Providing QoS for Publish/Subscribe Communication in Dynamic Ad-Hoc Networks

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von: Diplom-Informatiker Daniel Mahrenholz
geb. am 16.01.1976 in Wolmirstedt

Gutachter:
Prof. Dr. Edgar Nett
Prof. Dr. Jörg Kaiser
Prof. Dr. Wolfgang Schröder-Preikschat

Promotionskolloquium:
Magdeburg, den 2. Mai 2006

Zusammenfassung

Mit der Entwicklung und vor allem Miniaturisierung elektronischer Geräte hat auch die drahtlose Kommunikation in den vergangen Jahren eine immer stärkere Bedeutung zur Vernetzung dieser Geräte erlangt, die inzwischen nahezu alle Bereich des täglichen Lebens durchdrungen haben. Speziell durch die spontane (Ad-hoc-) Vernetzung und Interaktion ergeben sich eine Vielzahl neuer Anwendungsbereiche. Ein wichtiges Ziel ist dabei, drahtlose Kommunikation und die mobile Nutzung verschiedenster Dienste über grössere Bereiche hinweg ohne eine fixe Infrastruktur zu ermöglichen. Dies ist besonders dann wichtig, wenn eine Infrastruktur nur aufwendig errichtet werden könnte oder z.B. nach einer Naturkatastrophe zerstört wurde. Für die erfolgreiche Dienstnutzung müssen zuerst Dienstanbieter and Nutzer zueinander gebracht werden. Hierfür hat sich das Publisher-/Subscriber-Modell als sehr effektiv erwiesen, da es von physikalischen Knoten abstrahiert und Kommunikationsbeziehung anhand der verwendeten Daten definiert. Darüber hinaus erfordern einige Dienste eine definierte Mindestgüte der Kommunikation wie z.B. Bandbreite oder Latenz für eine erfolgreiche Ausführung. Dem gegenüber steht die inherent unzuverlässige Natur des geteilten Funkmediums und die Dynamik der Netzstruktur beim Einsatz mobiler Geräte.

Im Rahmen dieser Arbeit werden Kommunicationsprotokolle entwickelt, die die Kommunikation nach dem Publisher-/Subscriber-Modell unter Einhaltung bestimmter Dienstgüteanforderungen in einem drahtlosen Netzwerk ermöglichen. Dafür werden zuerst die Eigenschaften eines drahtlosen Multi-Hop-Netzwerkes nach dem IEEE 802.11 Standard untersucht, um die praktisch auftretenden Problem und Effekte beschreiben und in einem Modell nachbilden zu können. Die dabei gewonnen Erkenntnisse dienen gleichzeitig zum Aufbau einer realistischen Simulation als Basis für die Testung und Bewertung der entwickelten Protokolle. Die Simulation wird durch Praxistests für Teilaspekte ergänzt, um die ermittelten Eigenschaften zu verifizieren. Die Grundlage für die Durchsetzung von Dienstgütegarantien bilden eine kontinuierliche Zuverlässigkeitsmessung von Verbindungen zu Nachbarn, die verteilte Bestimmung und Koordination der Medienauslastung, sowie die kontinuierliche Messung von Ende-zu-Ende-Eigenschaften. Die Kommunikationsprotokolle vertrauen dabei nur auf gemessene Eigenschaften des Mediums. Aufbauend auf diesen Grundfunktionen wird die Publisher-/Subscriber-Kommuniktion mit Hilfe von Multicast-Bäumen realisiert. Hierfür wird eine einfach, maschinenlesbare Inhaltbeschreibung verwendet, um den Verarbeitungsaufwand auf den mobilen, meist leistungsschwachen Geräten und dadurch die Ende-zu-Ende-Verzögerung minimal zu halten. Die entwickelten Verfahren stehen als eine Menge eigenständiger Module einer portablen, ereignisbasierten Middleware zur Verfügung. Diese Middleware lässt sich dynamisch komponieren und konfigurieren, um nur die für den Anwendungsfall notwendige Ressourcen zu belegen.

Abstract

With the proliferation and miniaturization of electronic devices that today pervade almost all areas of our everyday life, wireless communication received a growing importance to interconnect them. Especially so-called ad hoc networks and collaboration allow for a variety of new applications. An important goal thereby is to enable wireless communication and access to different services without the need for a fixed infrastructure. This is especially important if such an infrastructure can only be deployed at high costs are has been destroyed in a disaster. For the successful use of a service it is first necessary to find matching service provers for a user. The publish/subscribe communication scheme provides an ideal solution for this problem as it abstracts from physical nodes and defines communication relationships based on the used data. In addition some services require a minimum quality of service to operate appropriately. In the contrary we have the inherent unreliable nature of the shared wireless medium and the dynamics of the network topology caused by the node mobility.

In this thesis we will develop communication protocols to provide publish/subscribe communication with certain QoS guarantees in a mobile ad hoc network. We will start with a investigation of the properties of a IEEE 802.11 standard compliant multi-hop network in order to describe the problems and effects that can be experienced in practice as a model. The hereby gained knowledge will also be used to build a realistic simulation that provides the basis for the testing and evaluation of the developed protocols. The simulation will be complemented by real-world experiments to verify the results of different aspects of the system. In order to provide QoS guarantees we will combine a continuous monitoring of the link quality to our neighbor nodes, a cooperative estimation and coordination of the bandwidth utilization, and an online monitoring of the end-to-end properties. The developed communication protocols only rely on actual measured properties of the network. Upon these basic functions we will realize a publish/subscribe communication using multicast trees. We will use machine-readable content descriptions to minimize the processing overhead on intermediate nodes and hence the end-to-end delay. The developed protocols will be implemented as independent modules of a portable, event-based middleware. This middleware allows a dynamic composition and configuration of individual modules at run-time in order to consume only the resources required for a certain use case.

Acknowledgements

At this point I want to express my feelings of gratitude to all those who have enabled me to complete this work. First of all, I would like to thank Prof. Dr. Edgar Nett for his guidance and many fruitful discussions during my time in the “Real-Time Systems and Communication” group. He taught me that understanding the core of a problem is the most important step to its solution.

I also have to thank all members of the working group for the prolific collaboration over the last years, for their important feedback and for many open-minded discussions even about some crazy ideas, and of course for contributing their time and skills for experiments and bug hunting.

I also want to give a big thankyou to my family and friends for their patience especially in the final stage of this work and for keeping my good mood up even in hard times.

Most of the work presented herein was conducted in the project “Eine Publisher/Subscriber-basierte Middleware mit Dienstgüte-Garantien zur Unterstützung kooperativer Anwendungen”, which was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant NE 837/3-1.

Contents

1. Introduction

1.1. Motivation

Wireless networks are becoming increasingly popular. More and more devices – from laptops and PDAs to home entertainment systems are equipped with wireless LAN (WLAN) transceivers. Even cell phones are starting to be equipped with WLAN transceivers to be used as hardware devices for IP-based voice communication (VoIP – Voice over IP). Wireless ad-hoc networks are used to provide a communication infrastructure in different areas – as a fast deployable, temporary replacement for destroyed fixed networks (*disaster recovery*) or in areas where wired LANs are impossible or only cost-intensive to deploy like protected historical buildings, or the venue of a conference. Currently wireless networks are mostly used only as a last-mile, that means a 1-hop extension of a wired LAN. But with the proliferation of wireless communication devices, multi-hop wireless networks (*MANETs*) will gain a practical relevance and acceptance in the future. They will provide a multi-hop extension to a wired backbone network, or a completely stand-alone network. In the scientific area they have been under heavy research for quite some time.

As we are accustomed to a lot of services that are currently used in a wired network, it is an obvious desire to use the same services in a wireless network at the same time we utilize the new possibilities like mobility or ad-hoc collaboration. This means that we have to provide communication services with a certain quality of service (QoS). Especially real-time sensitive multimedia applications like voice communication or video streaming require new concepts because wired and wireless network have fundamental different communication properties. Wired networks are almost static, i.e. stations rarely move and the topology is fixed, receivers of a transmission can be controlled by the cabling among them, and the probability of packet loss on such a cable link is very low. Wireless networks in contrast can be highly dynamic, i.e. mobile users often change their geographical position which permanently changes the topology, transmission are broadcasted into the air, i.e. we cannot control who receives it, and we have to consider a higher probability of lost packets. Apart from all multimedia applications, we have another class of applications with QoS requirements that operate over wireless networks – distributed SCADA (supervisory control and data acquisition) system or other types of command and control systems. Such systems have stronger real-time and reliability requirements compared with multimedia systems that have stronger bandwidth requirements.

