS	Commercial-Off-The-Shelf
CPU	Central Processing Unit
CRAWDAD	Community Resource for Archiving Wireless Data At Dartmouth
DARPA	Defense Advanced Research Projects Agency
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
DTN	Delay Tolerant Network
EDAT	Extensible Data Analysis Tool
ETX	Expected Transmission Count
EXC	EXperiment Control
FRANC	FRamework for Ad hoc Network Communication
FTP	File Transfer Protocol
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
GUI	Graphical User Interface
HLS	Hierarchical Location Service
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol

IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO	International Organization for Standardization
LAD	Least Absolute Deviation
LGR	Logical Grid Routing
LP	Linear Program
MAC	Medium Access Control
MANET	Mobile Ad-hoc NETwork
MLE	Maximum Likelihood Estimator
MRT	Multi-threaded Routing Toolkit
ns	network simulator
NTP	Network Time Protocol
ntpd	NTP daemon
ODMRP	On-Demand Multicast Routing Protocol
OLSR	Optimized Link State Routing
OML	ORBIT Measurements framework and Library
ORBIT	Open access Research testBed for next-generation wIreless neTworks
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PDA	Personal Digital Assistant
PDR	Packet Delivery Ratio
ppm	parts per million
PRNET	Packet Radio NETwork
RBP TCP	Rate-Based Pacing Transmission Control Protocol
RFC	Request For Comments
RISC	Reduced Instruction Set Computer
RMI	Remote Method Invocation
RSS	Received Signal Strength

SER	Simulation Emulation Real-world
SNR	Signal-to-Noise Ratio
SQL	Structured Query Language
SRI	Shared Research Infrastructure
STARA	System- and Traffic-dependent Adaptive Routing Algorithm
SURAN	SURvivable Adaptive Networks
TBRPF	Topology Broadcast based on Reverse-Path Forwarding
TCP	Transmission Control Protocol
TSC	Time-Stamp Counter
TTL	Time-To-Live
UDAAN	Utilizing Directional Antennas for Ad Hoc Networking
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VANET	Vehicular Ad-hoc NETwork
WLAN	Wireless Local Area Network
WMN	Wireless Multihop Network
WSN	Wireless Sensor Network
XML	eXtensible Markup Language

No one believes an hypothesis except its originator,
but everybody believes an experiment except the experimenter.
William Ian Beardmore Beveridge

Chapter 1

Introduction

Communication networks have strongly influenced everyday life since the invention of semaphores and telegraphs. Nowadays, telephones, the Internet and cellular wireless communication are used by billions of people around the world. All these networks have in common that they rely on infrastructure, be it the routers of the Internet, the circuits of the telephone network or the radio towers of the cell phone system. In the early 70's [LNT87], a different type of network paradigm was proposed. Instead of relying on infrastructure, the nodes in such networks cooperate and transmit packets for each other in a collaborative manner. The range of possible applications of this paradigm is constantly increasing and has led to a number of different network types. In vehicular ad-hoc networks (VANETs), cars exchange information about current traffic or emergency situations, thus allowing drivers to react early and appropriately. Via mesh networks the range of wireless access points can be extended to cover bigger areas with fewer cables, and multihop networks can continuously connect firefighters with each other and their headquarters to increase security and responsiveness. Wireless sensor networks (WSNs) enable new forms of sensing applications, e.g. surveying the activity of a volcano with high precision at low cost, and delay tolerant networks (DTNs) can transmit information to otherwise disconnected areas. All these networks are based on the same collaborative paradigm and use wireless technology to transmit information. These networks will therefore be summed up under the generic term *wireless multihop network* (WMN) in this thesis and the more specific term *mobile ad-hoc network* (MANET) will be used when focusing on WMNs with mobile nodes.

Although the basic idea of wireless multihop networks arose nearly 30 years ago, the two most common instruments to evaluate WMN algorithms are theoretical analysis and simulation, while real-world experiments are rare. Where theoretical analysis provides fundamental insights into the characteristics of the investigated approaches, simulation

enables their exploration in a dynamic environment. Both methods require a significant level of abstraction to reduce the real-world complexity of radio propagation, hardware, and mobility. It has been repeatedly shown that due to these simplifications, the direct transfer of findings, e.g. from simulations to real-world systems, is not advisable [LNT02, LYN+04, KNG+04].

To progress in the development of wireless multihop networks, it is necessary to complement theoretical analysis and simulation by real-world experiments: If the behavior of networks and algorithms is evaluated *on the devices* and *in the environment* for which they have been designed, none of the possibly misleading simplifications need to be made. Therefore, experimental evaluation is the key to a better understanding of this networking paradigm. However, an examination of existing experiments reveals that most real-world evaluations conducted so far lack a precise methodology. The result is the proof-of-concept of a certain algorithm or protocol but it is difficult to gain verifiable knowledge without a systematic approach. Furthermore, a lot of work is duplicated as the software tools created during these evaluations are quickly hacked prototypes. To overcome these issues, we concentrate in this thesis on methods and techniques for a structured experimental evaluation of WMN algorithms. We examine the different stages of an experiment and show how these stages can be supported by methodologies and tools that are all designed for extensibility and reusability. It is the goal of this thesis to make experiments with wireless multihop networks feasible and lower the effort necessary for them. Our main contributions are as follows:

- We perform the first large scale study of ring flooding and discover that even this simple algorithm exhibits a complex, unexpected behavior in realistic settings. It is shown that flooding with the network diameter d is not enough if all nodes should be reached and that the often used term of an n-hop neighborhood must be revised.
- We provide a guidebook that condenses information about the experiences made in existing experiments. It lists the most common tools, movement patterns, traffic models and concepts and thus lowers the duplication of work.
- Based on this knowledge as well as on our own experiences, we develop a methodology and the EXC toolkit to conduct tightly controlled real-world experiments in wireless multihop networks.
- With the help of EXC we perform the first systematic study on the repeatability of experiments with wireless multihop networks. In the context of this study, we

show how the differences of static and mobile topologies can be quantified solely based on packet level information.

- In the course of our experiments, we discovered that time synchronization is a recurring issue and therefore developed two time synchronization approaches that can be used either online, i.e. during an experiment or offline after an experiment. The NTP skew correction is an online approach that increases clock precision by two orders of magnitude with standard software tools, already reaching millisecond precision on PDA-class devices. The second approach is a post-experiment timestamp synchronization algorithm based on a maximum likelihood estimator (MLE). It can synchronize timestamps provided by unmodified clocks and has an error in microsecond range. Based on this approach, we have developed the tool `pcapsync` that synchronizes packet trace files in standard libpcap format.
- To cope with the need of flexible data analysis after an experiment, we have developed the modular data analysis tool EDAT. This tool follows a flow-based approach and produces graphs directly usable in scientific publications.

This thesis is structured as follows. In Chapter 2, we give an overview of existing evaluation strategies, present some of our initial experiments and condense this to a recommendation for the construction of WMN testbeds. Based on these recommendations, we have constructed our testbed EXC that is presented together with the underlying semi-automatic experiment control methodology in Chapter 3. The different time synchronization techniques developed in the course of this thesis can be found in Chapter 4. Thereafter, Chapter 5 discusses the extensible data analysis toolkit EDAT. The repeatability of experiments with wireless multihop networks is examined in Chapter 6, followed by concluding remarks in Chapter 7.

Chapter 2

Strategies, Experiments and Consequences

Chapter Outline

The first step in improving something is always the analysis of the state of the art, so we started with an examination of existing experiments with wireless multihop networks. This analysis reveals that a lot of work in this area is duplicated. To avoid such duplications in the future but also to introduce the reader to these existing approaches, we present in the first section of this chapter a guidebook for the experimental evaluation of wireless multihop networks. We list the different strategies that have been used to set up an experiment, the most important tools, commonly used topologies, input parameters, and metrics to characterize the outcome of the experiments. For this, we focus on mobile ad-hoc networks but also consider the simpler static setups. The content of this chapter has been published in [KM07a, KM07b], Appendix A reviews the experiments from which this knowledge has been extracted.

Besides surveying existing work, we started to perform our own WMN experiments. The first study [KTM05, KTM06] was an experimental evaluation of ring flooding, a technique often used in MANET algorithms and applications. Furthermore we created a number of tools to assess radio propagation in an experimentation area and performed measurements with them [Ker06]. On the one hand, these evaluations result in knowledge about the behavior of these techniques in real-world settings and are therefore valuable contributions for themselves. However, they are even more valuable in the context of this thesis because they provide hands-on experience with such real-world evaluations.

These experiences as well as those made by other researchers motivated us to think about the problems and requirements of experiments with wireless multihop networks in general. A summary of this process which has also been published in [KZTM05] is presented in the third part of the chapter.

2.1 Existing Strategies – A Guidebook

2.1.1 Topologies, Node Placement and Movement Patterns

From a network layer point of view, topology and topology changes are among the most prominent features of mobile ad-hoc networks. In simulations which often use a simplified radio model and no obstacles, network topology and topology changes are a direct result of node placement and movement. In a real MANET, a large number of interwoven factors have an additional influence on network topology, multipath propagation, small and large scale fading, and radio obstacles for example. Recent studies [KNG+04] have shown that the topology of a real MANET is complex: links are asymmetric, their quality can change rapidly over short periods of time and a link almost never provides a full-reliable delivery. As most of the influencing factors are hard to control or cannot be controlled at all during an experiment, most papers about MANET experiments only consider node placement and movement as influencing factors. In the following section, we give an overview of the node placements and movement patterns that are used most often in the literature. It should be kept in mind that these node placements and movement patterns are intended to directly result in a similar network topology, although this may not always be the case in the real world.

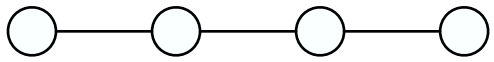


Figure 2.1: String placement.

In the *string* placement, the nodes are set up in a chain as shown in Figure 2.1. The intention is to let each node communicate only with its two neighbors such that, given a fixed number of nodes, the longest multihop chain is constructed. This placement has been used e.g. in [MBJ99, MBJ00, GWW04, Mös03]. In order to better separate the links from each other, obstacles like walls, buildings or even access points that create interference limiting the transmission range have been used [MFHF04, BCDG05].

When the nodes are aligned in a *grid* as shown in Figure 2.2, this should result in a topology where each node can only communicate with its four direct neighbors. As such a node placement requires a certain number of nodes to be available, it can be found most often in experiments with sensor networks. Nevertheless, there are also setups for MANET experiments like in the ORBIT testbed that aligns 400 nodes in a 20×20 grid [ORB, RSO+05].

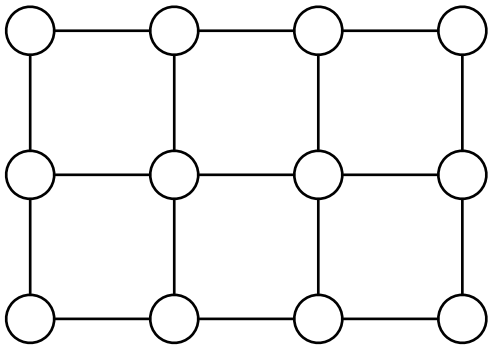


Figure 2.2: Grid placement.

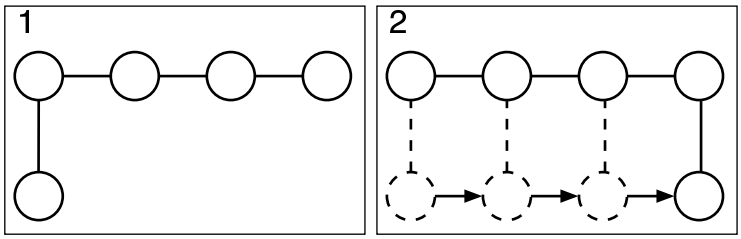


Figure 2.3: Roaming node.

In static experiments with commercial-off-the-shelf (COTS) 802.11 hardware frequently used for MANET experiments, the experimenters often deploy the nodes in a way that conforms best to the environmental conditions. This *random* placement consequently also leads to a random network topology. This is most often used in indoor settings when a larger number of nodes are deployed [DPZ04a, PAM+05] although outdoor examples exist as well [BABM04].

The static string also builds the basis for a number of simple movement patterns: the *roaming node* pattern (also called *circling node*) shown in Figure 2.3 is the simplest pattern with node mobility. Out of the n nodes in the network, n-1 are static and one is mobile. The static nodes can be set up either in a chain as shown in Figure 2.1 or be randomly placed. Packets are then exchanged between one of the static nodes and the moving node. As the mobile node moves in and out of the transmission range of the static nodes, the network topology changes constantly with respect to the mobile node [HJ02, CJWK02, PAM+05, SBSC03, DPZ04a, GWW04, STP+05, BCDG05]. Variations of the circling node pattern are *end swap* (the outermost nodes in a chain change places) and *relay swap* (two nodes in the middle of the chain change their place) [BCDG05].

The *chain on the fly* pattern shown in Figure 2.4 consists of a number of nodes that are

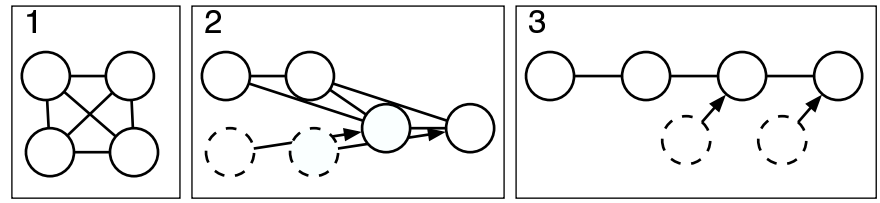


Figure 2.4: Chain on the fly.

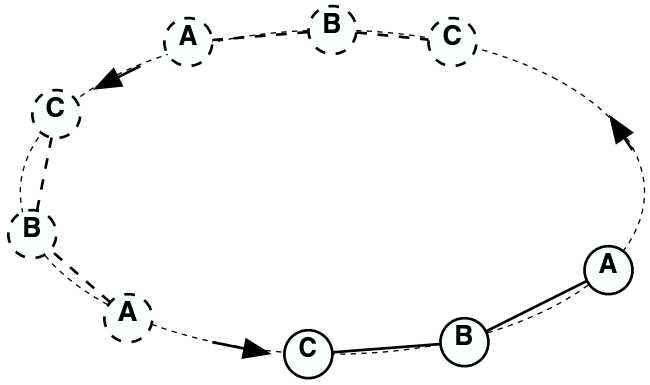


Figure 2.5: Mobile String.

close to each other in such a way at the beginning of the experiment that every node can communicate with every other node. Then the nodes start to move until they are aligned in a chain [MFHF04, STP+05]. The experiments presented in [LLN+02] use a variation of this topology in which clusters of nodes move around to form a chain of clusters.

The mobile variation of the static string is the *mobile string* movement pattern shown in Figure 2.5. Here, a number of nodes move around following each other without changing their relative position. This has been mostly used in experiments with cars [MBJ99, MBJ00, MFHF04].

The *mobile random* movement pattern is the mobile variation of the static random node setup. The nodes move around randomly corresponding to the random waypoint model used in a lot of MANET simulation studies. This pattern is most often used in experiments with a large number of nodes where it is difficult to explicitly control the movement of each single node [GKN+04, JBD+05, RRS+05].

Two different trends are visible in node placements and movement patterns used for MANET experiments. On the one hand, there are the *controlled* setups like string, grid, circling node or mobile string. These are built with either a small number of

nodes or with networks of small physical extension. *Random* setups are used as soon as the network becomes difficult to handle due to a large number of nodes, a great physical extension or high node speeds. We are not aware of experiments with a larger number of mobile nodes following a non-trivial, controlled movement pattern. As some of the presented patterns are used to create an especially challenging scenario for the tested protocols, they are rather artificial. Nevertheless, some patterns and placements also have realistic counterparts. For example, "chain on the fly" occurs in robotic surveillance systems [KOV+02, STP+05], "mobile string" occurs in unicast communication between cars on a highway, and the "random" placement is relevant for mesh networks [BABM04].

2.1.2 Traffic Patterns

As a protocol provides services for the next layer up in the protocol stack, testing the protocol requires that this layer requests these services. The question which WMN applications are most promising and therefore which traffic patterns to use for this is an open research issue. Here, we provide a short overview over the traffic patterns chosen by different researchers for their real-world experiments.

In systems built as demonstrators, *real traffic* produced by a real application is transported over the MANET. The intention of this setup is to show that the given application is working. Unfortunately, there is no demonstrator for a complex application over a MANET we know of that has been evaluated in detail. Examples are the Fleetnet demonstrator with "broadcasting of emergency warnings" [HFMF03], the live audio and video transmissions over DSR [HJ02], the work in [STP+05] where live video and robot control messages are transmitted over a MANET with one mobile node, the Centibots project where robots transmitted topology information and control messages over the MANET [KOV+02], or the DSR demonstrator [MBJ00] transmitting status information as well as GPS corrections over the MANET.

Artificial traffic is used to test the underlying protocol when no real application is available. This traffic is produced by a traffic generator following a certain distribution. There are experiments using the distribution of real applications as model such as the artificial voice traffic in [MBJ00] or the experiments presented in [GKN+04], where the traffic volume is modeled after a "prototype military application". A lot of experiments are performed to evaluate the maximum performance of a given protocol, e.g. to determine the highest possible throughput or the minimum round-trip-time. Therefore,

standard tools known from wired networks and designed to stress a protocol are often used. This can be traffic generators like netperf or iperf producing TCP and UDP streams [SBSC03, PAM+05, MFHF04, GWW04] as well as the ping utility [MBJ99].

2.1.3 Implementation Strategies

The first step towards protocol testing is the implementation of the protocol. As no code exists that can be (re-)used for protocols following a new paradigm, these are often implemented as *prototypes* fresh from scratch. Such implementations are mostly tailored for the experiment to be performed and are often neither published as source code nor available in binary form. Examples of this are the DSR prototype [MBJ99, MBJ00] or the Fleetnet position-based router [HFMF03, MFHF04]. Besides the "from scratch"-method for prototypes, a number of different implementation strategies for routing protocols as well as for protocols on other layers are described in the literature.

We classify all software tools built to support the task of implementing protocols as *frameworks*. The PICA API [CGM03] and the "user level framework for ad hoc routing" [AGSR02] shadow the calls to operation system specific functions. With this, a protocol can be developed once and used on different operating systems without porting the implementation. The MANET routing framework [NKSW02], FRANC [CSS03] and the ad-hoc support library [KZG03] extend this approach. Besides allowing platform independent implementations, they also offer some common services needed by a lot of algorithms and protocols. The idea is to implement flooding, neighbor discovery, packet buffering during reactive route discovery, reliable unicast and broadcast, queues, timers, packet sniffing or network emulation for testing purposes in the framework. This allows the programmer to concentrate on the specifics of the individual algorithm or protocol.

The Multi-threaded Routing Toolkit (MRT) is a framework that has been used for the DSDV implementation in [CJWK02] and also for the TBRPF implementation in [KOV+02]. Unfortunately, the MRT project seems to be discontinued and there is no documentation available.

The click modular router [KMC+00] provides a script language allowing the combination of simple modules to a router. Already existing modules in the click library have tasks like decrementing the TTL or recalculating the checksum of a packet. Furthermore it is easy to implement and add new modules. This approach accelerates the protocol development as it fosters reusability. Click has been developed for routing in fixed networks

but has also been used to implement routing protocols for mesh networks [BABM04] and MANETs [JBD+05].

As a large number of protocols intended for MANETs are already implemented[1], using existing code is another strategy that can lead to a working real-world protocol. First of all, it is possible to directly *use an existing implementation without modifications*, e.g. when a reference protocol is needed. This reusing reduces the workload and allows to compare the results with other experiments. In [LLN+02], publically available versions of AODV and OLSR are used for the experiments; [CJWK02] does the same with MAD-Hoc AODV. The authors of [SBSC03] use an openly available OLSR version as a reference point. The authors of [STP+05] utilize *code written for a network simulator* in their real-world experiment. They do this by providing a packet converter between simulator and real-world packet format together with additional wrapper code. In [GKN+04], existing implementations of four routing protocols are *used as model for a reimplementation*. This reimplementation is necessary because the authors need implementations that differ as little as possible to increase comparability. Furthermore, the existing code itself can be *adapted*. Examples of this are the signal-strength aware version of AODV [GWW04] based on AODV-UU, the modification of the Linux-kernel-TCP to TCP RBP in the same paper, SBRS-OLSR [SBSC03], a signal-strength aware version of OLSR, or the multimedia extensions to DSR described in [HJ02].

2.1.4 Tools

A number of tools exist that have been repeatedly used in existing MANET experiments. We summarize them in the following section. An *emulator* is a combination of soft- and hardware that mimics the behavior of a network with some of its components being implemented in the real world and others being simulated. Emulators can be either used to test protocols on real hardware or to prepare real-world experiments. In the latter case emulation is used to form a virtual topology among the nodes. This allows easy in-lab testing without moving the nodes physically around before conducting a full-scale experiment. For example, MAC layer emulators use real implementations for all network layers except the MAC and physical layer. These emulators simply determine the nodes that should receive a given packet: if a node is emulated to be within radio range of another node, a filter tool allows the exchange of packets between them; if the nodes are out of range of each other, the respective packets are dropped. Emulators for WMN experiments are often based on available network filter tools such as iptables,

[1] [wikb] lists ten AODV, four DSR and seven OLSR implementations.

e.g. as in MobiEmu [ZL02]. For a review on the different types of emulators refer to [KM07a].

Due to the complex topology of a real MANET, a number of *tools to classify the quality of existing links and to filter out bad links* have been built. This can be done based on the geographical distance between sender and receiver [HFMF03], based on signal quality [LNT02, CJWK02] or by taking the packet loss on a link into account [LNT02]. Furthermore, signal-strength awareness has been directly integrated in some routing protocols [HJ02, SBSC03, GWW04]. However, as bad links are an inherent property of a MANET topology, it may be worthwhile to take care of such links during protocol design rather than try to adapt the experiments to the expectations based on simulation models.

Monitoring tools allow the supervision of an experiment from a central point. These tools collect information such as battery state, traffic statistics or link quality and transmit it to one or several sinks. The monitoring information can be transmitted either in-band, i.e. over the experimental network itself [KGBK78, MBJ99, JBD+05] or out-of-band over an additional network [HFMF03, RABR05, SOSK05]. The information collected with monitoring tools has been mostly used for network visualization [Bey90, MBJ99, RBRA04]. This aids in the explanation of the network and furthermore supports its debugging. Debugging is also the purpose of the other projects that have implemented network monitoring without direct visualization [KGBK78, JBD+05, HFMF03].

In addition to monitoring an experiment, it is also necessary to *control an experiment*. The existing approaches can be classified according to their level of interactivity and automation. Trivially, all tools can be *controlled manually* by the experiments' participants. *Fully programming* the sequence of events in advance marks the other end of the design space. This can be done by utilizing shell scripts that start and stop all the software at a predetermined time, for example. As unforeseeable situations occur often during an experiment, this lack of interactivity can sometimes have severe effects while experimenters only recognize the resulting problems after the experiment is finished [GKN+04]. On the other hand, a certain level of automation is necessary and has been therefore used in a number of experiments. The Ad-hoc Protocol Evaluation testbed (APE) [LLN+02] also uses automated actions but interrupts the flow of actions at predetermined points. For this, the experiment is divided in runs. The events in each of these runs are specified by a script that starts traffic generators or tracing tools and prints out status messages to the experiment participants. Each of the runs then has to

be started manually one after the other by pressing a button. Another approach is to *steer the experiment based on experimental feedback* such as with the ORBIT Measurements Framework and Library (OML) [SOSK05]. In the presented example, a traffic source increases the data rate until the monitoring reports that loss exceeds a certain threshold. It is obvious that monitoring and controlling are most useful when working hand-in-hand: if there is no information available on the actual state of the network, adequate decisions are difficult to take. On the other hand, full information without the ability to interfere is also not satisfactory.

In order to have as much information as possible available for post-experiment analysis, *tracing tools* are important software components. For nearly all conducted experiments, standard packet-level tracers such as tcpdump [TCP] are a basic source of information as they record all packets sent and received by the recording node. This can be combined with a log file of the internal state of the tested protocols so that the decisions taken by these protocols can be replayed. Additionally, the performance on the layer requesting the service from the protocol to be tested can give insights in how well this service is provided. An important factor that strongly influences a WMN is the state of the physical environment. For the wireless channel, this state can be recorded by logging per-packet signal-strength or noise levels for example and the movement can be traced via GPS positions logs. This can then be correlated with the higher-layer information to examine the influence of the environment on the other performance parameters. As all this data is indispensable for latter analysis of the experiment and the loss of one node's traces can already make the interpretation difficult, care must be taken to correctly collect all trace files at the end of the experiment. A first step into this direction has been taken by APE [LLN+02] where all log files are automatically transmitted to a data collection node.

To properly understand the experiment, the recorded raw data needs to be analyzed and interpreted. A first approach is the calculation of performance metrics *using existing software*. With standard tools like netperf, iperf or ping, this is relatively simple because they directly calculate performance metrics such as throughput or round-trip-time. Furthermore, program packages such as the analysis scripts being a part of APE [LLN+02] can be used to calculate a number of metrics from recorded raw data. This is a concept also known from the simulation world where the tool TraceGraph [TRA] can compute a large number of different metrics and graphs for example. If custom performance metrics need to be calculated, e.g. because a special form of data aggregation is necessary or performance metrics for a new protocol have to be calculated, a *custom analysis* is required. This custom analysis is most often performed with programs written from scratch, an

approach called *manual analysis*. There are also network tracers like wireshark [WIRa] that allow the recorded packets to be examined in detail. Furthermore, these tools provide some customization of this examination through simple filtering. From the simulation world, *visualization tools* such as Adhockey [ADH] or Huginn [SFT+05] are known. These tools process the available trace files to visualize the events in the network in a way that allows a human to better understand the large amount of data. As such tools are often open source, adapting their parsers to the trace formats of recorded real-world data should be possible and can thus also support the analysis of experiments.

A *testbed* is a framework that supports testing, comparing and evaluating algorithms and protocols in the real world. Therefore a testbed should combine all of the above presented tools and be open for different protocols. The only existing testbed for mobile ad-hoc networks used to a larger extent is APE [LLN+02] that comes a Linux distribution directly bootable from CD on regular laptops. Each experiment participant is instructed to move according to a choreography script as described above. To a certain extent this makes experiments repeatable. The authors have integrated tools to collect traces about the experiments, to upload these traces at the end of an experiment to a central computer, and to calculate performance metrics.

2.1.5 Experimentation Strategies

The strategy that is pursued to perform a real-world experiment strongly depends on the goals that should be accomplished. To get an impression of the goals that have been reached with existing strategies and to facilitate the choice for future experiments, we discuss the most common approaches here.

If it should be demonstrated that a given protocol or algorithm works in a real MANET, a *proof-of-concept*-strategy can be applied. The goal of such an effort is the building of a working prototype with the required functionality. The focus lies on the implementation of the protocol and experimental design is of minor importance. Furthermore, the experiments are generally ended as soon as the protocol works under the specified conditions within the required parameters. As the evaluation here plays a secondary role, one should not expect quantitative results from such experiments.

An *in-detail-evaluation*-strategy requires a working protocol and can therefore be a follow-up to a proof-of-concept implementation. As such an evaluation requires the testing of the protocol under various conditions and in heterogeneous environments, experimental design becomes important. Before writing this chapter, we asked researchers

to estimate the time they spent during different phases of their experiments. Depending on the type of evaluation, it is quite likely that conducting and evaluating an experiment can require as much as or even more time than the implementation of the protocol. Furthermore, it has also been reported that the experiments themselves had to be improved in an iterative process, i.e. a number of "designing, conducting, evaluating an experiment"-cycles were needed.

The question how the experiment itself should be structured has been solved by nearly all researchers in a similar way: by *dividing the experiment in several runs*, often with similar runs being repeated multiple times during the same experiment[2]. As mobile devices like PDAs and laptops are used in most cases as hardware for the tests, the duration of the whole experiment is limited by their battery capacity [GKN+04]. Short runs tend to be used for small and low mobility setups, for example the routing protocol tests in [BCDG05] with a duration of 60 to 120 s or the static OLSR experiments in [PAM+05] with 60 s runs. In contrast, runs performed with cars are often of longer duration: [MBJ99] has runs of duration 220-1000 s and the experiments with the Fleetnet demonstrator [MFHF04] use runs of 400 s. On the other hand, the runs in [SBSC03] where only one mobile car is involved have a duration of 90 s. Runs performed with a larger number of mobile nodes also tend to be longer: the 40-node runs in [GKN+04] last for 900 s and the runs in [LLN+02] with 20 to 34 nodes have a length of 250 to 400 s. The general trend seems to be influenced by the complexity of the setup as here a certain time is required to achieve the necessary topology changes, thus increasing the minimum duration of a run.

An emulation can be either performed *instead* of an experiment, to *prepare* an experiment or to *reproduce* an experiment with trace data as input. For information on the first and last case, refer to [KM07a], here we concentrate on those emulations conducted to prepare experiments. The basic idea of this strategy is to test all soft- and hardware before the experiment under semi-realistic conditions. This can save a lot of work [MBJ99] and the importance of such an emulation is underlined by the number of experimenters that have used corresponding tools under different names: powerwave [CJWK02], APE mackill [ape], MobiEmu [ZL02], Fleetnet packet suppression mechanism [MFHF04] or FRANC virtual networks [CSS03].

Although it is very difficult to compare the outcome of two experiments, *performing baseline measurements* can be a strategy to improve comparability. The basic idea is to

[2] An exception to this is the "multimedia over DSR" demonstration presented in [HJ02] where the demonstration runs continuously without interruption for a whole day.

perform measurements with very simple or idealized protocols that can use more information than is normally available at a single node. With this it is then possible to either judge the quality of the environment, to determine upper bounds for the performance or to calculate performance metrics relative to the best possible performance. Different types of such baseline measurements are 1) measurement of the basic characteristics of all links in order to assess the maximum performance of the network without multihop effects [DPZ04a, BABM04], 2) measurement of the performance of a MANET protocol in a static scenario in order to have an upper bound for the performance under mobility [MFHF04, MBJ99], and 3) use of an idealized protocol that has global knowledge, e.g. for routing [FS01].

If the evaluation of a protocol comprises also a simulation and/or an emulation, *Simulation, Emulation, Real-World (SER) integration*, i.e. using the same protocol code for simulation, emulation and real-world experiment, may be worth considering. This requires some effort to adapt the code to different environments, but it also saves time: as the protocol has to be implemented only once, bugs can be found at an early stage in the development process and do not occur at the moment of the experiment. Furthermore, it is not necessary to reimplement the whole protocol for the experiment and double the effort. Existing SER integration approaches can be classified as follows:

1. Run encapsulated code and either use a packet converter between real world and simulation format [STP+05] or encapsulate the packets [DRSC05].

2. Write the code by using an API available in the simulator as well as in reality:

   a) Integrate the API in an existing simulator: nsclick [NJG02], GEA [HM05].

   b) Write a custom-made simulator that supports the API: SURAN [Bey90], Rooftop CPT (used by WINGS [GLA01] and GloMo DAWN [RH00]), "user level framework for ad hoc routing" [AGSR02], the routing protocol evaluation presented in [GKN+04, LYN+04].

3. Port the code manually: [RP00, LYN+04, OT05, RRS+05].

The approaches 1) and 2a) seem the most promising as they allow the use of a well established network simulator that normally contains a variety of protocols and radio layer models without making changes to the code.

2.1.6 Performance Metrics and Characterization

To judge the performance of a protocol and also to compare it with other protocols, performance metrics are used primarily. A number of such metrics are listed in RFC 2501 [CM99] for MANET routing protocols: 1) end-to-end data throughput and delay, 2) route acquisition time, 3) percentage out-of-order delivery, and 4) efficiency (internal efficiency, e.g. delivered bytes / total bytes). These performance metrics can be calculated as *averages over a whole experimental run*, e.g. the overhead of routing protocols [GWW04, GKN+04, STP+05], the packet delivery ratio [GKN+04, BCDG05], overall throughput [JBD+05, PAM+05, GWW04] or the end-to-end latency [GKN+04]. It is also possible to build performance metrics with respect to the *number of hops*, e.g. for ping requests and replies [MBJ99, LLN+02].

In order to characterize the environment in which the experiment has been executed, plots of physical layer parameters over time have been used. The parameters here can be received signal strength (RSS) or signal-to-noise ratio (SNR) [MFHF04, PAM+05, GWW04, CJWK02]. This has been extended by Lundgren et al. with their virtual mobility metric that is based on measured signal quality [LLN+02]. The idea is to use per packet signal quality to compute virtual distances between the nodes. These distances are used to describe the topology changes in the network as perceived by the nodes. Plotting the virtual mobility over time then produces a fingerprint of the experiment that can be used to determine how similar two repetitions of an experiment are with respect to connectivity.

The approach to plot parameters over time is also used for the network layer. The parameters here are loss rate or packet delivery ratio (PDR) [MBJ99, MFHF04, STP+05, LLN+02] and the link changes per second or connectivity [LLN+02]. In order to explain the inter-layer influences, such time-plots with parameters for different layers are often shown in parallel for the same run, e.g. in [MFHF04] where three plots with distance, hops and PDR are shown or the experiments in [LLN+02] where plots for PDR, connectivity and signal quality are combined.

2.2 Ring Flooding

The knowledge acquired in the previous section is complemented by the experiences made during our experimental evaluation of ring flooding, a dissemination technique that is frequently used in ad-hoc networks. In its simplest form, flooding is realized

by letting each node rebroadcast the flooded packet exactly once. To limit the scope of a flooded data packet, the sender of the packet may use *ring flooding*. For this, the packet's time-to-live (TTL) field is initially set to n. As the TTL of the packet expires after n hops, it only reaches all those nodes that are at most n hops away from the original sender. Ring flooding is used to distribute information only relevant in a certain area such as emergency messages in car-to-car networks, or to do an expanding ring search during route discovery as in AODV [PBRD03], for example.

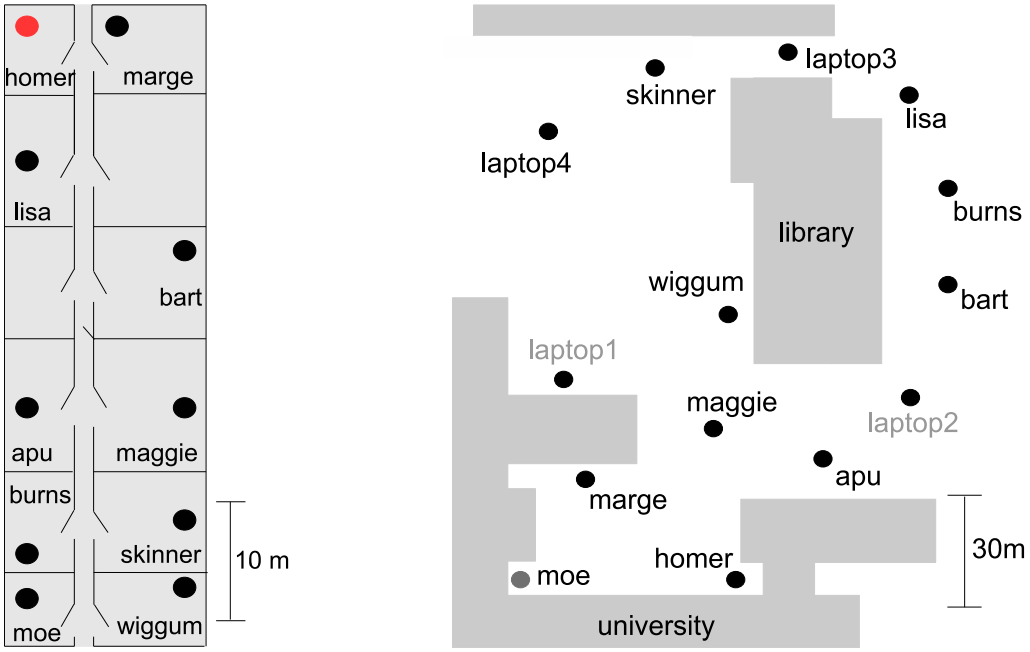
The examined scenarios consist of up to thirteen nodes and we study 1) how reliable a flooded packet reaches all nodes, 2) how long it takes the packet to reach the nodes, and 3) how many nodes are reachable when flooding with a certain TTL.

2.2.1 Experiment Setup

We have performed one indoor and one outdoor experiment on static multihop topologies with IEEE 802.11b equipped nodes. The parameters that have been varied are initial TTL, jitter and packet size. Jitter was used to delay the rebroadcasting for a random time from the interval [0; jitter] in order to reduce collisions. All nodes used Linux and the packets were traced with tcpdump. Flooding was implemented with click [KMC+00], and we used nsclick [NJG02] to test the implementation in ns-2 [NS2]. One node at the corner of the network acted as packet source. Each node repeated each packet exactly once and ignored duplicates; packets were dropped on TTL expiration.

The indoor experiment consisted of 10 iPAQ5550 PDAs distributed in two rows over 15×50 meters as shown in Figure 2.6(a). The flooded packets had a size of 100 bytes each, we chose 0 and 10 ms maximum jitter. For each jitter interval, we flooded 10 000 packets divided in sequences with TTL $(1, 3, 5, 7, 9)$ and 10 000 packets divided in sequences with TTL $(1, 2, 4, 8, 16)$. The minimum spacing between two flood attempts was 120 ms.

For the outdoor experiment, thirteen nodes (ten iPAQ5550 and three laptops) were distributed over an area of 110×145 meters on the university campus (Figure 2.6(b), "moe" was the packet source). For all experiments, we used linear ring flooding, i.e. the source increased the initial TTL from one to thirteen for each successive packet and then restarted from one. Packets had a size of 200 bytes each and the used maximum jitter values were 0, 5, 10, and 15 ms. For each of these values, a total of 3 000 packets were flooded in six runs. As minimum spacing between two flood attempts, we used 60 ms and increased this for higher TTLs and jitter. During run thirteen, the node "laptop1" (see Figure 2.6(b)) failed due to a lack of battery power, dividing the experiment in



(a) Indoor (b) Outdoor

Figure 2.6: Node setup in the ring flooding experiment.

TTL	0 ms jitter	[0;10] ms jitter
3	13.7%	14.1%
4	98.0%	99.1%
≥ 5	99.7%	97.0% (99.7%)

Table 2.1: Reliability for the indoor experiments.

two different topologies. We therefore eliminate the affected runs 8-15, leaving for each maximum jitter value 1 000 packets on each topology.

2.2.2 Results

Reliability is the percentage of packets that reach all nodes in the network. Although each node in the indoor experiment was theoretically reachable with at most three hops, packets with an initial TTL ≥ 5 achieved the highest reliability as shown in Table 2.1. Obviously, high reliability comes at the cost of letting each node repeat the packet. The behavior of packets with an initial TTL ≥ 5 for the [0;10] ms jitter runs is also interesting. One of the runs had a reliability of only 54.6%. With this run, the reliability was 97.0%, without it, it was 99.7%. We assume that this difference stems from temporary

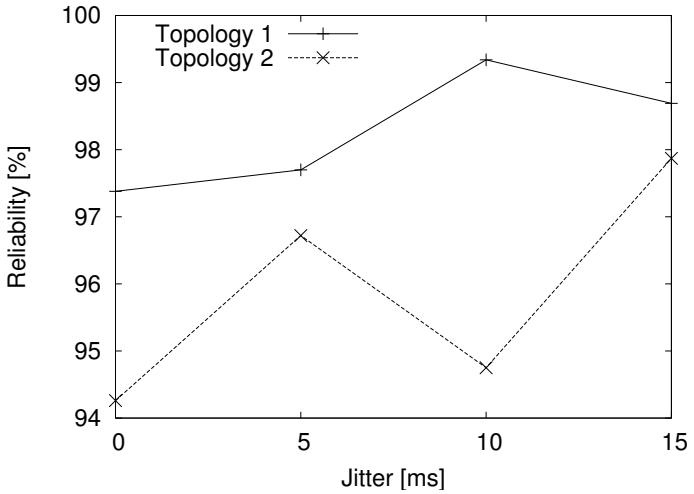


Figure 2.7: Reliability for TTL ≥ 6 in the outdoor experiment.

interference. Also for the outdoor experiment, the trend of the curves for the two topologies indicate that increased jitter has a positive effect on reliability as shown in Figure 2.7. However, it should be kept in mind that each data point is an average over only two runs, explaining the strong fluctuations that can be observed in the graph.

Latency is the time from the initial broadcast until the last node receives the packet. For the outdoor experiments, the latency is shown in Figure 2.8 for topology two. The highest latency in the experiment, over 140 ms occurred on topology one (not shown here). The reason for this was a rebroadcasting delay of 120 ms in "laptop1". Besides that, the highest latencies for both topologies were 42, 46, 67 and 74 ms for the increasing maximum jitter values.

Received copies are the number of copies a node receives of a flooded packet. As nodes at the border of the network do not receive packets with low initial TTL, we only consider packets with a sufficiently high TTL. For the outdoor experiment, a node received on average 3.8 copies on topology one and 3.3 copies on topology two (TTL ≥ 6). For the indoor network, the number of copies per node was 3.6 (TTL ≥ 5). This is interesting in relation to those copies sent but not received by any other node. While indoor runs with [0;10] ms jitter lost only 0.3% of all send events, 0.9% were lost for the runs without jitter. For the outdoor experiment (averaged over both topologies), this loss decreased slightly from 1.6% over 1.4% and 1.3% to 1.0% with increasing jitter. Due to the dependence on jitter, we suspect that this is an effect of collisions.

Neighborhood stability denotes how the allocation of a node to a n-hop neighbor-

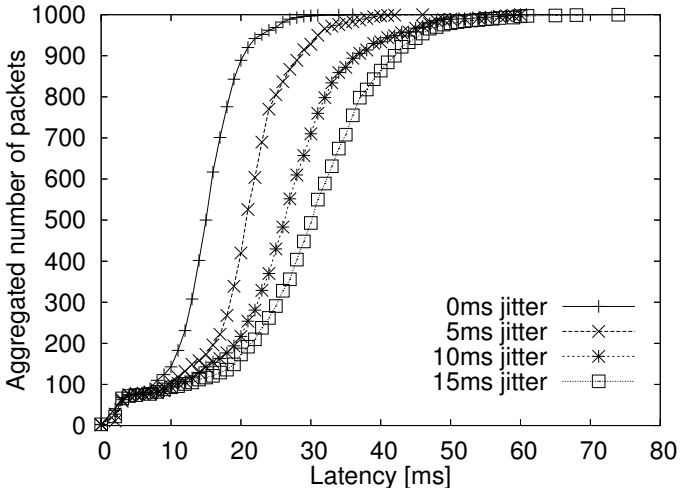


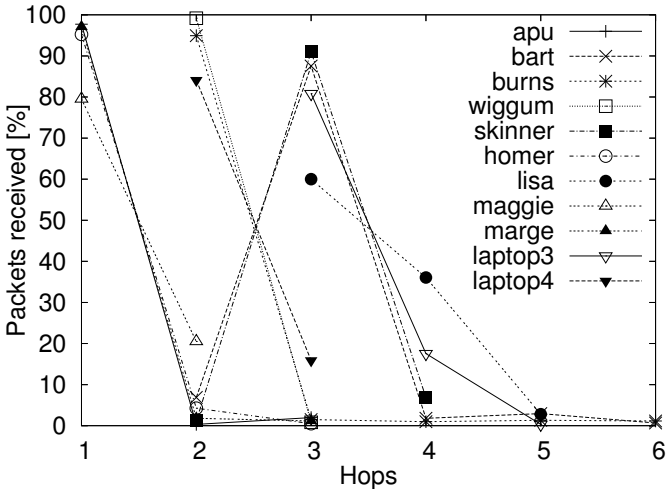
Figure 2.8: Latency for all packets in the outdoor experiment.

hood fluctuates. Figure 2.9(a) shows the minimum number of hops the flooded packets needed to reach the nodes. Six of the nodes were firmly attached to a certain neighborhood with reception rates of over 90%. Four other nodes received between 75 and 90% over a certain number of hops. One node ("lisa") received 62% of the packets over three and 38% over four or more hops. Figure 2.9(b) shows how this influenced node reachability: while 60% of the packets flooded with TTL 2 reached exactly seven nodes, a 2-hop ring flood might reach as few as four or as many as nine nodes. A similar behavior can be observed for the other TTL values.

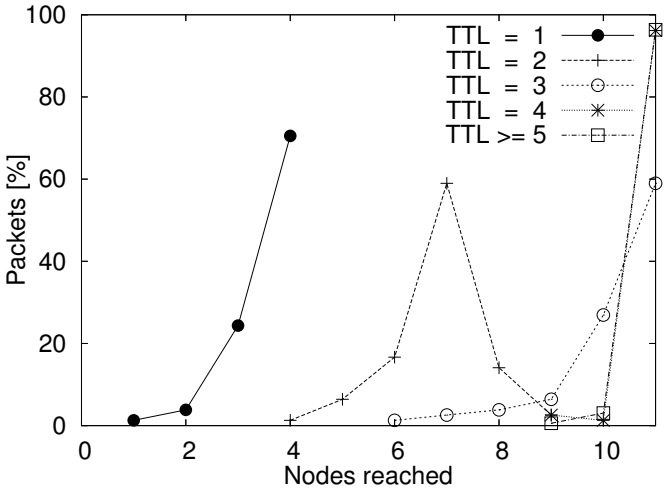
These experiments show that flooding with the network diameter d is not enough if all nodes should be reached: in the examined scenarios, flooding was most successful for $\text{TTL} \geq d + 2$. Furthermore, the number of nodes reachable with an n-hop flood varied from attempt to attempt even in our static networks. Thus the definition of the often used n-hop neighborhood should be revised.

2.3 Propagation Estimation

Our second experimental study concentrated on the estimation of the propagation characteristics of a certain area. This is important for the selection of an appropriate testing site during preparation of an experiment. Knowledge about these characteristics allows for better planning of the topology, for example for a setup with a certain number of hops. This information can also be used as input for an emulation that is closer to



(a) Min. hops/node, TTL ≥ 6



(b) Number of nodes reached

Figure 2.9: Neighborhood stability ([0;5] ms jitter) for topology two of the outdoor experiment.

reality than existing, distance-based connectivity assessments. The main problem here is that directly measuring these propagation characteristics between all point-pairs in the experimentation area is impossible: as the number of such pairs is unlimited, so would be the number of necessary measurements.

2.3.1 Basic Idea and Implemented Tools

To acquire the desired information with a limited number of measurements, we have developed a toolset that records the link quality only between selected point-pairs and uses interpolation to estimate the qualities for uncovered points. As directly measuring link quality is only supported by some wireless network interfaces, our tools use packet level information as link quality indicator. The whole measurement process is sped up by deploying multiple devices at the same time. The devices rotate through the different measurement positions and cover multiple point-pairs in one round. Thus, fewer measurement rounds are necessary to cover the whole area. However, this speedup comes at the price of a more complex experiment coordination. In total, we have developed three related tools for this task: *Mapconfig* is used for planing the measurements and calculating the sequence of positions for these. During the measurements, *Mapkit* provides a simple positioning service and also sends out the packets. The results of these measurements then can be directly processed by our library. To examine the propagation characteristics of the area in detail, we have implemented an *Analyzer* visualizing the measurements' outcome.

The first step in planning the measurements is to determine points for which measurements are to be made. To this end, Mapconfig positions a grid of adjustable size on a map where the grid points mark the measurement positions. Out of the set of grid points, the required subset of measurement points can then be selected. A screenshot of this tool during the planning of one of our experiments can be found in Figure 2.10; the selected points are marked white. For the measurement, the devices at the different positions have to send out packets successively such that they can be recorded without interfering transmissions of other devices. For the coordination of these tasks, we use control packets that are flooded in the network: as soon as a node has finished sending packets, the flooded control packet triggers the action in the successor node. To keep the network connected when the nodes rotate through the different measurement points, we use static *mesh nodes*. The mesh nodes must have a single- or multihop connection with every other mesh node and it must be possible to reach at least one of these from each measurement position. With some experience, these mesh nodes can be positioned

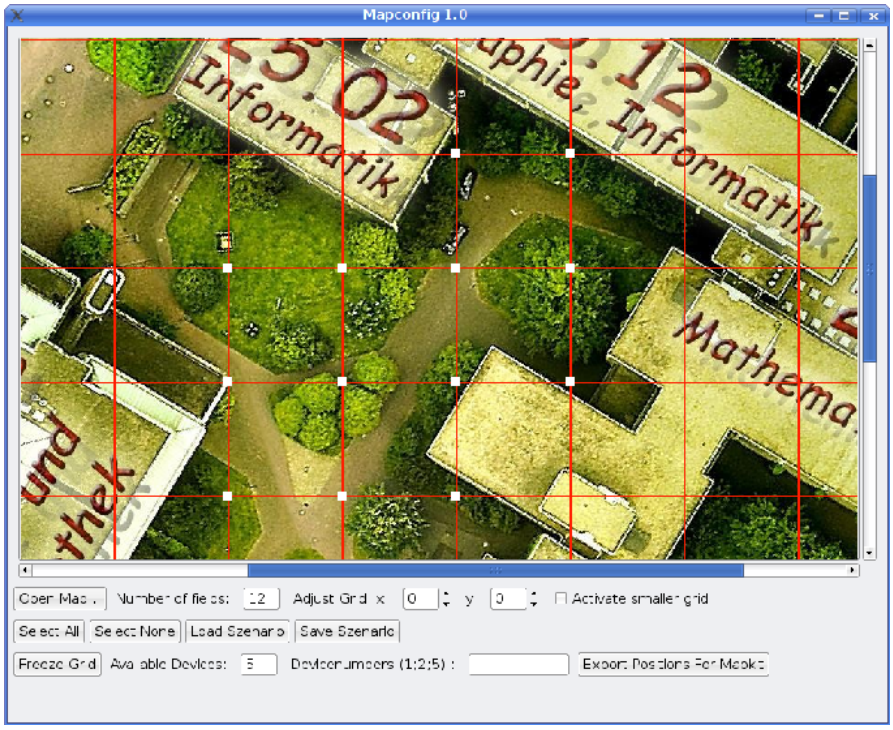


Figure 2.10: Mapconfig screenshot with activated, white-marked measurement points.

simply by visually inspecting the map. In case of doubt, however, it is advisable to conduct some pre-measurements.

Mapconfig also determines the movement sequences for the participating nodes. For this, the nodes are divided in two groups and rotated through the measurement positions as illustrated in Table 2.2. Here, a total of eight measurement points $x_1, x_2, ..., x_8$ are to be covered with four mobile measurement nodes $n_1, .., n_4$. The nodes n_1, n_2 that are part of the first group G_1 move to the first positions while the nodes n_3, n_4 of the second group G_2 rotate through the remaining positions. After the second group has finished, the nodes in G_1 move on to the next set of positions. The nodes in G_2 once again start rotating but skip those positions already covered by G_1 in the previous round. This process continues until all point-pairs have been covered[3].

The tool Mapkit is responsible for performing the measurements. It supports the users in correctly positioning the devices and also controls and performs the packet sending. For the first task, we use a map-based, visual positioning service: the target position is shown to the user who then manually marks its current location. In contrast to using

[3]For point-pairs such as x_1, x_2 where several measurements are made, the recorded link qualities are averaged during the analysis step.

Scenario/Position	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	n_1	n_2	n_3	n_4				
2	n_1	n_2			n_3	n_4		
3	n_1	n_2					n_3	n_4
4			n_1	n_2	n_3	n_4		
5			n_1	n_2			n_3	n_4
6					n_1	n_2	n_3	n_4

Table 2.2: Example positioning of the measurement devices.

GPS, this does not require additional hardware and also works when the view to the sky is obstructed, e.g. inside or in the vicinity of buildings. In our experiments, this simple approach allowed the devices' positioning with a precision of a few meters. Even though we use the mesh nodes, sometimes control packets do not arrive at the target nodes. Therefore, there is an additional monitor that coordinates the experiment and can repeat such missing packets. This monitor is also responsible for initializing the measurement.

The recorded values are used then as input for the interpolation in the analysis step. The different cases that can occur when the quality between a source S and a target T should be calculated are:

1. S is a measurement point, T is not.
2. S is not a measurement point, T is a measurement point.
3. Neither S nor T are measurement points.

The basic interpolation approach is identical for all cases and can be illustrated for the first case: in the initial step, a maximum of n measurement points around T are determined, $n \in \{1, 2, 3, 4\}$. As only measurements close to the corresponding point should be considered, points with a distance above a certain threshold can be rejected in this step. The qualities from S to these points are then weighted with their distance to T (where the weight decreases with the distance) and averaged, resulting in an approximation for the quality between S and T. Obviously, this calculation can be similarly performed for the second case. In the third case, this principle is applied in both directions, i.e. the above calculation is made four times for the neighbors of S, the results then are weighted with the distance of T to the measurement points around its own position. This is outlined in Figure 2.11 for $n = 2$. In the first step, the points s_2, s_4 that are closest to S are selected and the qualities to T are calculated by using the

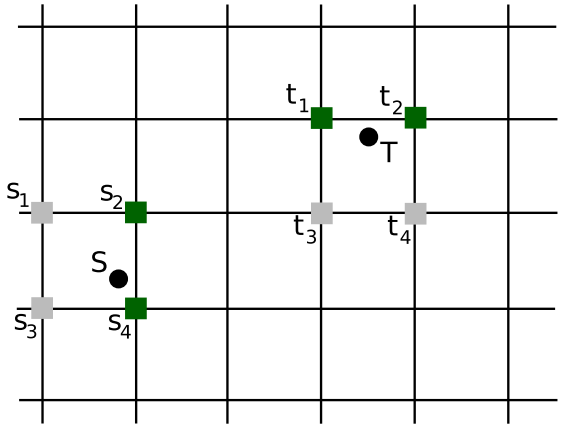


Figure 2.11: Example for an interpolation with $n = 2$. The algorithm in this case selects the points s_2, s_4 and t_1, t_2.

map size	1500×1500 pixel
area size	ca. 90×45 m
grid size	125 pixel $\widehat{=}$ 18.75 m
scale	1 pixel $\widehat{=}$ 15 cm

Table 2.3: Measurement details.

values provided by t_1, t_2. The results of this calculation are the interpolated qualities from s_2, s_4 to T. These are then once again weighted with the distances of s_2, s_4 to S and then averaged.

2.3.2 Evaluation

With these tools, we have examined the propagation characteristics in the area close to the university already known from the above flooding experiment. As map, an aerial photograph of the university campus has been used, for other areas maps or photographs with sufficient quality are available e.g. via Google Maps [Goo]. The area had a size of about 90×45 m and one grid square was 18.75 m wide; see also Table 2.3. On this grid, twelve measurement points and one mesh point have been selected, and a total of five Zaurus SL-6000 devices were used. This results in 15 different scenarios, i.e. measurement rounds. The measurement packets had a size of 256 bytes and the number of packets has been varied between 100 and 200 in the different scenarios. All devices used 802.11b network interfaces switched to ad-hoc mode on channel two.

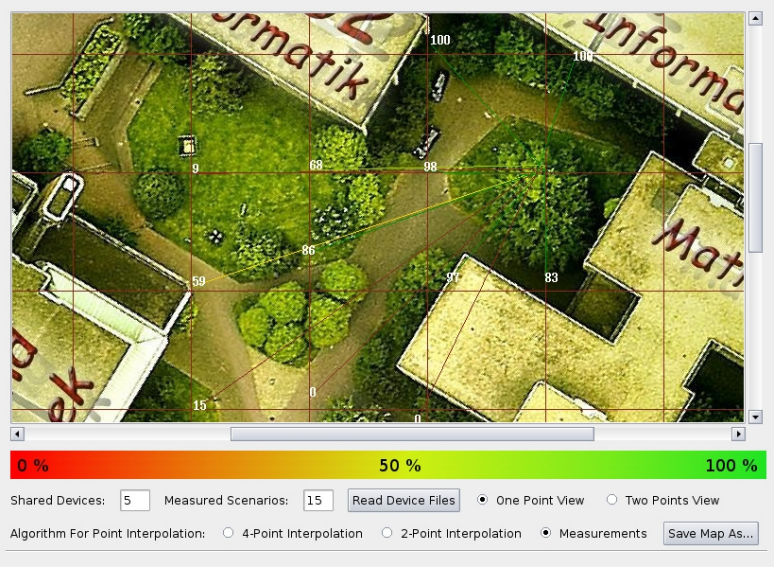


Figure 2.12: Measured link quality from selected point.

In the first step, we consider the measured values without interpolation as displayed in Figure 2.12. It shows a screenshot of the Analyzer, the single lines mark the links from the position under the tree in the upper right corner to all other points on the grid, the small white figures at the end of each line represent the measured link quality. In this scenario, it is obvious how trees and buildings impact the quality. One example here is the link passing through the group of trees in the middle, only reaching a quality of 15%. Those links to the bottom of the area are obstructed by buildings and do not even work at all. In contrast, links to points in the vicinity of the sender have a high quality. The influence of these obstacles on the interpolated link quality is displayed in Figure 2.13, here shown for an interpolation with $n = 2$. In this screenshot, the sender is positioned in about the same location as above, indicated by a white rectangle. Obviously, the coarse grid resolution does not allow to determine the quality in the proximity of smaller obstacles with enough precision. Nevertheless, e.g. the buildings' influence is clearly visible and the propagation map reflects our experiences made in this area.

The visual result of the interpolation shows that the propagation behaves as can be expected. However, it is not clear how close the interpolated values approximate the real link qualities. To clarify this matter, we have performed five additional rounds of measurements where the participants where asked to move to a new, randomly selected location in each round. This results in a total of 100 link measurements that can be compared to the interpolated values. Table 2.4 displays the results of this comparison.

Figure 2.13: Interpolated link quality.

Difference	$n = 1$	$n = 2$	$n = 4$
$\leq 20\%$	58	68	59
$\leq 50\%$	78	85	88
$\geq 50\%$	22	15	12
$\geq 80\%$	11	7	5

Table 2.4: Difference of the interpolated link quality in 100 measurements.

Here, the interpolation with two points approximates 68 out of the 100 measurements with a difference of less than 20%. On the other hand, only 15 of the measured links are misestimated with an error $\geq$50 %. In contrast, interpolating with more points reduces the number of good estimates and does not avoid many of the bad ones. If those links that are interpolated grossly wrong are examined in more detail, it becomes clear that most of these go through one of the obstacles like the group of trees in the middle or the building on the right. Thus, this has to be considered in future measurements, e.g. with a more fine grained grid resolution close to such obstacles.

2.4 Lessons Learned

In the following section, we sum up the experiences made and the lessons learned during the experiments with flooding and the propagation estimation. Inspired by [MBJ99], we investigated the radio ranges of our hardware before performing the above presented ring flooding evaluation. During these measurements, we discovered that the iPAQ radios were sometimes able to successfully deliver ping packets over more than 900 m while already a tree in the line-of-sight between two nodes can block a transmission. Thus, setting up a reliable, reconstructible 7-node/6-hop string topology for our preparatory tests was only possible by carefully positioning each device around the edges of a building.

Our next step was to set up a multihop topology where every node had multiple neighbors. After further measurement sessions, a suitable experimental site seemed to be the university parking lot. Only after the main experiment did we discover some undesirable properties of this location. As the library and other university buildings are nearby, there were other WLANs present requiring the careful selection of the radio channel (a problem also described in [RABR05]). Another issue were moving cars, possibly leading to frequent changes in the topology even though the nodes themselves did not move.

In the flooding experiment, the topology is determined at the start of an experiment by letting each node transmit a number of beacons. While one node transmits its beacon at a time, all other nodes record the packet reception. The necessary exact coordination is achieved with a *domino effect*. Prior to the experiment, a route is specified that includes all nodes of the network. The nodes use the sequence imposed by this route to coordinate their beaconing. The first node starts with its beaconing. Its successor will take over once this node has finished. This is repeated until the last node has transmitted its beacons. The domino approach worked well for the static setup in the flooding experiment but needed to be adapted for the propagation estimation where the topology changed during the experiment. As outlined above, the coordination was therefore performed by flooding the control messages, thus triggering each single action in the devices. Although this worked well in most situations, this resulted in a lot of control traffic. Furthermore, it required the supervision of each single control packet transmission from a central node as control packets sometimes were lost. For future experiments, it might therefore be necessary to develop a more generic mechanism and use a time-based coordination to reduce control traffic.

Often during our experiments, a link, the used software or a whole node failed. Every time this happened, we had to check each node manually. To ease this task, we im-

plemented a simple in-band one-hop status check. Each node wrote its current status to a file accessible via HTTP. To control the nodes' status, it was sufficient to walk around and use a Perl script to retrieve the status file from each node. Obviously, this approach has several limitations: it requires walking into the radio range of each node to be checked, the transmissions "contaminate" the experimental data, and the correction of an error still requires physical access to the affected node.

Another issue appeared during the postprocessing of the data from the experiments and the simulations. As the output format of the simulator (ns-2) differs from the trace format of the experiment (tcpdump), each tool for the analysis of the results had to be implemented twice. Furthermore there is currently no good solution for commenting and documenting the raw data so that special events during the experiments can later be remembered and reconstructed.

2.5 Refining the Experiences

Our own experiences outlined above as well as those of other researchers, see also Appendix A, show that most experimenters conducting real-world experiments have to face unforeseen difficulties: the failure of nodes during experiments was only discovered after the experiment, reproduction of results was difficult and required the unnecessary duplication of work by multiple work groups or the network showed unexplainable behavior. We believe that these problems can be alleviated if future experiments satisfy three key requirements of scientific experimentation.

- **Repeatability** is the "closeness of the agreement between the results of successive measurements of the same measurand carried out under the same conditions of measurement" [TK94]. For WMN experiments, this means that it must be possible to gain similar results in back-to-back measurements. Repeatability is also a prerequisite for reproducibility, meaning that other researchers should be able to recreate the experiments with comparable results. For mobile ad-hoc networks, repeatability is a significant challenge due to the complex impact of radio propagation and node mobility on the results of an experiment.
- **Comprehension** A scientist conducting an experiment must be able to access all relevant information to comprehend and explain the results of the experiment. There is a need for tools that collect information on different layers and combine this information to allow a detailed analysis.

- **Correctness** Any experiment may suffer from broken tools, errors with the setup and problems when conducting the experiment. While repeatability and comprehension will most likely reveal these problems, it is vital to the efficiency of a researcher to be able to verify whether any given experiment has produced valid results. This can be supported by an established methodology and a selection of appropriate tools.

To examine in detail how these requirements can be supported by a testbed, we divide an experiment in several phases: implementation, experiment specification, node configuration, setup verification, execution, and analysis. For each of these phases we discuss in the following how a testbed can support them. We assume that the testbed consists of two key elements: a number of physical devices (nodes) which may be moved around individually and the software to support and conduct the experiments. Note that this stands in sharp contrast to most existing approaches in which soft- and hardware are coupled to form a shared research infrastructure, e.g. [DRK+06, JSF+06, RSO+05].

The first phase of an experiment is the implementation of the algorithm to be tested. A good testbed will support this phase in three ways: it will 1) help to minimize the work required for the implementation, 2) seek to reduce implementation errors, and 3) encourage interoperability between algorithms implemented and evaluated by distinct research groups. As a lot of algorithms will be initially analyzed by means of simulation, reusing the simulation code instead of reimplementing it eases the workload and reduces the potential for errors. Thus a good testbed will allow the usage of SER integration tools. Encouraging interoperability is mainly a matter of interfaces and methodology. A good testbed will specify concise interfaces and best-practice methods for integrating new functionality. It will also support interoperability through a clean and simple architecture.

After the implementation is complete, the experimenter specifies the scenario used for the evaluation. In order to allow other research groups to verify the results, the specification should be a complete description of the experiment made available as a file in a standardized format. There exist at least two variants of scenarios, *strict* and *loose scenarios*. In a strict scenario each node follows detailed instructions on when each action is to be performed. Although a rigid description of the experiment fosters repeatability, there are setups in which this might not be suitable, e.g., if experiments are run as background tasks on devices primarily used for other purposes or if the number of nodes is too big to be controllable. A scenario with loose descriptions of the services and actions able to adapt to the current state of the node is better suited in this case.

When the scenario is prepared, the nodes need to be configured with the information required to run the experiment. This includes the implementation of the investigated algorithms as well as the specification of the actions and the movements of each node. This step mainly consists of the distribution of files and the configuration of nodes (setting of addresses for example), thus it should be automated as much as possible. The key to the autonomous configuration of the nodes is the experiment specification. Since this specification contains any relevant information on how each node should behave, a good testbed will be able to install the required software and perform the necessary configuration based solely on this information. This can either be done by directly distributing the specification to each node or it can require its "compilation" to gain configuration files that are specific for each node. If the nodes are physically accessible to the experimenter, the automatic file distribution can be provided with simple means, e.g., through a one-hop download. However, in loose scenarios the devices may not be available for a direct download and the required files therefore need to be distributed to nodes that are already in the field. One approach to do this is to let the nodes distribute the required files amongst themselves, i.e., whenever two nodes come in radio range, they will exchange information and files on the scheduled experiments.

The actual execution of an experiment that uses a strict scenario is extremely costly in terms of man-power and time. A verification of the test setup and the used hardware before the experiment in a controlled laboratory environment is therefore vital and should be supported by the testbed. The verification can be divided into tests involving one or multiple devices. Single device tests allow to avoid problems occurring due to lack of memory, low battery power or physical damages, for example. Tests with multiple devices can reveal problems that result form the interaction between devices. An important multiple device test which should always precede an experiment is running the complete setup installed on real devices under laboratory conditions in an emulation. Although these artificial conditions prevent the acquisition of quantitative results, the setup is not expensive, can be repeated easily, and allows the isolation of errors. A good testbed can use the position information in the scenario file to compute the virtual distances and control the topology accordingly.

The main phase of the experiment starts with the distribution of the devices. Each experiment will most likely consist of several runs in which the nodes move around. Finally, the devices need to be collected and the phase is concluded by downloading the trace files from the devices. The main phase has some properties which necessitate a dedicated support by the testbed: 1) the time in this phase is expensive: only an optimal usage of experimental time makes experiments economically feasible, 2) repeatability of

this phase is crucial for a scientific evaluation, and 3) all information available here is valuable. Due to these properties, the testbed should support the experiment by optimizing the usage of experimental time, by fostering repeatability and by collecting detailed information on the nodes' actions. The usage of experimental time can be optimized by automating tasks and by avoiding errors and therefore unnecessary repetitions. As device distribution and collection are physical tasks, the potential for automation here is small. This is different with tasks not requiring a direct (human) interaction like trace file collection. A large optimization potential also lies in the avoidance of the execution of erroneous experiments. By controlling that all nodes act within the parameters specified in the scenario, the testbed should assure that exactly the intended experiment is executed.

The repeatability of an experiment is provided if it is possible to rerun the same experiment such that the relevant parameters in both runs have sufficiently similar values. There are two ways to support repeatable experiments: comparing the parameters after an experiment to determine if it was a repetition of a prior experiment or steering the experiments to ensure that these parameters lie within an acceptable threshold. Open issues in this context are the determination of the relevant parameters and the question if it is technically and economically possible to record these parameters.

For all aspects mentioned so far it is crucial to trace the data on the behavior of nodes and on external influences as completely as possible. This data can be used for a detailed post-run analysis as well as for steering the experiment. The data to be recorded involves packet-level traces, timing and positioning information, states of higher level protocols as well as physical and MAC layer logging.

To steer the experiment, the *experiment control* component of the testbed should continuously compare the actual values of the relevant parameters to those specified in the scenario. The testbed therefore should provide a method to specify and control boundaries for these parameters, soft boundaries like "position between x-5 and x+5" as well as hard boundaries like "GPS daemon running". In case some of these boundaries are violated, the testbed can adjust the behavior of the node during the run or mark the run as invalid. If the violation is severe, this can render the whole experiment unusable and should therefore be known during the experiment. Thus, the testbed should support the transmission of status information to a central monitor station and it should also be possible to remotely correct errors or alter the configuration of the affected node.

Experiment monitoring and remote login necessitate the exchange of management messages between the nodes and the monitoring station, possibly during the experiment.

This counteracts a primary design goal of the testbed, i.e., minimizing the interference of the test equipment (both hard- and software) with the experiment. As outlined in Section 2.1.4, a solution to this is the "out-of-band"-transmission of all management messages. This can be achieved either via a separate network interface during the experiment or in the pauses between the single runs of an experiment using the tested network itself. Although the last method does not allow to stop erroneous runs directly, this is not problematic if runs are short. Furthermore, it is more practical than the more expensive first method.

The postprocessing can be divided into organizing the raw data gathered during the experiment and analyzing it. This phase should be governed by the principle that the raw data is a valuable resource. It needs to be documented, stored and published. Based on this data, independent researchers must be able to verify any conclusions that are drawn from the experiments. A good testbed will provide mechanisms to ease the documentation and support storage and publication of the involved files, e.g. via the repository of the CRAWDAD project [cra]. Furthermore it will provide or incorporate an extensible toolset for analyzing the raw data.

The first postprocessing step is the structured, permanent storage of all raw data. As there are also a lot of other files involved in the postprocessing such as scenario files or tools, the testbed's automatic file handling should include these. One possibility is the implementation of a file management framework that defines interfaces to access, view, annotate, process, and store these files. The organization of the raw data is concluded by the documentation of the events and conditions not recorded in the traces but perceived by the human participants. The tools to process the raw data should provide functionality for consistency checks, data analysis or for enabling trace-based simulation. All these tools can be used for multiple experiments, thus the testbed should foster reusability to reduce work. If the tools are modular and reusable, this also increases reliability as results can be easily reproduced. Therefore the goal has to be the creation of a reusable standard toolset for the postprocessing of WMN experiments which should be publically available, extensible and well documented. The analysis tools should support the different input formats of real-world traces and simulator traces to allow comparison of results from both evaluation methods.

2.6 Chapter Summary

In the first part of this chapter, we have presented a guidebook to existing WMN experimentation strategies. This guidebook contains an overview of the most common tools, lists important topologies, and surveys the metrics that have been used for the analysis of such experiments. This knowledge is complemented with the experiences made during our own experiments on ring flooding and the evaluation of a propagation estimation toolkit. In the third part of this chapter, we have then combined this knowledge. Repeatability, comprehension and correctness have been identified as the three key requirements that future experiments should fulfill and we have described how a testbed can support these requirements throughout the different phases of such experiments.

Chapter 3

The EXC Testbed

Chapter Outline

A lot of full-scale WMN experiments follow a "let's install the software and see what happens" approach that results in a proof-of-concept but does not allow the protocols to be investigated in detail [KM07a]. Instead, in order to acquire verifiable knowledge, protocol behavior must be examined in well-defined situations, e.g. on a specific topology or during transition from a partitioned to a connected network. Although the radio layer has a non-deterministic influence here, tightly controlling all other variables like movement or timing of the used software increases the chance of creating the desired situation. The coordination of such systematic experiments is a complex task: 1) The devices, spread over a possibly large area, have to perform actions at predetermined points in time. 2) The tested software as well as the protocols are prototypes, thus errors are likely to occur. 3) The used tools must not disturb the experiment as this would distort the results.

To cope with these requirements, we have developed the concept of *semi-automatic experiments*. While parts of the experiment are fully automated in *runs*, it still can be influenced between these runs. This concept is implemented in EXperiment Control (*EXC*), a toolkit to control and steer experiments with wireless multihop networks that is based on the knowledge outlined in the previous chapter. EXC is a pure software approach that runs on Linux-based systems supporting the Ruby programming language. This approach allows researchers to use their own hardware to build a testbed and execute experiments in a controllable fashion with moderate effort. EXC supports experiments under fully realistic conditions, as we do not require any modifications of either the environment or the used hardware. Thus, it is possible to perform controlled experiments in the very environment for which the algorithms have been designed. The

development of EXC has been guided by the belief that a WMN testbed should be open source, not restricted to special hardware, customizable, and not bound to any specific location. As outlined in the previous chapter, it must support repeatable, comprehensive, and correct experiments. Due to the complexity of such a testbed, we implemented it with a modular, extensible architecture. In the following chapter, we present this toolkit together with a description of the different experimental aspects supported by EXC. This chapter is based on a paper that has been accepted for publication [KOM08]. The analysis component of EXC is called EDAT and will be presented in Chapter 5; a full-scale study on experiment repeatability performed with EXC can be found in Chapter 6.

3.1 Movement and Control

Two of the most important questions to be solved during preparation of real-world experiments with (mobile) wireless multihop networks are 1) How to move a large number of devices in a large area? 2) How to control the actions of the devices and the timing of the actions?

3.1.1 Existing Concepts

In some existing testbeds, nodes are moved automatically e.g. by robots [DRK+06, JSF+06]. This fosters the repeatability of movement patterns but also leads to high initial costs and makes it difficult for other researchers to repeat the experiment. Furthermore, this does not allow for tests under realistic conditions such as they are necessary in the development of car-to-car communication for example. Another possibility is to let humans move the devices [LLN+02]. With this, the complexity of an automatic movement can be avoided, fixed costs can be kept low and the variable costs per hour of experiment mainly depend on the participants' salary.

For the coordination of the nodes' actions, there are currently two approaches: *manual* or *automatic control*. In manually controlled experiments, all actions on the participating nodes are triggered directly by the users. This allows for a very flexible control but either requires users with high qualifications and a lot of knowledge about the testsystem, or limits the complexity of actions. Besides, such experiments are error-prone and it is hard to obtain even a basic level of repeatability since exact timing of actions is difficult. In automatically controlled experiments such as the previously described

flooding experiments or the comparison of four MANET routing protocols described in [GKN+04], all actions to be performed are laid down in a script. The script is executed on startup and triggers all actions to be performed during the whole experiment without the need of further interference by the participants. This allows for complex actions and high timing precision with unexperienced users and possibly increases repeatability of actions. However, the automatic control approach is not flexible and no knowledge about the state of the experiment is available. Above as well as in [GKN+04], it has been reported that failing nodes reduced the number of actually participating devices. These errors can require the repetition of the whole experiment, thus they should be avoided or compensated for to the best possible extend.

3.1.2 Semi-automatic Control

As outlined above, protocol behavior should be examined in well-defined setups. The network size in most of these experiments will be in the order of some tens of nodes as this already allows the creation of interesting scenarios and at the same time limits complexity and cost. The movements depend on the type of network: when testing an emergency communication system among firefighters, participants will carry a personal digital assistant (PDA) or laptop, in a car-to-car scenario they will drive a car equipped with the respective device. The devices will be moved by unexperienced users that are only roughly briefed about the functioning of the experiment. In case a problem occurs on a device, the user would not be able to overcome the problem, and no one with sufficient knowledge would be close by.

To conduct such experiments in a systematic way, we have developed the concept of *semi-automatic control*. Each experiment here is partitioned in a number of runs that are only loosely coupled. In a run, all node actions, like sending certain packets or moving the node, are precisely scheduled based on a predefined script. As most actions can be executed without any interaction from the users, this increases repeatability and minimizes errors. Those errors that still occur only corrupt one run, not the whole experiment. After each run, the nodes pause and the person responsible for controlling the experiment (called *operator*) remotely queries the state of the nodes. If errors such as wrong position or a software crash have occurred, the operator has the chance to overcome these before continuing. If a run could not be conducted correctly, e.g. because the movement path of one of the nodes was temporarily blocked, it is also possible to repeat this run later on. After this control phase, the operator remotely

selects and starts the next run. As all the communication takes place between runs, the runs themselves are unaffected.

Besides dealing with unforeseen errors, this also allows the experiment to be steered based on experimental feedback that currently is only feasible with fixed nodes with an additional network for control traffic [RSO+05, SOSK05]. An example is the measurement of maximum throughput along a chain of nodes: after specifying runs for the whole range of possible throughputs (for a 802.11b based scenario, this would be 0–5.5 Mbit/s), one would start with a value just in the middle, e.g. 2.75 Mbit/s. After the end of the first run, the operator can query the last receiver for its reception rate. Depending on the result of the query, he then can select the next run to be executed similar to a binary search.

This semi-automatic control approach results in certain demands that the experiment control software must fulfill. It must allow the devices to be automatically steered based on command scripts, i.e. to start arbitrary actions either at predetermined moments or a selectable number of seconds after a certain event. Furthermore, it must be possible to communicate with the devices between runs because the operator at a central node needs to supervise the experiment by querying the nodes' state and by remotely influencing the nodes' behavior.

3.2 Implementation and Practical Aspects

In this section, we present our EXC software that implements the above described approach. As the software is too complex to be described in full detail, we concentrate on those aspects relevant for researchers that want to steer their experiment with EXC.

EXC is implemented in the scripting language Ruby because this allows for a similar code base on different platforms. The routing components are implemented with click [KMC+00]. For the GUI on the *monitor* (the node from which the experiment is controlled), we have used Qt and the corresponding Ruby/Qt bindings, the graphical user interface on our mobile nodes is implemented with GTK+ [QTT, QRB, GTK]. Configuration scripts, scenario files and movement files are mostly specified in XML and there is also a parser for movement files in ns-2 [NS2] syntax.

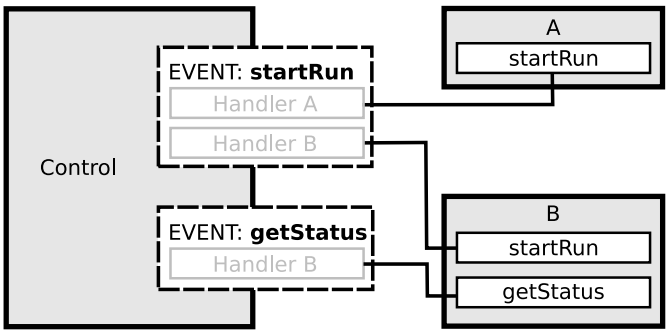


Figure 3.1: Schematic overview of two objects A and B that register their `getStatus` and `startRun` event handlers with `Control`.

3.2.1 Architecture

As outlined above, the devices move around and perform certain tasks during an experiment. Among these tasks are starting a routing protocol or a packet source, telling the user the next position to go to or setting the devices' clock. All these tasks can be regarded as actions triggered by an event. Therefore an event handler framework is the basic building block of the EXC architecture. The basic idea is to use a central class called `Control` that allows other classes to register their event handlers. All events that occur are raised within `Control`. This class delegates the handling sequentially to the registered handlers. This is schematically outlined in Figure 3.1. Here, the startRun handler is implemented by both objects A and B while only B has a handler for the getStatus event. These handlers are registered with Control and are called whenever the corresponding event occurs. Note that different handlers can react differently to the same event. For example, `startRun` can cause the handler of A to start counting the duration of the run while the handler of B starts writing a trace file. In order to determine which handlers (and with this which functionality) are needed during an experiment, we use an XML configuration script listing the required components.

This architecture has several advantages, the most obvious one is its generality and extensibility. The EXC scheduler is a good example: as it is designed to trigger an *event* at a certain moment in time, all events can be scheduled. All components of EXC follow this generality principle by working with such events. If other researchers want to use EXC and extend it with their own functionality, it is sufficient to write an appropriate handler and specify it in the configuration script. The new functionality is then fully integrated in the control software and can use all features offered by it. Furthermore, this architecture also allows to easily exchange certain components. An

example is the packet tracing for which we in general use tcpdump [TCP] during our experiments. At a certain point it was necessary to record packets at the moment they passed the firewall, a feature offered by ulogd [ULO]. For this it was sufficient to implement a second event handler that relied on ulogd but reacted to the same events as the one for tcpdump.

3.2.2 Plug-in Mechanism

The entire event handler management is performed by an object-oriented method plug-in mechanism. An object can register any of its own methods with the Control-class. After a method X has been registered, the event X can be raised by calling `Control.X()`. Whenever the method `Control.X()` is invoked thereafter, all methods that have been registered with this keyword and signature get called in their registration order. With this feature, it is thus possible to install handlers for single events.

Beside actions that can be performed in one single method call, certain programs and components can run for a longer time during an experiment. This may be a packet tracer like tcpdump or a new routing protocol implemented as Linux kernel module. Such long-running components are started at a certain instant, run in the background for a while and are stopped at some future moment. During runtime, the user or operator may want to know whether this component is still working correctly. For the routing protocol, starting means to load the kernel module, information about the state may be acquired via the "proc"-interface and stopping is performed by unloading the module. For programs with such requirements, it is sufficient implement a subclass of `Service` and appropriate `start`, `stop` and `status` methods. After these methods are registered via the plugin-mechanism, they are accessible within EXC and it is thus possible to fully control the routing protocol with EXC.

3.2.3 Control Scripts

To automate the sequence of actions as much as possible, it needs to be specified prior to the experiment. In EXC, this is done via an XML script. Each XML tag corresponds to the case-insensitive name of one method registered in `Control`. Method parameters are listed in child-tags and a tag also contains information on affected nodes and execution time. In this way, new methods registered via the plugin mechanism are callable from the scenario script without further programming. An excerpt from such a script looks as follows:

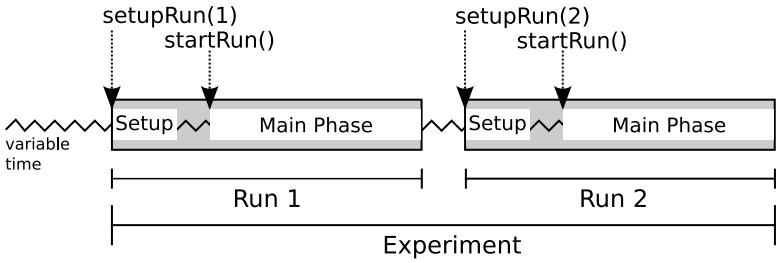


Figure 3.2: Semi-automatic experiment with setup and main phase. A phase is only started upon an explicit command.

```
<userinfo nodes="1-3" time="1" id="start">Starting...</userinfo>

<starttracing nodes="1-3" depends="start" time_delta="5">
   <param key="file">trace1</param>
   <param key="snaplength">200</param>
   <param key="interface">wlan0</param>
</starttracing>
```

The first tag specifies a userinfo event (this may print the given string on the command line or show it on the GUI) on nodes one to three at time one. The packet tracer is started five seconds later as specified by the second tag.

3.2.4 Semi-automatic Experiments

During the preparation of an experiment, each single run is specified in a control script. In our experiments, a run is composed of a setup phase and a main phase as shown in Figure 3.2. In the setup phase, the nodes start the necessary software, schedule all the actions for the main phase and the users move to their start positions. At the end of this phase, the monitor sends out a status request to verify that no errors occurred and then the run can be started. All this is available via two commands, `setupRun(<ID>)` to select a certain run from the experiment description and launch the preparation phase and `startRun` to trigger the main phase. Thus, two commands are sufficient to trigger the next run if no errors occur while the two phases are still decoupled. This allows for the delay of the main phase, e.g. in case one of the users takes more time than expected to reach his start position.

3.2.5 Remote Method Invocation

As the actions on the nodes should be accessible remotely, the methods of `Control` must be callable over the network. This is performed by the `RemoteCommunication` class that provides remote method invocation (RMI). For each node participating in the network, this class holds a stub serving as placeholder for the remote `Control` instance. If a method should be called on a remote node, the stub class constructs a message object that serves as a container for method name and arguments. The message is then transformed to a string via the marshaling feature of Ruby and transmitted to the remote node. There, the message is reconstructed and the corresponding method of `Control` is called. By convention, the return value of methods starting with the keyword `get` is transmitted over the network. In contrast, the return values of methods without this prefix are not returned remotely.

3.2.6 Communication

The EXC RMI allows the state of a node to be queried by executing a `getXY`-method and also provides the ability to influence the nodes' behavior by remotely executing methods. To this end, data packets must be exchanged with the devices. As outlined above, this is either possible by using an additional network and implementing out-of-band monitoring or by using the experimental network itself by means of in-band monitoring.

In EXC, both options are available because all control communication takes place over a *control network interface* specified in the node configuration file. Out-of-band monitoring is technically easy but requires additional network hardware, thus increasing cost and complexity. The central task when using in-band monitoring is routing, the transmission of control packets to the target nodes. The nodes in experiments with mobile multihop networks are spread over a larger area and the network topology is unknown. Thus only a limited number of these nodes is within direct reach of the monitor. The conditions under which the messages need to be delivered limit the choice of a suitable transmission method: 1) During a run, the method must not disturb the experiment. 2) Packet delivery must be robust and work on arbitrary multihop topologies.

At first glance, this looks like a task for one of the well-known MANET routing protocols. However, all proactive protocols continuously transmit packets and thus violate condition 1). Another option are reactive routing protocols like AODV [PBRD03] or DSR [JMH03] that fulfill their task in two steps. In the initial phase, a request is flooded

to discover a route to the target node, then the data is transmitted over this route in the second step. However, for our remote method invocation either method name and method parameters or a short return value need to be transmitted. As this information fits in one packet, we directly flood the control messages. As shown in Section 2.2.2, a wireless multihop network can deliver a high percentage of such control packets. To deal with losses still unresolved, each of these messages is sent out multiple times[1]. This method worked sufficiently well for all our experiments. If such losses are problematic, it is easy to use acknowledgments and retransmissions instead. This control packet exchange is based on the flooding implementation already used and tested in Section 2.2. The flooding itself is hidden behind a virtual Linux tun/tap interface that serves as control network card. All packets transmitted over this interface are encapsulated and then flooded in the wireless network. The target node unpacks the packet and delivers it over its own virtual network interface. This mechanism is called *monitor routing* because it serves the in-band transmission of experiment monitoring information

3.2.7 Trace Files

Experiments result in a large number of trace files: for each run there are position traces, packet traces or application layer traces. In order to associate these files with the correct run, they are stored in a special directory. EXC uses the `DirectoryGenerator` to keep track of the current trace-directory that is created new for each run and follows the notation *experimentname/runname/run-id_timestamp*. This also allows the use of the same run-specification multiple times without overwriting previous trace files, e.g. in case a run has to be repeated due to an error. For components that need to write such files, it is sufficient to call the `Control.getCurrentDirectory()`-method and write the file to this directory.

At the end of an experiment, the trace files can be automatically collected by the monitor. For this, EXC uses an FTP-server on the monitor to which the different nodes transmit their compressed trace-directory upon a call to the `Control.uploadTraces`-method. In addition, the file transfer mechanism is also available to remotely update the control scripts and even the EXC installation on the nodes.

[1] In the current implementation, it is sent out three times.

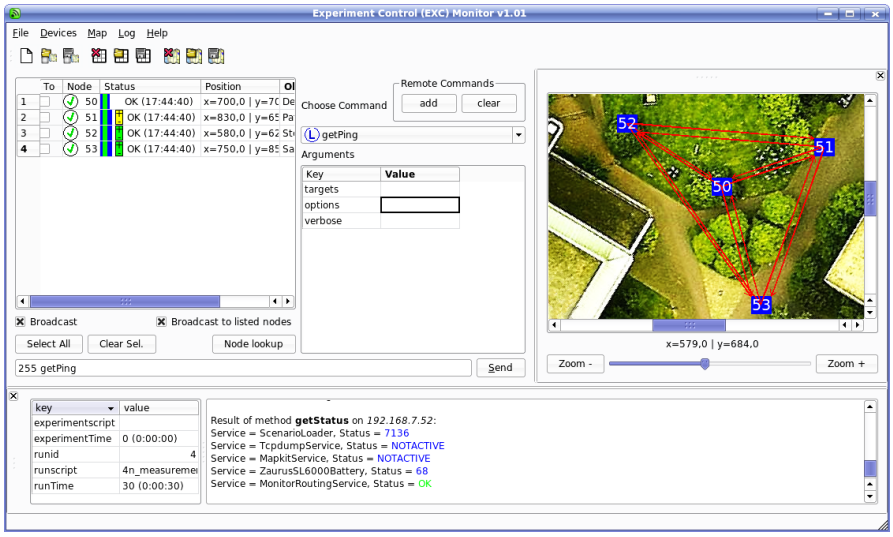


Figure 3.3: A screenshot of the EXC GUI that is running on the monitor used for an experiment with four nodes.