Network communication traditionally follows the client-server scheme where we have a known system that provides a service (*server*) and a system that uses this service (*client*). This scheme requires us to know who the server is, i.e. we have to know which node on the network runs the

required service. But what we actually require is the service and we would rather not care about what host is providing it. It would be much more convenient to tell a computer program that we are searching for a special information rather then to first go to one of the many search engine web sites and search for the information there. Another common example are instant messaging (IM) services that are used to exchange text messages or larger files between an arbitrary number of users. The most widely used IM system still require a user to know which service providers the other users are connected to, which is very inconvenient. Modern IM like the Jabber system [jhp] only require a user to know its own service provider in order to be able to communicate with other users or services using the *Extensible Messaging and Presence Protocol* (XMPP) as specified in the RFCs 3920–3923 ([Jab04a, Jab04b, Jab04c, Jab04d]). This is much more convenient but still not fully satisfying.

A solution is to shift the communication paradigm from client/server communication to publish/subscribe (P/S) communication. In a P/S system we do not longer talk about node addresses or service providers, we talk about the content of the communication. Communication partners are chosen based on the content they want to exchange. Thats why the publish/subscribe paradigm is also called content-based addressing. In such a system we have information providers (*publisher*) and information users (*subscriber*) and hold the runtime system responsible to deliver messages between them. An everyday-life example of a P/S system are newspapers. Usually the reader of a newspaper is not interested that a particular person writes an article or an author is not interested who actually reads the newspaper. What they agree on is the content – may it be financial news, sports, celebrities, etc. In a more technical context we can take an automated plant as an example. There we have a lot of different sensors that create events on state changes. To build a fire detection application we would specify our interest in all events that include temperature values and location information. So we are independent from the installed sensors, i.e. we do not need to adapt the application to different installations because this will be transparently handled by the run-time system. In addition to the Jabber example it should be noted that there is an ongoing effort [MSAM05] to provide P/S communication based on XMPP which currently is in a testing stage.

It is obvious that we cannot introduce a new communication paradigm with a higher level of abstraction without any additional costs. Especially if we intend to provide QoS guarantees in a P/S system we introduce another complexity dimension. QoS requirements can be very different depending on the application scenario and so it is highly unrealistic that we can create a universal solution that fits best in all cases. That means that we will have different algorithms and protocols to support different application scenarios and environments. At the same time we have to keep in mind the devices that we have in a wireless network – PDAs, Laptops, embedded devices with wireless adaptors – in general devices with limited resources. On such devices we cannot afford to waste resources by including the software components for all possible application scenarios. Instead we need a highly configurable, modularized software architecture that allows to exchange individual components and to construct a fully functional system on demand. Apart from the resource savings we have to keep in mind that QoS in MANETs is currently very active research field. So it is very likely that we will see improved algorithms for different task of such a complex system. And of course it should be possible to include such new algorithms when they

become available without the need for major changes (best without any changes at all) in other components of the system.

1.2. Problem Exposition

The problem to provide QoS guarantees for a publish/subscribe communication based system has to be split into two major sub-problems: first, the publish/subscribe communication, i.e. the system that brings matching publishers and subscribers together, and second all mechanism and protocols that allow for QoS guarantees.

A publish/subscribe communication system consists of several information providers (publishers) and information consumers (subscribers). Each subscriber specifies its own interests, i.e. the information that it intents to receive. The communication system has then to ensure that all data send by a publisher is delivered to all interested subscribers. This creates a n:m communication relationship between publishers and subscribers instead of a conventional 1:1 relationship in a client/server system. To realize such a communication we have first to define how we describe the content of the data created by a publisher and the interests of a subscriber. We then have to ensure that the data created by a publisher is delivered to all interested subscribers. From a global perspective this sounds quite easy – but we have to keep in mind that we have a highly dynamic distributed system. In a MANET nodes move around, join and leave the network at arbitrary points in time, and we do not have a central coordinator. Additionally we have to consider cases where a network splits into several unconnected parts or a network merges with another network. So we need a distributed system that keeps track of the network topology, publishers, subscribers, matches interests with offerings (published data), finds routes from the publishers to the interested subscribers, and finally delivers the data. The main challenge for this basic system is to create a global view of publishers and subscribers based on local knowledge without wasting network resources.

The basic P/S system gets much more complex if we introduce QoS requirements. We separate application-based and connection-based QoS. Application-based QoS describes properties of the delivered data, e.g. the accuracy of a sensor, the resolution of images, compression algorithms and so on. Connection-based QoS describes properties of the end-to-end delivery of the data, e.g. bandwidth and maximum delay.

When we look on the communication medium we encounter a broad range of problems that have to be solved in order to provide QoS guarantees. A wireless network uses a shared medium and so the nodes on the network have to coordinate their access to the medium, to reserve, or prioritize special communication flows. As every node has only a limited transmission range we will need a routing mechanisms that forwards data over some intermediate node to the target. This increases the complexity of the required coordination as we need to include all nodes along such a path and their respective neighborhood. A major problem for such a coordination is that it can only be done using communication among the nodes. The more nodes we have to involve, the more communication we likely need to coordinate them. The crux is that in many cases we

do not know how many nodes we have to involve. So we can either be conservative and assume a worst case number of nodes – which leads to a poor utilization of the network as we spend most resources for the coordination, or we can be optimistic, i.e. assume only a smaller number of nodes, and risk that our coordination will work insufficient which probably leads to a violation of the QoS guarantees. Furthermore we have to cope with the problem of network dynamics causes by node movement, nodes joining or leaving the network, or environmental changes that effect the wireless communication. That means that these dynamics make it hard or even impossible to predict the future evolution of the network based on the observations of the past. As a clear consequence we have to prepare for unexpected changes in the network and react as they happen. As we cannot solve all possible problems that may arise (if a laptop drops to the floor no software will revive it) we have at least to ensure an error detection within given time bounds.

A major problem for every network protocol development is the testing and evaluation. For wireless communication this is an even greater problem because we have more degree of freedom and so a much higher complexity. At the same time we are developing protocols for scenarios and topologies that are impossible to set-up in a test environment. But nobody will certainly roll out a new technology in a productive environment without a high confidence that it will function properly. So, our best solution is to use network simulations to test and evaluate the protocol under different conditions. Such a simulation is only meaningful if we manage to reproduce a realistic environment. This at least requires us to reproduce all effects (especially all problems) that can be experienced in a real-world scenario inside the simulation. Additionally we have to cross-check simulation results with real measurements as far as possible to increase the confidence in the simulation results.

1.3. Approach

To solve the problems discussed in the previous section we will use the following approaches:

Understanding and modelling wireless communication: Providing any kind of quality of service guarantees requires to understand the principle properties and problems of the communication medium and the communication devices – based on the IEEE 802.11 standard. We will study different ways to describe such a system and its components by combining knowledge and experience from the field of signal processing, communication engineering, network simulation, and of course network communication. The goal is to build a model of a wireless communication system that allows to describe it in a realistic way, and especially is able to produce all kinds of problems that can be experienced. We will supplement this study with field tests to verify the expressiveness and identify possible incompleteness of the model. Beside the theoretical model we will build an instance of it in form of a network simulation that will be used to test and evaluate the developed protocols.

Providing QoS for publish/subscribe communication: Based on the discussion of the discussion of the properties and difficulties of wireless communication we will develop a number of techniques and protocols to solve them. The main approach is to rely only on measured values in the

real system instead of special assumptions about the wireless medium. We will only use a neighborhood discovery protocol and measured link qualities and stabilities to define connectivity among nodes and the topology of the network. We will allow end-to-end bandwidth reservations based on a cooperative neighborhood coordination scheme and enforce it using local traffic shaping and prioritization. We will estimate bandwidth utilization using the developed model and cross-check it with the actual bandwidth utilization using a dynamic, monitoring-based approach to compensate for possible inaccuracies of the model as well as to detect and handle situations where coordination is necessary but communication is impossible. Based on the measured link qualities we will define an overlay network that satisfies our minimum reliability requirements and will be used for all communications tasks. Along this overlay network we will allocate and actively maintain point-to-point connections and combine them to multicast trees that will be used to build up the event dissemination network to perform publish/subscribe communication. We will distinguish between network QoS properties that will handled by the network layer and application QoS properties that will be opaque for the network layer and handled by the applications. The publish/subscribe communication will be divided into a best-effort communication based management part and the flow-based event dissemination process that is subject to certain QoS requirements. A monitoring system will continuously control the compliance of events with QoS requirements and give notifications about violations.

Multiplexing of QoS and best-effort communication: To guarantee end-to-end throughput we will use a conservative estimation of the medium. Will will exploit the actually unused redundancy for best-effort communication without impeding QoS flows. We will supplement it with a dynamic bandwidth allocation mechanism to allow for a fair sharing of non-QoS capacity.

Flexible, portable middleware integration: All components developed in this thesis have to be integrated into a flexible middleware platform. We will provide individual components that can be composed at run-time to provide the desired functionality without wasting resources for unused functionality. To increase portability and support testing and evaluation it will be implemented in an event-based manner that allows a unified execution on various native platforms and within a simulated network.