3.2.8 Graphical User Interfaces

Due to the large amount of complex information that must be handled by the operator on the monitor, EXC provides a graphical user interface that is shown in Figure 3.3. It has been implemented with Qt and Qtruby [QTT, QRB]. The GUI allows to monitor the participants' state, send RMI commands, visualize the network topology and keep track of other information regarding the experiment. The table on the left of the GUI contains a list of the participating nodes along with their status. The RMI command interface is located to the right and below. This interface adapts to the current configuration of EXC: if a plugin is loaded, the corresponding methods are available. The scrollable window on the right displays the current position of the participants and the topology if this information is available on the monitor. The component on the bottom stores information about the experiment state and displays the raw information exchanged between monitor and nodes. A detailed description of this GUI can be found in [Ogi07].

Besides the GUI for the monitor, EXC also provides a graphical user interface for the nodes that is shown in Figure 3.4. It is implemented with C++/GTK+ [GTK] as this allows for an easy cross-compilation for arm-based platforms like our Zaurus SL-6000 PDAs. The node GUI features a map display that provides a bird's-eye view on the area. On this map, the track the participant has to follow is displayed as white line.

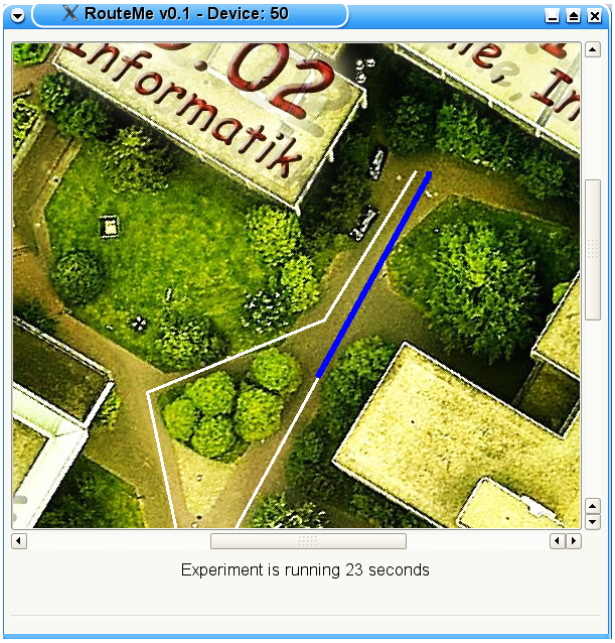


Figure 3.4: A screenshot of the EXC GUI for the node.

The current position on the track is indicated by a color change. Furthermore, the node GUI is able to display text messages that can be triggered by `userinfo` events.

3.2.9 Emulation

Emulation tools can create and manipulate a multihop topology without requiring actual node movement [KM07a]. Running such an emulation prior to the full-scale experiment is a valuable component as it helps to discover problems in the setup as well as bugs in the software.

Existing tools like MobiEmu [ZL02] use a central server to enforce the network topology as predefined in a script: if two nodes move out of each others' virtual radio range, a command is sent that makes these nodes block packets from each other. First of all, such an external tool complicates the setup, as e.g. the nodes actions and emulation need to be synchronized. Besides, such a centralized approach is problematic as it cannot be assumed that a central server knows the whole topology a priori. This is the case when a positioning system is used on the nodes. Examples for such systems are GPS (that must be replaced by an emulation component for the pretest) or tools like the manual positioning system implemented in Mapkit, see Section 2.3. The positioning component is part of the system to be tested, thus the position is only known on the

nodes. Furthermore, even without such a positioning system, only the node knows where it should be because it has to display the movement pattern to the user. By enforcing the movement via a centralized emulation, such components cannot be tested because errors affecting the node position simply do not occur.

Due to this, we implemented a new, mixed centralized/decentralized emulation component in EXC that only assumes that nodes know their own position. The component on each node continuously requests the position with local `getPosition` method calls and transmits it to a central machine. This central server collects the positions and integrates these to form the global position model. Based on this model, it decides which nodes can communicate. In the current implementation this is done by a simple distance metric. The connectivity information is transmitted back to the nodes that set and remove corresponding iptables rules. Problems like the time synchronization between emulation server and emulation clients as outlined in [ZL02] do not occur as the server calculates the topology based on the positions transmitted by the nodes. The communication between nodes and central machine takes place over the emulation network interface specified in the configuration file. Obviously, the emulation server should not be blocked by the nodes. Thus, the server either must be hosted on a dedicated machine that does not participate in the emulation or an additional (hardware or virtual) network must be available.

In all of our emulations, we treat the monitor as any other node. This creates a more realistic emulation as the monitor is also affected by topological effects such as network partitions in the real experiment. If the monitor cannot issue control commands to devices in other partitions, this might be a situation that should already occur in the emulation. To hide the complexity of the emulation setup from the experimenter, we have written appropriate plugins for EXC. Clients need to specify the `EmulationClient` service, the server `EmulationServer` in the configuration file. Both services react to the `startEmulation` event that triggers the above described mechanism and with this the emulation.

3.3 Experiments

In this section, we report on experiences made during case studies performed with the support of EXC, amongst them experiments with the `whoisthere` node presence detection approach presented in [TSM07]. A node using `whoisthere` regularly sends out a beacon announcing its own presence as well as the local information about other

nodes. Upon reception of a beacon, the information is incorporated in the local presence table. This way, the information spreads throughout the network. As the focus here lies on the behavior and performance of EXC, we will not go into the details of the tested algorithms. An EXC study concentrating on the experimental results themselves can be found in Chapter 6. For the following evaluations, both static and mobile scenarios were used, thus spanning the different setups encountered when conducting experiments with wireless multihop networks.

3.3.1 Integration

An important early task is the integration of the software to be tested; we use the `whoisthere` implementation as example here. In order to control this command line program from EXC, the following Ruby class had to be implemented:

```
class WhoisthereService < ServiceForExternalProgram

  def initialize()
    super()
    # Register methods with EXC
    Control.register(self,:startWhoisthere,
                     ["interval","nodeNumber"])
    Control.register(self,:stopWhoisthere, [])
    Control.register(self,:stop,[], METHOD_INTERNAL_TO_EXC)
    Control.register(self,:getStatus, [])
    @start_id = 1
  end

  def startWhoisthere(interval, nodeNumber)
    # Determine command line parameters
    nic     = Configurator.get('interface_name')
    b_addr  = Configurator.get('broadcast_ip')
    node_id = Configurator.get('nodeid')
    trace   = Control.getCurrentDirectory() +
                     "whoisthere_trace.txt"
    trace += @start_id.to_s()
    @start_id += 1
    # Assemble command
    command = "whoisthere␣-i␣#{nic}␣" +
      "-b␣#{b_addr}␣-bi␣#{interval}" +
      "-n␣#{nodeNumber}␣-id␣#{node_id}␣" +
```

```
      "-log␣#{trace}"
    # Start 'whoisthere' as subprocess
    io = IO.popen(command)
    @pid = io.pid()
    @last = OK
  end

  def stop()
    stopWhoisthere()
  end

  def stopWhoisthere()
    killchilds(@pid)
    stopService(self.class.to_s())
  end

  def getStatus()
    return {self.class.to_s() => status()}
  end
end
```

The constructor `initialize` is responsible for registering the implemented methods by means of the plugin-mechanism described in Section 3.2.2. As all input to `whoisthere` is performed over command line parameters, the `startWhoisthere` method collects the relevant parameters that are all available via EXC method calls. After the command line has been assembled in a string, `whoisthere` is started as a subprocess. The `stopWhoisthere` method uses a method provided by EXC to kill all child processes of the current process (identified by `@pid`) and then stops the process itself. The current state of `whoisthere` is determined in `getStatus` by inspecting the system process table with a method provided by EXC.

With this 45 lines of code, `whoisthere` is fully integrated with EXC: it can be started and stopped from an XML script at predetermined moments, the trace files are stored in the correct directory, the status is available in the GUI and commands can be executed remotely.

3.3.2 Experiment Setup and Network Topology

In most experiments, an IBM Thinkpad X40 laptop with Gentoo Linux was the monitor and Zaurus SL-6000 PDAs running OpenZaurus Linux in version 3.5.4.2 [opec] served as

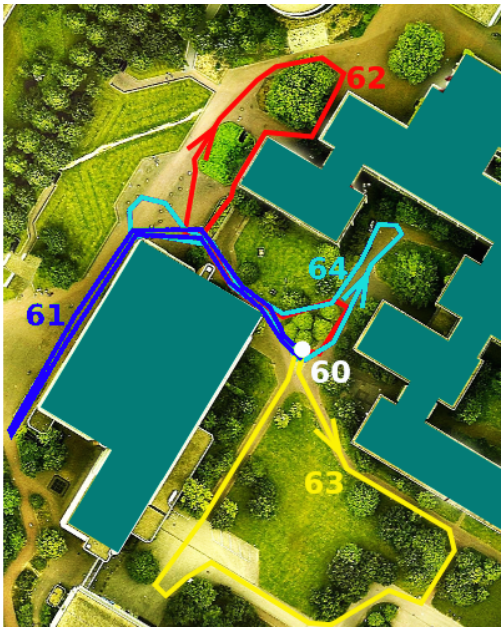


Figure 3.5: Movement of the four mobile nodes (61-64) and position of the stationary node 60 during a mobile experiment.

nodes. Besides, some experiments were conducted with an Xubuntu-based EXC Live-CD that allows to run EXC on x86- or PowerPC-based, CD drive equipped computers either in node or monitor mode, see Appendix C for details on the Live-CD.

The experiments consisted of up to eight nodes and one monitor. The topology and also the location of the network varied: some experiments have been conducted inside the offices of our university, others were performed in a students' hostel, on the campus and in a nearby residential area. An example for a mobile experiment can be found in Figure 3.5, one of the static setups is shown in Figure 3.6. The mobile experiment consisted of a total of five nodes: four mobile nodes (IDs 61-64) and one static node with the ID 60 that also served as monitor. The movement pattern was adjusted to normal walking speed and fully specified in advance. The volunteers that carried the mobile devices across the campus followed the pattern displayed on their nodes' GUI, see Figure 3.4. Due to the bird's-eye view on the area, it was easy to follow the pattern and from the visual impression, the error in the movement path was in the range of meters.

3.3.3 Detected Errors

One of the primary motivations behind the design of EXC is improving correctness by reducing errors and providing a fast recovery from errors that still occur during an experiment. As with every other evaluation, we encountered a number of such error situations during our measurements. These are described in the following to get an impression of the capabilities of EXC for these tasks.

During the preparation of one experiment, we deleted by mistake the basic trace directory on all participating nodes instead of its contents. When the first run of the next experiment was executed thereafter, the tracing component reported an error: the packet tracer (tcpdump in this case) needed to write its file to the trace directory which was not present. This was detected right after setting up the first run and thus could be fixed directly without abandoning the experiment.

Another error occurred during the mobile experiment due to a misconfiguration of the sleep cycle of the nodes. Per default, i.e. after flashing the standard operating system image to the SL-6000, the PDAs go to sleep mode after five minutes without user input. As our experimental setup did not require any user input in combination with EXC, this situation occurred shortly after the experiment was started. Here, the concept of EXC with an experiment divided in runs showed its whole potential: right after a run, some of the nodes did not answer to status requests. After requesting an error description from one volunteer over a walkie-talkie ("the screen suddenly went black"), we were able to wake up the devices again and synchronized the state between the nodes. The run was restarted remotely and the experiment set back on track. The misconfiguration itself could be overcome with a workaround ("click the touchscreen once per minute").

During one of the static indoor experiments, no power connection was available. In the middle of a run, some of the devices switched themselves off due to a lack of battery power. This was discovered right after that run as the devices did not answer any more to status requests. After recharging the battery, the experiment could be restarted with the run that failed first. Furthermore, as consequence of this incident, we implemented a service to remotely query the battery state and integrated appropriate feedback in the GUI, see Figure 3.3.

In the course of one presence detection experiment, the network interfaces of two nodes crashed due to unknown reasons. This ruined the current run and left the devices unaccessible. The error was detected right after the run, the devices had been rebooted manually and the runs could be repeated, leading to a full set of runs.

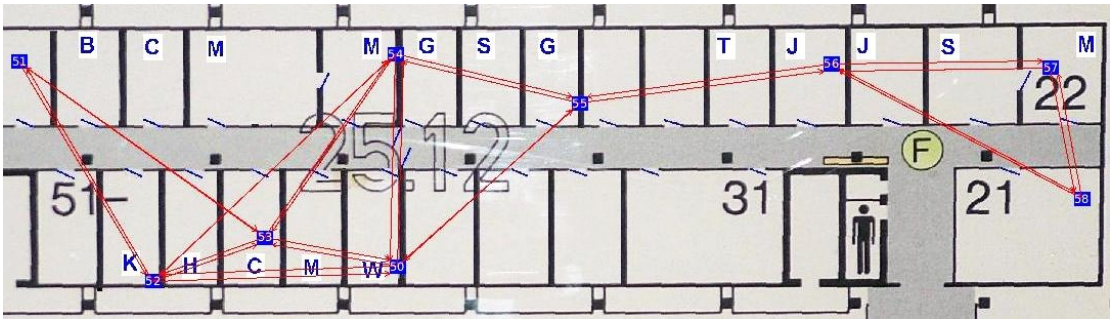


Figure 3.6: A screenshot of the EXC topology visualization showing a chain-like setup in one of the presence detection experiments with nine nodes.

3.3.4 Topology Visualization

Obviously, information propagation speed has a large impact on the performance of the `whoisthere` presence detection. Most interesting here is a string-like network topology that has some alternative paths. This is one of the scenarios outlined at the beginning of this chapter in which an experiment is to be performed on a well-defined topology. Setting up this topology was much simplified by the network visualization feature of EXC as shown in Figure 3.6. After the nodes had been distributed over the different offices of the second floor, we were able to test the topology. Some nodes were temporarily disconnected in the first setup. After two more "moving nodes and testing the topology" cycles, we were able to setup a topology that suited our experimental needs.

3.3.5 Communication

In the mobile experiment shown in Figure 3.5, all nodes moved effectively in a circle and returned to the starting point after the run. Thus, monitor communication generally took place over one hop here and therefore worked fine. In a second set of mobile experiments, the nodes moved from one position to another between runs. Also here the communication worked without problems as the network spanned multiple but nevertheless static hops. In fact, mobility itself and the resulting topology changes may not be an issue at all for the monitor routing of EXC: in all of our experiments that involved mobility *during* runs, the nodes were nevertheless stationary *between* runs.

The network with the largest diameter in our setups is the one shown in Figure 3.6. Because the monitor with id 50 is positioned in the middle, the monitor routing had to

cope with up to three hops here. This worked without perceptible performance degradation throughout the experiment. As the monitor routing is implemented as a simple form of flooding, it will very likely perform like ring flooding, see also Section 2.2.2. Furthermore, we also verified that all runs were free of monitor communication packets. Thus, EXC does not produce any packet influencing the protocols to be tested.

3.4 Related Work

The only testbed that can be directly compared with EXC as it is purely software-based and offers unlimited mobility in real environments is APE [LLN+02]. It comes as a Linux Live-CD and relies on shell scripts to fully automate the actions during the tests. After APE is booted on the participating laptops, the single runs can be started one after the other in a predefined sequence by pressing a button on any of the devices. In contrast to that, EXC follows the philosophy of a semi-automatic control of experiments. Only the actions *during* a run are automated, the decision which run is to be executed next can be made at runtime from a central point. The feedback on the experiments' status allows to encounter frequently occurring errors like node failures or wrong configurations already during an experiment and even permits to repeat runs. Furthermore, as stated in the REQUIREMENTS file of the APE distribution, APE needs "i386 compatible computers (preferably laptops) equipped with ORINOCO IEEE 802.11 WaveLAN cards". In contrast to that, EXC is highly portable and platform-independent and there are no further requirements on CPU architecture or network interface hardware models. The EXC base system (except for the platform dependent graphical user interfaces that require cross-compilation) works with similar code on the most recent Gentoo, Ubuntu and OpenEmbedded Linux distributions on i386-based laptops as well as ARM-based PDAs and also has been successfully tested on PowerPC Apple laptops.

All other testbeds that offer the ability to examine mobile nodes are *shared research infrastructures* (SRI) out of which we consider MiNT-m [DRK+06], Mobile Emulab [JSF+06], and ORBIT [RSO+05] here. The idea is to concentrate the physical resources at a central place and offer remote access to these resources to other researchers. This allows for sophisticated setups and a larger number of nodes, and researchers that want to perform experiments do not have to setup their own installation. Both MiNT-m [DRK+06] and Mobile Emulab [JSF+06] are indoor testbeds that use robots for node mobility. An important aspect in these systems is the combination of robot steering and

positioning that must be as precise as possible. As the multihop topologies are created inside a single room, both testbeds need radio hardware with limited range. While Mobile Emulab relies on sensor network hardware working in the 900 MHz band, MiNT-m uses attenuated IEEE 802.11a/b/g-cards. The ORBIT [RSO+05] testbed currently also consists of an indoor installation but follows a different approach to achieve mobility. Here, a total of 400 nodes are installed in a grid. Instead of moving nodes, the signal is switched from node to node. Although an outdoor installation is planned, it is not clear what it will look like.

Obviously, the approach and focus of EXC is orthogonal to such shared research infrastructures. While an SRI initiative results in one single testbed for all researchers, it is our goal to enable each researcher to set up an experiment with his own hardware. Where SRI testbeds need to rely on automatic mobility by means of robots, our goal is mobility by equipping each participating person with a device and thus allowing also tests with human centric networks. Furthermore, EXC makes it possible to perform experiments in the very environment for which the algorithms are designed while SRI approaches allow for tests in a controlled (indoor) laboratory setting.

Besides the above described testbeds, there is a lot of interesting work related to the experimental evaluation of wireless multihop networks. These are experiments performed with network prototypes, testbeds for mesh or sensor networks as well as software tools that support e.g. monitoring or emulation. Because these projects pick out certain special aspects of experimentation and therefore follow very different goals than a fully-fledged mobile testbed, we do not consider these here. For a detailed overview, refer to [KM07a] and Appendix A.

3.5 Chapter Summary

In this chapter, we have presented EXC, a software toolkit to conduct experiments with mobile and static multihop networks under fully realistic conditions. EXC is based on the new concept of semi-automatic control that divides each experiment in a series of runs that can be started individually. This is only possible with the central, flexible control feature of EXC that allows to request the nodes' status between runs. With this, errors occurring during the experiment can be detected and repaired. EXC is the first toolkit that allows to conduct real-world experiments with fully controlled mobility. By presenting the experiences made during different research projects, we

show how these features enable the evaluation of static and mobile ad-hoc networks under realistic conditions.

Chapter 4

Time Synchronization

Chapter Outline

A fundamental problem in real-world computer network experiments that also occurred repeatedly during our own evaluations is that each system uses its own local clock to timestamp events or schedule actions. As these clocks do not run perfectly synchronous, this can have negative effects not only when the experiment is running but also on its analysis. During an experiment, synchronized clocks can be a crucial factor for the coordination of the nodes' actions. After an experiment, event log files where all timestamps refer to a single reference clock instead of multiple local clocks are highly desirable for the investigation of timing related performance parameters, event correlation, and visualization. In production environments, this problem is often solved either by using a high-precision, special purpose external clock or by synchronizing the nodes' clocks over a network. A prominent example for such a synchronization protocol is the Network Time Protocol (NTP) [Mil92], but a number of other solutions exist as well [RBM05]. Both of these methods cannot be directly applied to experiments with wireless multihop networks because they either require the use of specialized, expensive hardware or a permanent, reliable network connection between a reference clock and the nodes that cannot be guaranteed during such experiments.

To improve clock precision during an experiment, we have developed a simple method that uses off-the-shelf software called *NTP skew correction*. It exploits the characteristic behavior of these clocks and the capability of the NTP daemon to correct clock speed when not connected to a reference clock. To evaluate the synchronization accuracy achievable with hardware typically used for real-world experiments, we have performed a measurement study that is presented in Section 4.2. It shows that clock precision can be improved by two orders of magnitude with simple means. A paper thereon has been published [KZM07].

Although the clocks' deviation can be limited to a few milliseconds with this method, the precision may still be too low for the analysis of time-related performance parameters like round-trip time. Furthermore, even if the clocks were perfectly synchronized, it takes some system dependent (and potentially non-deterministic) time from the occurrence of an event until it is actually timestamped and recorded. We call this the *timestamping delay*. While it may be possible to use customized hard- and software to bound the timestamping delay, such a solution cannot be employed for the off-the-shelf systems often used in network experiments.

In order to avoid these problems we have developed *MLE timestamp synchronization*, an algorithm to correct the timestamps of the individual log files after an experiment is completed which is based on a maximum likelihood estimator (MLE). For this synchronization, we take advantage of a specific characteristic of networks with local broadcast media: a transmission is often received by multiple nodes. Upon recording this transmission, each node uses its local clock to provide a timestamp for the same physical event. Such shared events can be used as *anchor points* that relate the different clocks to each other. The combination of multiple anchor points allows for a very good estimation of this relation and finally for an accurate post-experiment synchronization of the log files. A description of this approach is provided in Section 4.3, it is based on a paper that has been accepted for publication [SKR+08b].

4.1 Related Work

The relevant literature in the area of clock synchronization can be divided into online and offline clock synchronization protocols. The aim of *online clock synchronization* protocols, like the well-known Network Time Protocol (NTP) [Mil94a, Mil92], is to keep the clocks of the participating nodes synchronized while the network is up and running. By contrast, *offline clock synchronization* approaches correct timestamps that have been provided by unsynchronized clocks after the experiment is finished. *NTP skew correction* is a special application of NTP and an online approach, *MLE timestamp synchronization* clearly falls into the second category.

4.1.1 Online Clock Synchronization

As discussed above, most online approaches use explicit messages for clock synchronization. They are also constrained by the fact that they need to work in a distributed

fashion and may consume only very limited computational resources. Moreover, online synchronization can only exploit past information, whereas offline approaches can make use of all—previous as well as later occurring—events for the time estimates. For these reasons, online synchronization protocols are not an optimal solution if the goal of the synchronization are log files with a common time base. In the following section, we summarize the approaches that use the idea of events observed by multiple systems. A broader overview of the topic, with a focus on wireless sensor networks, can be found in [RBM05]. Although our NTP skew correction is based on NTP, the basic idea behind it can be adapted for other online protocols as well.

A number of online synchronization protocols [VRC97, MFNT00, EGE02] rely on the parallel reception of broadcasted packets by multiple systems. A broadcasted packet is received by all systems nearly at the same instant, and the only uncertainty in timestamping such packets is the signal propagation time and the timestamping delay. To synchronize the clocks, the recipients of a given broadcast communicate to exchange their respective reception times. By comparing these reception times, two nodes are able to compare and adjust their clocks. In [EGE02], for example, the clock skew is estimated using linear least-squares regression. A complete network can then be synchronized by synchronizing adjacent nodes pairwise along a tree structure, yielding, however, the disadvantage of accumulating the pairwise errors.

In [KEPS04], the pairwise synchronization of [EGE02] is extended to a global one. The authors present an online synchronization approach for sensor networks that is based on a global unbiased minimum variance estimator. They first introduce a version that considers only clock offsets, and then complement it with an idea on how to deal with clock rate differences. Their approach, however, is not able to handle offsets and rate deviations conjointly, but must rely on separate estimates on different time scales. This is feasible and appropriate in the considered context of online time synchronization for continuously running sensor networks, but is not optimal for the offline synchronization of the logs of time-limited experiments. In addition to avoiding the general drawbacks of using online approaches for the synchronization of log files, our MLE approach estimates offsets and rates in one single step, and can thus exploit all the available information to find the global optimum for both.

4.1.2 Offline Clock Synchronization

The first offline clock synchronization algorithm was proposed by Duda et al. [DHHB87] for generic distributed systems. The send and receive timestamps of messages between

nodes A and B are taken as coordinates of a point, the x-axis being the timestamp of A and the y-axis being the timestamp of B for the same packet. Due to the network delay, two point-clouds emerge with an empty corridor in between. Each point is either above the corridor (when sent from A to B) or below (when sent from B to A). The authors present two methods to fit a line in this corridor, thereby estimating the difference in clock speed and offset between A and B. The first method computes the separating line with linear regression; the other uses a convex hull approach. They also sketch a maximum likelihood approach but are not able to use it due to a lack of knowledge about the message delay from sender to receiver.

Duda's linear regression and convex hull approaches have been extended in [Ash95]. The author corrects the timestamps using experimental knowledge about the smallest round trip delays. This knowledge is incorporated in an algorithm that selects the two best points to estimate the skew and offset between the nodes. In [MST99], linear programming is used to compensate for clock skew that influences one-way delay measurements between two nodes over the Internet. A convex hull based approach able to cope with clock resets is presented in [ZLX02].

All of the presented offline synchronization algorithms can compensate linear clock deviations between two nodes without requiring additional network traffic. In contrast to our MLE approach, which exploits the broadcast nature of the medium, they can be used for all kinds of communication systems. However, this benefit is also their main drawback: all of them consider the comparison of send and receive timestamps. Thus, the network delay cannot be completely eliminated, as it is the case in our approach. Likewise, they cannot separate and handle the timestamping delay. Finally, while we use all the available data to compute globally consistent estimates for an arbitrary number of nodes in parallel, all these algorithms synchronize only two clocks directly. In order to synchronize more clocks, a successive synchronization of node pairs is necessary, a process in which errors can accumulate.

4.2 NTP Skew Correction

In the following we model clocks as twice differentiable functions, mapping some (virtual) global, absolute time t to the view of the respective clock. This model matches those commonly used in literature related to clocks and time synchronization [Mil92, MST99, EGE02], and is justified since only the limited timespan of a single experiment needs to

be considered. For the same reason and for the sake of simplicity we do not account for clock resets.

The *true clock* C_T is a clock which is correct by definition: $\forall t : C_T(t) = t$. The *offset* of a clock C at time t is the difference $C(t) - C_T(t)$ between C and the true clock C_T. If we use the term offset without referring to a certain point in time we refer to $C(0)$, the offset at time $t = 0$. $C'(t)$ is called the *rate* or *frequency* of C at time t. The difference between a clock's rate and the true clock's rate $C'(t) - C'_T(t) = C'(t) - 1$ is called *skew* or frequency error. Finally, the second derivative $C''(t)$ is called the *drift* of C. The clocks of devices participating in WMN experiments can have an arbitrary offset, but their skew is limited within production dependent boundaries.

The NTP daemon (*ntpd*) uses a network connection to a node with a high-precision clock to determine offset and skew of the local clock. Approximately one hour after ntpd acquires synchronization, the skew is laid down in the so called *drift file*[1]. Ntpd continues to correct the local clock with this skew estimate even if the device is disconnected from the reference clock. This continued NTP skew correction without network connection makes ntpd suitable for the synchronization of the clocks in a WMN experiment where no such connection is available. As long as the clock drift is small and thus the skew remains relatively constant over the time of an experiment, this should allow the local clock to maintain a good degree of synchronization with the reference clock.

To determine the accuracy achievable with hardware typically used in WMN experiments, we have conducted a series of measurements. On the one hand, the achievable absolute precision is of interest. On the other hand, it is also interesting how the skew and the related synchronization quality changes over time. Note that this skew correction approach is orthogonal to existing online synchronization algorithms as any approach able to determine clock skew with sufficient precision can be used. We have chosen NTP as it has been in productive use for a long time, is available on a large number of platforms and thus allows other researchers to easily adopt our method for their own experiments.

4.2.1 Measurement Setup

Two different kinds of handheld devices have been used for the measurements:

- HP iPAQ 5550 (Nodes 2-11)

[1] Although this file name may be misleading as the file contains the skew and not the drift, we keep this name to stay consistent with existing terminology.

- Sharp Zaurus SL-6000 (Nodes 51-64)

Both of these devices have integrated 802.11b WLAN network interfaces and were running a customized version of OpenEmbedded Linux [opeb] named *zaulux* [zau]. As not all devices were available for all experiments, some had been conducted with a subset of these nodes.

For the measurements, we used a setup similar to the one in [EGE02]: all nodes were set up in a single room, one node broadcasted a packet (beacon) once every second over 802.11b WLAN. The nodes receiving the beacon recorded it together with their local timestamp that has a resolution of 1 ms. Due to the proximity of the devices, the propagation delay is small enough to be ignored, each device thus records an event that has occurred at the same moment in time. By comparing the timestamps for this moment, it is possible to determine the differences of the clocks. For each run, we sent 140 000 beacons, resulting in an experiment duration of approximately 39h. During the experiment, we minimized background traffic in order to avoid disturbances. Note that this single-hop setup is used for *measuring* the clock behavior; the proposed approach, however, also works for multihop networks.

After the experiment, the trace files were gathered and evaluated with our EDAT analysis tool that will be presented in Chapter 5. For each clock, we then plotted the averaged difference to the timestamp of a reference node that is chosen from the receivers. To compensate for initial differences in the clock setting, the constant offset of the first packet is subtracted from all values[2]. These differences also occur in a real MANET experiment but can be compensated by setting the clock before disconnecting ntpd from the reference clock. The difference of a node's timestamp to the timestamp at the reference node is plotted on the y-axis, the elapsed time is shown on the x-axis. Thus, the y-axis always shows differences *relative* to the reference node. As we are interested in general clock behavior rather than that of individual nodes, the corresponding curve labels will be omitted in these graphs.

Clock Quality

Before evaluating the quality of the NTP skew correction, we determine lower and upper bounds for the synchronization quality. This upper bound is marked by a permanent NTP clock correction while running the clocks freely allows the quality of hardware clocks of our devices to be determined. The behavior of the clocks when these are

[2]The only exception is Figure 4.1 where such differences do not occur due to the setup.

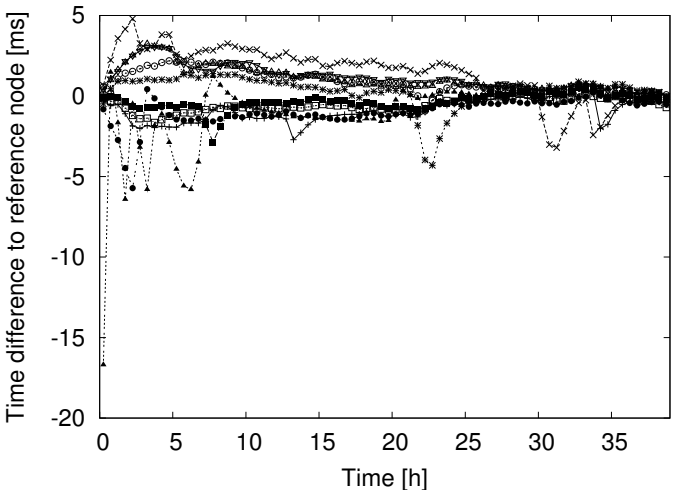


Figure 4.1: Deviation of the clocks when synchronized with NTP.

permanently corrected by NTP and connected over an additional wired network to a reference clock is shown in Figure 4.1 for ten of the Zauri. It can be seen that after an initial phase with larger oscillations, ntpd keeps the clocks well synchronized and adapts to changing clock skew such as the peak that occurs for one of the nodes around the 23rd hour. Without any correction, the clocks' quality is quite bad, as shown in Figure 4.2. Each device's clock soon diverges from the reference node. It is furthermore interesting to note that the clocks of the iPAQs (all lines ending above 6 000) tend to be faster than the clocks of the Zauri (the lines around zero and below). The clock spread in the group is about 77 ms/h for the iPAQs and about 128 ms/h for the Zauri. The overall spread is 307 ms/h. This strongly highlights the need for a simple mechanism that is able to correct the clocks.

Precision of the Drift Files

As long as ntpd has a connection to a reference clock, it updates the factor in the drift file to compensate for changing clock skew. This factor is measured as frequency error in "parts per million" (ppm)[3] and is updated once per hour. An important factor influencing the quality of NTP skew correction is the stability of the drift factor over time. The more stable it is, the more synchronized the nodes will stay even when disconnected from the reference clock. The stability of the drift factor for our devices is

[3] An error of 12 ppm corresponds to about one second per day.

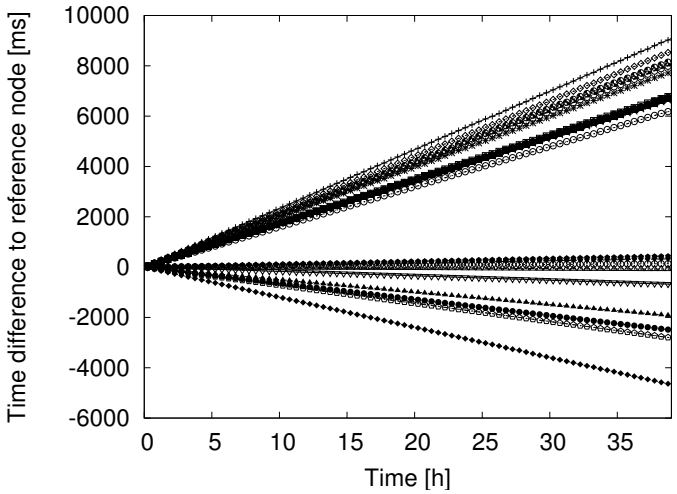


Figure 4.2: Deviation of uncorrected clocks.

shown in Table 4.1. It shows the changes of the initial correction factor for some of the participating devices over 46 hours, measured in ppm. For the majority of the devices, the initial frequency error never changes more than 0.8 ppm, so a longer synchronization time would not improve the correction factor significantly. However, synchronization for several hours is recommended since the error rate tends to stabilize in the first few hours, as also shown in Figure 4.1.

4.2.2 Evaluation

After knowing that the clocks behave as expected, we evaluated the performance of ntpd with the proposed NTP skew correction. We let ntpd create and refine the drift file for several hours and then shut down the network connection, relying only on the drift file to keep the clocks synchronized. The results of this experiment are shown in Figure 4.3 (note the different y-axis scale of this figure compared to Figure 4.2).

Obviously, the overall time deviation of the clocks can be reduced by this method, in our setup by two orders of magnitude. Nevertheless, the results are surprising as the plot shows some non-linearity. We repeated this setup several times and got similar results. The most likely reason for this is temperature fluctuation, as it has the largest short-term influence on the oscillator stability [Mil94b]. After examining this in more detail, we wondered if this non-linearity also exists for the non-synchronized case but could not be observed due to the different y-axis scales of the plots. We therefore

Node	Start value	Median freq. err.	Standard deviation	Max. difference from start value
60	95.792	95.888	0.031	0.12
5	18.893	18.946	0.116	0.256
51	79.851	80.1725	0.113	0.392
3	14.365	14.1855	0.105	0.412
7	4.965	4.9385	0.168	0.429
11	15.302	15.665	0.101	0.542
9	15.131	15.6005	0.127	0.616
2	7.458	7.0815	0.15	0.677
10	-1.866	-2.217	0.117	0.733
6	2.069	1.3145	0.173	1.056
61	80.94	82.611	0.361	1.785
63	61.367	59.374	1.07	7.105

Table 4.1: Changes of the initial correction factor (in ppm).

use linear regression to approximate the slope of each of the curves in the uncorrected plot in Figure 4.2. This approximated slope leads to a correction function that can be used to remove linear deviations, leaving mainly the non-linear components of this deviation. The result is shown in Figure 4.4. Due to the applied correction that results in a compression of the y-axis, it is now possible to compare this graph to the one in Figure 4.3 at the same scale. It is obvious that similar non-linear features are present in both cases. The discovered non-linearity is inherent to the devices' clocks, and the overall quality of the clocks is not influenced by the proposed skew correction approach.

The above results reveal a certain kind of non-linearity in the clocks and the available literature [Mil94b] suggests that changes in the environment temperature may be responsible for this. We therefore set up an additional experiment in which we also have recorded the temperature to examine its influence on the clocks. During this experiment, we have varied the environment temperature on purpose to provoke a reaction of the clocks. The evaluation method is the same as above: freely running clocks were corrected after the experiment with linear regression and we plotted averaged differences. The result can be seen in Figure 4.5, containing both temperature and the time differences. The influence of the temperature is immediately obvious: as soon as the temperature starts to decrease from 24 degrees to the lowest temperature of 15 degrees, the nodes change their relative clock speed. Different nodes show different reactions to this change in environmental conditions: some oscillators are more sensitive to tem-

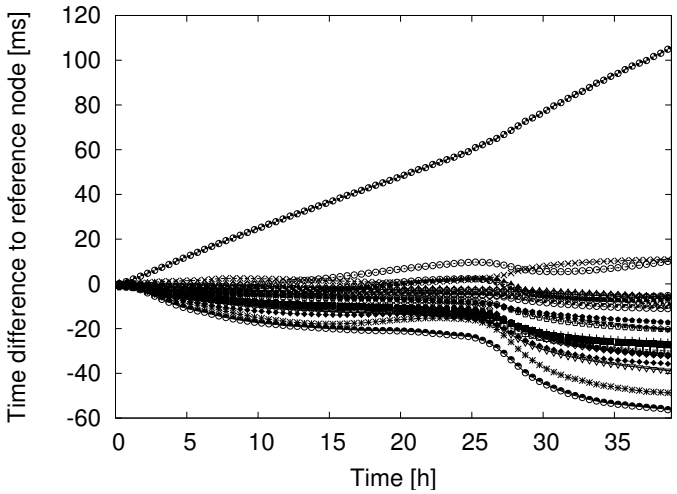


Figure 4.3: Deviation of clocks with the NTP skew correction. It is interesting that all clocks behave similarly except for one clock that is much faster and therefore has a difference to the reference node of more than 100 ms at the end of the experiment.

perature than others. As we chose an arbitrary node as reference, some clocks here slow down compared to this reference node (the lines below zero), while other clocks speed up in relation to this node. This experiment shows that it is advisable to avoid temperature fluctuations, e.g. by not exposing nodes to direct sunlight. Furthermore, it also shows that these clocks behave approximately linear if the temperature is stable.

To sum up, this measurement study reveals that NTP skew correction can reduce the clock frequency error of devices typically used in WMN experiments by two orders of magnitude. As this approach does not require a connection to an external reference time source, it is especially suited for such experiments where this connection is not available most of the time.

4.3 MLE Timestamp Synchronization

These experiments show that the clocks can be kept in sync within a few milliseconds over the duration of a whole experiment, a precision high enough for the *coordination* of actions. However, parameters like end-to-end delay have values in the same order of magnitude, their *analysis* therefore requires higher synchronization precision. For this,

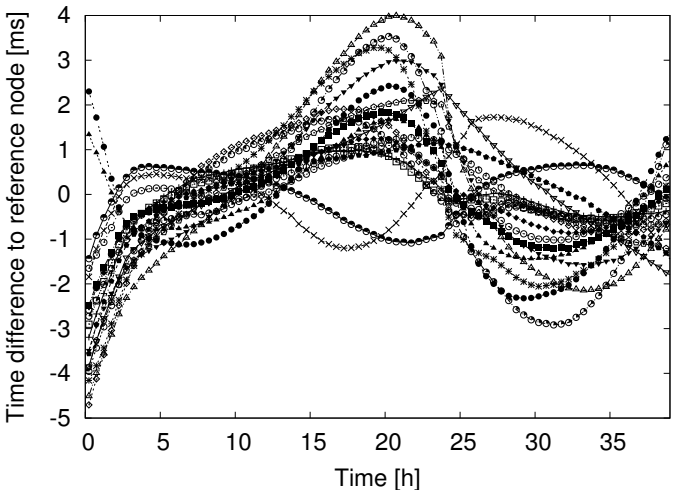


Figure 4.4: The clocks of Figure 4.2, now corrected after the experiment with a linear regression-based method.

we have developed the MLE timestamp synchronization approach. It employs a model for the clocks and the timestamping delays and uses physical events recorded at the same moment by multiple nodes (so-called *anchor points*) to estimate the parameters of this model. These parameters are estimates of the clock deviations and allow to derive estimates for the timestamps of all events on a common time basis. A maximum likelihood estimator is used for this purpose. It leads to a large linear program with a very specific structure. We exploit this structure to solve the linear problem efficiently in spite of its huge size. The solution then yields a synchronized log file where all entries are recorded with a common time basis. Analytical and numerical results show that the solution converges quickly to a good estimate for increasing input data sizes, and that it is robust if the assumptions made for its derivation are not perfectly fulfilled. Thus, in practice, a very reasonable amount of log data is typically sufficient to identify and eliminate clock deviations.

Our approach is applicable to all networks with local broadcast characteristics. It just requires that the clocks of any two nodes in the network can be—directly or indirectly—set into relation by anchor points. In particular, this includes experiments in wireless ad-hoc-, sensor- and mesh-networks for which it was initially designed. However, it is also suited for local area networks with multiple stations in each collision domain and satellite networks.

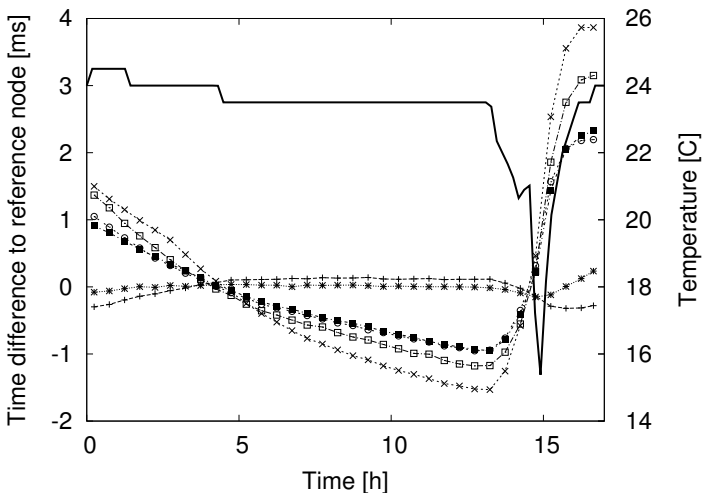


Figure 4.5: Experiment with temperature recording (solid line) and the time differences averaged over one minute.

4.3.1 Model, Terminology, and Applicability

Nodes and Events

In our terminology, an *event* is an incident that has been observed by one or more nodes and is recorded in their local log files. Of particular interest for us are packet reception events, since they can be observed by multiple nodes almost at the same time. We assume that parallel receptions of the same transmission can be identified as such and concentrate primarily on events that have occurred in more than one node, since they can serve as anchor points for the synchronization.

We denote the set of nodes participating in an experiment by J and the set of events that occur during the experiment by I. Each event $i \in I$ occurs at some "true" time T_i. The same event i can be observed by multiple nodes. In this case each of the nodes records its own timestamp for the event according to its local clock, i. e., event i is recorded by some of the nodes $j \in J$ with local timestamps $t_{i,j}$.

The recorded times define a relation $R \subseteq I \times J$ in the sense that $(i, j) \in R$ if and only if event i is recorded by node j. The subset of nodes that observe a certain event $i \in I$ is denoted by R_i, i. e., $j \in R_i$ if and only if $(i, j) \in R$.

Clocks

As stated above, the *true clock* C_T is a clock which is correct by definition: $\forall t : C_T(t) = t$. Our aim is to approximate this clock as closely as possible by the calculated global event timestamps.

The clock model we use for our estimator assumes that during the time interval of an experimental run, the local clocks in the nodes can be closely approximated by a linear function. We denote the rate of a node j's local clock C_j by $r_j > 0$ and its offset at time $t = 0$ by o_j. Thus, for all times t during the run we have

$$C_j(t) = r_j t + o_j. \tag{4.1}$$

The time span over which the linearity assumption holds is related to the clock's frequency stability. It is commonly specified using the Allan deviation [All87], which characterizes the rate variations over different timescales. One application area of offline synchronization are the experiments with wireless multihop networks this thesis concentrates on. As pointed out in Section 2.1.5, the duration of an experimental run typically does not exceed 1 000 seconds. This coincides with previous work, which shows that clock drift is typically negligible over time spans up to 1 000 seconds [VBP04]. Furthermore, also the experiments in Section 4.2.2 show that clocks are approximately linear for shorter intervals and stable temperature.

If the linearity assumptions do not hold, the accuracy of the results may deteriorate. As will become clear later, the degradation is graceful, i. e., the estimation is a good linear approximation. Note also that it is easily possible to synchronize arbitrary subintervals of longer logs, such that the assumption holds reasonably well within each sub-interval.

Furthermore, care must be taken that the linearity assumption is not thwarted by processes running on the nodes and manipulating the clocks. If online synchronization must be used—e. g., because it is part of the experiment—then it should record all non-linear modifications it made to the local clock, so that the effects of these changes can be eliminated from the log files of the respective nodes prior to synchronizing them. As NTP skew correction only applies linear corrections, this is not problematic here.