1.4. Structure of the Thesis

The remaining thesis is structured as follows:

Section 2 Will give an overview of concepts significant for the thesis, explain the motivating application scenarios, and will derive the functional and non-functional requirements for the developed system from these scenarios.

Section 3 Will discuss related work. If starts with an discussion of some routing protocols for mobile ad hoc networks, explaining their main ideas, problems, and practical experiences if available. This discussion provides a basis to identify valuable approaches to be adopted within this thesis and to avoid known pitfalls in the protocol designs.

Section 4 Explains the principles of wireless communication and how it can be modelled. The chapter starts with a discussion of how to model individual components of such a system. It is followed by an extensive discussions of the implications of the model used to describe signal propagation in the wireless medium. The chapter presents some real-world measurements to test the validity of the theoretical results and defines the model that will guide the protocol development throughout this remaining thesis. The chapter ends with a discussion of various test and simulation techniques applicable for wireless protocol developments as well as a brief discussions of practical problems that developers should be aware of.

Section 5 presents and discusses the concepts and protocols developed to provide quality of service guarantees for a publish/subscribe communication system. The chapter starts with a brief discussion of the objectives and the assumptions about the communication environment that are mainly results of the discussions in section 4. The section continues with the presentation of the publish/subscribe system followed by the protocols for the bandwidth coordination and link quality monitoring. Based on these basic protocols the multicast tree construction and maintenance protocol is developed. The chapter closes with a discussion of the multiplexing of QoS and best-effort communication.

Section 6 focuses on the implementation and discusses various design decisions that were made. The chapter starts with a presentation of the event-based run-time system extension that is the basis for the whole middleware implementation. The main part of the chapter discusses the internal and external APIs and how they contribute to fulfill the functional and non-functional requirements. The chapter closes with a brief discussion of the integration of legacy applications.

Section 7 Presents the results from various experiments that were made to test the functionality of the key components and to compare simulation with real-world results. Especially this comparison provides insightful results to evaluate the accuracy of the simulation model which is essential to create confidence in the solely simulation based results that follow the comparison experiments.

Section 8 finally summarizes the results of this thesis, the lessons learned and highlights potential for future improvements.

2. Concepts and Requirements

In this section we will briefly discuss some basic concepts and requirements that govern the work presented here.

2.1. Quality of Service

The term *quality of service* (QoS) will be uses for two different classes of properties – *application QoS* and *communication QoS*. As application QoS we will denote properties of a service or data provided by an application. Examples would be the resolution of a video image, its encoding algorithm, or the accuracy of a sensor. As network QoS we will we denote the properties of the communication between peers (applications or network nodes). Examples for such properties are *latency*, *bandwidth*, *jitter*, *reliability*, and *persistence*. As the communication QoS properties are the primary focus of this work, the term QoS will be used as the synonym for them. If we regard application QoS this will be stated explicitly. The network QoS properties are defined as follows:

latency	The time a data packet uses to travel from a source to the destination.
bandwidth	The capacity of a communication channel, i.e. the amount of information that can be transmitted within a given time period.
jitter	A measure of the variability over time of the latency of data packets.
reliability	The probability of a network packet to reach the destination.
persistence	The time an event is guaranteed to exist in the system. Nodes interested in the event joining during this time will receive it.
order	A message consumer is guaranteed to receive events in the same order they have been created by the producer.
global order	A message consumer receiving messages from different producers is guaranteed to received events in the same order they have been created by the producers with respect to an external clock.

Compared with traditional, wired networks, quality of service is very difficult to enforce in a MANET due to its completely different physical medium and dynamic behavior. This has to be considered in the definition of QoS properties. So, for QoS in a MANET we will use the definition for *soft guarantees* given in [Gup05] and extend it by a violation feedback as follows: *The ad-hoc network will choose the optimal existing mechanisms to support the desired quality for the service; and will reject the service request if the existing mechanisms are unable to provide the desired quality. Furthermore, the new service will also not be accepted if its introduction is*

expected to violate the QoS for the existing services. Once accepted, the compliance with the specified QoS properties will be constantly monitored and violations will be notified.

2.2. Publish/Subscribe Communication

Publish/subscribe (P/S) communication is a many-to-many communication paradigm that defines communication relationships based on the data communicated. It is anonymous in the sense that a publisher does not have to know the subscribers that receive its events, or other publishers in order to function appropriately. The communication between publishers and subscribers happens asynchronously and without a feedback for the publisher, that means a publisher creates an event (sends a data packet) and immediately resumes its control flow. It will never block because of the created event. In more general, P/S communication provides a decoupling of publishers and subscribers in terms of space, time, and synchronization. *Space decoupling* refers to the fact that a publisher does not need to know any other publisher or subscriber. *Time decoupling* means that publishers and subscribers do not need to actively participate in the interaction at the same time. Finally, *synchronization decoupling* means that a publisher will never block while producing an event, and that a subscriber can be asynchronously be notified of the occurrence of an event.

So P/S communication is very well suited for loosely coupled (distributed) applications in a dynamic, possibly large-scale network because it allows for an ad-hoc collaboration without the need for knowledge about the physical nodes, their addressing and connectivity. As it is a many-to-many communication in general, it obviously allows to model one-to-one or one-to-many interaction as well.

The publish/subscribe paradigm is not the only one suited for communication in distributed systems. Eugster et al. [EFGK03] presented a survey of different communication paradigms used for distributed systems and performed a comparison with the publish/subscribe paradigm. As alternative communication paradigms they explain *message passing*, *remote invocations*, *notifications*, *shared spaces*, and *message queueing*.

Message passing is a very basic interaction paradigm and nowadays rarely used directly because it does not provide much abstraction. It works asynchronously for the producer of a message and synchronously for the consumer. It does not hide the physical addressing and marshaling to the application. Both, the producer and consumer of a message have to be active at the same time. So we get a communication in which both sides know each other and are coupled in time and space.

Remote procedure calls (RPC) for procedural languages and *remote object invocations* (ROI) for object oriented languages are the most widely used interaction paradigm in current distributed systems. Three well known examples are SunRPC [Sun95] that is used in most UNIX-like operating systems, CORBA [Obj04b] that is used for a large number of object-oriented systems, and Java RMI [Sun04] used especially for Java-based systems. RPC/ROI systems hide aspects like the marshaling of messages and make remote invocations almost transparent to the user which makes them very appealing. Due to the synchronous nature of remote invocations they

introduce a strong time, space, and synchronization coupling. It should be noted that there are several attempts to break the synchronization coupling by using one-way calls (without return values or any other feedback), *future* or *promise* objects. That are objects that are returned as a result although their value will be set later.

Notifications are an interaction scheme that uses asynchronous messages in both directions. It is a distributed implementation of the so-called *observer design pattern* [BMR+96] where a consumer of an event directly registers its interest with the producer. If a consumer registers with a producer, it gives an implicit promise to notify the consumer of any state change. This can be regarded as a preliminary form of the publish/subscribe paradigm. Notifications provide a synchronization decoupling, but still are coupled in time and space.

Shared spaces like the *distributed shared memory* (DSM) provide hosts with access to a single virtual address space that actually is distributed over several hosts. Synchronization and communication take place using operations on shared data structures within the shared memory. *Tupel spaces* are a special class of shared spaces that allow to insert, remove, read, and write tupels in the shared space. An example for such tupel spaces is JavaSpaces [Sun02] that is a part of the JINI [Sun05] networking technology which is widely used in Java-based systems. Shared spaces provide time and space decoupling, but still have a synchronization coupling which limits its scalability.

Finally, message queuing systems are an interaction scheme that provides time and space decoupling, as well as synchronization decoupling for the produced side. As the name implies, we have a number of shared queues that can be seen as tuples with a timely ordered change history. Producers asynchronously append messages to a queue (usually a FIFO) and consumers retrieve them synchronously. The queue and its order is part of the system (middleware) and maintained independently from a producer or consumer.

Definitions

A publish/subscribe system (s. figure 2.1) logically consists of *publishers*, *subscribers*, *events*, and an *event service*. A publisher is an entity (mostly an application) that creates events whereas a subscriber consumes them. Each data packet transmitted is called an event. The event service denotes all software components and algorithms that manage to propagate events from publishers to subscribers based on their *interest*, i.e. a specification of the events that should be delivered to the subscriber.

Publish/Subscribe Models

The P/S communication paradigm does not include any guidelines on how a subscriber expresses its interest for events, or how the event service should work [EFGK03]. This allows for very different implementations. There are basically three subscription schemes (ways to describe a subscriber's interest) which leads to the distinction of *topic-based*, *content-based*, and *type-based* publish/subscribe systems.

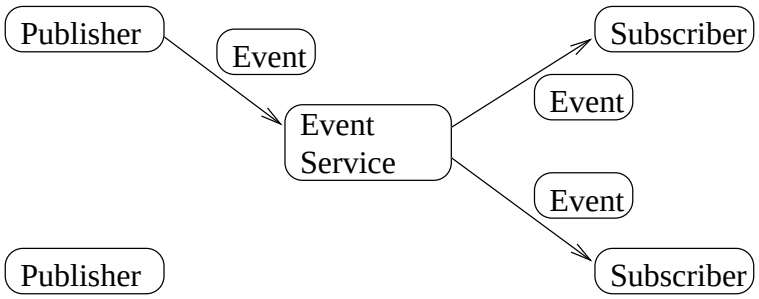

Figure 2.1.: Basic structure of a publish/subscribe system

In a topic-based (or subject-based) P/S system each event is augmented by subjects, i.e. human or machine readable keywords. The actual content is not of interest in this model. So it has strong similarities with *group communication* [Pow96]. A subscriber interested in a specific subject S can be seen as a read-only member of a group S whereas a publisher creating events with the subject S would be a write-only member of this group. Subject-based P/S systems can further be distinguished by the structure of the subjects, e.g. flat or hierarchical. But this shall not be discussed in more detail.

A content-based or *property-based* P/S system selects interesting events based on the whole content of the event. This is a much fine-grainer approach compared to a subject-based P/S system. Common ways to describe the interests of a subscriber is to use a subscription grammar like SQL or XPath. Such a description can than be used to parse an event and to determine if it matches the subscription. If the event is structured in a key-value-pair style, a template including required keys and/or key-value-pairs can be used to describe a subscriber's interest. Finally, if the event contains arbitrary unstructured data (from the point of view of the event service) a subscriber could describe its interest as a binary executable predicate object that is called for every event.

As third model we have the type-based P/S systems that form a special sub-class of the content-based P/S systems. In such a system an event is always an instance of a class and subscriptions are described based on types, or types with required values – similar to the template mechanism. This limits the expressiveness of the system but allows for a much tighter integration with object-oriented systems and languages. That means that syntax elements of an object-oriented programming language can be used to express the interest of a subscriber.

The three different P/S models are not complete complementary – they overlap in some cases. For instance, a content-based P/S system that uses a key-value-pair structure could also be modelled using a number of concurrent subjects. There are also a lot of variations that shall not be discussed further.

A more important aspect of a P/S system is the realization of the event service. It can be centralized or distributed. In a centralized system we have a dedicated network node that manages the subscriptions and collects all events from the publishers and forwards them to the appropriate subscribers. In a distributed event service this work is spread over a number of nodes. It can be partially distributed, that means that some but not all nodes implement the event service. An

example would be an overlay or backbone network with attached leaf nodes. Or it can be fully distributed which means that all nodes equally cooperate to implement the event service.

2.3. Motivating Application Scenarios

MANETs enable communication in a variety of situations. Many of them require a certain quality of service. In the following we will discuss different application scenarios and their importance that motivate the presented work.

Disaster relief

Disaster relief is one of the commonly cited scenarios for MANETs. Here a MANET serves two different purposes. First, it is thought to provide a temporary replacement for a destroyed, previously existing infrastructure. Second, it should aid mobile rescue personal in their work (e.g. by allowing voice calls, reliable transfer of status reports, possibly including high volume data like pictures or short video sequences). To provide a temporary replacement for a destroyed communication infrastructure a MANET is required to provide sufficient QoS (e.g. bandwidth, end-to-end delay, low packet loss) to support voice communication in addition to a best-effort Internet access. Especially to support rescue personal, the system has to be able to cope with the mobility and the thereby caused topology changes, although it can be assumed that the network will include a number of almost fixed stations (e.g. located at control rooms or emergency accommodations).

Sensor / Actor Networks

The second common application scenario cited for MANETs are sensor networks. Those networks consist of a possibly very large number of nodes that are deployed and then mostly remain stationary to observe their environment. Sensor networks that are only used to monitor environmental conditions usually use nodes with very limited capabilities. These nodes are out of scope of this work because they usually will not be equipped with 802.11 network interfaces and use different communication patterns. But in the area of industrial and home automation systems (e.g. SCADA systems) or for military command and control systems we will find mobile devices with sufficient capabilities. SCADA like systems will require only support for a low degree of mobility (e.g. mobile service personal) but have strong requirements for reliability and timeliness. This is especially true is they are also used to control the observed systems. Sensor/actor networks do not require us to provide QoS guarantees but are also an application that highly benefit from the publish/subscribe communication paradigm because data acquisition applications do not have to care about the deployed sensor nodes. Instead they only have to specify which data they are interested in. The same holds for the actors – an application needs only to specify which type of actor it requires to accomplish a task.

Office, Conference Networking

In an office environment, especially if the offices are locate in leased rooms or in protected historical buildings it can be too expensive or prohibited at all to deploy a wired LAN. So, a WLAN maybe the only feasible alternative to build up a network. The same holds for a network to provide Internet access at a conference venue for a short period (e.g. some days). Those networks will be used mostly to browse websites and to read EMails and so do not seem to require any QoS. But as most applications use TCP for communication we have to strive for reliable links with low packet loss rates. Otherwise users will experience a very low end-to-end throughput – this is a well known problem that has been discussed over the last few years. Admittedly, such networks are currently only a motivation to discuss the problem of reliable links in a MANET but not for publish/subscribe communication with the exception of instant messaging systems that are widely used.

Home networking

Home automation (e.g. the control of the heating installation or the shutters) have been in use for many years. These systems traditionally use a wired network or work over the power line. But a current trend is to install more WLAN-enabled devices. These devices include common multimedia devices like DVD players, settop-boxes, game consoles that exchange data with each other and the Internet, but also WLAN-based systems to transmit raw video and audio signals. There is even a noticeable trend for WLAN-enabled home automation and surveillance systems that can be controlled from the living room using a TV remote control. This goes that far that we already have microwave ovens and refrigerators equipped with WLAN interfaces to send a notification whenever a meal is finished or to place an order at your local delivery service if you run short of milk. Apart from the question if such devices make much sense their proliferation can cause significant problems. Current wireless video distribution systems are not prepared to cooperatively share the wireless medium with other devices. If we assume a home entertainment system that receives a video stream from the house's storage server or an Internet-based video on demand (VOD) service and forwards the video data to the TV we clearly see an urgent demand for a bandwidth management and medium access control. If we integrate more devices into our system without a coordination it very likely will collapse at some point because of the uncontrolled medium access. In this scenario we will also benefit from publish/subscribe communication because it eases the integration of several devices. All automation devices can publish their status or warnings without the need to know an actual receiver. This data can be received with an arbitrary device (e.g. the settop box, or a PDA) that can also be used to control them.

In this scenario we will see only a low degree of mobility but have requirements to support a mix of high volume multimedia streams and low volume status information and control data. Usually such a home information/control/entertainment system will cover only a small geographically area with a small number of hops.

2.4. Requirements

In this section we will take a closer look on all the requirements that are relevant for this work.

Functional requirements

Functional requirements are the requirements we desire of a problem solution. In our case this is the functionality that the communication system should provide, internally and externally. The *internal functionality* are algorithms, mechanisms, and protocols inside the communication system that are necessary for its interoperation with higher and lower layers on the same node and with remote nodes. The *external functionality* are the functions and protocols provided primary to higher layers, i.e. to the applications running on top of the communication layer and so for the end-user. The set of functionality that is provided, will be referred to as the *functional properties* of the system.

Non-functional requirements

Non-functional requirement in contrast refers to all requirements about the way the functionality is provided or a component is implemented. Examples are robustness, efficiency, or security (of the implementation itself not security functions provided for others).

Robustness in our case is the resilience of the system especially in situations that are not regarded as the normal mode of operation, e.g. system overload or unusual communication errors, or when components of the system have to handle invalid input. So the robustness is a value for the sensitiveness of a system to violations of the assumptions made for the normal mode of operation or the expected input data.

Efficiency is the ratio of processing done or output created to the amount of resources consumed for this task. Depending on the requirements the efficiency will be calculated with respect to the consumed CPU time, memory, or network bandwidth.

Security in our case should be differentiated into *software security* and *communication security*. Software security is a characteristic of the software to which extend it allows or actively restricts access to functions or data of a component for a specific user of the component. Communication security refers to the mechanism and protocols used to provide properties like confidentiality, integrity, non-repudiability, and anonymity.

In many cases, non-functional requirements arise from the target platform for which a software is implemented. An example is the communication interface. It can be synchronous or asynchronous. A preference for one implementation is a non-functional requirement. It does not change the provided functionality but can have a great influence on the behavior of the implemented system. So, the actual choice for an implementation strategy will be referred to as a *non-functional property* of the system. Such values like the CPU utilization or the processing time of a message will be referred to as *system properties*.