In practice, time in computer systems does not run continuously, but progresses in discrete steps. While the resolution of the timer-interrupt driven system clock is typically

relatively coarse—in the order of milliseconds—, more fine-grained time sources are often available and used. On the x86 platform, for example, the CPU's TSC register progresses with every CPU clock cycle. Thus, its granularity is very fine. It serves for generating the timestamps, for example, when using a Linux kernel and the widespread packet tracing library libpcap [LIB]. Thus, we can assume that the error introduced by the clock resolution is small in comparison to other sources of error. Our approach does not amplify such errors.

Timestamping Delay

When sending a message, a number of different delays occur from the moment the source application generates the message until the receiver timestamps it. As our approach uses these timestamps as synchronization anchors, we are interested in the delay *differences* experienced by distinct nodes. The deterministic components are not an issue in our context: if all timestamps in a node are recorded late by some fixed time, then this is the same as if they were recorded immediately with a correspondingly increased offset. So, the fixed delay components are equivalent to an additional clock offset.

The experienced delay can be decomposed into four components according to [KO87]: the time needed to compose the message and to assemble the packet; the time to access the medium; the propagation delay on the medium; and finally, after the transmission arrives at the receiver, the *receive time*, i. e., the delay for checking the message and recording the arrival timestamp. Obviously, the time until the packet leaves the sender is the same for all receivers and thus does not need to be considered.

The propagation delay depends on the distance between sender and receiver, and the propagation delay differences depend on the different distances between sender and receivers. As long as these differences are in the order of a few hundred meters, the propagation delay difference is in the order of, at most, microseconds and is therefore negligible[4]. The receive time is a result of the delayed recording of the timestamps in the nodes that does not happen immediately upon reception. It can be decomposed further into a fixed component (which equals the minimum path delay of the processing necessary at the receiver), and an additional, variable time that occurs because the timestamping is performed by the node's CPU, which may be busy with other tasks before the event is processed. The latter we call *timestamping delay*.

[4] If nodes are really far apart and the propagation delay is long, then it is often the case that the distances and thus also the delays are approximately known. This applies to satellite systems, for example. In this case it is possible to eliminate the delay prior to synchronization.

Note that the delay of an event is also "measured" by the recording node's clock, and thus is scaled with the rate of this clock. An event i at "true" time T_i that is recorded by node j with timestamping delay $d_{i,j}$ thus leads to a timestamp

$$t_{i,j} = C_j(T_i + d_{i,j}) = r_j(T_i + d_{i,j}) + o_j. \tag{4.2}$$

The timestamping delay is, like all delays, obviously nonnegative. Furthermore, it seems reasonable to assume that most timestamps are recorded with small latency and few are set after a longer time. We model the timestamping delays as exponentially distributed, pairwise independent random variables. Moreover, we assume that the exponential distributions of all delays share the same parameter λ. The latter is reasonable if the nodes participating in the experiment use comparable hard- and software for the timestamp generation, which will often be the case in a testbed.

In a real-world application, our assumptions about the timestamping delay just like those about the clocks' linearity will, of course, not perfectly hold. In fact, depending on the hard- and software of the devices, reality might look very different. We use the mentioned assumptions for the motivation and derivation of our method. It will later become clear that the resulting approach yields good results also under non-conformant circumstances.

Connectivity Constraints

As our proposed approach relies on anchor points to relate the clocks, it depends on the presence of events that can serve this role. Consider the case in which there is no common event between two groups of nodes. Here, it would be impossible for anchor point-based synchronization to tell if all clocks in one of the groups are, for example, early by one hour. A common, global time basis can thus not be established, whereas it remains nevertheless possible to synchronize the clocks *within* each group.

Note that the availability of anchor points does *not* imply that all pairs of nodes must share common events—clocks may also be related indirectly, over intermediate nodes. It also does not necessitate the network to be "connected" in the commonly used sense. For example, if there are two almost independent groups of nodes, one single node sharing events with both groups suffices. These shared events need not occur during the same time intervals, and thus there is no need for a fully connected topology at even just one single point in time.

Hence, the existence of anchor points is not a severe constraint in practice, and anchor point-based synchronization will be possible in the vast majority of experimental setups. If this condition is not met, artificial anchor points could be generated, e. g., by broadcasting "anchor packets". Doing so during the experiment might interfere with the experiment itself, just like running an online synchronization protocol. Anchor points, however, may also be generated before and after an experimental run.

4.3.2 Algorithm

The previous section introduced a model of the network and the timestamping delays. Now, we will formalize the problem and propose an approach for its solution via a maximum likelihood estimator (MLE). Given the recorded local timestamps, we wish to maximize the likelihood that our estimates of the true event times are correct.

Due to the exponentially distributed delays, the conditional probability density for measuring a timestamp $t_{i,j}$ for event i at node j given C_j and T_i is

$$f(t_{i,j} \mid C_j, T_i) = f(d_{i,j}) = \lambda e^{-\lambda d_{i,j}}. \tag{4.3}$$

Because of the independence of the delays the probability density for the whole set of measurements in our experiment can be written as

$$f((t_{i,j})_{(i,j)\in R} \mid (C_j)_{j\in J}, (T_i)_{i\in I}) = \prod_{(i,j)\in R} \lambda e^{-\lambda d_{i,j}}. \tag{4.4}$$

We can now express our problem as an optimization problem. Under a uniform prior, we want to find the optimal estimates $\widehat{T}_i$ of T_i for all $i \in I$, and, in parallel, the optimal estimates $\widehat{C}_j$ of C_j for all $j \in J$ such that the likelihood function L defined in the following way is maximized:

$$\begin{aligned} L &= L((\widehat{C}_j)_{j\in J}, (\widehat{T}_i)_{i\in I} \mid (t_{i,j})_{(i,j)\in R}) \\ &= f((t_{i,j})_{(i,j)\in R} \mid (\widehat{C}_j)_{j\in J}, (\widehat{T}_i)_{i\in I}). \end{aligned} \tag{4.5}$$

From (4.2) we can see that

$$\forall (i,j) \in R : d_{i,j} = \frac{t_{i,j} - o_j}{r_j} - T_i. \tag{4.6}$$

This relation must also hold for the estimates of T_i and C_j. Let $\widehat{r}_j$, $\widehat{o}_j$, and $\widehat{d}_{i,j}$ denote the estimates for r_j, o_j, and $d_{i,j}$, respectively. Then, in analogy to the above we have

$$\forall (i,j) \in R : \widehat{d}_{i,j} = \frac{t_{i,j} - \widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i. \tag{4.7}$$

Therefore, L can be expressed as

$$\begin{aligned} L &= \prod_{(i,j)\in R} \lambda e^{-\lambda \widehat{d}_{i,j}} \\ &= \prod_{(i,j)\in R} \lambda e^{-\lambda\left(\frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i\right)}, \end{aligned} \tag{4.8}$$

eliminating the estimates $\widehat{d}_{i,j}$ for the unknown quantities $d_{i,j}$.

Since all the delays are non-negative, the maximization of L is subject to the constraints

$$\forall (i,j) \in R : \frac{t_{i,j} - \widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i \geq 0. \tag{4.9}$$

Now we apply a standard technique in maximum likelihood estimation: maximizing L is equivalent to maximizing $\ln L$, because $L > 0$ for all valid parameterizations.

$$\begin{aligned} \ln L &= \ln \prod_{(i,j)\in R} \lambda e^{-\lambda\left(\frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i\right)} \\ &= \sum_{(i,j)\in R} \left(\ln \lambda + \ln e^{-\lambda\left(\frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i\right)} \right) \\ &= |R| \ln \lambda - \sum_{(i,j)\in R} \lambda \left(\frac{t_{i,j} - \widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i \right). \end{aligned} \tag{4.10}$$

Optimizing this expression with regard to λ and all the $\widehat{T}_i$ and $\widehat{C}_j$ is a difficult nonlinear optimization problem. However, we are not primarily interested in the parameter λ. Fortunately it turns out that the optimal $\widehat{T}_i$ and $\widehat{C}_j$ are independent of the value of λ. Let for the moment

$$k(x) := -\frac{\ln x - |R| \ln \lambda}{\lambda}. \tag{4.11}$$

k is strictly monotonically decreasing for any $\lambda > 0$ and $|R|$. Thus, it is easy to see that L is maximal if and only if $k(L)$ is minimal:

$$k(L) = -\frac{\ln L - |R| \ln \lambda}{\lambda} = \sum_{(i,j)\in R} \left(\frac{t_{i,j} - \widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i \right). \tag{4.12}$$

Therefore, instead of maximizing L, we minimize $k(L)$. We have thus eliminated the variable $\lambda > 0$. The constraints of the resulting optimization problem are still of the form (4.9).

From the clock model we know that the rates of the clocks are strictly positive. We exploit this fact and define

$$\bar{r}_j := \widehat{r}_j^{-1} \tag{4.13}$$

$$\bar{o}_j := \frac{\widehat{o}_j}{\widehat{r}_j}. \tag{4.14}$$

Equivalently, we have $\widehat{r}_j = \bar{r}_j^{-1}$ and $\widehat{o}_j = \bar{o}_j \widehat{r}_j = \frac{\bar{o}_j}{\bar{r}_j}$. Expressing $k(L)$ in terms of the variables $\bar{o}_j$ and $\bar{r}_j$ leads to

$$k(L) = \sum_{(i,j)\in R} \left(t_{i,j} \bar{r}_j - \bar{o}_j - \widehat{T}_i \right). \tag{4.15}$$

Similarly, the constraints (4.9) can be simplified to

$$\forall (i,j) \in R : t_{i,j} \bar{r}_j - \bar{o}_j - \widehat{T}_i \geq 0. \tag{4.16}$$

This is a linear objective function with linear constraints, which can be solved using standard linear program (LP) solvers like the simplex method.

For exponentially distributed errors, the maximum likelihood estimator is known to be nearly optimal. In our case, however, a different interpretation of the resulting approach is also possible. When comparing (4.12) and (4.7), we observe that

$$k(L) = \sum_{(i,j)\in R} \widehat{d}_{i,j}. \tag{4.17}$$

The optimal solution minimizes the sum of the estimated delays. Therefore, the resulting approach may also be understood as a form of constrained Least Absolute Deviation

(LAD) regression. Since this interpretation is completely independent from the assumption of exponentially distributed delays, it supports the expectation that the derived estimator is also well-suited for delays with other distributions.

Note that the optimization problem (4.15) and (4.16) has the trivial solution $\forall j \in J : \bar{o}_j = \bar{r}_j = 0$ and $\forall i \in I : \widehat{T}_i = 0$. This is because it is only possible to estimate the relative deviation between clocks from the information contained in the log files. We call this the *rate ambiguity*. To overcome the rate ambiguity, we add a normalizing constraint $\sum_{j \in J} \bar{r}_j = |J|$; in the average, the inverse clock rates are assumed to be accurate. This assumption, however, is not crucial at all: if the average takes some other value, the solutions are simply scaled accordingly.

Similar to the rate ambiguity, there is also an *offset ambiguity* in the log files. The right hand sides of (4.15) and (4.16) do not change when all $\bar{o}_j$ are replaced with $\bar{o}_j + \tau$ and all $\widehat{T}_i$ are replaced with $\widehat{T}_i - \tau$, where $\tau \in \mathbb{R}$ is a given constant term. Thus, like above for the rates, it is not possible to estimate absolute, but only relative event times and clock offsets (even ignoring the fact that there is, of course, no "absolute time"). We may set, without loss of generality, $\bar{o}_1 = 0$.

If a reference clock is available—e. g., because at least one node has a connection to an external time source like a GPS receiver and records appropriate data—absolute synchronization to this reference is possible. More specifically, if the correct, global time of one event occurrence in one single node is known, then the offset ambiguity can be overcome. If the global times of two events, or, alternatively, the time of one event and the rate of one node are known, then the rate ambiguity can likewise be eliminated. This is possible either by adapting the constraints for rates and offset, or by a respective transformation of the synchronization result.

The resulting linear program can be written in the form

$$\text{minimize} \quad b^{\mathsf{T}} y \quad \text{subject to} \quad A^{\mathsf{T}} y \leq c, \tag{4.18}$$

where y is the vector of the unknowns $\widehat{T}_i$ for $i \in I$, followed by the vectors $\bar{o} \in \mathbb{R}^{|J|}$ and $\bar{r} \in \mathbb{R}^{|J|}$ of the $\bar{o}_j$ and $\bar{r}_j$ for $j \in J$, i. e.,

$$y = \begin{pmatrix} \widehat{T} \\ \bar{o} \\ \bar{r} \end{pmatrix}. \tag{4.19}$$

The matrix A^T represents the inequality constraints (4.16) and the normalizing constraints.

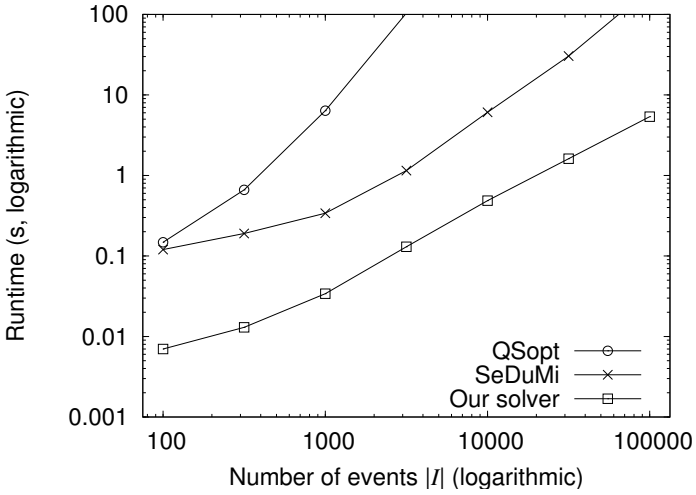
Events that have only been observed by one single node do not contribute information for the synchronization. Therefore, to keep the size of the linear program as small as possible, they should not be included in the optimization. Corrected timestamps for such events can easily be generated based on the rate and offset estimates.

4.3.3 Solving the Optimization Problem

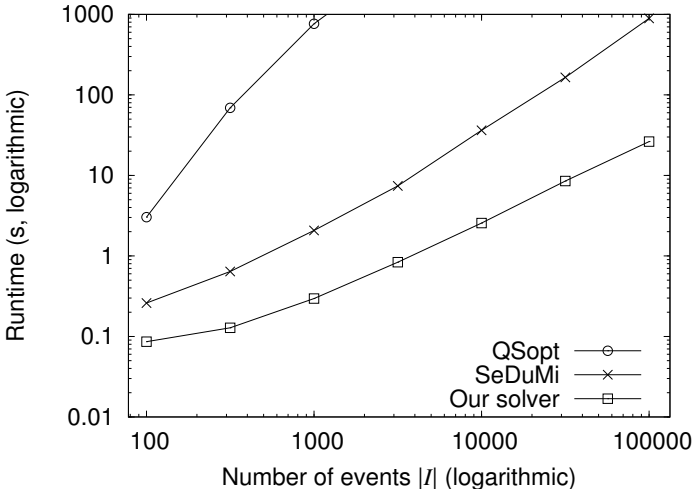
In (4.18), (4.19) the maximum likelihood estimator is defined as the solution of a linear program with $|I| + 2 \cdot |J|$ variables and $|R|$ linear inequality constraints. Due to the size of the linear program a straightforward application of the simplex method may result in a significant effort in terms of computational power and memory. When solving (4.18) with a standard simplex solver like QSopt [ACDM] the program takes hours to terminate even for relatively small problems. Therefore, we will now focus on the special structure of the linear program (4.18) and how it can be exploited to allow for a fast numerical solution. Below we outline the ideas behind our implementation of the synchronization approach. It is able to solve the linear program for data sets with $|J| \approx 100$, $|I| \approx 10^5$, and $|R| \approx 10^6$ on a standard PC within a few seconds.

Each row of A^T corresponding to a constraint (4.16) has exactly three non-zero entries and A is thus very sparse. The matrix A^T is closely related to the matrices arising in network optimization problems. In particular, it does not have full column rank. In the previous section, offset ambiguity was introduced. Since we set $\bar{o}_1$ to zero, the corresponding column of A can be eliminated prior to the optimization. Our implementation checks for further redundancies that depend on the particular instance R and eliminates additional linearly dependent columns of A^T if existent.

We use a modern interior-point algorithm for our solver, a variant of Mehrotra's predictor corrector algorithm [Meh92] that is particularly well suited to handle the structure of (4.18). The primary advantage of interior-point algorithms versus the simplex method is that interior-point methods do not suffer from degeneracy of the problem. Practical implementations very rarely take more than 70 to 100 iterations to solve a linear program. In our case, the particular structure of (4.18) can be exploited, making a single iteration very cheap. The concept of the algorithm as implemented here is based on Algorithm 14.3 in [NW99].



(a) $|J| = 20$.



(b) $|J| = 100$.

Figure 4.6: Performance comparison of QSopt, SeDuMi, and our own implementation as time synchronization LP solvers.

Apart from minor adjustments of the parameters proposed in [NW99], the main modification in our implementation concerns the storage format for the matrix A. Storing A directly would be extremely inefficient in terms of memory requirements as well as from a computational perspective. Our implementation comprises a specialized storage format for A, tailored to both the problem structure and the specific operations that appear in the interior-point algorithm. For matrices A arising from (4.18) this is a superior alternative to general purpose sparse matrix formats, as they are readily provided, e. g., by Matlab.

The main computational effort at each iteration of an interior-point algorithm is the computation and the Cholesky factorization of the matrix product $H = ADA^\mathsf{T}$. Here, D is a positive definite diagonal matrix that changes at each iteration. Due to our choice of setting up the variable y by first including $\widehat{T}$ and then $\bar{o}, \bar{r}$, the leading $|I| \times |I|$-block of H is a positive definite diagonal matrix; only the trailing $2 \cdot |J|$ rows and columns of H do have fill-in. This sparsity structure is also inherited by the Cholesky factor L of H. The leading $|I| \times |I|$-block of L can thus be computed in linear time. Given that typically $|I| \gg 2 \cdot |J|$, the computation of L and thus the solution of the overall problem is very cheap.

To demonstrate the huge gain in performance that is possible by using the tailored solver, we compare the runtime of our implementation with that of QSopt [ACDM] and SeDuMi [SRP]. QSopt is, as mentioned before, a solver that uses the simplex method. SeDuMi on the other hand is a Matlab interior-point code that, like our own solver, benefits from the special structure of H, but uses a more general—and therefore somewhat slower—storage format of the sparse matrix A^T, and a more general sparse Cholesky factorization.

Figure 4.6 shows the computation times for calculating the solutions of optimizations with 20 nodes and with 100 nodes, for different numbers of shared events. All measurements have been made on an AMD Athlon X2 BE-2300 CPU with 1900 MHz and 1 GB of main memory. From the figure it can be seen that our implementation actually works very well. The tailored solver brings a large performance gain—it reduces the computation time by typically at least a factor of 10–15.

Note that SeDuMi expects readily preprocessed input data in Matlab's sparse matrix format. The time needed for converting the data to this format is not included in the SeDuMi results in Figure 4.6. Especially for the larger problems, this can be substantially higher than the time needed for solving the problem. The processing times shown for our own solver do include the time for reading the input data and preparing the

optimization problem. For our specialized matrix format this step can be performed very efficiently; it accounts only for a negligible fraction of the total processing time.

In particular the results with QSopt underline that an off-the-shelf simplex solver is in fact highly unsuitable for the specific type of linear optimization problem that we deal with. Not only does the computation time grow rapidly with an increasing problem size, but also do the memory requirements. By contrast, our tailored implementation is very memory efficient, and its observed runtime increases approximately linearly with the number of events $|I|$.

4.3.4 Properties of the MLE

Now that we have seen that it is in fact possible to calculate a solution of the linear program and thus the maximum likelihood estimator within reasonable time, we are interested in the quality of this solution. In this section, we will thus tackle the question of how good the synchronization result is.

The available amount of data to estimate the clock deviations increases with an increasing number of network packets that have been received by multiple network nodes. Thus, intuitively, one could expect that the quality of the estimate improves with the availability of more experimental data. Similarly, it sounds reasonable that it is very unlikely that the result of the time synchronization process is grossly wrong if the input data is very accurate. In this section, we confine ourselves to a simplified variant of our estimator. For this simplified variant we can prove that these intuitive expectations are actually true. Since the complete proofs of these properties are quite complex and technical despite this simplification, we have not included them here. Instead they can be found in [SKR+08a]. Below we will discuss the results and their implications, and we give a rough sketch of the proofs' ideas. Our numerical results presented later underline that the results also hold for the fully featured estimator with clock rate estimates.

The simplified estimator does not take clock rate deviations into account, i. e., it assumes that for each node the clock rate r_j is (approximately) 1 and thus correct with respect to the "true clock". Under this assumption the recorded time for a node-event pair $(i, j) \in R$ becomes $T_i + d_{i,j} + o_j$. Thus, the simplified maximum likelihood estimator,

in analogy to the fully featured version, is the solution to the following problem:

$$\begin{aligned} &\text{maximize } L = \prod_{(i,j)\in R} \lambda e^{-\lambda\left(t_{i,j}-\widehat{o}_j-\widehat{T}_i\right)} \\ &\text{subject to } \forall (i,j) \in R : \widehat{d}_{i,j} = t_{i,j} - \widehat{o}_j - \widehat{T}_i \geq 0. \end{aligned} \tag{4.20}$$

The optimum is again independent of λ and the above is equivalent to

$$\text{minimize } k(L) = \sum_{(i,j)\in R} \left(t_{i,j} - \widehat{o}_j - \widehat{T}_i\right) = \sum_{(i,j)\in R} \widehat{d}_{i,j} \tag{4.21}$$

under the same constraints.

Here we will point out two desirable properties for this version of the estimator. First, we give tight error bounds on the estimation error that hold under the assumption of a bounded timestamping delay. In particular this means that the algorithm does not amplify errors. Furthermore, we show that the estimator is consistent: in other words, for increasing data set sizes the estimate converges (in probability) to the true values of the estimated features. It thus supports the intuition that the estimate improves for a larger amount of observed and logged events in the nodes.

Error bounds

In order to be able to give a bound for the estimation error, we need to make two additional assumptions. While the first guarantees network connectivity, the second establishes an upper bound on the timestamping delay. Note that an upper bound for the timestamping delay does not constrain the practical applicability of the results presented here: for any practical experiment there is a finite number of receptions, and thus also a maximum timestamping delay.

The existence of the offset ambiguity as introduced in Section 4.3.2 prohibits that the absolute event times and clock offsets are determined from the log files. This also holds for the simplified estimator considered here. From the offset ambiguity, it is easy to see that there is also no way to estimate the relative time between two separate partitions within the same experiment. If there are no anchor points between two sets of nodes, there will be an ambiguity of the offset between these partitions. Thus, in order to get a bounded maximum estimation error, we need to assume network connectivity in the sense of anchor points: the network nodes do not fall into disjoint partitions, between which no events are shared.

Under such an assumption we can prove that

$$\forall j_1, j_2 \in J : |(o_{j_1} - o_{j_2}) - (\widehat{o}_{j_1} - \widehat{o}_{j_2})| \leq (|J| - 1) \cdot D \tag{4.22}$$

if $D \in \mathbb{R}^+$ is an upper bound for the delays, i. e.,

$$\forall (i, j) \in R : d_{i,j} \leq D. \tag{4.23}$$

Note that the bound is on the difference between two estimation errors because of the offset ambiguity.

The basic idea of the proof is the following. Consider two nodes j_1 and j_2. Then it can be shown that there always exists a sequence of unique nodes $s_1, ..., s_n$, $2 \leq n \leq |J|$, $s_1 = j_1$, $s_n = j_2$, with a special property. In this sequence, for each pair of subsequent nodes s_q and s_{q+1}, there is an event observed by both s_q and s_{q+1}, for which the estimated timestamping delay in s_q is zero. This, together with the nonnegativity of the timestamping delays, allows for the construction of an upper bound for $(o_{j_1} - o_{j_2}) - (\widehat{o}_{j_1} - \widehat{o}_{j_2})$. Since j_1 and j_2 can be chosen arbitrarily, the same bound also holds with j_1 and j_2 interchanged. This yields a correspondingly lower bound for j_1 and j_2 and thus constrains the absolute value as shown above.

We are also able to show that under the mentioned assumptions the bound is the best possible, i. e., that no estimator can exist that achieves a smaller worst-case error. This proof is based on two explicitly constructed worst-case scenarios that result in identical log files. The point is that although the resulting local log files are identical for the two scenarios, the clock offsets differ so much that no estimate can be better than the worst-case bound given above in both cases.

From the bound on the clock offset estimation error it is then quite easy to come to a similar bound on the event time estimates:

$$\forall i_1, i_2 \in I : |(T_{i_1} - T_{i_2}) - (\widehat{T}_{i_1} - \widehat{T}_{i_2})| \leq |J| \cdot D. \tag{4.24}$$

No part of the proof exploits the exponential distribution of the delays. Thus, independent of the derivation of the estimator, it shows that if there is an upper bound for the timestamping delays, the estimates are close to the real values, regardless of the distribution of the delays within $[0, D]$.

Consistency

Differing from the results presented so far, we will now no longer assume an upper bound on the timestamping delays. Instead, we exploit their assumed exponential distribution. Under these premises, consistency of the simplified estimator can be established, which means convergence in probability to the correct offset values for an increasing number of observed events:

$$\forall j \in J : \operatorname*{plim}_{|I|\to\infty} \widehat{o}_j = o_j + x, \tag{4.25}$$

where $x \in \mathbb{R}$ again comes from the offset ambiguity.

Similar to the previous results, it is clear that such a result cannot hold if the network is not connected. We show the consistency of the simplified MLE under an additional regularity condition, defined as follows. We say that this regularity condition is fulfilled if there exists an undirected, connected graph $G = (J, V)$ and some positive constant β such that

$$\forall \{j_1, j_2\} \in V : E\left[\left|\{i \in I | \{j_1, j_2\} \subseteq R_i\}\right|\right] \geq \beta \cdot |I|. \tag{4.26}$$

This precondition can be seen as a somewhat stronger variant of the connectivity assumption used above. It is stronger in the sense that it requires an (in expectancy) ever-growing number of independent connections between all parts of the network with an increasing total number of observed events.

In order to prove the consistency result we show that the probability of the likelihood function having its optimum in an arbitrarily small environment around the correct clock offset estimates is arbitrarily high for a sufficing number of observed events. The key idea is to introduce a per-event decomposition of the objective function $k(L)$. Certain properties of these event-wise objective function terms form the basis of our proof. We have seen before that $k(L)$ is simply the sum of the $\widehat{d}_{i,j}$ for all (i, j) in R. Then a decomposition of $k(L)$ into event-wise components f_i is trivial:

$$f_i := \sum_{j \in R_i} \widehat{d}_{i,j} \qquad\qquad k(L) = \sum_{i \in I} f_i. \tag{4.27}$$

We then switch our point of view. We regard the f_i no longer as functions of the estimated latencies, but as functions of the estimation error. It is then quite straightforward to show that all the f_i are convex and that they are all Lipschitz continuous with a common Lipschitz constant. Furthermore, we show that the expectancy for each f_i—as a function of the estimation error—has a global minimum for the correct estimate, and we

give a non-negative lower bound for the difference between this expectancy in case of a non-zero estimation error and the minimum value. All these results in conjunction with the law of large numbers can then be used to establish the consistency of the estimator: for a given $\delta > 0$ there is a number of events N such that for $|I| > N$ the probability that the estimation error is greater than δ becomes arbitrarily small.

From the consistency result for the clock offset estimate, it is easy to obtain a result on the quality of the event time estimates in the same asymptotic setting. If the estimation error of the clock offsets is close to zero (neglecting the offset ambiguity), the remaining event time error for an event i is $\min_{j \in R_i} d_{i,j}$. This minimum of the independent, exponentially distributed $d_{i,j}$ is itself exponentially distributed with parameter $|R_i| \cdot \lambda$. In particular this means that the expected estimation error decreases with the number of nodes observing the same event.

4.3.5 Numerical Evaluation

While the previous section assessed the performance of the proposed time synchronization method analytically, we will now focus on numerical experiments with the algorithm. In particular, we will show that the asymptotic properties that have been proven for the simplified estimator hold also for the fully featured version with clock rate estimates. Moreover, it will become clear that the convergence is quick enough to yield accurate estimates even for small event counts. Finally, we will show that the algorithm is robust if the assumptions—in particular the exponential distribution of the timestamping delays and the negligibility of clock drift—do not hold.

Methodology

Although desirable, using log files from a real testbed for an evaluation that rigorously quantifies the numerical quality of the calculations and the convergence speed is not possible: for real hardware, the correct values for the rates, offsets, and event times cannot be determined. This is why we need post-experiment timestamp synchronization in the first place. Therefore, we use a two-step simulation in which the correct values are known. In the first step, the network is simulated to obtain globally consistent event times and a receiver relation R. Then, subsequently, we simulate the timestamping of the events in each node. Random clock rates, offsets, and timestamping delays are used to transform the correct timestamps, yielding a set of per-node log files. Like after a real experiment, our algorithm is then given these log files as input. The quality of the

solution can be determined by comparing the results to the correct times, rates, and offsets.

How a simulation should be set up for credible network protocol evaluation results is a highly controversial and heavily discussed topic. Since our focus here, however, is on supplying the numerical algorithm evaluation with an event set I, event times T, and a receiver relation R, and not on a realistic performance evaluation of some protocol, we constrain ourselves to a basic simulation scenario. We use the network simulator ns-2 [NS2] in version 2.30, which has been extended to support promiscuous mode-like packet tracing: if a data packet could be successfully decoded by a node's simulated wireless interface, the packet is timestamped and logged, regardless of whether the node was the intended destination or just able to overhear the transmission.

In our simulations, $|J| = 100$ nodes move on an area of 1200×1200 meters according to the random waypoint mobility model. AODV [PR99] is used as a multihop routing protocol. Five pairs of nodes communicate continuously over a simulation time of 10 minutes, performing FTP data transfers over TCP connections. The IEEE 802.11 MAC protocol is used at a fixed network bandwidth of 1 MBit/s. The radio range is set to 250 meters, the carrier sense radius to 550 meters.

For the generation of the local node log files, the clock offset and rate of every node were chosen randomly. The choice of the offset is not at all critical: our implementation actually exploits the offset ambiguity to achieve improved numerical stability and, as its first step, shifts all processed log files to start at time zero. Consequently, whichever offset is chosen for a node, the performed calculations and thus the accuracy of the estimates are virtually identical. In our simulations, we sample the offsets from a normal distribution with mean zero and standard deviation five seconds. For the clock rates, we used a gamma distribution[5] with mean one and different standard deviations. The gamma distribution has the advantages of yielding only positive rate values, and being concentrated around the expectancy. The probability density function of a gamma distribution is depicted in Figure 4.7. Unless otherwise stated, the parameters have been chosen to yield a standard deviation of 100 ppm corresponding to the maximum deviation that we have encountered during the measurements for the NTP skew correction,

[5] The gamma distribution is given by the probability density function

$$f(x; k, \theta) = x^{k-1} \frac{e^{-x/\theta}}{\theta^k \cdot \Gamma(k)} \quad \text{for } x > 0,$$

where Γ is the gamma function. The gamma distribution has two parameters, called the *shape parameter* k and the *scale parameter* θ. It has mean $k \cdot \theta$ and variance $k \cdot \theta^2$.

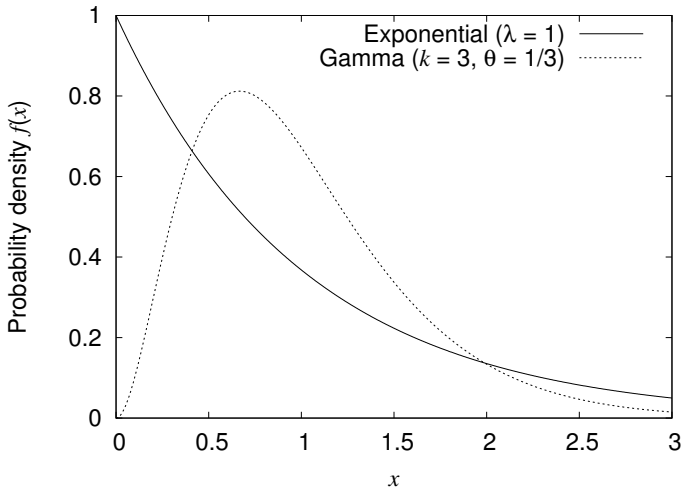


Figure 4.7: Probability density functions of exponential distribution ($\lambda = 1$) and gamma distribution ($k = 3$, $\theta = 1/3$).

see also Table 4.1. This means that on average, a clock is wrong by more than eight seconds per day.

To be able to compare the synchronization results directly with the correct values from the simulation trace file, the rate and offset ambiguities need to be overcome. As stated in Section 4.3.2, the normalization constraints lead to scaled and shifted results if the average inverse clock rate is different from 1, and if the offset of node 1 is different from 0. This scaling and shifting can easily be removed by a linear-affine transformation after the optimization, based on the average inverse clock rate in the simulation and the offset chosen for the first node.

Because ns-2 simulates the radio propagation delay, the arrival times of the same packet at different nodes actually differ slightly. For calculating the event time errors, we compare the estimated event time to the average ns-2 reception time. The differences are in the order of 10^{-7} seconds, and therefore significantly below the other errors that we are dealing with here.

Convergence and Numerical Accuracy

In our first set of experiments, we simulated the timestamping delays according to our assumptions, i. e., exponentially distributed. We varied the expected timestamping delay λ^{-1} between 10^{-3} and 10^{-5} seconds, and increased the number of events used for

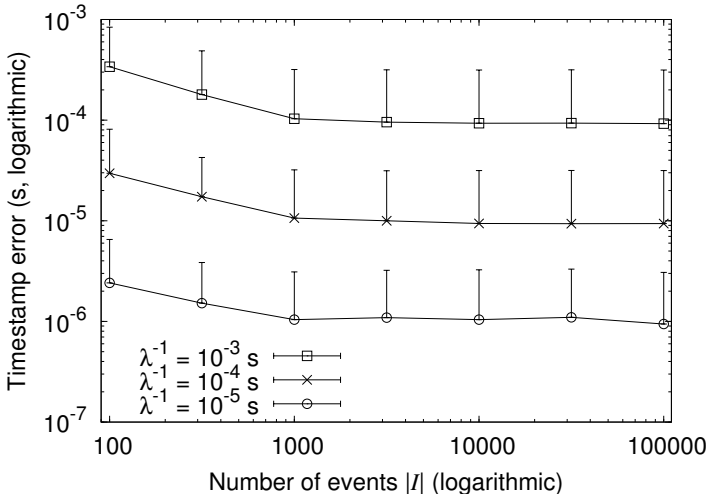


Figure 4.8: Event time errors depending on the number of events $|I|$ and the average timestamping delay λ^{-1}.

the synchronization. One central result in the previous section was that—at least for the simplified estimator—we can expect the quality of the estimates to improve if an increasing number of events is available for synchronization. In a practical implementation, numerical effects of, e. g., a limited floating point precision, may influence the results. The primary purpose of our first simulations is to verify that what we have shown in theory for the simplified case also holds for our practical implementation of the full estimator, and also to give an idea of the convergence speed.

Figure 4.8 shows the resulting average event time error with 95-percentile error bars. For better readability of the chart, only the upper part of the error bars is shown. The x-axis denotes the number of events that have been used for the synchronization. These have been chosen randomly from all transmissions with more than one receiver. Note that at the left hand side of the chart, for 100 events, there is only one sent packet per node on average. The randomly chosen clock errors are quite significant. Still, the synchronization eliminates them to an extent that allows for accurate event time estimates. If more events are available for the synchronization, the estimates improve further quickly, and the average event time errors are one order of magnitude below the timestamping delays. The convergence is so quick that for 1000 available anchor point events and more, only tiny fluctuations are left.

From these as well as from our other results, it can also be seen that the extent and the nature of the timestamping delays constitute the central limiting factor for the achievable

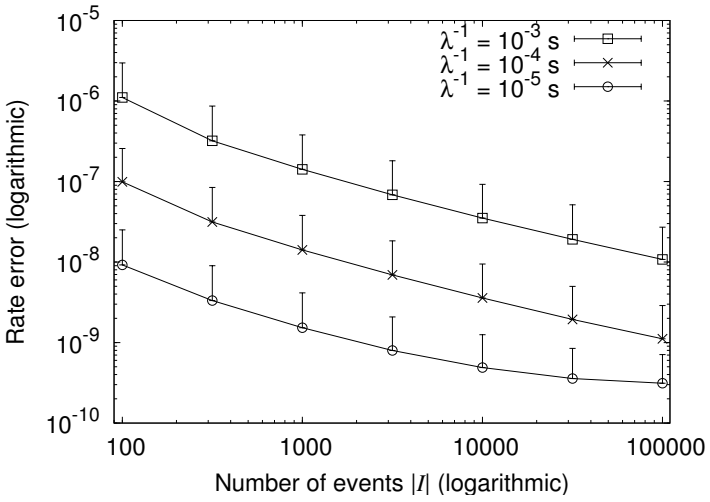


Figure 4.9: Rate errors depending on the number of events $|I|$ and the average timestamping delay λ^{-1}.

std. dev. of rates	avg. rate error	95-perc. max rate error
10 ppm	0.00352 ppm	0.00935 ppm
100 ppm	0.00358 ppm	0.00945 ppm
1000 ppm	0.00355 ppm	0.00927 ppm

Table 4.2: Clock rate estimation error for different clock rate standard deviations ($\lambda^{-1} = 10^{-4}$ s, $|I| = 10\,000$).

event time estimate accuracy. This is a common limitation to any synchronization solution. The error of the rate and offset estimates, however, tends to zero for large $|I|$. This is evident from Figure 4.9, which shows how the rate error develops in the same setting as before. The figure also displays the different levels of accuracy depending on the order of magnitude of the timestamping delays. The corresponding results for the offset estimates show the same behavior. For very small delays with $\lambda^{-1} = 10^{-5}$ seconds and a high number of available anchor points, it can be seen that the accuracy does no longer improve linearly; the implementation then approaches its numerical accuracy limits.

It also turns out that the estimation errors of rate and offset are largely independent from the true values of these parameters and their spread. For the offsets, this is clear from the problem structure. For the rates, however, this trait is not immediately obvious. Tables 4.2 and 4.3 show the accuracy of the estimation results. The small deviations in estimation accuracy are remaining statistical fluctuations.

std. dev. of rates	avg. offset error	95-perc. max offset error
10 ppm	1.56 μs	3.84 μs
100 ppm	1.50 μs	3.81 μs
1000 ppm	1.50 μs	3.96 μs

Table 4.3: Clock offset estimation error for different clock rate standard deviations ($\lambda^{-1} = 10^{-4}$ s, $|I| = 10\,000$).

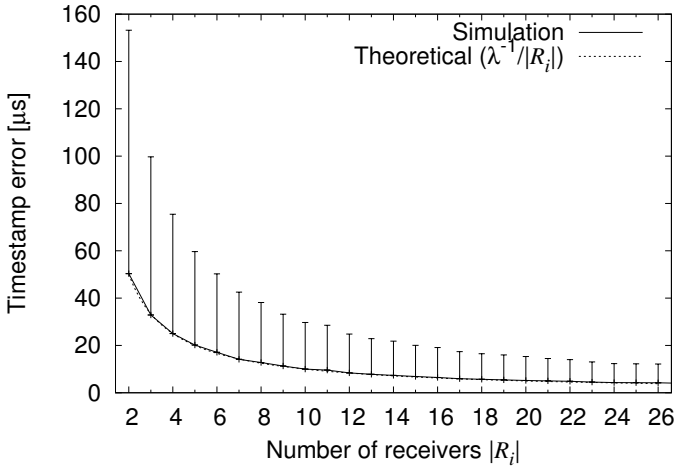


Figure 4.10: Theoretical and simulated event time estimation errors depending on the number of receivers $|R_i|$. ($\lambda^{-1} = 10^{-4}$ s, $|I| = 10\,000$)

Another theoretical result from the previous section is that the event time estimation accuracy for an event i increases with $|R_i|$. More specifically, the result said that for correct rate and offset estimates, the remaining expected event time error is exponentially distributed with parameter $|R_i| \cdot \lambda$, and thus is $\lambda^{-1}/|R_i|$ on average. Figure 4.10 shows simulation results from log files with $\lambda^{-1} = 10^{-4}$ seconds and $|I| = 10\,000$. They exhibit exactly the predicted behavior. The chart shows the average event time error and again the 95-percentile upper error bar, where the events are broken down along the x-axis according to the number of nodes $|R_i|$ that observed them. The chart shows also the theoretical average error given by the function $x \mapsto \lambda^{-1}/x$, and it is evident that the results match the theoretical expectations very closely.

We conclude that the convergence of the estimate is very quick, and a reasonable synchronization quality can be expected even if only a limited number of anchor points is available. The results also underline that the numerical performance of our implementation will not be the limiting factor for the accuracy in practical usage.

Robustness

So far, our simulations have used clocks and timestamping delays that match the assumptions made for the derivation of the approach. Now we assess how robust the estimator is if these assumptions do not hold. We thus use the very same estimator as before, but generate simulation data that intentionally contradicts the assumptions in different ways.

The event time estimation errors occurring in these experiments are shown in Figure 4.11. The results with $\lambda^{-1} = 10^{-4}$ seconds from the previous simulations, where all our assumptions hold, are used as a "baseline" for comparison (labeled "exponential"). Since the error bars in particular are difficult to identify in this figure, Table 4.4 provides a more detailed view on the exact values for $|I| = 10\,000$. As could be expected, the estimation results with exponential delays are slightly better than those where the assumptions do not hold.

First, we varied the distribution from which the timestamping delays were sampled. The result labeled "gamma" shows the estimation error for delays drawn from a gamma distribution with shape parameter $k = 3$. The scale parameter θ has been set to $1/(k \cdot \lambda)$. This yields a mean of λ^{-1} and therefore allows for a direct comparison to the results with exponentially distributed delays with the same mean. The probability density functions of these two distributions are shown in Figure 4.7, both adjusted to mean 1.

In the "multi-modal" simulations we assess the robustness to outliers in the timestamping delays. The majority of timestamping delays follows an exponential distribution with $\lambda^{-1} = 10^{-4}$ seconds. 10 %, however, are instead drawn from a gamma distribution with $k = 10$ and a mean ten times higher. 1 % are "heavy outliers", sampled from a gamma distribution with a mean 50 times higher and $k = 100$. From these results, it can be seen in particular that our proposed method is very robust against outliers.

Heterogeneous hardware with different timestamping delay characteristics is simulated in the "two groups" setup. The simulated network nodes are divided into two groups of 50 nodes each. The delays are exponentially distributed here, but the values of λ^{-1} differ by one order of magnitude: one half of the nodes uses $\lambda^{-1} = 10^{-4.5}$ seconds, and the other half uses $\lambda^{-1} = 10^{-3.5}$ seconds.

We have also simulated the effects of a bad clock accuracy. As stated earlier, clocks in computer systems sometimes have a rather coarse resolution. In the simulations labeled "lim. resol.", the timestamping delays are again exponentially distributed with $\lambda^{-1} = 10^{-4}$ seconds, but the timestamps' resolution has been reduced to 0.1 milliseconds

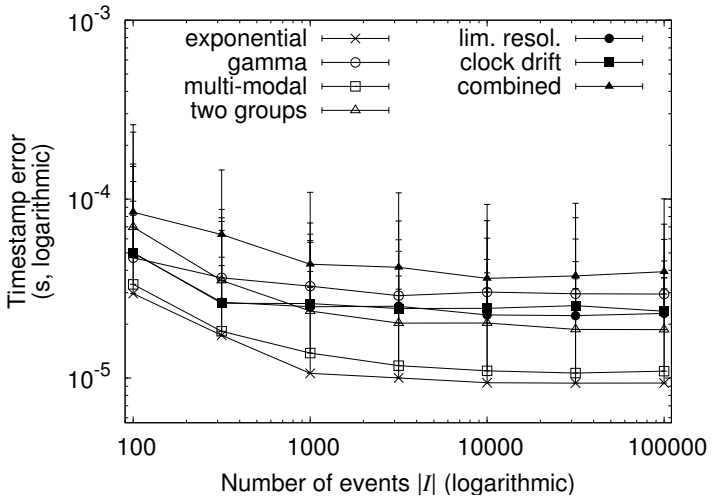


Figure 4.11: Event time estimation errors for increasing I if assumptions do not hold.

distribution	**avg. timestamp err.**	**95-perc. max timestamp err.**
exponential	9.4 μs	31.6 μs
gamma	30.2 μs	60.5 μs
multi-modal	11.0 μs	35.9 μs
two groups	20.3 μs	38.7 μs
lim. resol.	22.5 μs	46.0 μs
clock drift	24.5 μs	75.8 μs
combined	36.0 μs	93.5 μs

Table 4.4: Event time estimation errors for $|I| = 10\,000$ events if assumptions do not hold.

prior to performing the time synchronization. The estimator again deals very well with this effect. It is particularly remarkable that the availability of measurements from multiple nodes with different offsets allows for an estimation of the event times that is more accurate than the resolution of a single node's clock.