There are can be more non-functional requirements like extensibility, maintainability, configurability, and portability.

Extensibility is the ability to add new features to a component or the system without disturbing any existing code. This does not necessary mean that the added feature will not modifies the provided functionality. But it has to modify it in a pre-defined way without causing unexpected side-effects, i.e. modifications of the behavior that are not included in the design specification.

Maintainability is the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. This can hardly be measured as a value.

Configurability is a characteristic of the system that specifies in which way and to what extend it supports the rearrangement of features and modification of attributes of its components to create multiple variants of the system. The flexibility of a system, i.e. the range of conditions under which it can deliver the required functionality, highly depends on the ability to compose a system from different components and the availability of configuration parameters, that are external accessible attributes of a component that modify the internal behavior in a well-defined way.

Portability is the ease which with a system can be adapted to run on a new platform, i.e. the effort required to modify the system. A new platform can be a new hardware, operating system, or middleware – depending on which layer a system runs. The portability mainly depends on the requirements a system places on the lower layers. A software written in a high-level language is decoupled from the assembly code of the hardware and only depends on the availability of a compiler and a runtime-system and so is much more portable then a software written in the assembly language itself. If the software only uses common operating system functions, e.g. that functions specified in the POSIX (*Portable Operating System Interface*) standard, it is also better portable.

2.4.1. Functional Requirements

The developed this has to provide the following functional properties:

A publish/subscribe communication interface

Publish/subscribe communication is the primary goal of this work because it provides a convenient basis for the development of mobile applications.

Client/server communication interface

For applications that do not use the publish/subscribe communication paradigm but can be ported to the new platform, a conventional client/server communication interface has to be provided. It allows a step-by-step transition to the new middleware.

Legacy application interface

An integration of applications that cannot be ported to the new middleware should be supported, even if they cannot utilize the full functionality of the middleware. This is an important requirement to solve the chicken-or-egg problem.

Multi-hop routing

The developed communication protocols have to provide the possibility to communicate with nodes that are out of direct communication range and only reachable via some intermediate nodes.

QoS support

The developed communication protocols are required to support the specification and enforcement of quality of service requirements on an end-to-end basis. Namely these are bandwidth, reliability (loss rate), and delay.

2.4.2. Non-Functional Requirements

The implemented communication middleware has to fulfil the following non-functional requirements:

Portability

As we intent to use heterogeneous devices in the network the software should be portable. This includes two aspects. First it should only rely on portable system APIs and has to avoid platform specific abilities. Second, no transmitted data structure has to rely on a specific host byte order or alignment. A single external data representation has to be used and a local conversation between the internal and external representation has to be applied that respects local requirements. This is especially important to support platforms like ARM that do not allow unaligned memory access.

Efficiency

The implementation should avoid computational or memory expensive algorithms to allow a deployment on embedded / portable devices. Internal data should be stored and processed in machine-readable formats. Human-readable data representations like XML have to be avoided in the communication core and should be considered only for optional extensions – the system has to be usable without.

Configurability / Tailoring

The middleware software should be structured into independent components that can be composed to a system that only includes components necessary for a specific task. The composition should be possible dynamically, i.e. the system has to allow dynamic loading of components to construct the full system on-demand. The structure of network packets has to be configurable, i.e. should be defined depending on the component selection. Components providing an equal functionality (i.e. routing) have to use the same interfaces to allow an interchange of components that is necessary to be able to quickly integrate future developments and improvements.

Extensibility

The middleware should provide mechanisms to add extensions without the need to modify existing components. Especially it has to be possible to extend the headers of network packets without modifying other components. The extensibility mechanisms should be available through a well-defined interface to allow independent maintenance of third-party software components.

There are two properties that are intentionally excluded: security and energy consumption.

3. Related Work

A publish/subscribe system provides n:m communication. In principle this can be implemented using protocols providing 1:1, 1:n, n:m, or broadcast communication. So we will take a closer look on available communication protocols with and without QoS properties providing those relations. As we use a subject-based publish/subscribe model this has some similarities with service discovery in dynamic ad-hoc networks. But publish/subscribe communication is much more than simple service discovery so we will take an additional look on various publish/subscribe systems intended for different network architectures to study their properties and assumptions about the network.

3.1. Communication Protocols for MANETs

Research in the area of routing protocols is very active. In this section some protocols will be presented. Additionally their actual performance in practice and significant observations that contribute to this thesis will be discussed.

AODV (Ad hoc On-demand Distance Vector Routing)

AODV [PBRD04] is a reactive routing protocol, i.e. it will search routes only on demand. When a route is established it will be maintained as long as needed using a soft-state approach. That means it will expire automatically if not used for a specific time. AODV uses *Hello* messages to announce a node in its neighborhood and to determine connectivity. AODV ensures that a node will send at least one hello message within each time interval. If a node does not receive a hello message within a given time (default is twice the hello interval) it considers the link to the neighbor to be broken. To search for a route, AODV floods the network with *RREQ* message that record the routes they use to the target. The target replies with a *RREP* message that takes the reverse route. If the originator of the search receives the RREP it uses the discovered route. If it receives multiple replies it chooses the route with the shortest hop count.

AODV assumes symmetric links, that means if it discovers a route it assumes that the reverse route will be stable. AODV only considers the current connectivity among nodes, not the link stability or quality. Field tests [CBR02, Mul04] clearly show that AODV does not perform well in practice using its default behavior because it reacts too sensitive to lost hello messages. Furthermore it selects routes without considering the quality of their individual links which can

lead to a very low throughput. For large scale networks or high mobility it does not scale well because of the network-wide flooding.

There is ongoing research to add QoS features to AODV. The first approach, AODVQOS tried to transmit QoS requirements inside the RREP packets and intermediate nodes were only allowed to forward the request if they can provide the required QoS. This work seams to be discontinued as it inherits the scalability problems of the original approach. A newer approach, AODVng [MGPT02] tries a *Differentiated Services* approach that provides QoS properties based on traffic classes that are dynamically assigned on a per-hop basis. There are numerous other modifications that all vary certain aspects of this basic extension and so shall not be discussed.

DSR (Dynamic Source Routing)

DSR [JMB01] is a reactive protocol that discovers routes on demand and maintains them as long as they are needed. It is officially defined in [JMH03]. DSR distinguishes between *route discovery* and *route maintenance*. DSR is able to operate on networks containing unidirectional links but can optimize if they are assured to be bidirectional. It is explicitly designed to cope with mobility. A sender has full control over which routes to choose for the transmission. It works without any periodic transmissions like beacons for neighborhood discovery.

To discover a route, DSR floods the network with a RREQ (route request) message that includes a serial number and the address of the originator. If the destination node is found it replies using a known route back or by another RREQ flooding piggybacked by the RREP (route reply). It can also answer using the reverse route stored in the RREQ packet but only if bidirectional links can be assumed. The route maintenance works by using acknowledgements along the forwarding path. Each packet that is forwarded has to be acknowledged by the receiver. If not, it will be retransmitted several times. If all retransmissions fail, a route error message if send back to the originator with an indication of the broken link. The originator has to handle this error.

A DSR extension with QoS support is MP-DSR [LLP+01]. It basically tries to forward packets on multiple paths with certain end-to-end reliability requirements. MP-DSR assumes to know the probability for a successful transmissions between two arbitrary nodes and uses multiple disjoint paths (path that have the same source and destination but do not share intermediate nodes) to increase the combined end-to-ent transmission probability. It basically works by accumulating hop-by-hop transmission probabilities during the RREQ flooding and stops forwarding if it falls below a threshold defined by the source. In addition to the maintenance of each individual route it performs an end-to-end reliability repair. That means if individual routes break it automatically tries to establish new redundant routes.

OSLR (Optimized Link State Routing Protocol)

OLSR is a proactive routing protocol that works on the basis of defined links. But in contrast to a pure link state protocol where the link state information is flooded through the whole network

it selects multipoint relay nodes that are used to forward messages. OLSR uses a neighborhood discovery that periodically transmits beacons (*Hello* messages) that contain a list of known neighbors, the link state, and their multipoint relay selectors. With this information nodes can learn their 2-hop neighborhood. OLSR considers the existence of one-way links and checks the state of the link in both directions before it is declared a valid link. Based on the neighborhood information the multipoint relays are selected dynamically. Routing is done based on a hop-by-hop basis using the most recent local information. As OLSR is a proactive routing protocol each node ideally has full knowledge of the whole network. In return this makes the protocol sensitive for rapid topology changes caused by mobility of nodes.

FAST-OLSR is an extension to the protocol to handle individual fast-moving nodes. A node has to detect for itself that is moving fast. If so, it switches to the fast mode which means it will send *FAST-HELLO* messages at a higher frequency to faster discover its neighborhood. Nodes in fast mode cannot be selected as multipoint relays. That means that only a fraction of all nodes can switch to fast mode. Otherwise the whole network will collapse.

There is also a QoS extension called *QOLSR* [BA04, HBA04] which has been submitted as IETF drafts [BA05]. It uses the 20 bit flow label defined by IPv6 to store its QoS properties. The authors present a route selection algorithm that is based on Dijkstra's shortest path algorithm. It uses the available link bandwidth and delay to select routes. The authors claim that this even allows to give hard real-time guarantees using standard IEEE 802.11 devices. Given the non-deterministic behavior of the IEEE 802.11 MAC this is hard to believe although the presented results indicate such a behavior. A closer look on the evaluation process clearly shows the source for these results. First, there is no native implementation of the protocol available. All tests have been performed using simulations. In [BA03] the authors explain the simulation setup in more detail. They use the OPNET simulator but with a self-developed IEEE 802.11 MAC model that is described to be "very close to real operations" with simplified acknowledge and RTS/CTS schemes. The physical propagation model is defined to be distance-based (a circular, flat-earth model). The problems associated with this assumption will be discussed in section 4.5. A look on the routing algorithm shows that they do not consider problems like gradual link qualities, and more important interference.

The only available but very meaningful real-world results regarding the OLSR protocol come from a large OSLR installation in Berlin, Germany. An OLSR-based network consisting of more than 250 stationary access points (AP) provides free network access within several areas of the city. The project has been covered in [Sie05] and its status is available from the project homepage [ff]. One of the first observations was that standard OLSR performs significantly worst in the real-world than in a simulation – mostly because OLSR also chooses routes that actually where not able to reach a usable throughput. This could be easily explained – OLSR does not consider the link quality and so is not able to distinguish between good and bad (high and low throughput links). Only after the addition of a link quality measurement based on the ETX (expected transmission count) metric [CABM03] it became usable. This clearly shows the importance of a link-quality based routing approach.

IQRouting

Interference (see section 4.3.1) is one of the major problems one has to cope with in an MANET. QoS routing protocols traditionally used in wired networks very likely will not work in a MANET because they do not consider interference on the shared medium. Gupta et al. [GJTW05] present a routing protocol that explicitly considers the interference in a MANET. It uses two graphs, a connectivity and a conflict graph. The connectivity graph defines the connectivity among the wireless nodes. The conflict graph additionally specifies which nodes will be affected by interference. The routing algorithm now tries to find routes in a way that they cause a least possible interference. Usual shortest path algorithms tend to choose counterproductive routes. The author explain that also so-called shortest-widest path algorithms that try to choose the shortest path that is wide enough to cause minimal interference with existing paths are sub-optimal in the long run because they can be the reason for significant increased interference for further routes. So the newly proposed algorithm uses a number of different source-routing strategies and compares them using a heuristic that tries to minimize the overall interference. The used source-routing algorithms are:

- widest shortest path (WSP); take the widest path if multiple alternatives exist with a minimal hop count,
- WSP complement; remove the WSP in a graph freed by the links used in the original WSP,
- shortest feasible path (SFP), compute the shortest path feasible under the assumption of a simple 2-hop interference model,
- OSFP-like weighted path cost; assign a cost based on the available bandwidth to each link and choose path with minimal cost,
- shortest widest path; look for the widest path and choose the shortest if there are multiple alternatives.

The authors show that they could increase the admission rate by about 30% compared with other distributed routing algorithms. They additionally showed that this routing scheme together with a distributed admission control can be used to provide routing with bandwidth guarantees [JGWV05]. It should be noted that an inherent assumption of this work is that links in a wireless network have a statable range, i.e. if a node transmits a packet we have a constant loss probability up to a certain distance (the transmission range) and that it abruptly drops towards zero after this point.

3.2. Publish/Subscribe Systems

In this section we will take a closer look on some of the well known publish/subscribe systems that are also well documented and used for research projects. But this does not mean

that P/S systems are limited to the research area. There are also industrial-strength P/S systems available for many years. An example is TIBCO's Rendezvous [tib06] (originally named TIB/Rendezvous) that has been grown from a subject-based P/S messaging system into a fully-fledged distributed, event-based middleware platform that supports P/S communication among other interaction schemes. It claims to support a wide range of QoS properties from real-time message delivery to reliable transactions, but detailed information is not available.

Corba event / notification service

CORBA (*Common Object Request Broker Architecture*) [Obj04b] is a language-independent object model and specification for a distributed applications development environment. It includes an event service [Obj04a] that provides a simple event producer/consumer model. The CORBA event service uses an *event channel*, producers and consumers can connect to. Events are records and the event channel can be aware of the record structure (*typed channel*) or not (*untyped channel*). Most implementations use the untyped version. That means that every consumer that connects to the channel will receive all events submitted by any producer. The CORBA notification service [Obj04c] extends this event service by a content-based subscription mechanism that allows filtering of received events.

Siena

SIENA (*Scalable Internet Event Notification Architecture*) [Tar98, CRW01] is a publish/subscribe system that has been developed since 1998 and is still maintained. Its source code is public available [sie] under the GPL licences. This systems aims for a scalable wide-area P/S system. Events (named *notifications* in SIENA) consist of a set of typed tupels. An example event looks like (from [CRW01] fig. 2):

Type	**Name**		**Value**
string	class	=	finance/exchange/stock
time	date	=	Mar 4 11:43:37 MST 1998
string	exchange	=	NYSE
string	symbol	=	DIS
float	prior	=	105.25
float	change	=	-4
float	earn	=	2.04

Subscriptions are handled using *filters* and *event patterns*. The filters define which events a subscriber intends to receive. It looks similar to the event itself and consists of a set of attributes and constraints on the value of the attributes – so we get a content-based P/S system. Filters can use binary predicates on the values – $=$, $\neq$, $>$, $<$, etc. for numerical values; prefix ($> *$), suffix ($< *$), and substring ($*$) for string types. Event patterns provide a mechanism to select groups of events based on individual filters. From the sequence of events the system selects for each

filter the first event that matches and discards further matches for the same filter until the whole pattern is completed. The SIENA API provides the following functions:

- `publish(notification)` to publish a new event,
- `subscribe(identity, event pattern)` to register a subscription for a certain client,
- `unsubscribe(identity, event pattern)` to remove a subscription,
- `advertise(identity, filter)` to register a publisher,
- `unadvertise(identity, filter)` to remove a publisher.

SIENA does not provide any QoS guarantees at all. The service is declared to be best-effort without any special measures to ensure event order or reliability. SIENA uses either a partly distributed, hierarchical, acyclic peer-to-peer, or general peer-to-peer event server backbone. In the hierarchical topology one server is the *root server* and all other servers are associated with exactly one parent server. In the acyclic peer-to-peer topology servers are interconnected without cycles (this topology has to ensured by the operators of the backbone network). In the general peer-to-peer case cycles can appear. Each client is connected to exactly one event server but does not participates in the event brokerage process. The used topology has to be defined beforehand and will be used by the system to apply routing optimizations. Event forwarding is done using a shortest-path routing with an event filtering and replication on the server backbone. That means, publishers and subscribers propagate their event filters in the network that decides how far an event has to be forwarded. SIENA relies on a static server backbone but can tolerate single server failures in the general peer-to-peer topology.

READY

READY [GKP99] is an synchronous event notification system developed at AT&T Labs–Research and can be regarded as a content-based P/S system. It does not strive for scalability, instead it is certainly one of the first P/S systems that explicitly address the problem of QoS guarantees. READY is designed to work on wired networks using TCP/IP and a reliable broadcast protocol. It uses a hierarchical, partially distributed event service. For the reliable broadcast READY uses IONA's OrbixTalk [ION02], another P/S system. READY supports the QoS properties specified for the OMG Notification Service [Obj04c]. These can be distinguished into persistence related properties and event related properties. The persistence properties control how the system reacts to error conditions and stores temporary data:

- *None*: event may be lost due to node, link, or process failures
- *Retry*: individual message losses are handled by retransmissions

- *Reconnect*: retry plus re-establishing of broken connections
- *Persistent Session*: all information like QoS settings, established sessions is recorded to persistent storage
- *Persistent Event*: all events are recorded to persistent storage

The event related properties influence the delivery of events:

- *Priority*: a static priority that defines the order of events
- *Validity*: a time-to-live value defined as relative or absolute time value that controls when events should be discarded
- *Ordered*: buffer messages in the system in a defined order

It should be noted that READY has been discontinued and superseded by OrbixTalk that fulfills the OMG Notification Specification itself in the current version.