The "clock drift" simulations show the effect that randomly drifting clocks have on the accuracy of the estimates. Instead of linear clock functions, we use second order polynomials. Clock drifts are chosen independently from a Gaussian distribution with mean zero and standard deviation $3 \cdot 10^{-9}$. With this, the speed change of a clock can easily sum up to several ppm during a ten-minute simulation. Nevertheless, as our results show, the effect on the synchronization accuracy is very limited.

Finally, the "combined" simulations incorporate all of the above sources of inaccuracies. In this data set, the timestamps are delayed according to the outlier-prone "multimodal" distribution, there are two groups of nodes with different expected timestamping delays like in the "two groups" simulations, the simulated clocks drift as described above, and the timestamps' resolution is again limited to 0.1 ms. Even this combination of effects—all of which heavily contradict the foundations on which we have initially built our method—results in a degradation of the estimation quality by substantially less than one order of magnitude.

In summary, the estimator has proven to be very robust and yields sensible results also if the various assumptions made for its derivation do not exactly hold. Although the quality of the estimates degrades to a certain extent as could be expected, they are still very good, and the estimator converges quickly to a high accuracy in all cases.

4.3.6 Real-World Experiments

The previously presented robustness assessment has shown that our proposed time synchronization method is able to deal well with a whole range of adversarial effects in the log data. Still, however, these evaluations were based on artificially generated simulation data. We will thus now complement them with an application of our method to real-world experimental data. While this does not allow to rigorously determine the remaining errors due to the unknown true values, it nevertheless provides a good intuitive understanding and shows how well the method can handle real data.

Our experimental setup consist of seven PCs with rather heterogeneous hardware both in terms of CPU/memory and the wireless interface card. One of these nodes periodically broadcasts one packet per second, over a total of 20 minutes. The other six

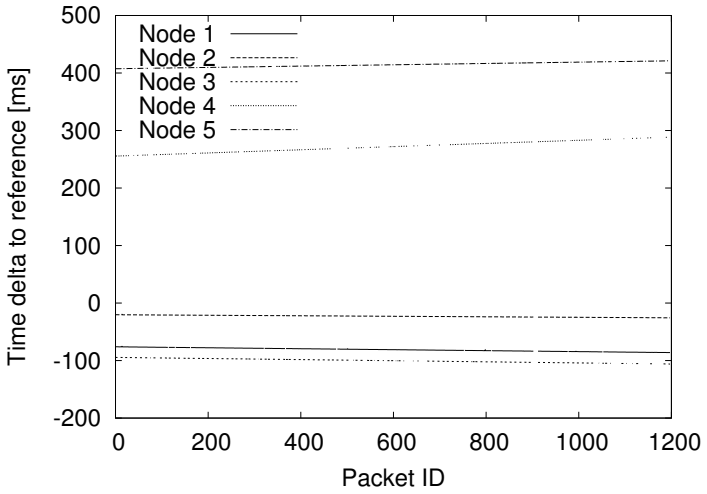


Figure 4.12: Unsynchronized timestamp differences in real-world experiments.

record and timestamp the received packets. Initially, the offsets have been reduced by approximately setting the clocks by hand.

In our figures, we use one of the receivers as a reference, and plot the differences in the recorded timestamps between this receiver and the other five. Figure 4.12 shows how—for unsynchronized clocks—this difference develops during the experiment. The almost exactly linear relative clock errors are clearly visible. Where the lines in the plot are interrupted, the respective nodes have missed packets.

The data from the above experiment has then been used as input for `pcapsync`, an implementation of our approach for real-world log files that will be presented in Section 4.5. This synchronization yields estimates $\widehat{r}_j$ and $\widehat{o}_j$ for the rates and offsets of the six receivers. We use those to eliminate the estimated linear clock deviations from Figure 4.12, by computing

$$\frac{t_{i,j} - \widehat{o}_j}{\widehat{r}_j} \approx T_i + d_{i,j} \tag{4.28}$$

for each timestamp. In Figure 4.13, we show the results of this correction. Again we plot the timestamp differences to the reference node, the y-axis uses the same scale as in Figure 4.12.

The approximation in (4.28) is exact if the estimates $\widehat{r}_j$ and $\widehat{o}_j$ of r_j and o_j are exact. Remaining clock deviations or estimation errors would therefore be visible in Figure 4.13: they would result in remaining timestamp differences to the reference node.

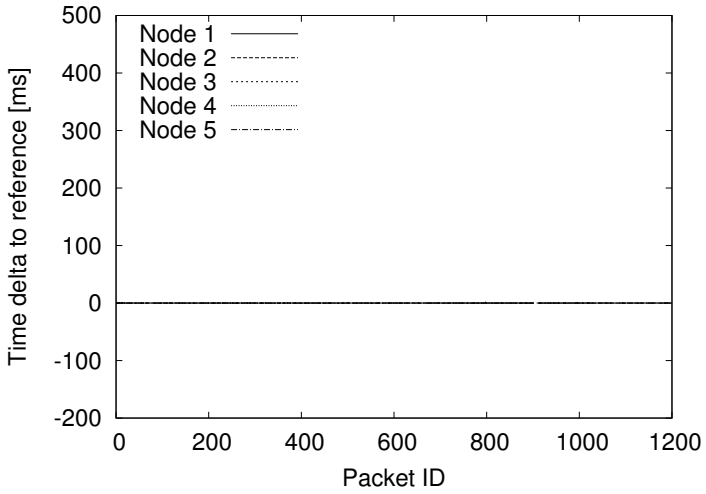


Figure 4.13: Synchronized timestamp differences in real-world experiments.

That such errors are in fact virtually non-existent becomes clear if we zoom the y-axis further in, as we do in Figure 4.14. It can be seen that the timestamping differences are typically in the order of some ten microseconds, with occasional outliers of up to 1–2 milliseconds. There is, however, no sign of a systematic (i. e., rate or offset estimation) error, like clocks drifting apart over time. This indicates that the rate and offset estimates are indeed correct.

Note that eliminating the estimated linear clock deviations according to (4.28) leaves the timestamping delays in the data. In a practical application, our approach would have been able to also eliminate long timestamping delays with very high probability. Recall that given exact rate and error estimates, it removes all but the shortest timestamping delay that occurred for events with multiple observers. Since we do not know the true times of the packet receptions in the experiment, we cannot tell how large exactly the then remaining deviations are. It seems reasonable, however, to assume that the smallest timestamping delay for a packet is within the same order of magnitude as the minimal timestamp difference to the reference node[6]. Considering (4.28) this difference is simply the difference of two timestamping delays. In the discussed experiment, the average of the per-packet minimum timestamp difference is 5 microseconds; for 95 % of the packets, it is below 29 microseconds.

In order to examine the impact of clock drift, we have performed a second experiment

[6]This does of course not hold when the minimum path delay is included in the timestamping delay. This, however, is not a problem here: recall from Section 4.3.1 that the minimum path delay in a node is equivalent to an additional clock offset, and may thus be eliminated.

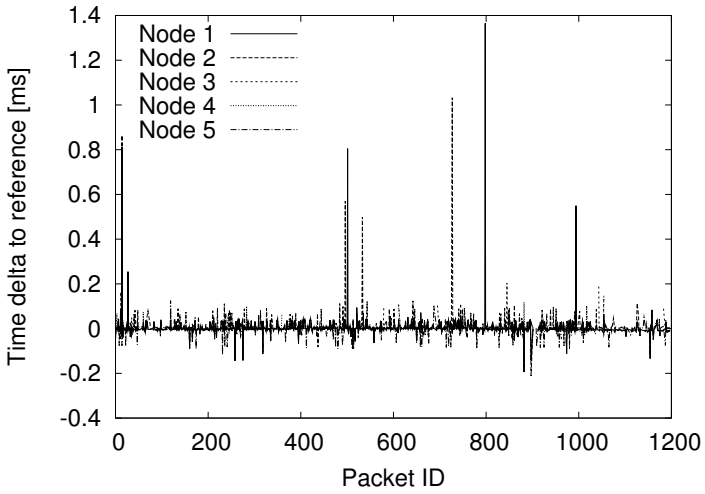


Figure 4.14: Synchronized timestamp differences in real-world experiments (zoomed).

with a duration of 100 minutes. Note that this significantly exceeds the time span over which drift can be neglected. The experimental setup as well as the used evaluation methodology are the same as above. The results of this experiment after correcting the timestamps according to (4.28) are shown in Figure 4.15. The non-linear deviations—effects of clock drift—are clearly visible. Nevertheless, our approach provides good estimates for the offsets and rates. Moreover, 95 % of all timestamp differences are below 142 microseconds.

4.4 Least Squares Timestamp Synchronization

In addition to the above presented MLE timestamp synchronization, we have developed an approach that performs the same task with least squares regression. This approach avoids the assumption about exponentially distributed timestamping delays and we were able to proof its convergence to the correct values for a fully featured version. We give a short overview in the following section, for details refer to [JKM+].

By reformulating (4.2) for an event $i \in I$ and $k, l \in R_i$ it follows that

$$\begin{aligned} r_l t_{i,k} &= r_k r_l (T_i + d_{i,k}) + r_l o_k \\ r_k t_{i,l} &= r_k r_l (T_i + d_{i,l}) + r_k o_l. \end{aligned} \tag{4.29}$$

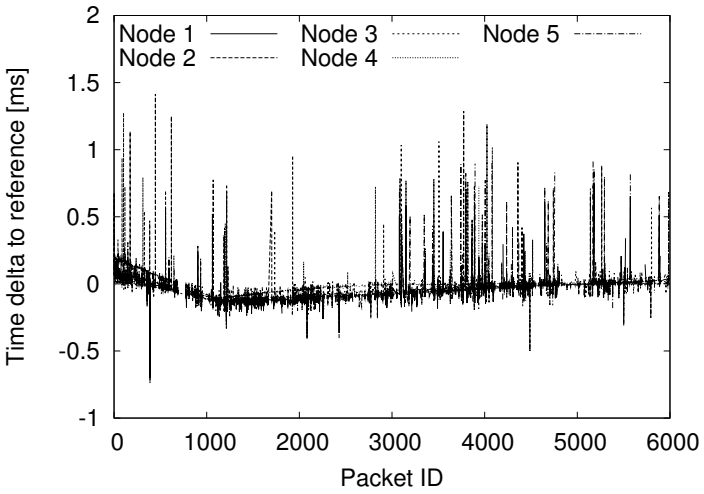


Figure 4.15: Synchronized timestamp differences in a real-world experiment with an overall duration of 100 minutes.

Subtracting the second from the first equation yields

$$r_l t_{i,k} - r_k t_{i,l} = r_k r_l (d_{i,k} - d_{i,l}) + r_l o_k - r_k o_l. \tag{4.30}$$

In (4.30), the unknown quantities T_i are eliminated. To obtain a set of linear equations, we substitute $\bar{r}_j := 1/r_j$ and $\bar{o}_j := o_j/r_j$ for all $j \in J$. Multiplying relation (4.30) by $1/(r_k r_l)$, we obtain

$$t_{i,k}\bar{r}_k - t_{i,l}\bar{r}_l + \bar{o}_l - \bar{o}_k = d_{i,k} - d_{i,l}. \tag{4.31}$$

Consider a fixed pair $k, l \in J$ with $k > l$ and where the set

$$R^{k,l} = R^{l,k} := R^k \cap R^l$$

of events i that are recorded in both nodes k and l is nonempty, $R^{k,l} \neq \emptyset$. By adding (4.31) for all $i \in R^{k,l}$ we obtain

$$\bar{t}_{l,k}\,\bar{r}_k - \bar{t}_{k,l}\,\bar{r}_l + |R^{k,l}|\,\bar{o}_l - |R^{k,l}|\,\bar{o}_k = \Delta d_{k,l}, \tag{4.32}$$

where

$$\begin{aligned} \bar{t}_{l,k} &:= \sum_{i \in R^{k,l}} t_{i,k} \\ \bar{t}_{k,l} &:= \sum_{i \in R^{k,l}} t_{i,l} \\ \Delta d_{k,l} &:= \sum_{i \in R^{k,l}} (d_{i,k} - d_{i,l}). \end{aligned} \tag{4.33}$$

We obtain up to $|J| \cdot (|J| - 1)/2$ equations of the form (4.32). These are summarized in the matrix equation

$$M \begin{pmatrix} \bar{o} \\ \bar{r} \end{pmatrix} = \Delta d, \tag{4.34}$$

where $\bar{o}$ and $\bar{r}$ denote the vectors of the $\bar{o}_j$ and $\bar{r}_j$, respectively.

To overcome the existing rate and offset ambiguity, corresponding rows are added to the matrix as normalization constraints. As an estimator for the unknown parameters $\bar{o}_j, \bar{r}_j$, a least squares solution to the resulting system is computed. Once these estimates—and thus the $r_j = 1/\bar{r}_j$ and $o_j = \bar{o}_j/\bar{r}_j$—are readily available, the estimates for the event times T_i can be obtained, for example, from (4.2) by setting

$$\widehat{T}_i = \min_{j \in R_i} \frac{t_{i,j} - o_j}{r_j} = \min_{j \in R_i} (\bar{r}_j t_{i,j} - \bar{o}_j). \tag{4.35}$$

Under connectivity assumptions similar to the ones in Section 4.3.4, it can be proven that the least squares algorithm converges to the correct rate and offset values. As this proof has been conducted with the fully featured least squares algorithm, this is a significantly stronger result than the proof for the simplified MLE approach. However, with the same methodology as in Section 4.3.5, the two estimators can be compared numerically. The results of this comparison are shown in Figure 4.16 and Figure 4.17 for the rate estimation for exponential and gamma distributed timestamping delays. It is obvious that the MLE approach yields better estimates for both distributions, and these differences are even more pronounced for the offset estimates in Figure 4.18 and Figure 4.19.

These results show that the MLE approach yields significantly more accurate estimates even in cases where the underlying assumptions are not satisfied. The theoretical bounds that have been derived for the least squares estimator can thus be regarded as an indirect justification of the MLE.

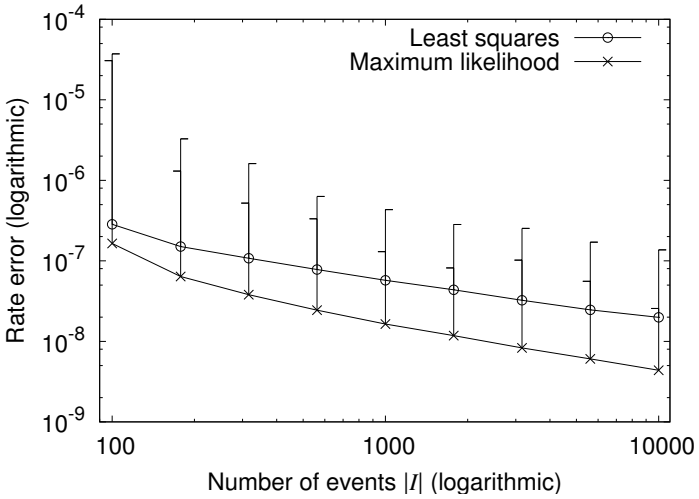


Figure 4.16: Rate estimation errors for exponentially distributed timestamping delays with least squares and MLE ($\lambda^{-1} = 10^{-4}\,\mathrm{s}$).

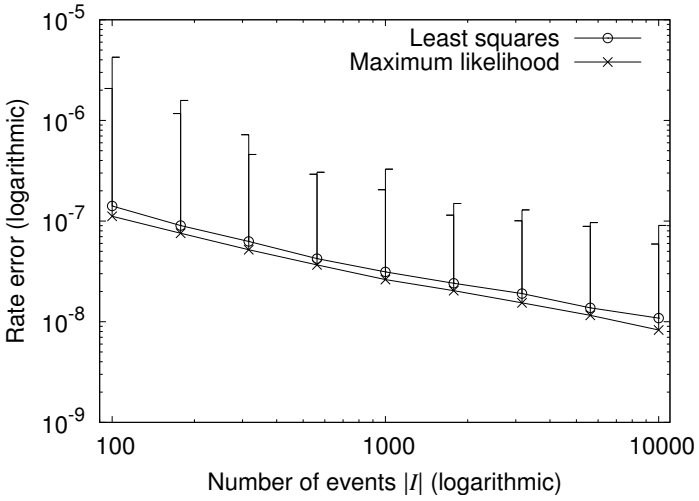


Figure 4.17: Rate estimation errors for gamma distributed timestamping delays with least squares and MLE ($\lambda^{-1} = 10^{-4}\,\mathrm{s}$).

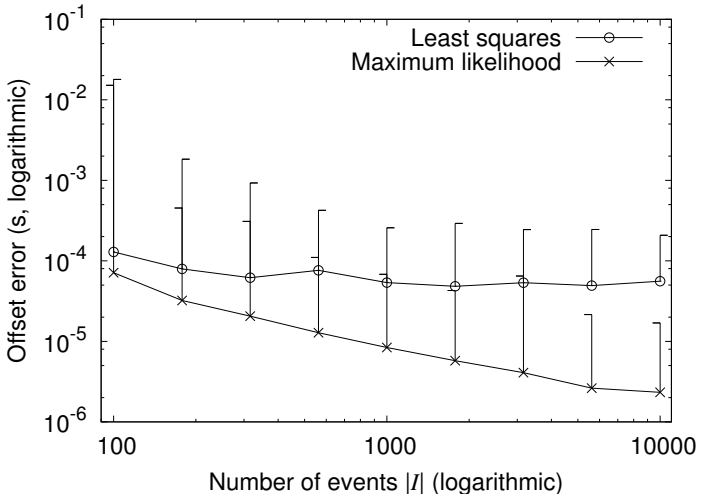


Figure 4.18: Offset estimation errors for exponentially distributed timestamping delays with least squares and MLE ($\lambda^{-1} = 10^{-4}\,\mathrm{s}$).

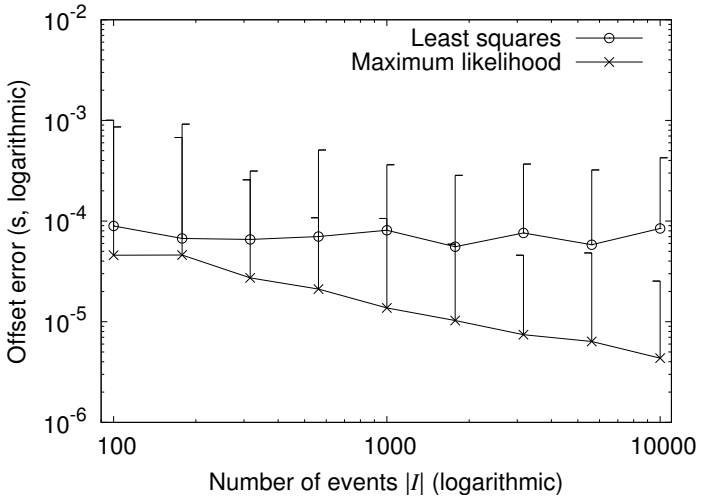


Figure 4.19: Offset estimation errors for gamma distributed timestamping delays with least squares and MLE ($\lambda^{-1} = 10^{-4}\,\mathrm{s}$).

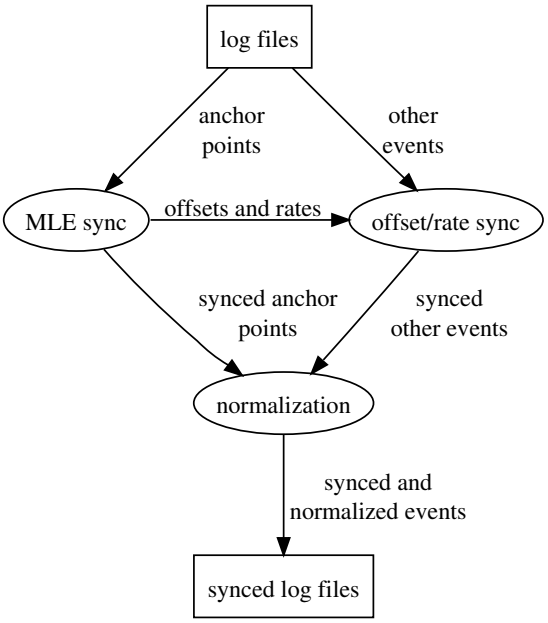


Figure 4.20: Structure of `pcapsync`.

4.5 Pcapsync - Applying MLE Timestamp Synchronization to Real-world Data

Up to now, it has been just assumed that anchor points can be identified as such. In the following section, we examine how they can be identified in real-world data. We concentrate on experiments with IEEE 802.11 networks [802] documented in libpcap format [LIB] (used by tcpdump [TCP] and wireshark [WIRa], for example), as this is the most common type of WMN experiment. This method together with the MLE timestamp synchronization has been implemented in the `pcapsync` tool. The single steps for this are shown in Figure 4.20: `pcapsync` reads a set of libpcap log files, identifies potential anchor points in them, applies our offline time synchronization algorithm, maps the recorded local timestamps to a common, global time scale, and finally writes back a corresponding set of synchronized libpcap files. Its output can thus be immediately used for further analysis with standard tools. The following section is based on a paper on `pcapsync` [MKS+08].

As stated in Section 4.3.1, an anchor point is an event observed and timestamped at nearly the same moment by multiple nodes. These anchor points are the foundation of MLE synchronization, and its performance crucially depends on correctly identifying them. When examining the tracing behavior of libpcap shown in Figure 4.21, it becomes

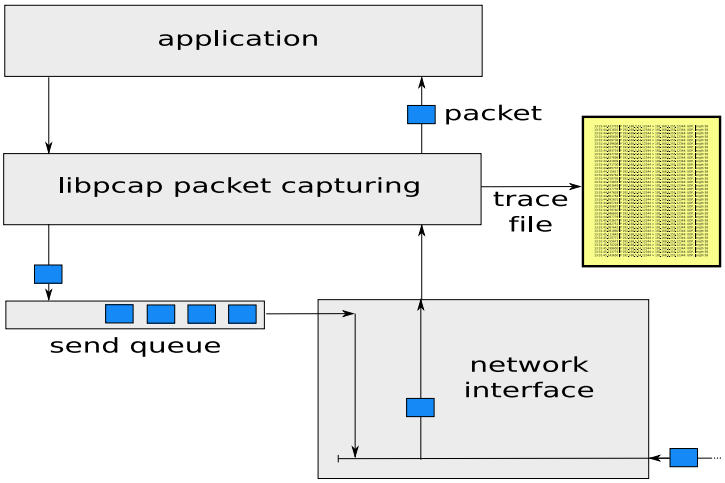


Figure 4.21: Packet recording with libpcap.

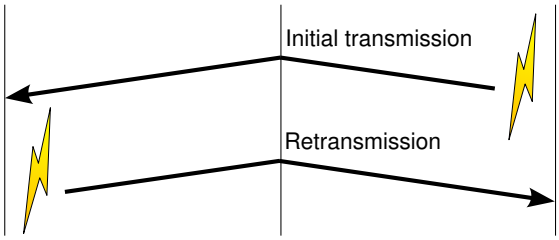


Figure 4.22: Ambiguous reception times in case of retransmissions.

clear why outgoing packets are not suited as anchors. These packets are recorded by libpcap before being added to the send queue, and the delay from its recording to its transmission can be quite large, e.g. if the medium is occupied. Thus, a packet recorded due to a send event marks a different physical event than the corresponding packet reception(s). In contrast, the logging for received packets takes place after their reception with a relatively small delay. If several nodes receive and timestamp the same packet, this happens at nearly the same moment in time. In `pcapsync`, we therefore use parallel *receptions* of the same transmission as anchor points. For a real wireless network, it is thus necessary to identify groups of timestamped packet receptions in the libpcap files belonging to the same physical transmission. One central duty of `pcapsync` is the identification of such events.

The link layer reliability mechanisms in IEEE 802.11 retransmits unicast packets up to seven times if an acknowledgment is missing [802]. If multiple nodes receive the same

unicast transmission[7], these events therefore do not necessarily belong to the same physical layer transmission: for example, as shown in Figure 4.22, it may well happen that one node receives only the first transmission attempt, while another node receives only the second one. Unfortunately, it is not possible to record the number of performed retransmissions in a hardware-independent way. A packet reception log entry does also not reveal which (re)transmission attempt has been received. Thus, in the specific case of 802.11, unicast packets cannot be used as anchor points. For broadcast packets there is no automatic retransmission. Nevertheless, it can happen that identical broadcast packets recorded in the log files refer to different transmissions since higher layers may generate multiple copies of the same packet. ARP [Plu82], for instance, often broadcasts identical requests when the same address is resolved again. However, broadcast packets generated multiple times can easily be identified using the records about sent packets in the log files. Consequently, they are not used as anchor points. In summary, `pcapsync` is able to use parallel receptions of globally unique broadcast transmissions as anchor points for the synchronization in 802.11 networks.

Based on these rules, the events that can be used as anchor points and those which are not suitable for this purpose can be identified and separated. For the anchor points, synchronized timestamps are estimated by the MLE algorithm. As it also yields estimates for clock rates and offsets of all nodes, the timestamps of all other events can be corrected by applying a linear transformation. For an event observed at local time t by some node with estimated clock rate $\widehat{r}$ and estimated offset $\widehat{o}$, it can easily be seen from (4.1) that the corrected timestamp $\widehat{T}$ is given by

$$\widehat{T} = \frac{t - \widehat{o}}{\widehat{r}}. \tag{4.36}$$

After calculating synchronized timestamps for all events, `pcapsync` normalizes them such that the first event in the experiment occurs at time zero. For this normalization, the globally earliest synchronized event timestamp is subtracted from all timestamps. Finally, `pcapsync` writes the data to new, synchronized per-node log files. To simplify the evaluation and visualization of a network experiment, the tool also offers the option to write the synchronized data into one global log file.

[7] Note that this is generally possible if the log files are recorded in promiscuous mode.

4.6 Chapter Summary

In the first part of this chapter, we have shown that the clock skew of typical PDA-class devices can be reduced by two orders of magnitude when NTP skew correction is used. The clocks' deviations are reduced from 12 s to 0.2 s over a time of 39 hours. For this, it is only necessary to let the NTP daemon use a previously generated drift file to correct the clocks without requiring any connection to an external reference time source. The non-linear behavior of the clocks that became obvious due to the correction is inherent to the clocks themselves. We have shown that it can be provoked by changing the environmental temperature. However, as long as an experiment does not span several hours with significant temperature changes, the impact of this is rather small. To sum up, this purely software-based method improves the clock quality to a point that makes it suitable for the coordination and monitoring of WMN experiments.

In the second part, we then consider offline timestamp synchronization for networks with local broadcast media. We have proposed a method to combine separate event log files from nodes in such a network into one single log file with a common time basis, in spite of deviating local clocks and latencies that occur when the timestamps for the events are generated. The key issue is how the deviations of the clocks and the latencies can be addressed without the necessity of additional communication between the network nodes. Our algorithm utilizes transmissions that have been received by multiple nodes as anchor points. It has been shown how the synchronization can be formulated as an optimization problem, and how this problem can be expressed as a linear program. The special structure of this linear program can then be exploited by an efficient solution algorithm. We have presented an implementation of a specialized solver, and comparisons to other LP solvers underline the performance of the employed solution techniques. Furthermore, we have presented analytical results on the quality of the synchronization for a simplified variant of the estimator. In particular, these results include a bound on the maximum possible synchronization error, depending on the maximum timestamping delay, and a consistency proof. A subsequent numerical evaluation shows that the convergence to accurate estimates is quick, and that the solver is robust even if the underlying assumptions do not hold. The performance of this first approach is also underlined by the numerical comparison with the least squares approach that makes weaker assumptions. To apply the generic MLE approach to data from experiments with wireless multihop networks, we have implemented `pcapsync`. The main focus of this tool is the correct identification of suitable anchor points in real-world data, and we have shown how globally synchronized timestamps can be obtained

also for packets not suitable as anchors. With this tool, we have synchronized data from a number of experiments demonstrating that the MLE estimator is able to correctly handle real-world data in the presence of timestamping delays and clock drift.

We consider the presented MLE approach generally applicable whenever event data is distributed over multiple sources, and common events can be used as anchor points for a maximum likelihood timestamp synchronization. Apart from supporting the interpretation of experimental results in networks with local broadcast media that we have primarily considered, adaptations of the proposed technique for example in the field of network forensics can be envisioned. Here, data of, e. g., multiple intrusion detection systems (IDS) or firewall logs can be combined as certain events will often have been observed in parallel by multiple systems.

Chapter 5

Trace File Analysis

Chapter Outline

After a real-world experiment or a simulation is completed, it has to be interpreted based on the recorded trace files. The result of such an analysis are graphs usable e.g. for scientific publications. As an example for this process, consider the calculation of the packet delivery ratio between two stations A and B of a protocol X based on data from a real experiment. For this, the information in the two packet traces has to be processed by: 1) parsing the traces, 2) removing packets not sent by A or not belonging to protocol X, 3) counting the number of packets for each time interval in both files, 4) dividing the matching values through each other, and 5) producing a plot of these values.

Obviously this requires a significant, custom-made processing of the input data. Therefore, such analyses are most often performed with custom-made software tools written in scripting languages like Perl, Python, Sed, or Ruby. When such a program is built from scratch, it can be crafted for the current analysis. However, these programs are only a by-product of an examination and their creation can be time consuming. As a result, such analysis programs are often "quick hacks" and lack a good software design, thus reducing maintainability and reusability. Furthermore, every small change in the analysis process requires the manual adaptation of the program source code and the subsequent rerunning of the whole analysis.

After examining our own programs written over the last years for the analysis of simulations and real experiments, it became obvious that a lot of simple as well as complex operations recur in modified form in nearly each program. Therefore, we have developed the *Extensible Data Analysis Toolkit* (EDAT) to encapsulate and reuse these recurring functionalities in so-called *operators*. EDAT treats the data to be analyzed as a flow

that runs through a chain of concatenated operators. Each operator in the chain modifies the incoming data stream and hands the result over to the next operator. EDAT itself provides more than 50 operators and can be easily extended within a few minutes by either implementing new operator classes or combining existing operators to more complex ones. The tool can produce graphs in postscript-format and features a graphical user interface to combine operators by simple drag-and-drop. EDAT is designed for 1) the rapid development of analyses with 2) detailed control over the whole process of data manipulation with 3) the same power as directly programming it in a scripting language. The toolkit is in productive use and has already been used for the analysis of our own experiments, among them those for the different time synchronization approaches in Chapter 4 and for the repeatability examination in Chapter 6. A paper on the toolkit is in preparation [KCWM].

5.1 Related Work

The tasks in the post-experiment analysis based on recorded data can be divided into parsing, processing, data modeling/mining, and visualization. After extracting the raw data from a collection of files in the parsing step, this data is processed by combining, filtering, and transforming the information. Data modeling/mining is the application of statistical methods or clustering techniques to discover patterns and dependencies in the data, and visualization consists of producing a two or three-dimensional graphical representation of the result.

The processing on a per-packet basis is supported by network tracers like tcpdump [TCP] or wireshark [WIRa] in an offline-mode. In this mode, packets in capture files can be filtered, and these tools also provide some basic manipulation of the files' content like cropping or altering packet timestamps. If the data has been parsed and inserted into a relational database, processing can also be performed by means of SQL with operations like joining or averaging. For more complex processing, the network data mining tool CoMo [Ian06] can be used. CoMo manages the recording and storage of raw capture data as a data flow and provides callbacks in this flow. Via these callbacks, the user can insert custom functions written in C to implement the analysis.

The idea to combine generic elements to a processing pipeline can already be found in tools like the Unix shell bash or in dataflow programming languages. Here, we concentrate on tools that are somewhat related to the task of network data analysis and that use this approach mainly for visualization or data mining. Huginn [SFT+05], a

3D visualizer for simulation trace files, allows to combine a predefined set of processing components within a fixed pipeline. The resulting information is used to alter the properties of the displayed network nodes. In the data mining tool KNIME [Kni], the necessary statistical calculations are created by graphically combining processing components. KNIME also supports simple (pre-)processing of the input data, e.g., via filtering or sorting. For more complex processing however, KNIME would have to be extended with new components. As their creation requires a lot of work [Kni] this may be too time consuming for an analysis process requiring such extensions on a regular basis. OpenDX [OPEa] is a data visualization tool for 2D or 3D plotting and also allows the production of animations. It provides a visual program editor to configure the plotting with predefined, connectable components. LabView [lab] is a commercial visual programming language mainly used to build measurement and control applications. The analyses provided by LabView concentrate on the description of physical phenomena, e.g., by means of signal or image processing or wavelet transform.

Instead of providing a framework to create analyses, TraceGraph [TRA] defines standard evaluations. This trace file analyzer is designed for files produced by the network simulator ns-2 and supports over 200 different analysis types. If the data has already been processed, plotting can be performed with gnuplot [gnu]. It requires an input file in a table style format and some configuration parameters and is able to plot the corresponding graph in a large number of output formats.

5.2 Philosophy, Architecture and Implementation

An examination of custom-made, handwritten programs for the analysis of network simulations and experimental traces shows that these programs share a lot of similar functionality. Instead of implementing each of these recurring operations from scratch upon design of a new analysis, EDAT provides a framework to encapsulate them in so called *operators*. These operators can be combined to form a data processing pipeline where the data is handed from one processing element to the next and successively transformed in each step. To adapt to the processing needs, each operator can be configured by means of certain parameters. With this approach, an analysis is a concatenated sequence of operations on the input data.

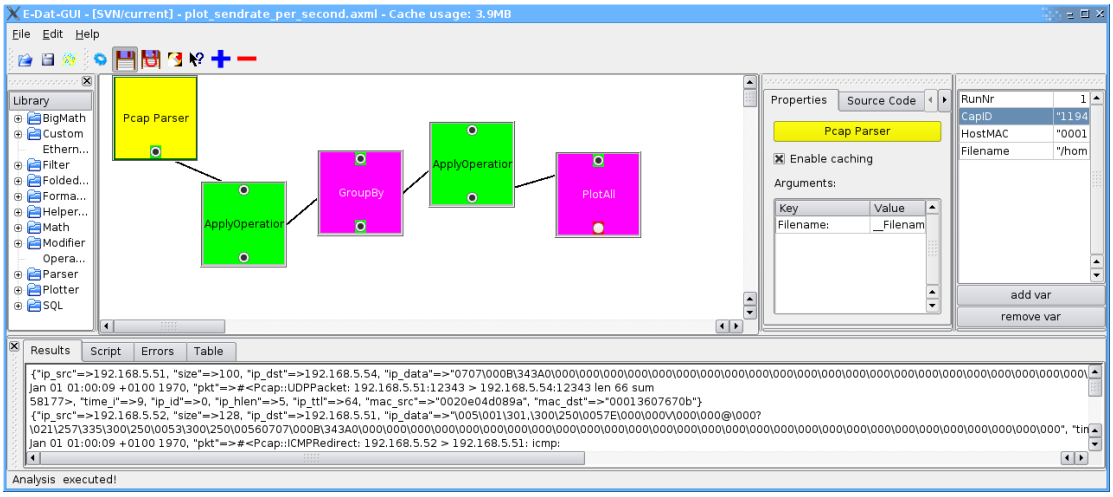


Figure 5.1: Screenshot of the EDAT GUI. The analysis shown on the workbench produces a plot of the number of packets a node has sent per second.

5.2.1 Graphical User Interface

A data processing pipeline is created with the EDAT *graphical user interface* (GUI) as shown in Figure 5.1. The GUI provides boxes as visual representations of the operators that can be added and combined by simple drag and drop. The user interface is divided into four main areas, the operator library on the left, the workbench in the center, the operator inspector on the right and a result and feedback view at the bottom. Once an operator is on the workbench, selecting it shows its current configuration in the inspector. To integrate an operator into the analysis, its input and output ports must be connected to other operators. Existing connections are represented by lines drawn from an output port at the bottom of one box to an input port at the top of another box. Each of the boxes provides a context menu to delete the operator or view and modify the operator's source code. The "execute" entry in the context menu triggers the processing of this operator. The result is then shown in the text box at the bottom, allowing for an inspection of the data in the flow.

A noteworthy feature of the EDAT GUI is *operator folding* that allows to create a new operator by combining other operators. As a simple example, imagine that an analysis of all packets with a size between 500 and 1 000 bytes should be performed. To extract these packets, two consecutive filters can be inserted in the data flow, the first filtering out all packets below 500 bytes while the second discards all packets above 1 000 bytes. However, if such a range filter is needed in another analysis, the same two filters would

have to be configured once again. Instead, these filters can be combined to a new operator: once they have been selected on the workbench, their context menus provide the "Create compound operator" option. This inserts a new operator into the library that internally uses the two filters to perform its tasks. In order to create a generic range filter, special tags can be used as configuration parameters. Each of these tags is used as argument of the newly created range filter and internally handed over as configuration option to the internal filter operators.

5.2.2 Operators and their Data Format

Under the hood, an EDAT operator is implemented as class in the scripting language Ruby. The input ports are realized as references to other operator objects, and the remaining configuration is performed via additional constructor parameters. Between operators, data is exchanged in generic data containers. The format of these containers is related to the tables of relational databases but allows for "rows" with arbitrary structure. To this end, each row is implemented as an associative array of key-value pairs, and all rows together are stored in an array that preserves the input order. The processing in an operator is performed in three steps: 1) get the container with the output of the preceding operator; 2) modify the data and put the result in another container; 3) return the result container to the subsequent operator.

This mechanism is best explained by means of an example: consider an experiment where packets are transmitted over a network with lossy links. The goal of the analysis is to calculate the length of the occurring error bursts based on the receiver packet trace. The corresponding EDAT parser for this trace format produces a data container in which each row corresponds to one packet. The different header fields are stored as key-value pairs. A row from this container looks as follows:

```
{"ip_src"=>192.168.5.50, "time"=>Wed Jul 25 09:03:02 +0200 2007,
"ip_dst"=>192.168.5.255, "id"=>46, ...}
```

To compute the length of the error bursts, we use the `CalculateInterrowDifference` operator. It calculates the difference of the same field between each consecutive pair of rows. The idea is to compute this delta for the packet-id field: if the id of consecutive packets (rows) differs by n, $n-1$ packets are missing. The implementation of this operator looks as follows:

```
class CalculateInterrowDifference < Basic
 def initialize(input, key)
  super(input, key)
  @input     = input              # preceding operator
  @key       = key                # target field
  @delta_key = key + "_delta"
 end

 def process()
  input_container  = @input.getResult() # input data
  result_container = Array.new()
  previous_row = nil
  input_container.each do |row|  # loop over input data
   if previous_row.nil?()        # first row: no delta ...
     row[@delta_key] = nil       # ... calculation possible
   else                          # else: calculate delta
     row[@delta_key] = row[@key] - previous_row[@key]
   end
   result_container.push(row)
   previous_row = row
  end
  return result_container        # return the result
 end
end
```

The constructor has two input parameters, a reference to the operator that serves as `input` and the `key` to the field for which the difference should be calculated. The whole processing is then performed in the `process` method. The call to the `getResult` method of the preceding operator triggers this operators' `process` method and returns the input data container. The calculation of the difference is then performed in a loop over all rows in this container. The result is stored in a new container and returned at the end of the method.

With the correct configuration, this operator transforms the input data into

```
{"ip_src"=>192.168.5.50, ..., "id"=>46, "id_delta"=>36, ...}
{"ip_src"=>192.168.5.50, ..., "id"=>54, "id_delta"=>8, ...}
```

where the new field `id_delta` contains the delta between consecutive ids. In the current example this difference is eight, thus a total of seven packets have been lost between these two successively recorded packets.

5.2.3 Creating an Analysis

From an implementation point of view, an analysis is a concatenation of operator instances in a stand-alone Ruby script. This script consists of commands to instantiate and configure the different operators and a call to the last operators' `getResult` method. If the GUI needs to execute an analysis, it generates the corresponding script and then executes it in a separate process. Due to this architecture, an analysis can also be invoked from shell or Ruby scripts to perform batch processing, or it can even be integrated into other programs.

5.2.4 Example Operators

Up to now, the general architecture of EDAT and the idea behind the operator concept have been described. In order to get an impression of the capabilities of this concept as well as on the different operations that are already supported, this section presents some of the operators implemented up to now.

In our experiments, it was often necessary to partition the input data into subsets and perform a certain operation on each subset. An example is the calculation of delivery ratio per second, where the packets sent in a certain second are the subset and the operation is counting these packets. This operation is implemented by the `GroupBy` operator that is inspired by the `group by` clause of SQL. It builds such subsets and executes an operator on each subset. However, it is not limited to simple aggregate functions but can also apply any operator to the data in the groups. Similar to `GroupBy`, the `Join` operator is inspired by the corresponding SQL expression. `Join` combines or matches the input from two flows to one flow based on a configurable key that serves as join-criterion. The problem of duplicate keys that occurs when two rows of data with the same content are joined is solved by appending "1" or "2" as a suffix to each key.

In order to interact with an SQLite database, the `SimpleSQL` operator can be used. It is configured with an SQL statement and returns the result in the well-known form of an array of hashes. Furthermore, there also exists an `Insert` operator to load the data in the flow into an SQL table. For this, the structure of the flow is analyzed and an appropriate table is created, then the single lines are inserted into this table. A concatenation of these two operators even allows arbitrary SQL statements to be executed on the data in the flow: `Insert` integrates the data in the database and `SimpleSQL` executes the target query.

The important task of visualizing analysis results is performed by the `PlotAll` operator. It is designed to produce two dimensional plots with one or more curves. To do this, the operator assumes that each row in the input flow contains the data for one point on the x-axis. The field representing this point can be configured as one of the input parameters. All other values in a row are plotted as data points that belong to the configured x-value. The plotting itself is performed by gnuplot: the preprocessed data is written to a file in appropriate format and then an instance of gnuplot is created for plotting. As the gnuplot files as well as the resulting postscript graphs are stored in the filesystem, `PlotAll` implicitly creates a chronologically sorted archive of all plots.

5.3 Advanced Features

5.3.1 Automated Caching

When analyzing data and working with it, a user often just changes the parameter of one operator (e.g., the scope of a filter) and re-executes the whole analysis to examine the influence on the result. Furthermore, most analyses are developed in a step-by-step process in which the user adds a new operator to the end of the current analysis-flow and then re-executes it to examine the output. In both cases, the same calculations would be repeated over and over again, a rather time consuming task when large amounts of data are processed.

To cope with this, EDAT supports the caching of previous computations. This is implemented in the base class `Operator` from which all other operators are derived. Thus, newly created operators automatically inherit this feature. For the caching, the result of a computation is serialized and written to a file-system directory. If the configuration of an operator has changed, a recalculation is necessary, otherwise the result can be loaded from the cache. To determine whether a recalculation is necessary, EDAT uses a fingerprint of the operators' configuration that changes as soon as the configuration parameters change. Depending on their type, these parameters are treated differently when included in the fingerprint: 1) in order to determine whether the input provided by another operator has changed, its fingerprint is used. 2) If the parameter is a file, the modification timestamp is considered. 3) Parameters like strings or numbers can be directly included. All this different information is then concatenated. In order to avoid

that the fingerprint becomes too large due to these concatenations, an md5sum [Riv92] over all these values then represents the operators' fingerprint[1].

5.3.2 Executable Pieces of Code

As EDAT is implemented in a scripting language, it is possible to configure operators with pieces of code that are evaluated and executed at runtime. In contrast to a compiled language like C/C++ or Java in which the operations must be specified at compile time, this allows for more generic operators. With EDAT, the data can be modified in ways that would normally require a significant amount of manual programming. For example, in one of our experiments packets of varying size have been sent over a multihop network. To allow for a unique identification as a packet travels from node to node, each packet carries a consecutive number. This number is wrapped in a UDP packet that is itself wrapped in an IP packet. Extracting this number thus requires to take the payload of the IP packet (i.e. the UDP packet), strip the eight UDP header bytes, and convert the rest to an integer. For this task, the operator `ApplyOperation` is configured with an executable piece of code as third argument:

```
ApplyOperation.new(output, "ip_data", "[8..-1].to_i()")
```

The important code snipped from `ApplyOperation` that performs all the modifications in the `process` method looks as follows:

```
lines.each do |line|
   line[@key] = eval("line[@key]" + @operation)
   result.push(line)
end
```

The `eval` method provided by Ruby evaluates the expression that it gets as argument. Instance variables start in Ruby with the at sign "@". In the current example `@key` is the second and `@operation` the third argument of the `ApplyOperation` constructor. Due to the above configuration, the performed operation is thus

```
line[ip_data] = (line[ip_data])[8..-1].to_i()
```

[1] Note that a harmful fingerprint collision is very unlikely. For this to happen, 1) an md5sum collision has to occur and 2) the wrongly loaded result must not lead to a crash of the rest of the analysis. However, to fully avoid this, caching can be either switched off for the final analysis or the "detect fingerprint collision" option that verifies the fingerprint can be activated.

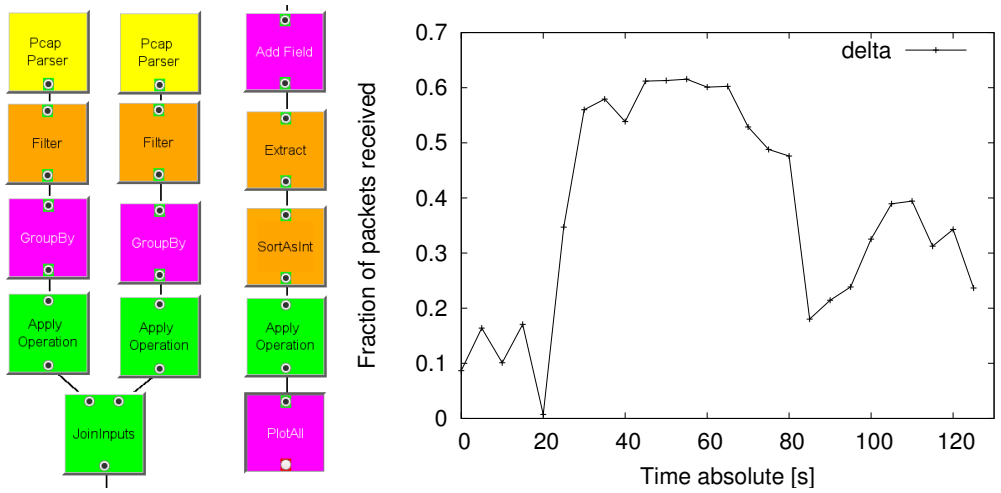
Figure 5.2: Configuration of the ApplyOperation operator.

Thereby the bytes starting at position eight until the end are extracted from the payload and the "to_integer" method is applied to the resulting substring. The configuration of this transformation via the GUI is shown in Figure 5.2. After the data flow has passed the corresponding operator, the `ip_data` field contains the decoded packet sequence number instead of the IP payload.

5.4 Case Study

In the introduction to this chapter, the calculation and plotting of delivery ratio has been used as an example for a standard analysis. How this analysis is conducted with EDAT will be demonstrated with data from one of our experiments. Here, we use the libpcap trace files from two laptops equipped with 802.11b network interfaces. The first laptop broadcasted 80 packets/s and the second recorded these packets with tcpdump. A screenshot of this analysis on the EDAT workbench can be found in Figure 5.3.

The first half of the analysis shown in Figure 5.3(a) starts with parsing the files and filtering the packets according to the correct sender address. In the `GroupBy` operator, the packets are first sorted according to their timestamps in 5-second buckets, followed by counting the number of packets in these buckets. After the results are extracted from the special "GroupBy" data structure, they are joined according to their bucket id. The data flow now contains lines of the form `[value_input1 = 416, value_input2 = 220, time_i = 1185347050]` where the first value represents the sent packets, the second the received packets, and the third the timestamp of the bucket. The second half of the processing can now be found in Figure 5.3(b). After the ratio between the two input-values has been added as additional field to each line of the flow, only the fields



(a) Part 1 (b) Part 2 (c) Graph produced by this analysis

Figure 5.3: Example analysis for plotting the throughput between two nodes.

required for the plot are extracted. The following operator sorts the rows according to their timestamp which is then normalized to start at time zero. The result of this analysis as it is plotted when executing `PlotAll` can be found in Figure 5.3(c).

To get an impression of the impact of caching, the runtime of the above analysis has been measured and averaged over 100 repetitions. Without caching, analyzing the 14 000 packets in the two files took 3.2 s on a 2 GHz Opteron. If caching is activated, the runtime with empty cache is 5.5 s as new results must be written to the cache files. This improves if cached data can be reused, e.g., if the bucket size in the `GroupBy`-operators is changed from five to ten seconds. Here, the results of the two previous `Filter`-operators are reused, resulting in a duration of 2.5 s. This shows the tradeoff: although twice as fast as the analysis with empty cache, the improvement is only moderate compared to the execution without caching. This changes if more of the previous calculations can be reused, e.g., if the delta calculation in the first operator of Figure 5.3(b) is changed to a subtraction instead of a division. Here the analysis only takes 0.02 s.

5.5 Chapter Summary

In this chapter, we have presented the extensible data analysis toolkit EDAT that is designed for the evaluation of network simulations and experiments. It follows a flow-based approach where modular operators are combined to form the analysis. The tool

is easily extensible, comes with a rich set of operators, and has already been used to produce the graphics for a number of our own scientific publications. Although extending the tools' capabilities requires some programming knowledge, analyses that use existing components can be created by non-programmers with its graphical user interface. EDAT eases the task of evaluating network simulations and experiments, allowing researchers to concentrate on the original problems rather than on the development of tools to analyze them.

Chapter 6

Repeatability

Chapter Outline

In Section 2.5, comprehension, correctness and repeatability have been identified as key requirements of scientific experimentation. A more detailed examination of these requirements shows that comprehension can be achieved if sufficient information is available. Thus appropriate tools are required to record this information which also depends on the experimental setup. Furthermore, an experiment must be correct, for broken tools, errors with the setup and other problems will corrupt the experimental result. As demonstrated in Section 3.3.3, a number of such errors can be detected by EXC, thus improving and supporting correctness. While this shows that solutions for the first two requirements exist, repeatability (and reproducibility that is founded thereon) has been neglected up to now.

Repeatability is the "closeness of the agreement between the results of successive measurements of the same measurand carried out under the same conditions of measurement" [TK94]. In existing experiments, most often it was just implicitly assumed that if all controllable factors are similarly set, i.e. all devices perform the same actions, also the outcome will be somewhat similar and can therefore be compared or averaged [SBSC03, BCDG05, GKN+04]. However, this is a risky assumption as such experiments take place in an unstable environment where uncontrollable factors play an important role. For example, the movement of obstacles, changing humidity or interference from external sources all strongly affect electromagnetic wave propagation and thus the experiment.

Due to the importance of these uncontrollable factors for the experiments' outcome, their influence should be monitored. However, separately monitoring all these factors, e.g. on the physical layer, can be extremely expensive and it is not clear if and how all

relevant factors can be recorded. Besides, these factors can fluctuate over very short intervals, making it nearly impossible to perform exactly repeated experiments. Instead, repeatability should be considered and verified at layer two of the ISO/OSI stack as reflected in the network topology. This has the advantage that all physical layer factors, even the unknown ones, are represented therein. If the topology is considered for intervals of appropriate length, also short-term fluctuations can be eliminated. Furthermore, instead of requiring special hardware or modifications to the used software, the corresponding values can be recorded and calculated with standard tools and devices.

In this chapter, we examine the repeatability achievable on a topological level. For this, we study the variability of topologies in identically executed experiments. These are performed back-to-back, i.e. within a short time period to limit environmental changes influencing physical layer parameters due to long term effects. In the initial step, a metric to quantify the topological similarity of experimental runs with wireless multihop networks is developed. It can be computed with link quality information recorded with standard packet tracers like tcpdump [TCP] for static as well as mobile setups. We then show that this metric is able to classify repetitions of the same experiment according to the heterogeneous topology caused by interference and distinct node behavior. This metric is used to examine – in strictly controlled experiments – how much topology variations occur in real-world environments. Based on these measurements we are able to identify classes of links that have a strong negative influence on topological repeatability. Finally, in a setup with multihop UDP traffic, the relation between an application layer and the topology metric is examined. The experiments show that it is in fact possible to repeat experiments with wireless multihop networks.

For this study, it was necessary to combine the different methodologies and tools developed throughout this thesis: the experiments have been controlled with EXC and the precise control and repetition of all actions was crucial here. The metric has been implemented with EDAT components and the evaluation required the `pcapsync`/MLE timestamp synchronization. The content of this chapter is based on a paper that is currently in preparation [KOTM].

6.1 Related Work

In the context of the APE project [LLN+02], the differences of repetitions of experimental runs with mobile ad-hoc network are considered. For this, signal strength measured with a custom driver is used to calculate the “virtual mobility” as perceived on the

radio layer. This calculation results in a two-dimensional time-mobility graph for each run that averages link quality changes over all nodes. By comparing graphs for different runs, an experimenter can visually assess the similarity of the average quality change in the runs. In contrast, our metric works with arbitrary 802.11 interfaces and only requires layer two information. Similarity is considered on an individual per-link basis, thus it is also suited for experiments where nodes follow individual movement paths. Furthermore, our metric *quantifies* the similarity between run pairs and thus allows for an automatic comparison of runs.

In [GKHS05], it is demonstrated that the physical layer behavior of different network interfaces strongly varies and should therefore be calibrated. Furthermore, the authors examine throughput and signal strength in experimental runs in the ORBIT testbed. They overlay time/value plots of throughput and signal strength to visualize their variation and calculate the overall mean and standard deviation in these runs. In contrast to this descriptive approach for specific experiments, we provide a generic method based on network topology that considers the influence of interference and mobility.

For network anomaly detection, it has been proposed to view a network as graph and model changing topologies as series of graphs [SKR99]. Techniques like graph edit distance are used to discover the one outstandingly different graph (i.e. topology) in a coarse-grained time series of graphs. By contrast, the examination of topological repeatability requires that a fine-grained series of *differences* between graphs considering the time dimension as well as the average difference be assessed on a per-link basis.

6.2 A Metric for Topological Similarity

For the remainder of this chapter, we assume that an experiment is divided in R runs. Two runs have a similar *type* if the used hardware and software, the actions (e.g. starting of packet generators), and the movements of the devices are similar. However, an equal configuration is only the prerequisite for repeated, similar runs. The second condition is the similarity of the resulting topologies that can be used as coarse-grained indicator for the influence of uncontrollable physical layer factors. If this similarity is high, it can be assumed that the uncontrollable factors have not changed significantly.

To determine the topological similarity of two runs, we propose a metric quantifying topology differences and following a bottom-up approach. In the first step, a measure for the similarity on a per-link basis is deduced. The measures for all links in the run pair are then combined to an estimate for the whole topology.

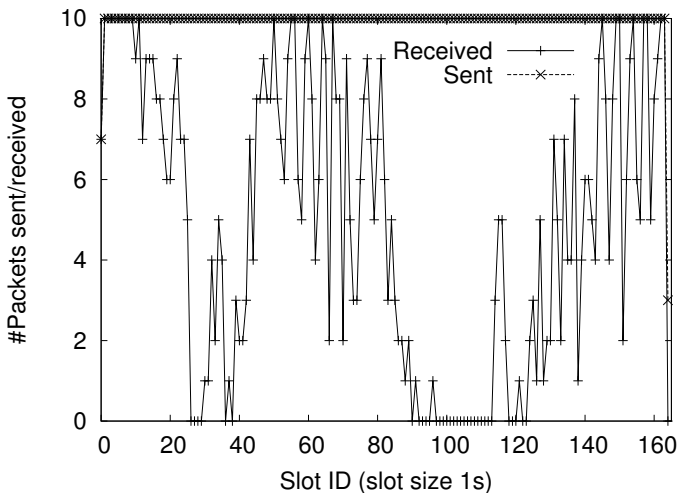


Figure 6.1: Link for a slot size of one second and ten packets/s.

6.2.1 Link Quality

A link in a wireless network is not simply "up" or "down" but rather exhibits a certain quality that varies over time [LLN+02, ABB+04]. This quality can be determined by calculating the fraction of packets successfully delivered in a certain interval i. If node a has sent s_i packets and node b has received r_i out of these packets, the link quality q_i^{ab} is:

$$q_i^{ab} = \frac{r_i}{s_i}. \tag{6.1}$$

To be able to easily compare the quality of a link in different runs, each run is divided in S similar-sized intervals called *slots*.

The examination of time variation of loss rates in [ABB+04] indicates that short-time fluctuations occur for at least a slot-size of one second. Furthermore, if the number of packets sent is low as shown in Figure 6.1, the link quality estimate will be rather coarse. To overcome these issues but still be sensitive to movement-induced quality changes, the quality is averaged over a sliding window in our metric.

Based on [ABB+04] and our own experiences, we parameterize the quality calculation with a slot size of one second and a window length of five seconds. Thus, the averaged quality w_i^{ab} can be computed as

$$w_i^{ab} = \frac{\sum_{k=i-s}^{i} q_k^{ab}}{s+1} \tag{6.2}$$

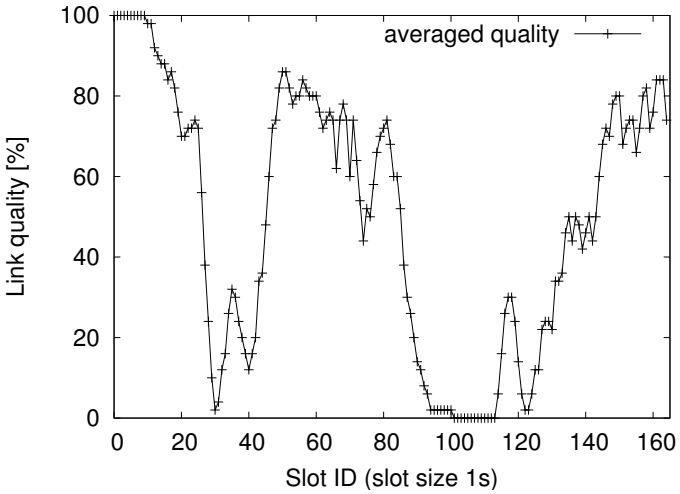


Figure 6.2: Link quality of Figure 6.1 after averaging with a sliding window of five seconds.

where

$$s = \begin{cases} i, & i < 4 \\ 4, & i \geq 4 \end{cases}$$

The quality for a link over the whole run is then $\vec{ab} = \{w_i^{ab} | i = 0, ..., S-1\}$. For the link in Figure 6.1, this results in a quality as shown in Figure 6.2.

6.2.2 Comparing Links

The next step is to assess the similarity for the same link in different runs. The problem is visualized in Figures 6.3 and 6.4 that show the qualities for two links A and B together with the inter-run differences. For a human, it is possible to discover that link A only has small differences and that this difference is stable while link B shows large differences and intermediate fluctuations. However, for an automatic comparison it is necessary to quantify this in a single figure. Therefore we decided to separately calculate measures for the two intuitive dimensions *difference* and *fluctuation* and then combine these to an overall measure for link similarity.

The basis for these measures is the difference of the link qualities as it is also displayed in Figures 6.3, 6.4, and 6.5. It can be calculated as $d_{rj}^{ab} = |\vec{ab}_r - \vec{ab}_j|$ and will be abbreviated as $d = d_{rj}^{ab}$ in the following. The first dimension, *difference* can then be expressed as

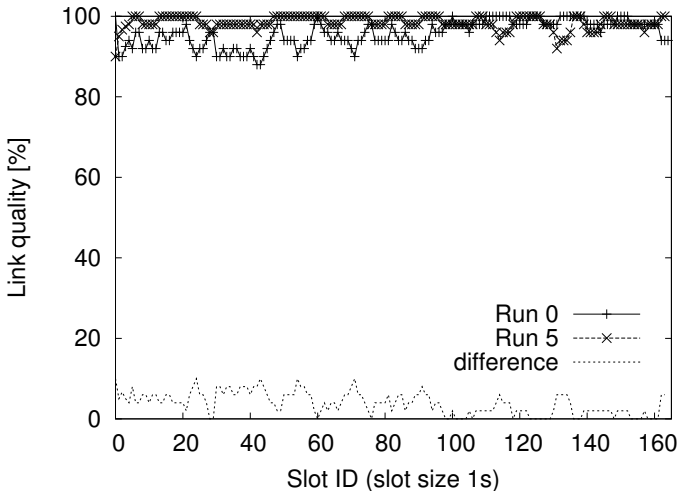


Figure 6.3: Link A with small differences and high similarity.

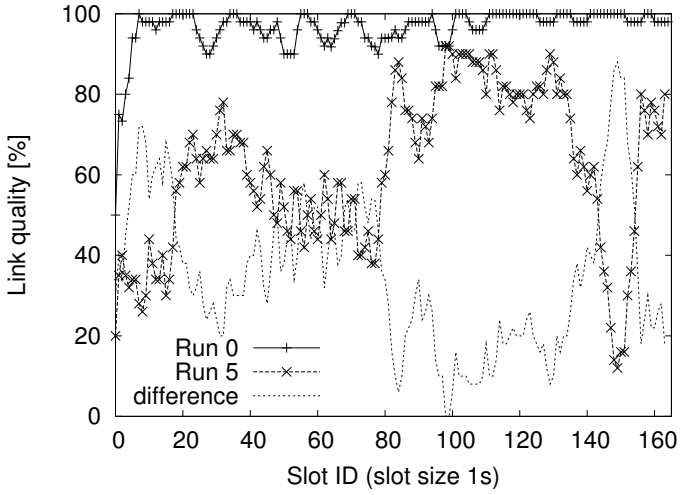


Figure 6.4: Link B with high differences and small similarity.

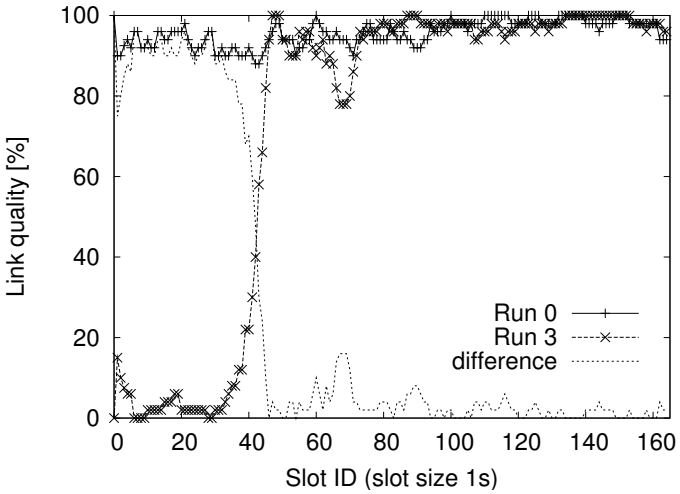


Figure 6.5: Link C with abrupt quality change in run 3.

average of these differences:

$$diff = \frac{\sum_{i=0}^{n} d_i}{n} \tag{6.3}$$

For link A in Figure 6.3, this is 3.8%, and the link in Figure 6.4 has an average difference of 35.1%.

The intuition to be expressed in the calculation of *fluctuation* is that a link with a stable difference of 10% between two runs is more similar than a link that has no difference at all during 90% of the time and a total outage during 10% of the runtime. For this, consider the example link in Figure 6.5. The average difference here reaches a value of 25.0% suggesting a higher similarity than the 35.1% average difference of the link in Figure 6.4. However the total outage in the first 40 seconds of run 3 can have an especially severe impact. To appropriately consider such cases, the variance can be used because it captures the dispersion of the values and overweights outliers. However, in order to assess the spread of link quality in the same unit as the quality itself, the standard deviation (the square root of variance) is used:

$$fluct = \sqrt{\frac{n \cdot \sum_{i=0}^{n-1} {d_i}^2 - \left(\sum_{i=0}^{n-1} d_i\right)^2}{n(n-1)}} \tag{6.4}$$

For link A in Figure 6.3, this standard deviation is 2.8%, link B in Figure 6.4 has a value of 19.0% and the link in Figure 6.5 has 37.4%.

Together, average difference and standard deviation allow to measure how different the link has been between the runs and how much this difference changed. In the remainder, we will call these two figures the *average* and the *deviation* of a (link) comparison, or shorter *AD metric* and use the notation (avg, dev) to characterize such a comparison. To quantify the similarity of a link in two runs in a single figure, average and deviation need to be weighted and combined appropriately. In the remainder of this paper we use equal weights and characterize the difference by the sum of these values, $avg + dev$. The estimate for the link differences is thus for link A (3.8,2.8) = 6.6%, for link B (35.1,19.0) = 54.1%, and for link C (25.0,37.4) = 62.4%.

6.2.3 Comparing Runs

Based on the AD values of single links, the overall similarity of a run pair is computed by averaging these values. The lower this average, the smaller the overall difference and the higher the similarity between the runs. Depending on the type of experiment and the links that are used, this evaluation can be based on all $n \cdot (n-1)$ links or on a subset thereof. The latter allows to focus on certain parts of the topology, e.g. on the path from a sender to a receiver, and to exclude other links. This can be helpful if these links are far away or otherwise do not influence the path but would negatively effect the expressiveness of the AD similarity metric. In this case we speak of examining the *relevant links*.

6.2.4 Applying the AD Metric to Real Data

As link quality can only be calculated when at least one packet is sent in the corresponding time interval, slots in which no packets are sent need to be considered. One approach is to appropriately design the experiment such that no empty slots occur. However, this limits the range of experiments that can be examined. Furthermore, even with a permanent packet source, it can still happen that no packets are sent, e.g. because the receiver of a unicast packet cannot be reached. In our own experiments, we therefore use linear interpolation to cope with empty slots.

With the AD metric, it is possible to compute similarity estimates for all run pairs in an experiment. However, comparing these estimates for all runs manually can become quite laborious as the number of possible combinations grows quadratically. On the other hand, this task cannot be fully automated. The resulting estimates are a relative measure where it is difficult to give absolute numbers for “similar” or “non-similar”. The

assessment of similarity also should consider the experimental conditions. Therefore, we use a graphical representation of the similarity estimates that will be called *AD plot* in the remainder. For this, the similarity estimates between all $\frac{R\cdot(R-1)}{2}$ run pairs in an experiment are sorted and then plotted as shown in Figure 6.10. The x-axis shows the IDs of the compared runs while the y-axis displays the AD similarity estimate for this pair. If all runs in an experiment are very similar as in the left half of the figure, the curve is flat and does not exhibit major jumps. A steep curve and abrupt jumps like in the right half indicate that the topologies exhibit more differences.

6.3 Experiments

All of our experiments have been executed with EXC to guarantee for runs as similar as possible. For the mobile experiments, we used the EXC node GUI so that the person carrying the device could precisely follow the movement path. All devices in the network used Linux as operating system and the integrated IEEE 802.11b cards in ad-hoc mode as network interface. The laptops used were IBM Thinkpads and the personal digital assistants (PDAs) are ARM-based Sharp Zaurus devices. For every run type, a total of ten repetitions have been performed.

6.3.1 Validation of the AD Metric

To validate our metric, we have performed an experiment with deliberately provoked interference. For this, the laptops 52-55 where distributed on our office floor as shown in Figure 6.6. The microwave displayed in the graph operates in the same frequency band as our networking hardware and therefore produces some interference when running. A run in this experiment has a duration of 60 seconds. In runs of type A the interferer was switched off while it was switched on in runs of type B. Each node broadcasted 40 packets/s with a size of 100 bytes and recorded all incoming packets. The strongest reaction on the provoked interference can be observed between the nodes 54 and 55, see Figure 6.7. In each second run (runs of type B with interference), the quality for 54→55 is 40 to 70% while it exceeds 95% in type-A runs. In contrast, the opposite direction 55→54 is not affected.

The AD plot for all of the 190 possible comparisons between runs of type A and type B can be found in Figure 6.8. As the links in the two run types behave differently, the 90 comparisons for which the AD metric computes the lowest differences are those

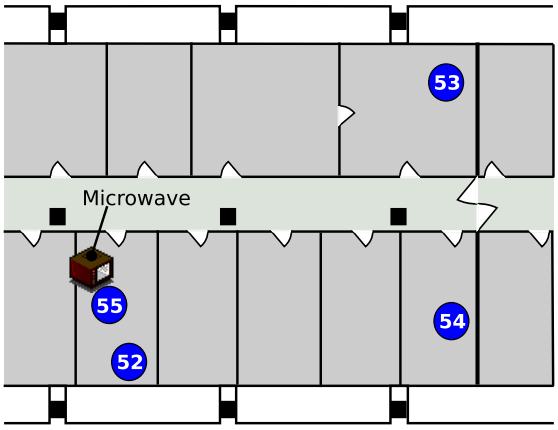


Figure 6.6: Positions in the validation experiment.

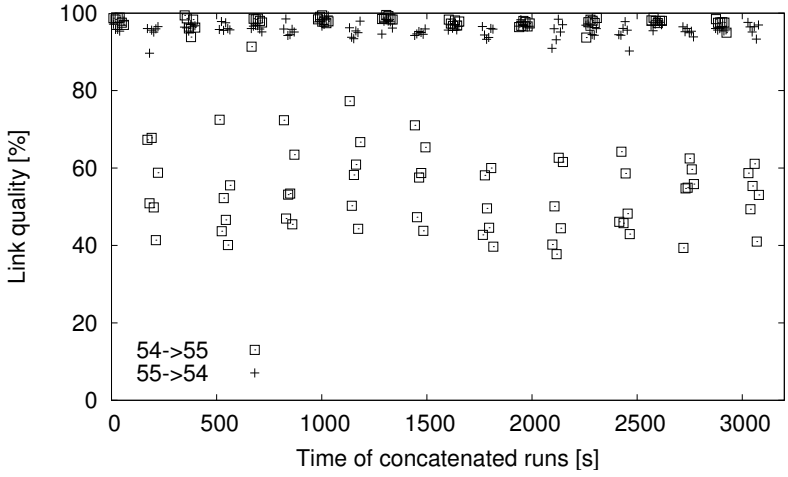


Figure 6.7: Link quality of the links 54→55 and 55→54 in the validation experiment. Each dot marks the average over ten seconds.

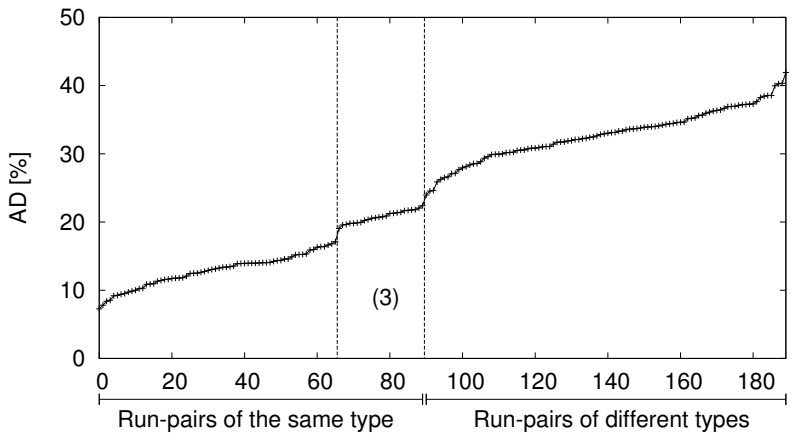


Figure 6.8: AD plot for all runs pairs in the validation experiment.

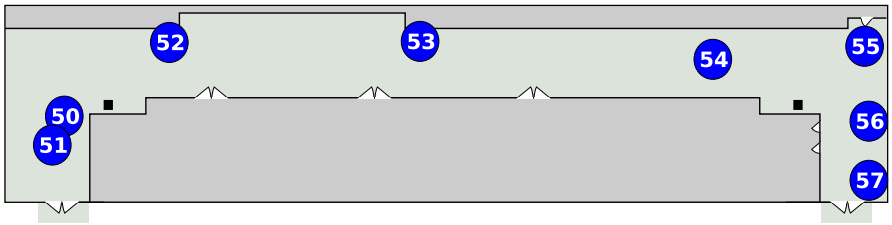


Figure 6.9: Setup in the basement experiment with node 50 as sender.

between runs of the same type (A,A or B,B comparisons). The 100 runs with the highest differences are those of differing type (A,B or B,A). Thus, the metric is able to distinguish runs with and without interference solely based on the information that is available at layer two.

An interesting observation can be made for those comparisons marked in Figure 6.8 with (3). This group has a higher difference compared to other similar-type run pairs. All these comparisons take place between runs of type B, i.e. with interference. An examination of the microwaves' interference pattern with a Wi-Spy spectrum analyzer [WiS] reveals that this interference is not constant. Instead, it periodically switches to frequencies outside the 802.11b band with a period 15-20 seconds. This periodicity is reflected in the link quality and leads to the higher differences if these are not synchronized between the compared runs.

6.3.2 Static Setup with a Single Sender

To evaluate the behavior of single, isolated links, the next experiment has been conducted in the basement of our university. For this area, we verified with the spectrum analyzer that the 2.4 GHz interference was low and that no other 802.11 traffic could be received. As shown in Figure 6.9, we set up seven PDAs and the laptop with the ID 50 as sender. As we use only one sender, the other nodes do not produce packet collisions. The source broadcasts sequences of packets with fixed, different sizes of 50, 100, 500, 1000, and 1400 bytes at a rate of 80 packets/s.

The AD plot for this experiment is shown in Figure 6.10. The values of about half of the possible run comparisons are indeed low and close together but there are also run-pairs that exhibit higher dissimilarity towards the right end of the plot. To localize the source of these differences, the values of the AD metric for the single links have to be examined. A graph of this, sorted similarly to the previous AD plot, can be found in Figure 6.11. This graph shows that there are a number of links that nearly contribute no error at all while some links are responsible for most of the deviation. The links with the highest error contribution are those to the nodes 55 and 56. The reason for this is the bad link quality that varies between 0 and 60% during the experiment. This bad quality is most likely the result of low signal-to-noise ratio that is affected already by the slightest change in environmental conditions. The influence on the similarity between two runs on a link level is shown in Figure 6.12 for the link 50→55. The quality of the link between the runs 0 and 9 strongly varies and therefore negatively affects topological repeatability. The AD plot in Figure 6.10 shows that the runs 0, 1, and 2 are the most different to the right of the visible gap. These are the runs with the lowest repeatability.

Besides such *unstable links*, there are three other classes visible: 1) *perfect links* to the nodes 51 and 52 with an average difference of 0.7% and a link quality of nearly 100% throughout the whole experiment, 2) *intermediate links* to the nodes 53 and 54 with differences of 3.7% and 6.0% and a quality of above 90% for most of the time, and 3) a *zero link* to node 57 that produces no variations as it receives nearly no packets at all.

In a follow-up experiment on our office floor, a similar setup with one node sending out 100 byte packets at a frequency of 50 packets/s and eight receivers was performed. Also in this office environment with more interference, the same link characteristics were found.

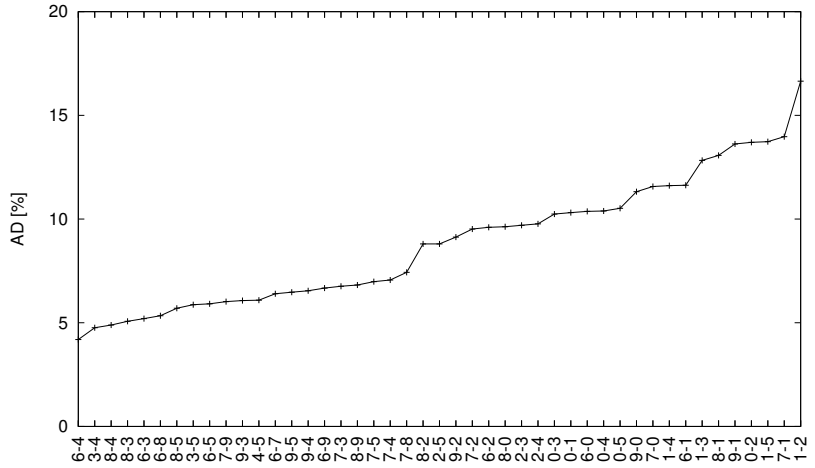


Figure 6.10: AD similarity for the basement experiment.

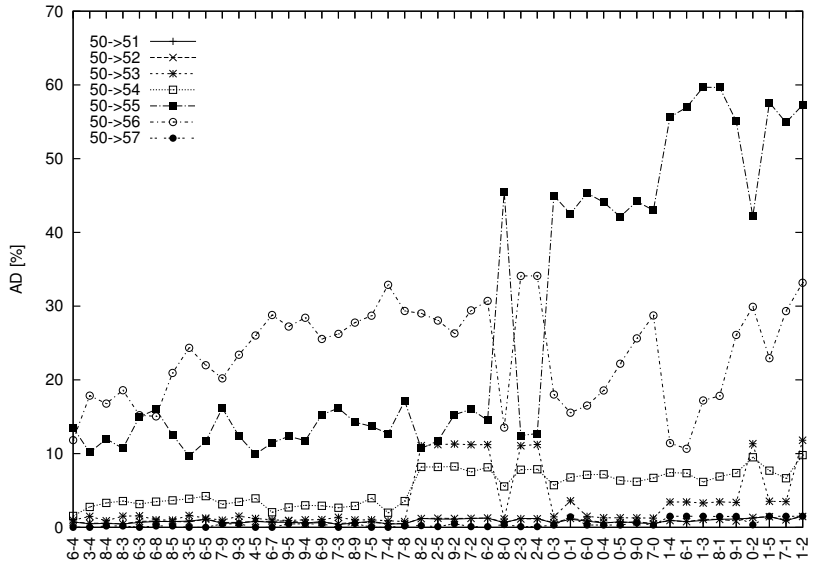


Figure 6.11: All links AD metric for the basement experiment, sorted by overall run similarity.

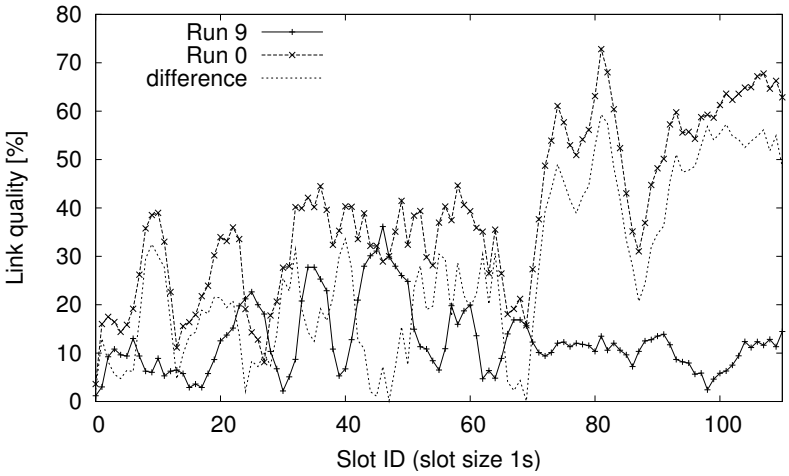


Figure 6.12: Comparison of the link 50→55 in the runs 0 and 9 in the basement experiment with an AD value of (26.9, 17.3).

From these experiments we can thus conclude that the similar behavior of static links between different runs has a direct relation to link quality: if this quality is very high ($> 90\%$) or close to zero, the link will be stable throughout the repetitions while an intermediate quality leads to strong variations and instability. Thus, the number of perfect, intermediate and zero links in relation to unstable links strongly affects overall repeatability.

6.3.3 Mobile Setup in an Office Environment

After examining the behavior in shielded as well as unshielded environments with a single sender, in the next step we move on to runs with mobility and multiple senders. As shown in Figure 6.13, we use the roaming node movement pattern here. The three static nodes 51 to 53 are set up within each others radio range for this and the mobile node 59 follows the movement path indicated by the displayed line. The track is designed such that the mobile node looses connection to the static nodes in the outermost areas. A run has a duration of 160 s. In this setup, each node permanently sends out broadcast packets at a rate of 10 packets per second with a size of 100 byte.

The interesting point here is the repeatability achievable for links between mobile and static nodes. The best link for the direction static→mobile (s→m) is shown in Figure 6.14. The average difference in the link qualities is 3.2% with a standard deviation of 3.9% resulting in an AD value of 7.1%. The similarity for s→m links is in general

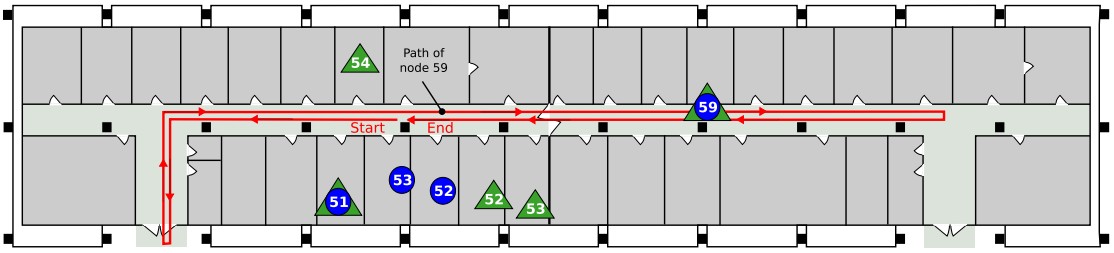


Figure 6.13: Positions and movement in the first two experiments with mobility. The nodes for the first mobile experiment with three static and one mobile node are marked with circles while the movement and positions for the second experiment with two different types of runs are marked with triangles. In both experiments, the mobile node 59 follows the same path indicated by the line in the corridor.

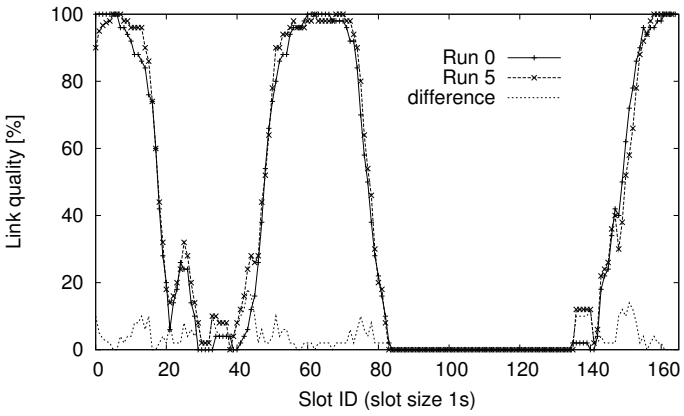


Figure 6.14: The most similar static→mobile link comparison from 51→59 with an AD value of (3.2, 3.9) in the office setup.

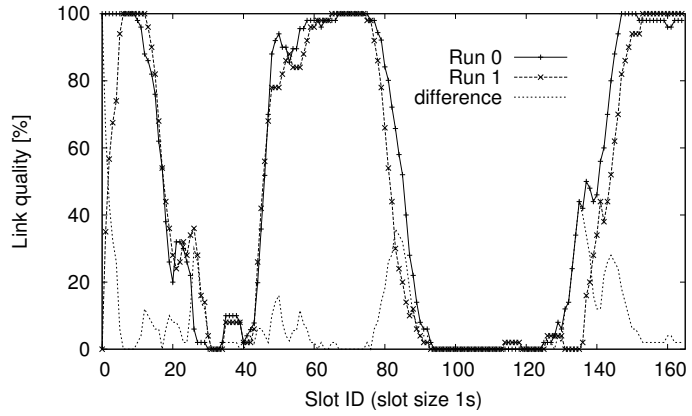


Figure 6.15: The most unsimilar static→mobile link comparison from 53→59 with an AD value of (8.4, 13.3) in the office setup.

rather high: even the s→m link with the biggest difference has an AD value of (8.4, 13.3) = 21.7% as shown in Figure 6.15. In both graphs, the pattern of connected as well as unconnected phases induced by mobility is clearly visible and exhibits a high repeatability for the links in the displayed runs. The highest differences between the curves occur in the transition phases from the connected to the non-connected state and thus also have the biggest contribution to the AD value. As the links here work in the same quality spectrum as the instable links, this is not surprising.

For the current experiment, the overall error contribution of the s→m links is lower than that of all other links. An examination of the connections between static nodes reveals however that these all belong to the classes of intermediate or unstable links. This lower error contribution of s→m connections is thus an artefact of the setup that did not comprise perfect links. Another observation can be made for the m→s links in this experiment. In only 9% of all possible run combinations, these links have a higher similarity than the links in the opposite direction. In the vast majority however, their repeatability is much worse.

Based on these experiments, several general observations can be made: 1) Static links with a high delivery rate (well above 90%) are very stable. 2) Static links with intermediate and low delivery rates are likely to vary during repetitions and strongly contribute to the overall difference. 3) Depending on the quality, mobile links can exhibit a higher stability than static links. 4) Mobile links exhibit their highest differences in the transition phase from connected to non-connected states.

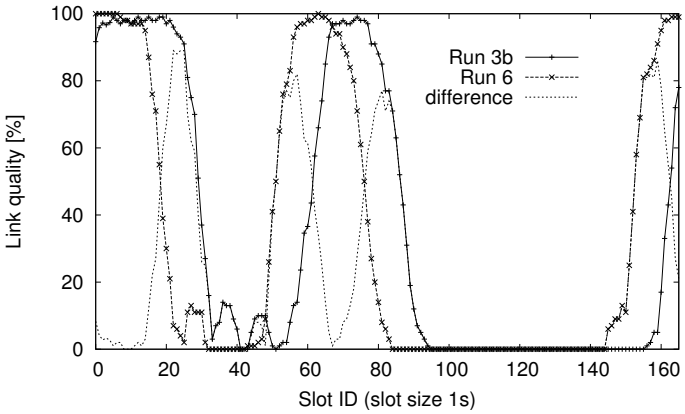


Figure 6.16: Comparison of the link 51↔55 for the runs 3b vs. 6 with different movement patterns. The AD value is (23.4, 29.4).

6.3.4 Mobile Setup with Runs of Different Type

In order to examine the effect of slight topology changes on repeatability, we performed an experiment with two types of runs. For both configurations, the setup consists of four static and one mobile node that follows the same path as in the previous experiment. The detailed setup is also shown in Figure 6.13. In runs of type A, the mobile node 59 directly starts to follow the indicated path when the run begins, whereas this is delayed for ten seconds in type B runs. In both run types, each node broadcasts 20 packets of 100 bytes per second and all other settings are equally similar. Thus, the only difference between the two types is the 10 second delay in starting the movement of the mobile node. We have performed ten runs of both type A and B, each with a duration of 170 s.

The effect of this small change in the mobility pattern on the similarity of mobile links is severe, as can be seen in Figure 6.16. The graph shows the comparison of link 51→55 for the runs 3b and 6a. Similar to the delayed movement, the link-quality curves are shifted by 10 seconds between the different run types, resulting in a large AD value of 52.8%, and thus a large difference. This effect is equally pronounced for all links between mobile and static nodes, see Figure 6.17. It shows the AD values for all these links, sorted according to their average, for all 190 possible run combinations. The combinations on the left of the gap are those for runs of the same type while those to the right with an AD value of well above 40% are combinations of differing type. Thus, already small changes in the movement can have a severe impact on topological repeatability.

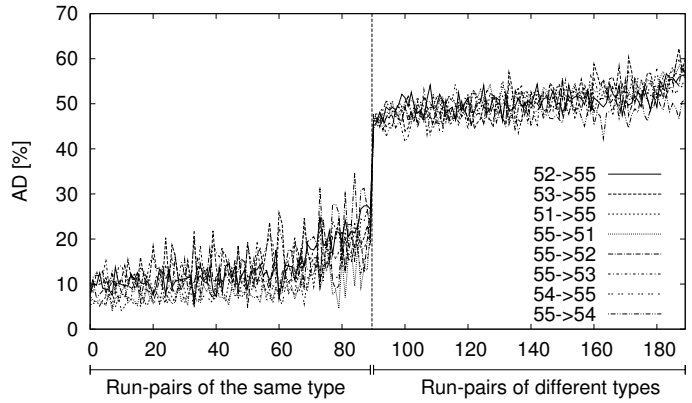


Figure 6.17: The AD values for all links between mobile and static nodes in both directions.