Elvin

Elvin, a content-based routing/messaging system [SA, FMK+99, SAB+00] has been developed since 1995 at the University of Queensland, Australia and is still maintained [elv]. It is available under a commercial license and free of charge for academic and research projects. Elvin uses a partially distributed, general peer-to-peer system with explicit event broker servers. Clients are connected to exactly one event broker. To ease the connection process, Elvin supports a IP multicast-based server discovery process. Events in Elvin consist of a set of typed and named values (similar to the events in SIENA). Subscriptions are also very similar. They consist of a set of attributes and constraints on them. A difference to SIENA is the support for POSIX Extended Regular Expressions (ERE) as defined in POSIX 1003.2 for all string values. The server backbone relies on stable TCP connections but provides the ability to use encryption on them. Elvin does not requires a publisher to register before publishing an event. Instead it allows a publisher controlled *quenching*. A publisher can provide the event broker backbone network with additional filters that control which subscription updates have to be handled by the distributed subscription database. These quenching filters include sets of attribute names that have to be present in a subscription – a kind of positive list of tuples the events created by a publisher will include. Elvin does not support any QoS guarantees but instead focuses on scalability and security.

Hermes

Hermes is an event-based middleware that supports publish/subscribe communication [PB02]. It has been developed by Peter Pietzuch as part of his PhD thesis and aims for an Internet-scale

communication system. Hermes puts a special focus on the integration with applications so that they can utilize the event-based nature of the middleware. Hermes uses a partially distributed event brokerage system, i.e. a peer-to-peer network of event broker servers. Both, publishers and subscribers are connected to exactly one of the servers. The backbone system relies on stable TCP connections between the nodes but can handle individual node and link failures. The server backbone is structured as an overlay network similar to Pastry [RD01], a distributed hash table. This overlay is used to define *rendezvous* nodes for every event type that are known to the publishers and subscribers of the event type to build an event dissemination tree. The overlay additionally provides a reliability mechanism to cope with individual node and link failures. Hermes supports a type-based event system to allow a tight integration with object oriented languages. This system includes support for type hierarchies and inheritance. The subscription process requires two steps. First publishers and subscribers have to register a new type in the system. This defined the rendezvous node that will handle this event type. Afterwards publishers use an advertisement and subscribers a subscription message that they send to the rendezvous node. The event routing is done using a shortest-path algorithm on the neighborhood relationship defined by the overlay network. Hermes provides support for QoS (reliability and timeliness) that is build on-top of the basis system. Lower layers do not consider QoS guarantees at all. To ensure the reliability and fault tolerance, Hermes uses a heartbeat protocol between the event brokers to detect node and link failures and a replication of the subscription information stored on the rendezvous nodes. The authors admit that it is an open problem to keep the subscription information consistent at all time, even for rare link and node failures. So it is well suited for a large wired backbone but not for mobile systems.

Herald

Herald [CJT01] is to successor project of Hermes. Herald inherits Hermes design principles, i.e. it uses Pastry for the broker overlay network and a type-based publish/subscribe system. Herald introduces no additional QoS properties but strives to improve the resiliency, scalability of the system and its ability to self organize. Herald uses replication of subscription data to ensure fault-tolerance. But it limits the amount of subscription data that a single event broker will store. The system allows to perform a load balancing among rendezvous nodes. That means, if a rendezvous node notices a significant imbalance in the assignment of event types to rendezvous nodes it can delegate new type registrations to other rendezvous nodes. Herald additionally allows for a migration of connected clients to replicated rendezvous nodes. The automatic control of the imbalance detection and migration is still an open question. Herald additionally addresses the problem of the synchronization of the replicas because this was one of the major limitation in the Hermes system. Herald does not actively synchronizes replicas of rendezvous nodes. Instead it requires periodic refreshes of the type and subscription information by the clients and accepts temporal inconsistencies in the time between a backbone reconfiguration (joining, leaving server, or modification of replica assignment). Herald extends the reliability of message deliveries by a persistence property – nodes that loose their connection to the network for a short period are able to retrieve a history of missed events. Of course this can violate a guaranteed end-to-end

delay but is sufficient for mobile clients using the best-effort event delivery. Future research in the Herald project will focus on the security and stability of the system, namely the replication process of rendezvous nodes in the case of a small number of malicious nodes, or the significant change of the assignment of types and clients to rendezvous nodes and the amount of events they have to handle which can be caused by a "sudden fame" of an application or service using the network. Both effects are very common on the Internet and so have to be considered for a system of that scale.

Gryphon

Gryphon started as a research project at IBM's T. L. Watson research center in 1997 [gry] and has grown into an industrial-strength event broker that has become a part of IBM's WebSphere suite as the WebSphere MQ Event Broker [wsm]. The Gryphon event broker has proven its ability for large event dissemination at major sport events like the Olympic Games in Sydney (2000) or the Wimbledon tennis championship (2001) with more than 100.000 concurrently connected clients.

Gryphon is a mature publish/subscribe system that provides a Java Messaging Interface (JMS) with a topic- or content-based subscription scheme. Gryphon uses a fixed, partially distributed redundant event broker network but is able to automatically detect and handle node and link failures. The system is based on a logical (global) information flow graph that specifies the delivery of events from producers to consumers [BKS+99]. This graph can be altered using filtering (using the announcements and subscriptions) and transformations (aggregation of filters) and is mapped onto the physical broker topology. For the event dissemination the broker network implements a multicast technique [BCM+99] called *link matching* that aggregates subscriptions on the brokers to decide if an incoming event needs to be forwarded over one or more outgoing links. Therefore the matching algorithm [ASS+99] performs an optimized partial matching of the event content and stops immediately if a definitive decision can be made.

Gryphon guarantees exactly-once delivery of event as required by the JMS specification. Events are persistently kept in the system, that means events for temporary disconnected subscribers will be kept timely ordered on the last broker until the client reconnects. As long as a client is connected it is guaranteed to receive a gapless, timely ordered flow of events. Apart from the event order and persistence, Gryphon provides no other QoS guarantees.

JEDI

JEDI stands for *Java Event-based Distributed Infrastructure*. It is an content-based publish/subscribe system implemented in Java [CNF01]. Events in the JEDI system have a name and a set of named attributes. Subscriptions are specified using a simple pattern matching language. As there is nothing like the advertisements in other P/S systems, subscriptions are propagated in the whole network to produce a global knowledge. JEDI does not use a single dissemination tree for all events like the previously discussed systems. Instead it dynamically builds individual trees based on the concept of *Core Based Trees* [BFC93]. The event broker system is partially distributed,

i.e. again we have an event broker network. This broker network uses another overlay network for its internal communication and coordination in addition to the event dissemination trees.

JEDI explicitly addresses mobility. A subscriber that wants to disconnect temporary from an event broker can suspend its subscriptions using `moveOut` and reactivate it using `moveIn`. The event broker will store the incoming events until the node reconnects. This is similar to the persistence feature in Gryphon but on an on-demand basis. There is no automatic process to handle mobile nodes or anything like neighborhood or topology discovery. JEDI additionally provides the `move` command that can be used to migrate code from one client node to another e.g. to implement mobile agents. JEDI does not provide any QoS guarantees other then the event order and limited persistence. The communication in the JEDI network uses TCP between the clients, and the event brokers (called *Event Dispatcher*) to allow for non-Java clients and Java RMI among the event dispatchers.

XMLBlaster

XMLBlaster [Ruf00] is a *Message Oriented Middleware* (MOM) that provides a topic-/subject-based P/S interface. It has been under active development since 2000 and is licensed under the LGPL (Lesser GNU General Public License). Events in XMLBlaster are defined in XML and consist of three parts – the key, the content, and optionally a specification of QoS requirements. The key includes the topic, a unique string identifying the content, and an arbitrary number of fields with detail information about the properties of the content. The content itself can be anything from an XML document to binary image data. The QoS specification includes an arbitrary number of XML encoded attributes that describe the QoS properties (e.g. the time since the creation) and requirements for each event (e.g. persistent storage, transport security). Currently QoS support is limited to the persistence of events which is realized using relational databases. The XMLBlaster engine does not handles QoS itself, instead it delegates it to specialized plugins (e.g. an encryption module). Subscriptions can be made on the topic of events (*Exact Subscription*) or using XPath expressions on an arbitrary number of fields in the key of an event (*XPath Subscription*). XMLBlaster is designed to use different underlying communication protocols for event delivery, including TCP/IP, SMTP, HTTP, Java RMI, XMLRPC, SOAP, and Corba. This list can be extended by a plugin mechanism. It supports security features like encryption and authentification.

Miscellaneous

So far we discussed only P/S systems that are primary focused on wired networks. We saw the mobility support in the JEDI system but this is only a method to inform a system of a mobile client. We did not see any build-in support for mobility or a fully distributed event service running on a MANET. This does not mean that there is not such system. Quite the contrary, research in this area is very active as the publish/subscribe paradigm is very well suited for communication in such an environment. But currently we see only research prototypes.