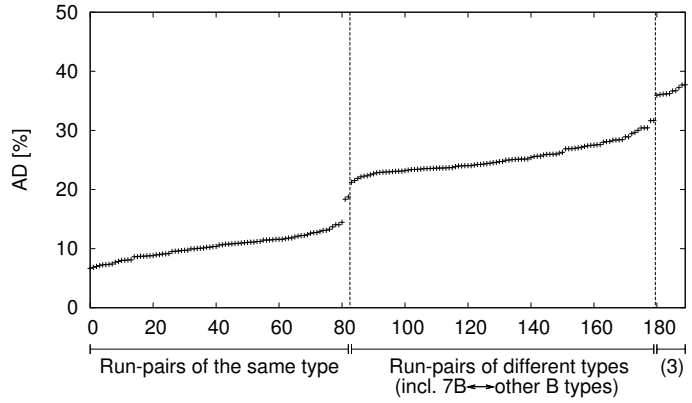


Figure 6.18: Comparison of all possible combinations of type A and type B runs sorted according to the average link difference (AD metric). Part (3) represents runs of type A with run 7B.

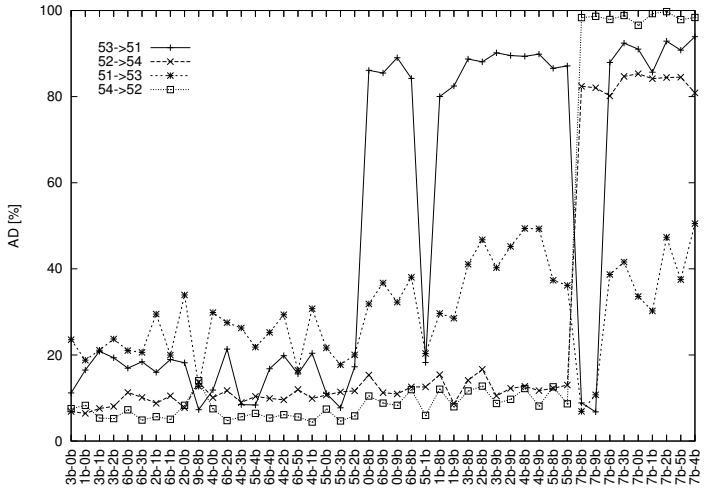


Figure 6.19: The AD values for the links 51↔53 and 52↔54 in all type B runs sorted according to the type-B AD plot.

The AD plot in Figure 6.18 shows that both run types achieve an equal level of repeatability. For the first 81 entries of the graph (read from the left), the similarity values are close together and only exhibit small variations. All of these entries correspond to comparisons of runs of the same type, i.e., either comparisons of type A with type A or B-B comparisons.

With 81 comparisons of similar-type runs, there are a total of nine combinations of the same type missing. In this case, these are all combinations of run 7B with the other type-B runs. Two of these are prominently positioned as the two crosses in the middle of the gap. The others are intermixed with the A-B/B-A comparisons to the right of the gap. For this diversion the links 51↔53 and 52↔54, all between static nodes, are responsible. The AD values of these links in type-B runs are shown in Figure 6.19 that is sorted according to the overall AD value of the run pairs. Also here, the run 7B has the lowest similarity, the comparisons containing this run are all at the rightmost end of the plot. Out of all these comparisons, the links 51↔53 still have rather low AD values for 7B-8B and 7B-9B. As in the basement experiment, three runs should be removed due to their high dissimilarity. If 7B, 8B, and 9B are left out, the overall range of the AD values for the type B comparison drops from $[6.6, 24.8]$ to $[6.6, 11.6]$.

The influence of this dissimilarity is also reflected in the A-B/B-A comparisons in the AD plot for the whole experiment, see Figure 6.18. It is visible as a sharp increase in the displayed value close to the right border of the graph. In this area marked with (3), the

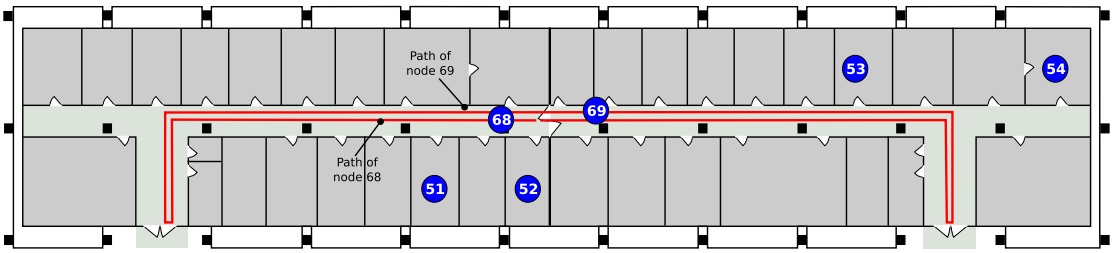


Figure 6.20: Positions and movement in the experiment with two mobile nodes. The mobile nodes move along the line in the corridor and start in different directions.

comparisons of run 7B with all possible runs of type A are located. Thus, the diversion of the different movement pattern and that produced by the differing links adds up.

The conclusions that can be drawn from this experiment are as follows: 1) Although the movement is delayed in type-B runs, they achieve equal similarity estimates as runs of type A and vice versa. Thus the topological similarity for runs with small variations achieves equal levels. 2) However, these small mobility variations strongly affect repeatability. 3) The AD metric is also sensitive to movement induced topology variations.

6.3.5 Two Mobile Nodes

In the next step, a setup with a second mobile node has been examined. For this, we altered the movement path and increased the distance between the static nodes as shown in Figure 6.20. In this setup, the two mobile nodes follow the same path but in opposite directions. Furthermore, a higher number of packets (100 packets/s) have been transmitted by each node and a run had a duration of 160 s. In addition to the broadcast packets, the nodes 51 and 54 sent out multihop unicast packets to each other, however their transmission did not work correctly due to a non-working MAC address resolution for one of the mobile nodes. Therefore, unicasted packets are examined in the next experiment and we concentrate here on the repeatability of the link between the two mobile nodes based on the broadcast packets.

The similarity of this link between different repetitions can be high as demonstrated in Figure 6.21. It shows the link in direction 69→68 in a run comparison with an AD value of (4.4, 4.0) = 8.4%. The most unsimilar link comparison in the experiment has an AD value of (15.7, 13.2) = 28.9% and is shown in Figure 6.22. Although the phases of

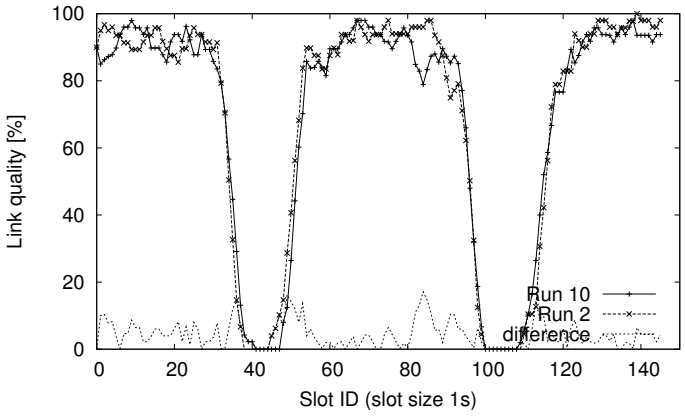


Figure 6.21: The most similar mobile→mobile link, in this case in direction 69→68. The AD value is (4.4, 4.0) here.

disconnection are slightly shifted here and the quality in the connected phases exhibits a higher level of variability, the similarity is still high. This is also reflected by the average AD values for this link that lie between 17% and 20% for type B runs and between 16% and 18% for type A runs.

When examining the different types of links, certain patterns can be observed. The links with the highest similarities all are the static→static links. The only exception to this being 52→51 that fluctuates between 40% and 95% and 52→53 that has a strongly varying quality below 60% most of the time. This underlines the statement in Section 6.3.3 that the low similarity of s→s links was an artefact of the setup there. The m→s links in most cases have a higher similarity than the links in the opposite direction s→m. This stands in contrast to the experiment in Section 6.3.3 where the s→m links had a higher similarity.

6.3.6 Mobility, Unicast and an Application Layer Metric

The experiments presented up to now focused on broadcasted packets and topologies with a maximum of two hops. Now we examine unicast packets in a scenario with four hops that is displayed in Figure 6.23. The four laptops (51-54) are set up similar to the previous experiment and we verified by means of pings that the links 51↔52 and 53↔54 have a high quality and the nodes 52 and 53 were separated until no ping could be transmitted any more. This results in a disconnected chain. The mobile PDA with ID 55 moves from one end of the chain to the other and can act as forwarder between

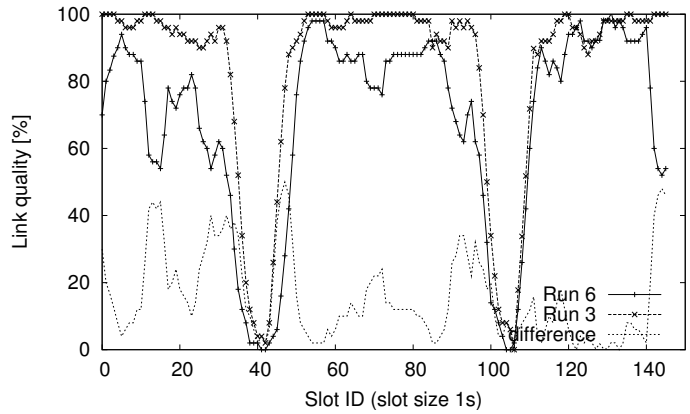


Figure 6.22: The most unsimilar mobile→mobile link, here from 68→69. The AD value is (15.7, 13.2).

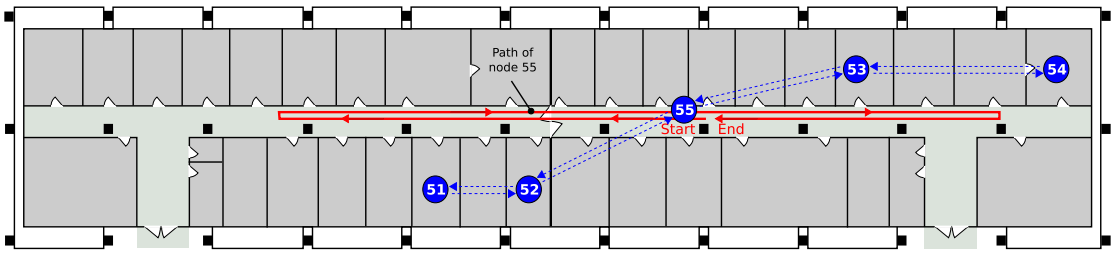


Figure 6.23: Positions and movement of the unicast experiment. The topology consists of four static nodes and one mobile node. Packets are sent from node 51 to 54 and vice versa, the static route is shown by the (blue) dotted arrows. The mobile node moves along the (red) line in the corridor and acts as a connection between the two partitions ({51, 52}, {53, 54})

52 and 53 if in radio range of both. To allow for a communication over four hops, the static route 51↔52↔55↔53↔54 is set up. The nodes 51 and 54 at both ends of the chain transmit 100 byte UDP packets to each other via this route at a frequency of 20 packets/s, a run has a duration of 170 s. Also in this experiment, two types of runs are performed where the mobile node delays its movement until second 10 for runs of type B. In the first half of the experiment, five repetitions of type-A runs are executed and then five runs of type B. In the second half of the experiment, this order is reversed. All nodes trace packets in promiscuous mode, thus they also record unicasted packets that are not directed towards themselves.

In contrast to the first two setups on the office floor, the distances between the static nodes are much longer, as reflected in the link qualities. There is no link between node 54 and the nodes 51 and 52 on the left of the gap, and also for 53→51, the quality is close to zero most of the time. However, the link in the opposite direction, 51→53 changes from a bad to a good quality around the middle of the experiment, and the link 53→52 fluctuates during the whole experiment as shown in Figure 6.24. Furthermore, this behavior is more pronounced for the reverse direction 52→53. Thus, although the nodes were positioned such that no ping traffic could be transmitted over 53↔52, the tracing in promiscuous mode still exhibits instable links here. A plot of the similarity estimates for the links of the mobile node can be found in Figure 6.25. The variability is much higher than in the previous experiment. This is most likely due to the fact that the timespan for which these links operate in the critical range between connected and unconnected state is longer in this setup. As we are interested in the interplay between single links and multihop behavior on the application layer, only links transporting packets along the chain will be considered in the following analysis. The relevant links for the evaluation are: 51↔52, 52↔55, 55↔53, 53↔54.

As metric for the application layer, the delivery rate over time is used, i.e. the percentage of packets sent in a certain second that arrive at the other end of the chain. An example plot of this metric can be found in Figure 6.26 for the runs 0a and 3a that achieve one of the highest AD values in the experiment. The movement of the mobile node is clearly reflected on the application layer: starting in a connected state, the chain is interrupted when the node moves out of 53's radio range. Around second 80, this connection is reestablished. But then the link to 52 is (nearly) interrupted as the node reaches the second turning point. At the end of both runs, the chain is fully connected again. As measure for the similarity of two runs on the application layer, we use the averaged differences between these delivery rate curves. In Figure 6.26, this difference is shown as dotted line, the application layer difference for the displayed runs is 5.7%.

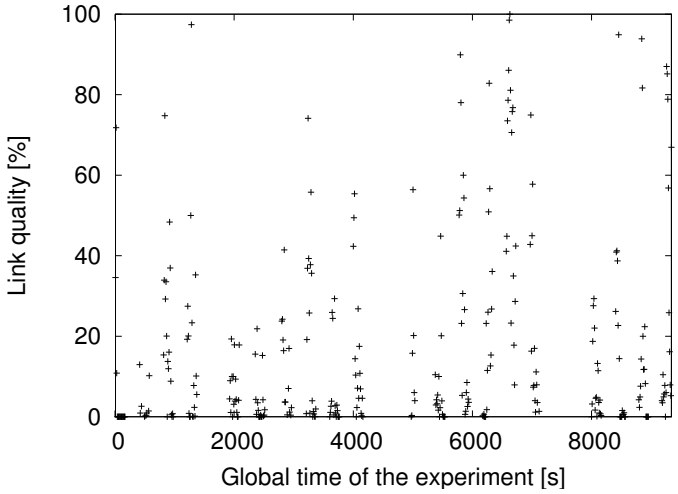


Figure 6.24: Link quality of 53→52 in the unicast experiment. Each point marks the average over 10 seconds, the runs are clearly visible.

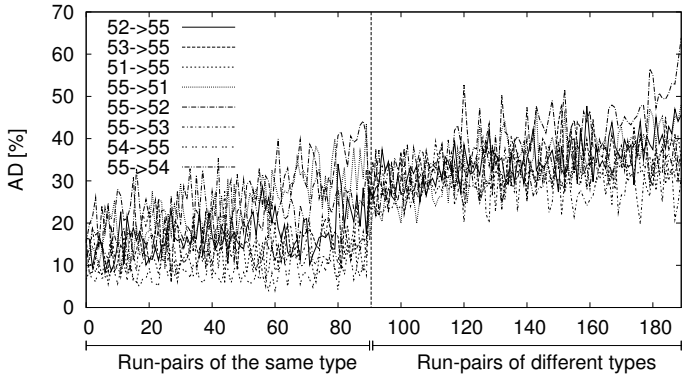


Figure 6.25: AD values for all mobile links in the unicast experiment.

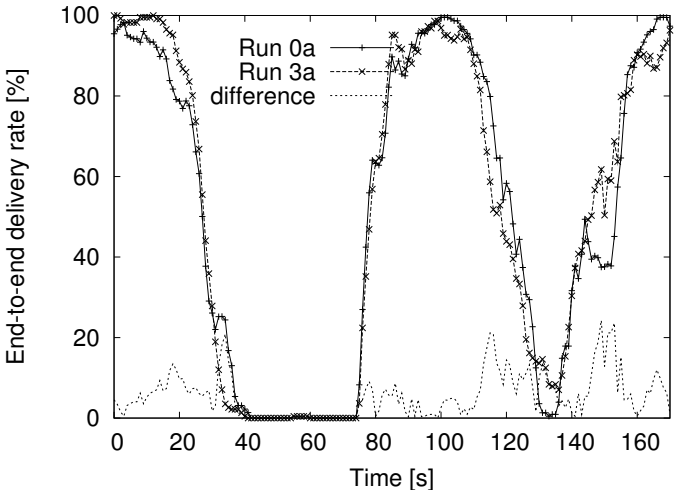


Figure 6.26: Overall end-to-end delivery rate for both directions for the runs 0a and 3a with the difference that is 5.7% on average.

The AD plot for the whole experiment together with the similarity of the end-to-end delivery rate in these run pairs is shown in Figure 6.27. The AD similarity estimate of run combinations of equal type (A-A, B-B) is higher than that of all the combinations where the types differ. Thus, also here the AD metric correctly classifies the runs according to the mobility induced topology differences. However, the transition is not as extreme as in the previous experiment. Especially the quality fluctuations in the mobile link 52↔55 in runs 7a and 9a have a strong influence and their topological similarity with other type-A runs is therefore rather low. As consequence, the seven similar typed comparisons with the highest differences left of the horizontal line in Figure 6.27 all include either run 7a or run 9a.

The application layer metric shows a strong relation to the topology. In runs of type B in which the movement is delayed, the delivery rate curves are also shifted in time. Nevertheless, as multiple links are involved, their differences amplify, resulting in high variations between the different runs, as shown in Figure 6.27. In comparison to the similarity estimate of the topology, the curve for the application layer displays high fluctuations. In spite of these variations, a general trend is visible. The delivery rate comparison shows small differences for similar topologies (the values on the left of the plot) and tends towards greater differences if also the topologies are more different. This visual impression is reflected in the correlation between the two displayed metrics that has a value of 0.82 here. Considering topological similarity in this experiment thus allows for the identification of those run pairs in which the application layer metric

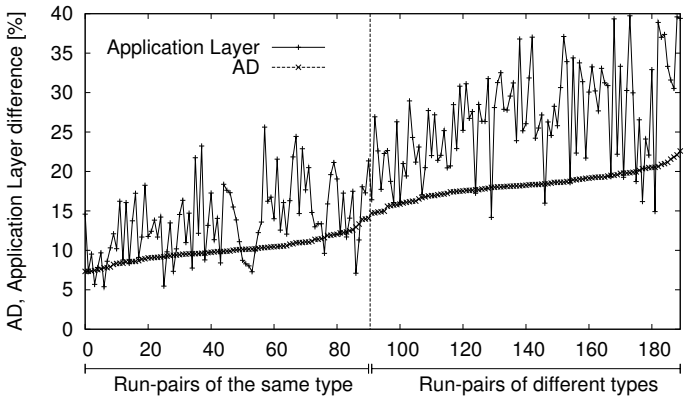


Figure 6.27: The run comparisons in the unicast experiment sorted according to the AD metric, only relevant links. The correlation between the AD metric and the application layer metric is 0.82.

variations are the effect of topology changes.

6.4 Chapter Summary

In this chapter, we have examined the repeatability of experiments with wireless single- and multihop networks. For this, the AD metric is used that quantifies the similarity of static and mobile network topologies based on layer two information. We have shown that this metric produces good similarity estimates and is sensitive to both interference and mobility induced topology variations. With this metric it is thus possible to identify runs in which external factors distort the topology. We discovered that certain classes of links show different behavior with respect to repeatability and found a relation between topological similarity and similarity on the application layer.

As the WMN community moves from simulative evaluation to tests with real hardware in realistic environments, experimental methodology becomes increasingly important. One of the cornerstones thereof is repeatability as it is a prerequisite of valid and sound experiments. We believe that the presented metric is an essential tool for the validation of experimental repeatability and that it can also be a starting point for the examination of reproducibility.

Chapter 7

Conclusion

In this thesis, techniques for the experimental evaluation of wireless multihop networks have been developed. The focus has been laid on solving the practical problems that up to now prevented the systematic evaluation of algorithms and protocols in realistic settings.

To this end, existing techniques and approaches have been examined in Chapter 2. These have been complemented by our own experiences made during the experimental study of ring flooding and the development of a toolkit to create a radio propagation map of an experimental area. Based on these experiences, the features of an ideal testbed were derived.

Chapter 3 presented our own testbed EXC that is purely software-based and follows the new approach of semi-automatic controlled experiments. We have shown how EXC can be adapted for a new experiment and discussed the different ways in which it supports experiments and can be used to discover errors.

Due to their inherently distributed nature, computer network experiments suffer from the imprecise clocks of the participating devices. In Chapter 4, a number of solutions to this problem have been presented. For the synchronization of clocks during the experiment and thus for a better coordination, NTP skew correction has been proposed and experimentally evaluated. This purely software-based approach improves clock precision by two orders of magnitude. However, for the investigation of metrics like round-trip time, this precision is still too low. Therefore we created MLE timestamp synchronization. It uses a mathematical model of the devices' clocks and estimates the parameters of this model based on anchors, i.e. packets received by multiple nodes at the same moment. We have presented an analytical assessment of the synchronization error, performed numerical experiments that show the robustness of the synchronization for

cases in which the underlying assumptions do not hold and complemented this evaluation with a number of real-world experiments. These show that the synchronization error for standard soft- and hardware is likely in the order of a few tens of microseconds. Furthermore, we also derived a second solution to this synchronization problem for which we can proof the error bounds. However, numerical comparisons indicate that the MLE approach achieves a higher synchronization precision.

In Chapter 5, the analysis of WMN experiments was examined and the extensible data analysis toolkit EDAT has been presented. It follows a flow-based analysis approach in which existing components can be combined to an analysis. It is shown in a case study how EDAT can be used to produce graphs that are directly usable in scientific publications.

The different tools and methodologies created throughout this thesis are then combined in a study on the repeatability of WMN experiments in Chapter 6. We discussed the AD metric that assesses the level of topological repeatability and performed a large set of experiments with the help of EXC. This study shows that repeatability on a topological level can be achieved if all controllable parameters are controlled and that runs where external factors distorted the topology can be identified with the AD metric.

We consider the tools, methods and algorithms created during this thesis as our main contribution to the field of the experimental evaluation of wireless multihop networks. The resulting implementations will therefore be made publically available on our institutes' website under an appropriate open source license. This allows other researchers to freely use and adapt these tools for their own evaluations and thus to profit from our work. We will also support others in setting up their own EXC-based testbeds and using the tools.

Interest in the experimental evaluation of WMN technology also increases due to its potential in car-to-car communication. This is best documented by the large amounts of money that soon will be invested in this area and by the large experiments planned for the next few years. Among them is a field operational test on a European scale as well as the German SIM-TD project by a consortium of all major car manufacturers involving a total of 500 equipped cars. These projects profit from the knowledge acquired throughout this thesis; the examination about repeatability and the ability to relate events with sub-millisecond precision are especially obvious assets here. In this context, a number of research challenges become clear, among them the time-synchronization of long-running experiments and the extension of the repeatability examination to setups where cars move in real traffic. As preparation for these experiments, we are currently

setting up an EXC-based testbed for inter-vehicular communication in cooperation with the Volkswagen AG in Wolfsburg. The goal of this effort is to perform tests with up to ten cars on real roads. At the time of writing, initial tests with a one-car prototype system are performed and EXC has already been extended to support UMTS-based out-of-band monitoring.

During our examination of repeatability, it became obvious that the second big challenge is *reproducibility*, the ability to reproduce and verify the result of an experiment both in another environment as well as by other researchers. On the one hand, the question here is on what level results can be reproduced due to the chaotic nature of electromagnetic wave propagation. On the other hand, the necessary techniques for the description of an experiment need to be developed, i.e. an experiment must be described with enough detail. We believe that considering level two information as proposed for repeatability verification can contribute to this effort; however, it does not seem to be enough for achieving reproducibility.

Although we advocated throughout the whole thesis for experimentally evaluating WMNs, ironically simulations will profit from this work. The need for experimentation is mainly a result of the insufficient modeling of real-world behavior in simulations, which in turn results from a lack of understandable, sound experiments. As a consequence of this thesis, we would like to see the detailed calibration of simulation tools with the results from realistic measurements, and finally the creation and spread of a closed-loop evaluation approach that encompasses simulation, emulation, and real-world experimentation.

Appendix

Appendix A

Overview of Existing WMN Experiments

The following sections give an overview over the WMN experiments examined to derive the guidebook presented in Chapter 2. It covers work published up to the creation of the corresponding book chapter [KM07b] and also lists the knowledge acquired through these experiments.

A.1 Historical Overview

Research on multihop wireless networks (which were initially called packet radio networks) started in the early 1970's. The ALOHA [Abr70] project at the University of Hawaii was among the first demonstrations of feasibility for using packet broadcasting in a single-hop system. Based on the knowledge acquired through ALOHA, the DARPA funded PRNET project [KGBK78, JT87] was started in 1973. PRNET was a multihop Packet Radio NETwork system that reached a size of around 50 nodes and allowed some nodes to be mobile. It contained features still present in todays MANETs, e.g. a routing protocol employing mechanisms that are currently used by DSR and AODV. Other PRNET features were the remote debugging capability and the ability to remotely load code to the nodes. The PRNET was in daily experimental use for at least ten years [JT87]. An in-depth discussion of the packet radio network technology in the early to mid 80's with a special focus on findings of the PRNET project can be found in the *Proceedings of the IEEE, Special Issue on packet radio networks* [LNT87].

The follow-up project of PRNET was SURAN (SURvivable Adaptive Networks, 1983-1990) [Bey90] which had the goal to develop techniques enabling the operation of a packet radio network in the presence of electronic counter measures. For the SURAN project, a number of routing algorithms, an in-lab emulator and a real-world demonstrator based on custom made hardware were developed. For the demonstrator, a total of

180 custom made nodes were produced, the largest experiment with the demonstrator involved 22 nodes (some fixed, some car mounted, one airborne). The protocols developed for the SURAN project were intended for large networks with up to 10000 nodes but these large-scale settings were not evaluated in real-world tests. The knowledge acquired during SURAN was used by the US army to enhance existing radios with packet switching capability. A survey on further MANET projects and experiments conducted by the military can be found in [Per01].

With the broad availability of WLAN hardware and small-scale, low-cost portable devices in the late 90's, interest in MANETs increased dramatically. In the following we focus on results from that time up to now.

A.2 Wireless Sensor Networks

Wireless sensor networks consist of small, low-power, low-energy (stationary) nodes used for monitoring parameters such as temperature, humidity, and motion. Algorithms and protocols for these networks often focus on energy conservation and techniques for data aggregation. Nevertheless, sensor networks are wireless multihop networks and therefore face the same fundamental problems.

In [ZG03] a sensor network consisting of up to 60 nodes was used to measure the packet delivery rate with respect to distance and time in office and outdoor environments. The authors identified an area in the shape of a ring close to the maximum transmission range they call *gray area*. This area covered 20-30% of the radio range. In this gray area, packet reception was possible but the packet loss rate had a high variance both in time and space: loss rate varied between 10% and 50%.

The behavior of basic flooding (i.e., every node rebroadcasts each packet exactly once) has been examined in detail in [GKW+02]. The authors present a sensor network with over 150 nodes deployed in a dense grid topology. Contrary to expectations, some of the nodes did not receive the flooded packet. Furthermore, a treelike representation of the flooding process reveals that the packet was received by some nodes via *backward links* (i.e., by nodes farther away from the source than themselves) or via *long links* (i.e., the packet is received over a distance longer than the assumed radio range). In a MANET setting, this behavior could have severe impact on routing protocols such as AODV or DSR which rely on flooding as a means to find nodes and discover routes.

The authors of [YCK+02] present a sensor network with up to 91 nodes intended to collect votes from congress participants. The measurements were performed in a laboratory environment on a grid topology. Routing was performed with a single-destination variant of DSDV: the network's traffic sink regularly flooded route requests to the network. With this, the nodes were able to select the best next hop to the sink based on a link quality metric. The authors discovered significant end-to-end loss rates over multiple hops. They implemented passive acknowledgments (i.e., a node retransmits a packet if it does not hear the same packet being forwarded by its downstream neighbor) to reduce the losses. In a small setup with 24 nodes, the passive acknowledgments decreased the loss rate. However, in larger experiments with 48 and 91 nodes the loss rate increased. The authors conclude that congestion caused by duplicate packets was responsible for this.

In [HKS+04], a 70 node sensor network intended for the tracking and detection of vehicles is presented. The authors discovered that asymmetric links can lead to instable reception rates and that an initial idea to overcome this, link layer handshaking, is expensive. Therefore, they used a different method to avoid asymmetric links. During the creation of a diffusion tree, spanning all nodes, the packets which allow the nodes to detect their parents were sent with a lower transmission power. Thus, the nodes selected parents which are close. The resulting link is therefore symmetric with a high probability.

In [CBE03], experiments with up to 55 sensor nodes are performed to determine the radio characteristics of one indoor and two outdoor environments. The gray areas here span 50% to 80% of the radio range. Furthermore, 5% to 30% of the links were found to be asymmetric. The authors also present evidence that asymmetric links may be caused by differences in hardware calibration: when the positions of two nodes connected by such a link were swapped, the link asymmetry was inverted in 91% of the tested cases.

One of the largest sensor networks has been deployed in the context of the ExScal project [Aea05] for intruder detection. The network consisted of more than 1000 sensor nodes and about 200 IEEE 802.11b nodes that served as backbone network, creating a 2-tier network structure. Each of the backbone nodes was responsible to relay the messages of a certain number of sensor nodes to a base station. The authors discovered that the distance vector protocol initially used transported only 33.7% of the sensor nodes' messages to the next backbone node. Therefore, the authors switched to LGR (Logical Grid Routing). LGR selects the routes according to a spanning tree that is

computed during a setup phase. In combination with a custom transport protocol, 99% of the sensor nodes' packets could be delivered to the next backbone node.

Summarizing, the experiments done with communication in sensor networks show that physical layer effects must be considered when building multihop wireless networks. The findings on gray areas, asymmetric links and congestion are particularly interesting, since the number of nodes used in the experiments was comparatively high.

A.3 Mesh Networks

The most mature wireless multihop networks with respect to real-world deployment are mesh networks. Mesh networks are composed of stationary nodes equipped with radio hardware and connected to the power supply system. Commonly, their aim is to provide multihop access to the Internet.

In the MIT Roofnet project [ROO], two scientific mesh networks with up to 29 (indoor) and 38 (outdoor) nodes have been deployed. On the 29-node indoor network, the properties of links between 802.11b-equipped nodes were evaluated [CABM03]: out of 124 existing links between the nodes, there were 28 links where forward and reverse delivery ratios differed by at least 25%. Furthermore, the impact of different packet sizes on the delivery ratio of single-hop transmissions has been evaluated and it could be shown that larger packets have a much lower probability of being delivered than smaller ones. It is also shown that a purely hop-count based selection of end-to-end routes often results in suboptimal routes [DACM03, CABM03]. Therefore, the outdoor Roofnet network uses Srcr [BABM04] as routing protocol, a variant of DSR modified to find routes with high throughput. In [ABB+04] the links in the outdoor network are examined by letting each node send a number of 1500 byte packets. The authors show that it is difficult to strictly distinguish between neighbors and non-neighbors as there are a lot of links with intermediate or high loss rates. Signal-to-noise ratio and distance exhibit only a weak correlation to the delivery rate and experiments with a physical layer emulator [JS04] reveal that multipath fading may be responsible for these loss rates. The end-to-end performance of Roofnet (here with 37 nodes) is evaluated in [BABM04]. Lossy high throughput links seem to be a good choice in multihop paths as this provides better overall throughput than high quality links with a low bandwidth. Furthermore, short-distance, high-throughput links are preferable to long-distance, low-throughput links in this respect.

In [RPD+05] a three-node multi-radio mesh network is studied. As nodes, Linux workstations with up to four 802.11b network interfaces are used. It is shown that the throughput is reduced by up to 33% if more than two network interfaces are installed in one node (one interface transmits, the other interfaces are only switched to a passive state). The authors suspect that radiation leaking from the passive cards and board crosstalk is responsible for this. Then a two-hop experiment is described in which each hop can be performed on a different channel as the middle node uses two network interfaces. It is discovered that it is not possible to operate the two network interfaces at full capacity regardless of the channel used. This became feasible with a minimum antenna separation of at least 35 db (corresponding to 1 m of antenna separation).

The experiments presented in [DPZ04a] study the impact of four different link-quality metrics on overall end-to-end TCP throughput in a 23-node indoor mesh network. The two most important of those metrics are hop count and the Expected Transmission Count (ETX) metric. ETX uses single-hop broadcasts to determine per hop loss rates and, based on this information, calculates the path with the lowest overall number of (re-)transmissions. The nodes in the examined network are Windows XP machines equipped with 802.11a radios. Routing is performed with a variant of DSR adapted to the corresponding metric. Initial baseline measurements reveal the existence of asymmetric links in the network: for about 50% of the one-hop links with two directions, the reverse and forward bandwidth differs by more than 25%[1]. In the vast majority of the presented measurements, the ETX metric achieves the best throughput, followed by the hop count metric and the other two metrics. The only exception to this is a TCP throughput measurement from a single mobile node carried around the periphery of the network to one of the static nodes. Here, the hop count metric exhibits a better performance as ETX does not adapt fast enough to link quality changes. The testbed was modified for a follow-up experiment [DPZ04b] to support two network interfaces per node. Initial baseline measurements on the testbed with single-hop transmissions reveal that two radios of the same 802.11 dialect (a/b/g) in one node interfere with each other regardless of the used channel. Similar to the measurements presented above [RPD+05], throughput drops significantly due to this.

Apart from scientific approaches, there are also private mesh networks that mainly serve as access networks to the Internet. Examples include the efforts to cover the Dutch city of Leiden [wirb] or the city of Melbourne, Australia [MEL] with a mesh network. Further information on mesh network implementations can be found at [WIKa, BCG05].

[1] The radios were allowed to dynamically select their data rate.

The experiments performed on mesh networks are of particular interest since most of these networks are in real-world use. Thus very practical issues such as routing metrics and routing stability is investigated under realistic constraints. As a result the findings are also useful when considering MANETs.

A.4 Mobile Ad-Hoc Networks

A.4.1 DSR at Carnegie Mellon University, Pittsburgh

The work on the DSR prototype [MBJ99, MBJ00] started in 1998 at the Carnegie Mellon University. It comprised five mobile nodes installed in cars moving at top speeds of 40 km/h, a mobile node connected via mobile IP and two stationary nodes installed 671 m apart at opposite ends of the course traveled by the mobile nodes. The nodes were equipped with 900 MHz WaveLAN-I radios with a nominal range of 250 m and GPS for tracking purposes, routing was performed with DSR. To overcome the missing link layer acknowledgments of the WaveLAN-I radios, acknowledgments on the routing layer were implemented, lowering the per-hop loss rate from 11% to 5% by means of retransmissions.

The designers of the DSR prototype identify several tools and utilities which have proven to be valuable for the analysis and debugging of the prototype [MBJ99]:

- a GPS receiver at each mobile node enabling the tracking of individual nodes
- a visualization tool that displays the status of the nodes and allows a birds view on the experiment
- tcpdump to track all packets for a detailed post-run analysis[2]
- a per-packet signal-strength recording
- a per-packet state-tracing, recording the internal states of the used protocols, namely TCP and DSR
- a MAC filter for emulating movement without actually moving the nodes

[2] The authors emphasize that the additional processing time due to the usage of tcpdump has influenced the results of some experiments as this delayed acknowledgments.

In an initial test of the DSR prototype ping packets were sent from the first stationary node to the second stationary node via the five nodes circling between them. With a loss rate of about 5% for the first hop, the overall end-to-end loss rate is reported to be 10%. About 90% of the packets used two and three-hop routes. Due to the variability in the environment, roughly 10% of the ping packets were exchanged directly between the two nodes over a distance of 671 m producing a loss rate of 22.3%.

During the evaluation of a TCP transfer in a static two-hop scenario [MBJ99, MBJ00], *fluctuating links* with a range longer than the specified radio range, a poor quality and a high variance both in time and space occurred. These links led to poor performance in the evaluated scenario: three nodes were set up in a chain topology, with the two outer nodes being positioned such that they were as far away from the middle node as possible but still able to successfully transmit ping packets to the middle node. Temporarily, the two outer nodes were able to communicate directly leading to a significant amount of packet loss. The use of a macfilter prohibiting the use of this one-hop route improved the throughput by 30%. Therefore, the authors emphasize the necessity of a mechanism to prevent the use of fluctuating links.

The authors of [MBJ99] also mention some additional general lessons learned:

- packets controlling the routing protocol should be delivered with high priority (e.g. by implementing multi-level priority queues)
- management of human experiment participants is difficult and time consuming
- wireless signal propagation is highly variable

The DSR prototype implementation was extended to support real-time traffic such as audio and video [HJ02]. In a network consisting of one mobile and seven fixed nodes with 802.11 Lucent WaveLAN adapters the mobile node transmitted an audio and a video stream over up to three hops to one of the fixed nodes. The experiment showed that the transmission of real-time traffic over an ad-hoc network is possible if the routing protocol is adapted to the specific scenario.

A.4.2 AODV/DSDV at Sydney Networks and Communications Lab

An experiment conducted with implementations of AODV and DSDV is described in [CJWK02]. Two routing protocols were tested in a scenario with four fixed and one mobile node and the roaming node movement pattern. For the experiment Linux PCs and laptops with 802.11b adapters were used. The maximum transmission rate was

limited to 1 Mb/s to avoid automatic rate changes by the 802.11b adapters. Furthermore, the adapters were wrapped with metallic anti-static bags to limit the transmission range to 5 m thus allowing in-lab testing[3]. Two tests were performed in this setup, sending UDP packets from the mobile node to one of the fixed nodes at the end of the chain and transferring a file with FTP in the other direction. It was discovered that both routing protocols frequently selected very unreliable links which resulted in poor performance. The reason for this problem was that both routing protocols prefer routes with a low hop count. Implicitly this leads to a preference for unreliable long range links. The DSDV implementation did not suffer as much as AODV as it used a handshake before accepting a link. To overcome the unreliable links, the *powerwave* tool was implemented as a sub-layer below the routing layer: nodes regularly exchange echo packets with each other to filter those links with a bad signal-to-noise ratio. The authors have detected two shortcomings in their tool: the high network load due to echo packets and the insufficient interaction with the routing protocols as they are not informed about link breakage but have to detect this situation by employing their own timers.

A.4.3 Centibots at Artificial Intelligence Center SRI International, Menlo Park / University of Washington, Seattle / Stanford University

One of the largest MANETs was deployed within the scope of the Centibots project [KOV+02]. The goal of the project was to deploy a team consisting of 100 autonomous robots for the surveying of an indoor area. The robots used 802.11b network interfaces, routing was performed with TBRPF, a pro-active link-state routing protocol. The largest number of robots running at the same time were 72 with a maximum route length of five hops and a throughput of about 1 Mb/s [Vin04]. The robots were moving at 30 cm/s in an area of 650 m^2. When the experimenters tried to run all robots at once, the network broke down. The problem was solved by bringing 10 to 18 nodes up at a time. The final reason for this problem was not fully identified, the experimenters name three potential sources: the network interfaces, TBRPF and the TBRPF implementation they were using.

A.4.4 GPSR at University of Mannheim

The Fleetnet Router [HFMF03, Mös03, MFHF04] implements the greedy forwarding strategy of the position-based routing protocol GPSR, i.e. a node selects the neighbor

[3] Due to the different transmission range, the AODV timers had to be adapted.

closest to the target as next hop. The target's position is discovered by flooding a position request. On reception of the request, the target sends a reply containing its position. Nodes are installed in cars and have the following components: a Windows-based application PC, a Linux-based 802.11b router, onboard GPS and GPRS to monitor the internal state of the node. Furthermore, packets received from nodes farther away than 220 m are dropped to avoid the fluctuating link problem. In a static three-hop experiment with the Fleetnet router [MFHF04], it was discovered that the maximum achievable throughput of 400 Kb/s depends on the size of the packets as smaller packets lead to more collisions. In the same setup with mobile nodes, it has become evident that unacknowledged broadcasts are often lost. Thus, flooding used to discover the target's position took a long time to reach all nodes. Furthermore, the lack of feedback from the MAC layer about broken links was an issue. In [Mös03], the experimenters evaluated the router in a static three-hop setup with laptops without the cars and the application PC and found some additional problems. During one of the test runs, a bursty loss occurred blocking nearly all packets. The authors suspect interference and attenuation by large objects between sender and receiver to be responsible for this. Furthermore, high round trip times occurred for the first packet of each test run. This was due to process scheduling of Linux.

A.4.5 Routing protocol evaluation at Dartmouth College / Colorado School of Mines / University of Illinois at Urbana-Champaign / Bucknell University, Lewisburg

In [GKN+04], an experimental comparison of four MANET routing protocols (APRL, AODV, ODMRP and STARA) can be found. The network consisted of 40 laptops equipped with 802.11b cards running at a fixed rate of 2 Mb/s. Nodes had GPS receivers attached to track their position for later emulation and simulation. The nodes flooded beacons with their own position and timestamped positions of other nodes to the whole network. Emulation was performed by placing all nodes in the same room and using packet filtering to emulate a dynamic topology. The experiment itself was conducted on a rectangular athletic field of size 225×365 meters on which the nodes were moved around according to the mobile random movement pattern. The results only take 33 of the 40 nodes into account as seven nodes did not work correctly. The outdoor experiment revealed that the two reactive protocols AODV and ODMRP deliver much more messages than the two proactive protocols. However, even those protocols produced a high overhead and achieved a low absolute delivery rate. The repetition

of the experiment by means of simulation showed large differences to the real experiment [LYN+04]. This has been extended in [LYN+05] to examine how different radio layer models affect the simulation results. A simple stochastic radio propagation model with standard outdoor parameters produces results that are closest to the real experiment while the other models (also those enhanced with the connectivity information from the experiments) differed more.

A.4.6 DSR at Rice University, Houston

In [STP+05] the authors use unmodified DSR routing code from the ns-2 simulator in a real network. In order to achieve this, the code is encapsulated in a user-level process that provides a simulator/real-world packet format converter. Using this technique the ability to handle real-time video traffic over a mobile ad-hoc network is investigated. The network used in the experiments consists of four stationary and two remotely controlled mobile nodes. The communication at each node is performed over 802.11b equipped Linux laptops that use DSR for routing, the mobile nodes have an additional Windows laptop that handles the live video. The average packet delivery ratio during the demonstration is above 95% at an overall latency of about 30 ms, thus validating the presented implementation technique.

A.4.7 DSR at University of Colorado, Boulder

The MANET examined in [JBD+05] is composed of 10 nodes out of which some are mounted on remote-controlled miniature airplanes. The nodes are composed of single board computers equipped with 802.11b network interfaces and GPS, routing is performed with DSR. The authors demonstrate in this work that it is possible to combine airborne and ground nodes in a MANET. They achieve a throughput of about 250 kb/s at a latency of 30 ms over up to three hops.

A.4.8 Ad Hoc Networking with Directional Antennas at BBN Technologies, Cambridge

In [RRS+05], a system for ad-hoc networking with directional antennas is described. The implementation of this system used the same routing code for the real experiment as for the simulation. For routing, the link-state routing protocol HSLS was used. An experiment was conducted with 20 nodes (cars) that drove around a 4×3 km area. Each

car was equipped with four directional antennas selectable on a packet-per-packet base and 802.11b as physical layer. According to the authors, their system outperformed a similar setup (20 cars but with omnidirectional antennas and OLSR) although more details on this are not available. In a second experiment, a helicopter was added as aerial node.

A.4.9 Signal strength aware OLSR at Stanford University, Robert Bosch Research and Technology Center, Palo Alto, University of Cincinnati

In [SBSC03], the authors present and experimentally evaluate a signal strength aware modification of OLSR, SBRS-OLSR. This OLSR variation estimates, based on the rate of change of signal strength between two nodes, how long these nodes will be probably able to communicate. The experiments are based on the roaming node movement pattern and use four 802.11b-equipped nodes. The mobile node is installed in a car and moves around a building at 15 mph. Then UDP packets are sent from one of the stationary nodes to the mobile node. By analyzing the throughput over time, the authors show that SBRS-OLSR is more responsive to topology changes than OLSR. The authors conclude that this is due to the fact that SBRS-OLSR selects another next hop before the route breaks.

A.4.10 OLSR at INRIA Rocquencourt, CELAR Bruz

The authors of [PAM+05] describe their experiments with OLSR on the CELAR MANET platform. This is a military test network that consists of a total of 18 nodes equipped with 802.11b interfaces out of which some nodes are mobile within vehicles. The OLSR version used implements a link hysteresis mechanism with two predefined signal strength thresholds: a node is accepted as neighbor if the signal strength is higher than the high threshold and removed from the neighbor list if the signal strength drops below the low threshold. The authors start with examining TCP flows over ten randomly placed static nodes. They show that the throughput of concurrent flows is very unreliable: 1) A one-hop vs. a three-hop flow captures nearly the whole bandwidth. 2) Two concurrent three-hop flows share the bandwidth fairly while the first of two concurrent two-hop flows gets nearly no bandwidth at all. They furthermore show that for concurrent UDP and TCP traffic, UDP is highly favored as TCP reduces its data rate due to its congestion control.

In the mobile experiment shown, the roaming node movement pattern is used with ten static and one mobile node mounted in a vehicle. One of the static nodes sends TCP and UDP traffic to the mobile node. The presented plots show that there is a network connectivity problem lasting twelve seconds in which a neighbor to the mobile node is only shortly available with a bad link quality, furthermore there are some short network outages in this run.

A.4.11 TCP/AODV at University of Calgary

The authors of [GWW04] evaluate the performance of TCP in combination with a signal-strength aware variant of AODV. Furthermore, they also examine TCP rate-based pacing (RBP TCP), a mechanism that reduces the burstiness of TCP traffic by artificially introducing inter-packet delay to improve multihop performance. The signal-strength aware variation of AODV is intended to provide more stable routes. For this, AODV-UU is modified to accept control packets only if their signal strength is above a certain threshold.

A number of indoor experiments with 802.11b equipped laptops running Linux are performed. In a setup with five nodes and a roaming node movement pattern, the overall throughput of TCP from the mobile node to the first static node in a chain is examined. The authors find out that the throughput degrades with higher node speeds and conclude that there are two reasons for that: 1) The faster the client, the lower the time spent in the proximity of the server where the throughput is high due to fewer hops 2) The time spent for route discovery in which no data traffic can be transfered has a higher percentage on the whole transmission time. In a second experiment, they evaluate TCP rate-based pacing on a five node indoor string node setup. They find out that RPB TCP does not perform better than Reno TCP in this setup although prior simulations suggest this. Several potential reasons are identified: 1) Real-world wireless propagation is more challenging than simplified simulation models. 2) The simulation does not take a routing protocol into account. 3) RPB TCP had to be implemented on the TCP layer instead of the link layer.

A.4.12 AODV/OLSR/P2P at Institute for Informatics and Telematics, Pisa

The routing algorithms AODV and OLSR and CrossROAD, a cross-layer peer-to-peer (P2P) system for ad-hoc networks are examined in [BCDG05] in setups with up to eight nodes. These nodes are laptops with 802.11b network interfaces at 11 Mb/s. In the first

part of the experiments, the authors evaluate the performance of AODV and OLSR for the roaming node, end swap and relay swap movement patterns with four nodes and node speeds of about 1 m/s. Interesting is the time it took the two routing protocols to discover new routes in presence of a changing network topology. In the roaming node scenario, OLSR needs five seconds to discover the two-hop path after the one-hop link is broken and ten seconds to discover the three-hop path after the loss of the two-hop path. AODV has a delay of two and seven seconds in the respective situations. In the reverse situation when the routes get shorter due to the movement, both protocols directly find the shorter routes. For the end swap scenario, the highest delay for OLSR are 15 s and 10 s for AODV. The highest delay for the relay swap movement pattern are 15 s for OLSR and 11 s for AODV.

CrossROAD is a cross-layer P2P system that piggybacks its information on the control packet of a proactive routing protocol. The authors show with experiments on a 8-node static ad-hoc network that the control traffic for CrossROAD over OLSR is much lower than for a legacy P2P approach. Furthermore, a 5-node experiment is performed on a chain topology with the middle node starting to move to one end of the chain, creating two network partitions. The authors thus demonstrate that CrossROAD can handle network partitions correctly.

A.4.13 AODV at Uppsala University and Ericsson Research/Switchlab, Stockholm

The indoor experiments presented in [LLN+02] were conducted with 9 to 37 nodes by using the APE testbed. The nodes were divided in at most four independently moving groups which split up and reunited in the course of the experiment according to the chain on the fly movement pattern. The authors ran several experiments with OLSR and AODV. By comparing the virtual mobility graphs of the distinct experiments the authors conclude that their approach of choreographing the movement of the nodes is suitable to produce comparable test runs.

A four node experiment with APE [LNT02] revealed the existence of *communication gray zones* in 802.11b based ad-hoc networks. A node X is said to be in the communication gray zone of a node Y if it is listed in the neighbor table of Y but Y cannot forward any data traffic over X. The reason for this lies in the different reception characteristics of broadcasted beacons used for neighbor discovery and unicast data packets in 802.11b-based ad-hoc networks: 1) 802.11b broadcast packets are normally sent at

a lower bit rate than unicast packets, thus they can be received over greater distances. 2) Broadcast packets are not acknowledged and can thus be transmitted over unidirectional links. 3) The small size of beacons results in fewer packet losses due to bit errors and collisions. 4) Fluctuating links lead to entries in neighbor tables about nodes which are only occasionally reachable. The authors also evaluated the impact of three different strategies to overcome gray zones, exchanging neighbor tables, accepting a neighbor only after the reception of three beacons and discarding beacons received with a low signal quality. They show that all three strategies improve the packet delivery rate significantly.

A.5 Summary of Results

Even though the existing experiences with real-world implementations of mobile ad-hoc networks are quite heterogeneous, there are several observations that can be generalized:

- A lot of available links in a wireless network are asymmetric. This has been shown for sensor networks [CBE03, HKS+04], mesh networks [DACM03, CABM03, DPZ04a] and MANETs [KNG+04].
- In has been shown for sensor networks that the direction of an asymmetric link can be switched by switching the positions of the two affected nodes [CBE03].
- Distance may only exhibit a weak correlation to the packet reception rate. In sensor networks, this is known as *gray areas* [CBE03, ZG03], in MANETs as *fluctuating links* [MBJ99, MBJ00, LNT02, LA03, MFHF04]. The problem has also been verified for mesh networks [ABB+04]. The emulator experiments in [ABB+04] suggest that this may be an effect of multi-path fading. The size of such gray areas depends on the environment [CBE03].
- Even simple flooding does not behave as expected [GKW+02].
- Experiments are time-consuming and expensive [MBJ99, Lun02, LYN+05].
- 802.11 radio interfaces have a circular *gray zone* at the border of the transmission range in which broadcasts can be received but unicasts cannot [LNT02].
- Current simulators are not accurate because the assumptions on which simulators are built are too simple, therefore simulation results can differ significantly from real-world experiments [GKN+04, KNG+04].

- Packet delivery is influenced by the distance of the nodes from the ground [ABC+05].
- If multiple TCP connections from one source or to one sink are present, one-hop connections capture nearly all of the available bandwidth [LA03].
- Switching on all nodes in an ad-hoc network at the same time can overload the network [Vin04].
- Battery power and wireline power supply are a bottleneck during experiments [Lun02].
- Emulation tools like a MAC filter are essential to save time during the preparation of an experiment [MBJ99]. The importance of this is underlined by the number of implementations existing under different names: powerwave [CJWK02], APE mackill [ape], MobiEmu [ZL02], Fleetnet packet suppression mechanism [MFHF04], FRANC virtual networks [CSS03] and the MAC filter used in [HBRB05].
- Every tool should be tested for its influence on the experiment, e.g., tcpdump is reported to consume lots of resources and may have an impact on the performance of the investigated routing protocol [MBJ99].
- Packets for the control of the routing protocol should be delivered with high priority which can be achieved by implementing multi-level priority queues [MBJ99].
- Current 802.11 drivers do not report broken links to upper layers: to use link-layer acknowledgments on higher layers, the driver needs to be patched [HJ02, MFHF04].
- A routing protocol that uses hop count as route metric may select suboptimal routes. In particular this has been investigated for mesh networks [DACM03, CJWK02, CABM03, DPZ04a, DPZ04b] but also shown for sensor networks [YCK+02].
- Two network interfaces of the same type integrated close to each other in one computer interfere regardless of the used channel [DPZ04b, RPD+05].
- [DPZ04a]: "... static and mobile wireless networks can present two very different sets of challenges, and solutions that work well in one setting are not guaranteed to work just as well in another."

Appendix B

Propagation Estimation API

To integrate the results of a Mapkit measurement and the corresponding interpolation in other programs (see Section 2.3), we provide a simple API. It is implemented C/C++ and can be used by including the *analyze.h* header file. The functions available via this file here will be shortly presented.

The function

void readData(char path);*

is used to read the measurement data that serves as basis for the interpolations. The argument is the path to the measurement-file.

With the function

int getQuality(int x1,int y1, int x2, int y2,int algo);

it is then possible to get the link quality in percent between two arbitrary points in the experimentation area. Thus, it returns an *int*-value in the range 0 and 100 between the two points (x_1/y_1) and (x_2/y_2) calculated according to the selected algorithm, refer to Section 2.3 for details. According to our own experiences, the algorithm that considers the two closest measurement points as interpolation input performs best, thus we advice to use $algo = 2$ as input here. In addition to point-to-point link quality, the library is also able to return the quality between a single point and all other points in an area around this point via the function

*int** getQualityAround(int x, int y, int width, int height, int algo);*

The details of the functions that directly return the measured values can be found in the *analyze.h* header file.

Appendix C

EXC Live-CD

C.1 Structure

The EXC Live-CD is based on a Xubuntu 6.10 Live-CD and therefore has a similar structure. It consists of an initial ramdisk (initrd) [INI] and a SquashFS file system [SQU]. The initrd is treated by the Linux kernel like a hard disk partition, it contains the files necessary during startup of the Live-CD and in general has a size of a few megabytes. The SquashFS filesystem contains all other files and programs and has a size of about 500 MB in the case of the EXC Live-CD. For the adaptation of the Live-CD, new files can be installed in the initrd as well as in the SquashFS. As the initrd can be re-compressed within a few seconds due to its small size, often-changed files should be better installed there. In contrast, newly creating the SquashFS can take 10 - 30 minutes.

C.2 Build Script

The *exc-build-scripts* allows to easily adapt an existing EXC Live-CD to new needs. It is based on scripts of the TUn!x [TUN] project that have been extended and combined to a single, menu-controlled script for the Live-CD. Its usage requires root-privileges. The menu looks as follows:

```
please choose next step
[1] make new Live-CD environment
[2] chroot into Live-CD environment
[3] (re)build squash filesystem
[4] (re)build initrd
[5] create new iso image
[6] steps 3 - 5 in a row
[7] steps 4 - 5 in a row
[8] exit to shell
```

`make new Live-CD environment`: is the first step that has to be performed when a Live-CD should be adapted. After selecting this entry, the user has to specify the path to an existing ISO-image of the EXC Live-CD. The SquashFS file system in this image is decompressed in the folder *new/*, the extracted initrd will be stored in the folder *init/*. Furthermore, the script generates the folder *live-cd/* that contains a compressed version of the SquashFS and further files that are not important for the adaptation of the Live-CD. All other steps of the adaptation process work on this *new/* and *init/* folders.

`chroot into Live-CD environment`: this entry allows to log in to the Live-CD via `chroot` with root-privileges. Besides changing configuration files, it is also possible to use the aptitude packet manager [APT] of the Ubuntu-Linux distribution to install new software.

`(re)build squash filesystem`: compress the SquashFS in *new/*; this is necessary if changes have been applied to it. As several gigabyte of data are compressed in this step, this can take 10 - 30 minutes depending on the available CPU.

`(re)build initrd`: compress the initrd in *init/*; this is necessary if changes have been applied to it.

`create new iso image`: creates a new ISO image based on the SquashFS in *new/* and the initrd in *init/*. The following naming convention is used: exc-*year-month-day-version*.iso. The version number is set to the next free number. Example: if there are already two versions „*exc-2007-06-01-r1.iso*“ and „*exc-2007-06-01-r2.iso*“, the next ISO image gets the name„*exc-2007-06-01-r3.iso*“.

C.3 Initrd

As we have used the Live-CD throughout the development of EXC, the experiment control has been stored in the initrd. The corresponding folder can be found in *init/home*, it is copied to the home-folder of the user upon startup of the Live-CD. To update EXC, it is thus sufficient to copy it to the corresponding folder.

In the home-directory, there is a *start.sh* shell script that is automatically executed when the Live-CD is booted. It initializes the WLAN-interface and is able to detect the used device via its MAC-address. This can be useful if device-specific operations are to be executed, e.g. mounting the local harddisk for storing trace files. At the end, the script starts EXC.

Basic changes to the system or to the startup sequence of the Live-CD can be performed by adapting the scripts in *init/scripts/*.

Appendix D

Pcapsync

As outlined in Section 4.5, `pcapsync` provides the synchronization of trace files in libpcap format [LIB] by using our MLE timestamp synchronization approach. This section shows how the tool is called from the command line and explains the details of the used file formats.

D.1 Command Line Parameters

`Pcapsync` provides the following command line parameters:

`pcapsync [-hsgtl] [-o <output directory>] <list of input files>`

- h : help, displays a help text.
- s : single mode, produces one synchronized file per input file.
- g : global mode, produces one global file containing all packets. This mode requires the additional "-o <output directory>" option.
- t : test mode, produces for each input file a file in a format that can be directly read by the MLE implementation "TSC".
- l : list changes mode, produces for each input file a file that lists packet sequence changes resulting from the synchronization. This mode requires the additional "-s" option.
- o <ouput directory> : redirect the output of `pcapsync` in the output directory.

Example calls:

- `pcapsync -s -g -o ./output/ ./source/*.cap`: produces for each file in directory "source" that ends with ".cap" a synchronized file in the output directory "output" and produces an additional global log file in the current directory.
- `pcapsync ./source/*.cap`: produces for each file that ends with ".cap" a synchronized file in the current directory.

D.2 File Formats

D.2.1 Text File Accompanying the Global Log File

When global mode (the `-g` option) is used, the resulting global log file contains only one entry for each packet, even if this packet has been received multiple times. To document which packet in the global file has been received by which node, `pcapsync` produces an additional text file called `global_observer.txt`. This file lists for each packet the nodes that have received this packet and uses the format:
<packet ID global log> <source file name> <packet ID source file>

An excerpt from such a file looks as follows:

```
140     00028AB7BB53.cap     24
141     00028AB7BF6E.cap     23
142     00028AB7BF6E.cap     24
143     00028AB7C0BF.cap     24
144     00028AB7B89B.cap     24
145     00028AB7BB53.cap     25
145     00028AB7BF6E.cap     25
145     00028AB7BF89.cap     26
```

In this example, the packets with the global ID 140 to 144 all have been recorded by only one receiver. In contrast, the packet with the global ID 145 has been recorded by three receivers.

D.2.2 Text File for the List Changes Mode

When synchronizing libpcap trace files with `pcapsync`, the relative order of packets can change when they are written back to the output files sorted by the calculated, synchronized timestamps. This can happen due to the different treatment of outgoing

and received packets by libpcap (see Section 4.5) or due to the different synchronization methods for anchor and non-anchor points[1]. In the "list changes mode" available via the `-l` option, these changes can be documented for each source file in an additional file, marked with the ending ".swapped". This file contains lines of the following format:
<*initial packet ID*> `->` <
textitnew packet ID> diff <*difference of the IDs*>.

Example:

```
352 -> 351 diff -1
351 -> 352 diff 1
438 -> 437 diff -1
437 -> 438 diff 1
```

In this ".swapped"-file, it is documented that four packets have changed their relative position due to the synchronization. The packets 351 and 352 swapped their positions, and so did the packets with the IDs 437 and 438.

[1] Non-anchor points are only corrected with rate and offset as there it is not enough information to estimate their timestamping delay while anchor points are corrected with rate, offset and timestamping delay

Bibliography

Own Publications

[JKM+] Florian Jarre, Wolfgang Kiess, Martin Mauve, Magnus Roos, and Björn Scheuermann. Least squares timestamp synchronization for local broadcast networks. Under review.

[KCWM] Wolfgang Kiess, Nadine Chmill, Ulrich Wittelsbürger, and Martin Mauve. Modular Network Trace Analysis. Work in progress.

[KFWM04] Wolfgang Kiess, Holger Füssler, Jörg Widmer, and Martin Mauve. Hierarchical Location Service for Mobile Ad-Hoc Networks. *Mobile Computing and Communications Review*, 8(4):47–58, October 2004.

[KM07a] Wolfgang Kiess and Martin Mauve. A Survey on Real-World Implementations of Mobile Ad-Hoc Networks. *Elsevier Ad-Hoc Networks*, 5(3):324–339, April 2007.

[KM07b] Wolfgang Kiess and Martin Mauve. Real-world evaluation of mobile ad-hoc networks. In Marco Conti, Jon Crowcroft, and Andrea Passarella, editors, *Multi-hop Ad hoc Networks from Theory to Reality*, pages 1–22. Nova Science Publishers, Hauppauge, NY, USA, 2007.

[KOM08] Wolfgang Kiess, Thomas Ogilvie, and Martin Mauve. The EXC Toolkit for Real-World Experiments with Wireless Multihop Networks. In *EXPONWIRELESS '08: Proceedings of the 3rd Workshop on Advanced Experimental Activities on Wireless Networks Systems*, June 2008. Accepted.

[KOTM] Wolfgang Kiess, Thomas Ogilvie, Andreas Tarp, and Martin Mauve. On the Repeatability of Experiments with Wireless Multihop Networks. Work in progress.

[KOTM07] Wolfgang Kiess, Thomas Ogilvie, Andreas Tarp, and Martin Mauve. The EXC toolkit: conducting realistic experiments with wireless multi-hop networks. In *MobiSys 2007: The 5th International Conference on Mobile Systems, Applications, and Services 2007, Demo Session*, June 2007.

[KRM07] Wolfgang Kiess, Jedrzej Rybicki, and Martin Mauve. On the nature of Inter-Vehicle Communication. In *WMAN '07: Proceedings of the 4th German Workshop on Mobile Ad-Hoc Networks*, pages 493–502, March 2007.

[KTM05] Wolfgang Kiess, Andreas Tarp, and Martin Mauve. Real-World Evaluation of Ring Flooding. In *MobiCom '05 Poster Session*, September 2005.

[KTM06] Wolfgang Kiess, Andreas Tarp, and Martin Mauve. Real-World Evaluation of Ring Flooding. *Mobile Computing and Communications Review*, 10(4):11–12, October 2006.

[KZM07] Wolfgang Kiess, Stephan Zalewski, and Martin Mauve. Improving System Clock Precision With NTP Offline Skew Correction. In *MedHocNet '07: Proceedings of the 6th Annual Mediterranean Ad Hoc Networking Workshop*, pages 159–164, June 2007.

[KZTM05] Wolfgang Kiess, Stephan Zalewski, Andreas Tarp, and Martin Mauve. Thoughts on Mobile Ad-hoc Network Testbeds. In *REALMAN '05: Proceedings of the 1st International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality*, pages 93–100, July 2005.

[MKS+08] Daniel Marks, Wolfgang Kiess, Björn Scheuermann, Magnus Roos, Martin Mauve, and Florian Jarre. Offline time synchronization for libpcap logs. In *WMAN '08: WMAN Fachgespräch 2008*, April 2008. Accepted.

[RSK+07] Jedrzej Rybicki, Björn Scheuermann, Wolfgang Kiess, Christian Lochert, Pezhman Falahi, and Martin Mauve. Challenge: Peers on wheels — a road to new traffic information systems. In *MobiCom '07: Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, pages 215–221, September 2007.

[SKR+08a] Björn Scheuermann, Wolfgang Kiess, Magnus Roos, Florian Jarre, and Martin Mauve. Error bounds and consistency in maximum likelihood time synchronization. Technical Report TR-2008-001, Department of Computer Science, Heinrich Heine University Düsseldorf, Germany, 2008.

[SKR+08b] Björn Scheuermann, Wolfgang Kiess, Magnus Roos, Florian Jarre, and Martin Mauve. On the time synchronization of distributed log files in networks with local broadcast media. *IEEE/ACM Transactions on Networking*, 2008. Accepted.

Other References

[802] IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Std 802.11-2007.

[ABB+04] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. *ACM SIGCOMM Computer Communication Review*, 34(4):121–132, 2004.

[ABC+05] G. Anastasi, E. Borgia, M. Conti, E. Gregori, and A. Passarella. Understanding the real behavior of mote and 802.11 ad hoc networks: an experimental approach. *Pervasive and Mobile Computing*, 1(2):237–256, July 2005.

[Abr70] N. Abramson. The ALOHA system - another alternative for computer communications. In *Proceedings of the Fall 1970 AFIPS Computer Conference*, pages 281–285, November 1970.

[ACDM] David Applegate, William Cook, Sanjeeb Dash, and Monika Mevenkamp. QSopt linear programming solver. Version 1.01. `http://www2.isye.gatech.edu/~wcook/qsopt/`.

[ADH] Ad-hockey and ns faq. `http://www.monarch.cs.rice.edu/ns-faq/faq.html`.

[Aea05] Anish Arora and Rajiv Ramnath et al. ExScal: Elements of an extreme scale wireless sensor network. In *RTCSA '05: Proceedings of the 11th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 102–108, August 2005.

[AGSR02] J. Allard, P. Gonin, M. Singh, and G. G. Richard. A user level framework for ad hoc routing. In *LCN '02: Proceedings of the 27th Annual IEEE International Conference on Local Computer Networks*, pages 13–19, November 2002.

[All87] David W. Allan. Time and frequency (time-domain) characterization, estimation, and prediction of precision clocks and oscillators. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, UFFC-34(6):647–654, November 1987.

[ape] How to build, install and run the APE testbed. TeX-file in the APE distribution, `http://apetestbed.sourceforge.net/`.

[APT] Aptitude paket management tool. `http://www.debian.org/doc/manuals/reference/ch-package.en.html`.

[Ash95] Paul Ashton. Algorithms for off-line clock synchronization. Technical Report TR COSC 12/95, Department of Computer Science, University of Canterbury, December 1995.

[BABM04] John Bicket, Daniel Aguayo, Sanjit Biswas, and Robert Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *MobiCom '04: Proceedings of the 10th Annual ACM International Conference on Mobile Computing and Networking*, pages 31–42, September 2004.

[BCDG05] E. Borgia, M. Conti, F. Delmastro, and E. Gregori. Experimental comparison of routing and middleware solutions for mobile ad hoc networks: Legacy vs cross-layer approach. In *E-WIND '05: Proceedings of the 2005 ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis*, pages 82–87, August 2005.

[BCG05] Raffaele Bruno, Marco Conti, and Enrico Gregori. Mesh networks: Commodity multi-hop ad hoc networks. *IEEE Communications Magazine*, 43(3):123–131, March 2005.

[Bey90] David A. Beyer. Accomplishments of the DARPA SURAN program. In *MILCOM '90: Proceedings of the IEEE Military Communications Conference*, pages 855–862, September 1990.

[CABM03] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03: Proceedings of the 9th Annual ACM International Conference on Mobile Computing and Networking*, pages 134–146, September 2003.

[CBE03] Alberto Cerpa, Naim Busek, and Deborah Estrin. SCALE: A tool for simple connectivity assessment in lossy environments. Technical Report 21, Center for Embedded Networked Sensing, University of California, Los Angeles (UCLA), September 2003.

[CGM03] Carlos T. Calafate, Roman Garcia Garcia, and Pietro Manzoni. Optimizing the implementation of a MANET routing protocol in a heterogeneous environment. In *ISCC '03: Proceedings of the 8th IEEE International Symposium on Computers and Communication*, pages 217–222, July 2003.

[CJWK02] Kwan-Wu Chin, John Judge, Aidan Williams, and Roger Kermode. Implementation experience with MANET routing protocols. *ACM SIGCOMM Computer Communication Review*, 32(5):49–59, November 2002.

[CM99] S. Corson and J. Macker. Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. RFC 2501 (Informational), January 1999.

[cra] CRAWDAD, the Community Resource for Archiving Wireless Data At Dartmouth. `http://crawdad.cs.dartmouth.edu/`.

[CSS03] David Cavin, Yoav Sasson, and Andre Schiper. FRANC: A lightweight java framework for wireless multihop communication. Technical report, EPFL, Lausanne, April 2003.

[DACM03] Douglas S. J. De Couto, Daniel Aguayo, Benjamin A. Chambers, and Robert Morris. Performance of multihop wireless networks: shortest path is not enough. *ACM SIGCOMM Computer Communication Review*, 33(1):83–88, 2003.

[DHHB87] Andrzej Duda, Gilbert Harrus, Yoram Haddad, and Guy Bernard. Estimating global time in distributed systems. In *ICDCS '87: Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 299–306, September 1987.

[DPZ04a] Richard Draves, Jitendra Padhye, and Brian Zill. Comparison of routing metrics for static multi-hop wireless networks. In *SIGCOMM '04: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 133–144, September 2004.

[DPZ04b] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *MobiCom '04: Proceedings of the 10th Annual ACM International Conference on Mobile Computing and Networking*, pages 114–128, September 2004.

[DRK+06] Pradipta De, Ashish Raniwala, Rupa Krishnan, Krishna Tatavarthi, Jatan Modi, Nadeem Ahmed Syed, Srikant Sharma, and Tzicker Chiueh. MiNTm: an autonomous mobile wireless experimentation platform. In *MobiSys '06: Proceedings of the 4th International Conference on Mobile Systems, Applications, and Services*, pages 124–137, June 2006.

[DRSC05] Pradipta De, Ashish Raniwala, Srikant Sharma, and T. Chiueh. MiNT: A miniaturized network testbed for mobile wireless research. In *INFOCOM '05: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 2731–2742, March 2005.

[EGE02] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. In *OSDI '02: Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, pages 147–163, December 2002.

[FS01] Andras Farago and Violet R. Syrotiuk. Merit: A unified framework for routing protocol assessment in mobile ad hoc networks. In *MobiCom '01: Proceedings of the 7th Annual ACM International Conference on Mobile Computing and Networking*, pages 53–60, July 2001.

[GKHS05] Sachin Ganu, Haris Kremo, Richard Howard, and Ivan Seskar. Addressing repeatability in wireless experiments using ORBIT testbed. In *TRIDENTCOM '05: Proceedings of the 1st International Conference on Testbeds and*

Research Infrastructures for the Development of Networks and Communities, pages 153–160, February 2005.

[GKN+04] Robert S. Gray, David Kotz, Calvin Newport, Nikita Dubrovsky, Aaron Fiske, Jason Liu, Christopher Masone, Susan McGrath, and Yougu Yuan. Outdoor experimental comparison of four ad hoc routing algorithms. In *MSWiM '04: Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 220–229, October 2004.

[GKW+02] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical Report UCLACSD-TR 02-0013, Computer Science Department, UCLA, July 2002.

[GLA01] J.J. Garcia-Luna-Aceves. Wireless internet gateways (WINGs) for the internet. Technical report, University of California, Santa Cruz, 2001.

[gnu] Gnuplot, a portable command-line driven interactive data and function plotting utility. `http://www.gnuplot.info/`.

[Goo] Google Maps. `http://maps.google.com/`.

[GTK] GTK+ : The GIMP Toolkit. `http://www.gtk.org/`.

[GWW04] Abhinav Gupta, Ian Wormsbecker, and Carey Williamson. Experimental evaluation of TCP performance in multi-hop wireless ad hoc networks. In *MASCOTS '04: Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications System*, pages 3–11, October 2004.

[HBRB05] Marc Heissenbttel, Torsten Braun, Tobias Roth, and Thomas Bernoulli. GNU/Linux implementation of a position-based routing protocol. In *REALMAN '05: Proceedings of the 1st International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality*, pages 25–33, July 2005.

[HFMF03] Hannes Hartenstein, Holger Füßler, Martin Mauve, and Walter Franz. Simulation results and proof-of-concept implementation of the FleetNet position-based router. In *PWC '03: Proceedings of the IFIP-TC6 8th International Conference on Personal Wireless Communications*, pages 192–197, September 2003.

[HJ02] Yih-Chun Hu and David B. Johnson. Design and demonstration of live audio and video over multihop wireless ad hoc networks. In *MILCOM '02: Proceedings of the IEEE Military Communications Conference*, pages 1211–1216, October 2002.

[HKS+04] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks.

In *MobiSys '04: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, pages 270–283, June 2004.

[HM05] Andre Herms and Daniel Mahrenholz. GEA: A method for implementing and testing of event-driven protocols. In *MeshNets '05: Proceedings of the 1st International Workshop on Wireless Mesh Networks*, July 2005.

[Ian06] Gianluca Iannaccone. Fast prototyping of network data mining applications. In *PAM '06: Proceedings of the Passive and Active Measurement Conference*, March 2006.

[INI] Initrd - initial ram disk. `http://www.ibm.com/developerworks/linux/library/l-initrd.html`.

[JBD+05] Sushant Jadhav, Timothy Brown, Sheetalkumar Doshi, Daniel Henkel, and Roshan Thekkekunnel. Lessons learned constructing a wireless ad hoc network test bed. In *WiNMee '05: Proceedings of the 1st International Workshop On Wireless Network Measurement*, April 2005.

[JMH03] David B. Johnson, David A. Maltz, and Yih-Chun Hu. The dynamic source routing protocol for mobile ad hoc networks. Internet-Draft, draft-ietf-manet-dsr-09.txt, April 2003. Work in progress.

[JS04] Glenn Judd and Peter Steenkiste. Repeatable and realistic wireless experimentation through physical emulation. *ACM SIGCOMM Computer Communication Review*, 34(1):63–68, 2004.

[JSF+06] David Johnson, Tim Stack, Russ Fish, Dan Flickinger, Leigh Stoller, Rob Ricci, and Jay Lepreau. Mobile emulab: A robotic wireless and sensor network testbed. In *INFOCOM '06: Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1–12, April 2006.

[JT87] John Jubin and Jane D. Turnow. The DARPA packet radio network protocols. In *Proceedings of the IEEE*, volume 75, pages 21–32, January 1987.

[KEPS04] Richard M. Karp, Jeremy Elson, Christos H. Papadimitriou, and Scott Shenker. Global synchronization in sensornets. In *LATIN '04: Proceedings of the 6th Latin American Symposium on Theoretical Informatics*, pages 609–624, April 2004.

[Ker06] Markus Kerper. Eine Plattform zur methodischen Vermessung und Modellierung der Funkausbreitung in einem Experimentierszenario. Bachelor's thesis, October 2006. (in german).

[KGBK78] Robert E. Kahn, Steven A. Gronemeyer, Jerry Burchfiel, and Ronald C. Kunzelman. Advances in packet radio technology. In *Proceedings of the IEEE*, volume 66, pages 1468–1496, November 1978.

[KMC+00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[KNG+04] David Kotz, Calvin Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. In *MSWiM '04: Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 78–82, October 2004.

[Kni] KNIME, The Konstanz Information Miner. `http://www.knime.org/`.

[KO87] Hermann Kopetz and Wilhelm Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Communications*, 36(8):933–940, 1987.

[KOV+02] Kurt Konolige, Charles Ortiz, Regis Vincent, Andrew Agno, Michael, Eriksen, Benson Limketkai, Mark Lewis, Linda Briesemeister, Enrique Ruspini, Dieter Fox, Jonathan Ko, Benjamin Stewart, and Leonidas Guibas. DARPA software for distributed robotics. Technical report, DARPA, December 2002.

[KZG03] Vikas Kawadia, Yongguang Zhang, and Binita Gupta. System services for ad-hoc routing: Architecture, implementation and experiences. In *MobiSys '03: Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services*, pages 99–112, May 2003.

[LA03] A. Laouiti and C. Adjih. Mesures des performances du protocole OLSR. IEEE SETIT, March 2003. (in French).

[lab] LabVIEW for Measurement and Data Analysis. `http://zone.ni.com/devzone/cda/tut/p/id/3566`.

[LIB] Libpcap File Format. `http://wiki.ethereal.com/Development/LibpcapFileFormat`.

[LLN+02] Henrik Lundgren, David Lundberg, Johan Nielsen, Erik Nordström, and Christian Tschudin. A large-scale testbed for reproducible ad hoc protocol evaluations. In *WCNC '02: Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 337–343, March 2002.

[LNT87] B. Leiner, D. Nielson, and F. Tobagi, editors. *Proceedings of the IEEE (Special Issue, packet radio networks)*, volume 75, January 1987.

[LNT02] Henrik Lundgren, Erik Nordström, and Christian Tschudin. Coping with communication gray zones in IEEE 802.11b based ad hoc networks. In *WoWMoM '02: Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia*, pages 49–55, September 2002.

[Lun02] David Lundberg. Ad hoc protocol evaluation and experiences of real world ad hoc networking. Master's thesis, University Uppsala, 2002.

[LYN+04] Jason Liu, Yougu Yuan, David M. Nicol, Robert S. Gray, Calvin C. Newport, David Kotz, and Luiz Felipe Perrone. Simulation validation using direct execution of wireless ad-hoc routing protocols. In *PADS '04: Proceedings of the 18th Workshop on Parallel and Distributed Simulation*, pages 7–16, May 2004.

[LYN+05] Jason Liu, Yougu Yuan, David M. Nicol, Robert S. Gray, Calvin C. Newport, David Kotz, and Luiz Felipe Perrone. Empirical validation of wireless models in simulations of ad hoc routing protocols. *Simulation: Transactions of The Society for Modeling and Simulation International*, 81(4):307–323, April 2005.

[MBJ99] David A. Maltz, Josh Broch, and David B. Johnson. Experiences designing and building a multi-hop wireless ad hoc network testbed. Technical Report CMU-CS-99-116, School of Computer Science, Carnegie Mellon University, 1999.

[MBJ00] D. Maltz, J. Broch, and D. Johnson. Quantitative lessons from a full-scale multi-hop wireless ad hoc network testbed. In *WCNC '00: Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 992 – 997, September 2000.

[Meh92] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[MEL] Melbourne wireless. `http://www.melbournewireless.org.au`.

[MFHF04] Michael Möske, Holger Füßler, Hannes Hartenstein, and Walter Franz. Performance measurements of a vehicular ad hoc network. In *VTC '04-Spring: Proceedings of the 59th IEEE Vehicular Technology Conference*, pages 2116–2120, May 2004.

[MFNT00] Michael Mock, Reiner Frings, Edgar Nett, and Spiro Trikaliotis. Continuous clock synchronization in wireless real-time applications. In *SRDS '00: Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, pages 125–132, October 2000.

[Mil92] D. Mills. Network time protocol (version 3) specification, implementation and analysis. RFC 1305 (Draft Standard), March 1992.

[Mil94a] David L. Mills. Internet time synchronization: The network time protocol. In Zhonghua Yang and T. Anthony Marsland, editors, *Global States and Time in Distributed Systems*. IEEE Computer Society Press, 1994.

[Mil94b] David L. Mills. Precision synchronization of computer network clocks. *ACM SIGCOMM Computer Communication Review*, 24(2):28–43, 1994.

[Mös03] Michael Möske. Real-world evaluation of a vehicular ad hoc network using position-based routing. Master's thesis, Department of Computer Science, University of Mannheim, July 2003.

[MST99] Sue B. Moon, Paul Skelly, and Donald F. Towsley. Estimation and removal of clock skew from network delay measurements. In *INFOCOM '99: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 227–234, March 1999.

[NJG02] Michael Neufeld, Ashish Jain, and Dirk Grunwald. Nsclick: Bridging network simulation and deployment. In *MSWiM '02: Proceedings of the 5th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 74–81, September 2002.

[NKSW02] Jürgen Nagler, Frank Kargl, Stefan Schlott, and Michael Weber. Ein Framework für MANET Routing Protokolle. In *1. deutscher Workshop über Mobile Ad-Hoc Netzwerke (WMAN)*, pages 153–165, March 2002. (in German).

[NS2] The ns-2 network simulator. `http://www.isi.edu/nsnam/ns/`.

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, Berlin, 1999.

[Ogi07] Thomas Ogilvie. Control of MANET-experiments with a graphical interface. Bachelor's thesis, University of Düsseldorf, Germany, March 2007.

[OPEa] OpenDX - Open Data Explorer. `http://www.opendx.org/`.

[opeb] The OpenEmbedded Project. `http://www.openembedded.org/`.

[opec] OpenZaurus: Linux distribution for sharp zaurus palmtops. `http://www.openzaurus.org`.

[ORB] The ORBIT testbed. `http://www.orbit-lab.org`.

[OT05] Evgeny Osipov and Christian Tschudin. A path density protocol for MANETs. In *REALMAN '05: Proceedings of the 1st International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality*, pages 69–76, July 2005.

[PAM+05] Thierry Plesse, Cedric Adjih, Pascale Minet, Anis Laouiti, Adokoe Plakoo, Marc Badel, Paul Muhlethaler, Philippe Jacquet, and Jerome Lecomte. OLSR performance measurement in a military mobile ad hoc network. *Elsevier Ad-Hoc Networks*, 3(5):575–588, September 2005.

[PBRD03] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental), July 2003.

[Per01] Charles E. Perkins. *Ad Hoc Networking*. Addison Wesley Professional, 2001.

[Plu82] David C. Plummer. Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware. RFC 826 (Standard), November 1982.

[PR99] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *WMCSA '99: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[QRB] QtRuby bindings. `http://developer.kde.org/language-bindings/ruby/`.

[QTT] Qt Development Framework. `http://www.trolltech.com/products/qt/`.

[RABR05] Krishna Ramachandran, Kevin Almeroth, and Elizabeth Belding-Royer. A novel framework for the management of large-scale wireless network testbeds. In *WiNMee '05: Proceedings of the 1st International Workshop On Wireless Network Measurement*, April 2005.

[RBM05] Kay Römer, Philipp Blum, and Lennart Meier. Time synchronization and calibration in wireless sensor networks. In Ivan Stojmenovic, editor, *Handbook of Sensor Networks: Algorithms and Architectures*, pages 199–237. John Wiley & Sons, September 2005.

[RBRA04] Krishna Ramachandran, Elizabeth Belding-Royer, and Kevin Almeroth. DAMON: A distributed architecture for monitoring multi-hop mobile networks. In *SECON '04: Proceedings of the 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, pages 601–609, October 2004.

[RH00] Ram Ramanathan and Regina Hain. An ad hoc wireless testbed for scalable, adaptive QoS support. In *WCNC '00: Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 998–1002, September 2000.

[Riv92] R. Rivest. The MD5 message-digest algorithm. RFC 1321 (Informational), April 1992.

[ROO] The MIT roofnet project. `http://www.pdos.lcs.mit.edu/roofnet/`.

[RP00] Elizabeth M. Royer and Charles E. Perkins. An implementation study of the AODV routing protocol. In *WCNC '00: Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 1003–1008, September 2000.

[RPD+05] Joshua Robinson, Konstantina Papagiannaki, Christophe Diot, Xingang Guo, and Lakshman Krishnamurthy. Experimenting with a multi-radio mesh networking testbed. In *WiNMee '05: Proceedings of the 1st International Workshop On Wireless Network Measurement*, April 2005.

[RRS+05] Ram Ramanathan, Jason Redi, Cesar Santivanez, David Wiggins, and Stephen Polit. Ad hoc networking with directional antennas: A complete system solution. *IEEE Journal on Selected Areas in Communications*, 23(3):496–506, March 2005.

[RSO+05] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In *WCNC '05: Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 1664–1669, March 2005.

[SBSC03] Jatinder Pal Singh, Nicholas Bambos, Bhaskar Srinivasan, and Detlef Clawin. Proposal and demonstration of link connectivity assessment based applications to routing in mobile ad-hoc networks. In *VTC '03-Fall: Proceedings of the 58th IEEE Vehicular Technology Conference*, pages 2834–2838, October 2003.

[SFT+05] Björn Scheuermann, Holger Füßler, Matthias Transier, Marcel Busse, Martin Mauve, and Wolfgang Effelsberg. Huginn: A 3D visualizer for wireless ns-2 traces. In *MSWiM '05: Proceedings of the 8th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 143–150, October 2005.

[SKR99] Peter Showbridge, Miro Kraetzl, and David Ray. Detection of abnormal change in dynamic networks. In *IDC '99: Proceedings of Information, Decision and Control*, pages 557–562, February 1999.

[SOSK05] Manpreet Singh, Maximilian Ott, Ivan Seskar, and Pandurang Kamat. ORBIT measurements framework and library (OML): Motivations, design, implementation, and features. In *TRIDENTCOM '05: Proceedings of the 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, February 2005.

[SQU] Squashfs - a compressed read-only filesystem for Linux. `http://squashfs.sourceforge.net/`.

[SRP] Jos F. Sturm, Oleksandr Romanko, and Imre Pólik. SeDuMi. Version 1.1R2. `http://sedumi.mcmaster.ca/`.

[STP+05] Amit Kumar Saha, Khoa To, Santashil PalChaudhuri, Shu Du, and David B. Johnson. Physical implementation and evaluation of ad hoc network protocols using unmodified simulation models. In *ACM SIGCOMM Asia Workshop*, April 2005.

[TCP] Tcpdump: a tool for network monitoring, protocol debugging and data acquisition. `http://www.tcpdump.org`.

[TK94] Barry N. Taylor and Chris E. Kuyatt. Guidelines for evaluating and expressing the uncertainty of NIST measurement results. Technical Note 1297, National Institute of Standards and Technology (NIST), 1994.

[TRA] Trace graph - Network Simulator NS-2 trace files analyser. `http://www.tracegraph.com/`.

[TSM07] Thi Minh Chau Tran, Björn Scheuermann, and Martin Mauve. Detecting the presence of nodes in MANETs. In *CHANTS '07: Proceedings of the 3rd ACM MobiCom Workshop on Challenged Networks*, pages 43–50, September 2007.

[TUN] The TUn!x linux distribution. `http://pluto.htu.tuwien.ac.at/Hauptseite`.

[ULO] Ulogd: a userspace logging daemon for netfilter/iptables related logging. `http://www.netfilter.org/projects/ulogd/`.

[VBP04] Darryl Veitch, Satish Babu, and Attila Pàsztor. Robust synchronization of software clocks across the Internet. In *IMC '04: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 219–232, October 2004.

[Vin04] Regis Vincent. Re: Centibots: communication among robots. Personal Communication, June 2004.

[VRC97] Paulo Veríssimo, Luís Rodrigues, and Antonio Casimiro. Cesiumspray: a precise and accurate global time service for large-scale systems. *Real-Time Systems*, 12(3):243–294, 1997.

[WIKa] Wikipedia, wireless community network. `http://en.wikipedia.org/wiki/Wireless_community_network`.

[wikb] List of ad hoc routing protocols at wikipedia. `http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list`.

[WIRa] The wireshark network protocol analyzer. `http://www.wireshark.org/`.

[wirb] The Wireless Leiden foundation. `http://wirelessleiden.nl`.

[WiS] The Wi-Spy spectrum analyzer. `http://www.metageek.net/Products/Wi-Spy`.

[YCK+02] M.D. Yarvis, W.S. Conner, L. Krishnamurthy, J. Chhabra, B. Elliott, and Alan Mainwaring. Real-world experiences with an interactive ad hoc sensor network. In *Proceedings of the International Workshop on Ad Hoc Networking*, pages 143–151, August 2002.

[zau] The zaulux project. `http://www.cn.uni-duesseldorf.de/projects/ZAULUX`.

[ZG03] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 1–13, November 2003.

[ZL02] Yongguang Zhang and Wei Li. An integrated environment for testing mobile ad-hoc networks. In *MobiHoc '02: Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 104–111, June 2002.

[ZLX02] Li Zhang, Zhen Liu, and Cathy Honghui Xia. Clock synchronization algorithms for network measurements. In *INFOCOM '02: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 160–169, June 2002.

Index