Huang and Garcia-Molina present an overview of the currently used techniques [HGM04]. There main observation is that all currently used systems that support mobility to some degree, rely on a fixed broker network and hence are not suitable for MANETs. We have systems that use centralized and distributed event services that are independent from publishers and subscribers. That means clients are only users of the event service and not an active part. Clients are usually connected to exactly one event broker using a unicast link. Within the event broker network we see two basic protocols – distributed broadcast like in the SIFT [YGM99] system, or distributed multicast that is used in most systems.

This is not very surprising as we currently do not see any widely deployed MANET. With the proliferation of third-generation (3G) mobile phones that are focused more on data-driven services than traditional voice communication we will see more P/S-based services. These will still be based on a fixed infrastructure backbone but will provide a playground to develop services that can be later extended to MANET systems.

Cugola and Jacobsen in [CJ02] present an overview of the challenges and possible directions in the development of services for mobile telecommunication systems. For such a system the authors identified the following expected requirements:

- management of mobility of application components,
- management of changes in the underlying network topology (for ad hoc networks),
- scalability to millions of information consumers (subscriptions),
- scalability to a large number of information providers,
- propagation of events to a large number of consumers simultaneously,
- management of high volatility of users interests (“sudden fame”),
- processing / conversion of divers content formats,
- support for heterogeneous notification channels (end-user technologies),
- support for “approximate subscriptions” and “approximate events” to increase flexibility of the systems to handle fuzzy user demands,
- support for high available event dispatching systems,
- support for accounting, pricing, and advertisement,
- support for authentication and secure content delivery.

The authors argue that currently no system fulfills a significant set of these requirements because they are either focused on wired networks or are research projects without business and/or practice relations. They present the Toronto Publish/Subscribe System (ToPSS) [ALJ02], a content-based P/S system with a centralized event service that would be applicable for a mobile telecommunication system. The system is designed to support HTML-based content. Based on this core

system the current development focuses on the addition of XML support, approximate matching, mobility support, location awareness, etc. (s. [top] for more details).

Cugola also co-authored two other papers [CPM02, CN01] that discuss improvements of the JEDI system to overcome the problem of the reconfiguration of the event dissemination trees. JEDI as discusses earlier only supports (to some degree) mobile client nodes. With a reconfigurable backbone it would become applicable for mobile ad hoc networks. It would additionally allow to add and remove event brokers dynamically. The paper discusses that we have to deal with two different cases – the removal of a link and the replacement of a link. A replacement would be the case if a link breaks but there is a nearby alternative (however this is found). The default behavior to deal with a link break is the synthesis of unsubscription events to be send along the disconnected part of the dissemination tree. It is obvious that such a reaction is suboptimal if there is a chance for a local reconfiguration of the tree because it will waste a lot of messages. The reconfiguration works quite simple. If a link break is detected and an alternative link is found, one end of the broken link builds a connection over the new link to the other end to initiate a tree merge operation. This redirects the traffic over the new link and updates the subscription tables. This operation requires lossless connections with ordered messages on each link which is true for JEDI as it uses TCP for the connections. The authors omit the answer how to find the alternative link and refer the different MANET routing protocols that have to solve this problem too and so delegate the problem which can be sufficient if the routing layer provides a method to perform just a route discovery with feedback to the higher layer.

Fiege et al. [FGKZ03] present an improved version of the REBECA P/S middleware, a research prototype to provide the publish/subscribe paradigm for location-dependent applications. Rebeca uses an almost static event broker backbone as many other systems. They extend events by a location attribute that has to match the location of a client to take effect. Mobility of clients is handled by the broker network, clients do not have to announce their movements as in the JEDI system. The broker network forwards events based on the subscriptions stored on all broker nodes. Clients can receive events that are not valid at a time because they are out of the target area – those events will be filtered by the clients. To handle mobility, the system uses a proactive approach. It assumes a mobility graph, a per-client graph that includes all possible brokers the client could move to, given a maximum speed and a constant time interval. Every time a node attaches to a new broker, this graph is build and the subscription is forwarded to all brokers in the mobility graph. These brokers now have the time to update their subscription databases. As a result events are delivered to the broker the client is currently attached to and to all brokers nearby. If the client moves to a nearby broker (it is a basic assumption that it cannot move to other brokers) it already provides the client with fresh events. This will waste a significant amount of bandwidth but allows for a continuous event delivery under ideal conditions. If the client attaches to a new broker, this will create an automatic unsubscribe message for the previous one. The authors show that this scheme produces a significant amount of traffic but is still some oder of magnitude better then a pure flooding-based approach. Additionally they show that it becomes more efficient for a lower degree of client mobility (event brokers are not allowed to move). It should be noted that if one would omit all event parameters except for the location this would be come a location-based multicast routing protocol.

Apart from the fact that we do not see widely deployed MANETs for testing, the main problem for a P/S system in such a network is the management of subscriptions and the dissemination tree. The usual approach to use a single dissemination tree for all events does not scale well if we consider mobility. Furthermore it is an inappropriate basis to provide end-to-end QoS properties. But there are many approaches to solve individual problems in such a system.

Anceaume et al. [ADGS02] presented a distributed tree maintenance algorithm that can be used for single- and multi-subject trees. Its main focus is the *anonymity of the publishers* – QoS is out of scope of this work. The algorithm requires a neighborhood discovery (that is not further specified) and stable links between nodes – message loss is treated equivalent to a complete link break which is an unrealistic assumption in a MANET and would cause permanent tree reconfigurations. The proposed algorithm is only discussed on a theoretical basis and abstracts totally from actual problems in a MANET, but given the assumed conditions, it would provide an acyclic graph that is the basis for many of the protocols used in backbone-based systems.

Aguilera et al. [AS00] present a distributed solution for the problem of *economical atomic broadcast*. An atomic broadcast system allows consumers to receive messages in a total global order. It is call economical if it requires messages only to be send if an event is transmitted, i.e. clients should not send any messages to comply with this definition. Their approach relies on approximately synchronized clocks of all producers (e.g. by using the GPS time signal) and the knowledge about the approximate rate of published events on all nodes. Furthermore the algorithm requires reliable links between nodes. The authors present two algorithms for a deterministic merge of the event sequences from the producers that only run on the consumer nodes to preserve the global order of messages. They claim that running the deterministic merge on all nodes results in an atomic broadcast. The merge algorithm is described to be resilient to small clock skews and varying transmission delays at the cost of event throughput. The authors present an theoretical prove of correctness of the algorithm that is sound but never tried it in a simulated or real network. They also show how this algorithm can be applied to multicast communication. Considering the usual latencies experienced in a MANET the algorithm will not scale well but could be useful for communication in a small region if we can ensure reliable links.

A P/S middleware explicitly designed for communication in MANETs is STEAM [MC02]. In STEAM all nodes participate in the event service, but nevertheless is not a fully distributed event service. STEAM builds on the concepts of *proximity groups* [KCM+01]. That are groups of nodes within a geographical region that are build under functional aspects, i.e. several groups can exist in the same region to solve different tasks. STEAM supports three type of event filters – subject filters for a coarse-grained event selection, content filters for a fine-grained event selection, and proximity filters. Subject and proximity filters are installed on the producer nodes to decide if an event needs to be propagated. The client node performs the fine-grained and probably time intensive content-filtering.

Within the CORTEX (*CO-operating Real-time senTient objects: architecture and EXperimental evaluation*) project [cor] the STEAM middleware has been extended to provide real-time delivery of events within a proximity group [HC03b]. The real-time guarantees are realized using

a deterministic TDMA-based MAC protocol [CC02] that provides time guarantees for single-hop communication. On top of this MAC layer an area-based bandwidth admission control and deadline-driven scheduler control the transmission of events on the medium [HC03a]. To increase the coverage area of such a network events have to be forwarded over multiple hops. This functionally is provided by a middleware layer that includes proactive routing functions, a prediction model to anticipate the changes caused by the node mobility, and a resource reservation component to care about the local resources (CPU and memory) of the client nodes.

Summary

From the presentation above we see that publish/subscribe system have been in use for many years. Traditionally they are intended for wired networks where a common strategy is to deploy one or more event broker server that build up a logical backbone (overlay) network to serve a large number of clients. The main focus for most traditional systems is to provide a scalable system that can serve a large number of clients (publishers and subscribers). The primary requirement on the service quality is reliability and temporary persistence of events, secondary we sometimes have requirements on the timely delivery of events. The delivery of multimedia streams or other bandwidth sensitive information seem to be out of scope of these systems or bandwidth is assumed to be available in a sufficient amount. For wired networks, especially with event brokers placed on the backbone of large network providers this is an acceptable assumption even for a large user base. Table 3.1 provides a short overview over the discussed systems.

System	**Network type**	**Event type**	**QoS properties**
CORBA event service	partially distributed	none, type	none
CORBA notification service	partially distributed	type, content	persistence, priority, timeliness
ToPSS	centralized	content	none
Siena	partially distributed	content	none
Elvin4	partially distributed	content	none
Hermes	partially distributed	type	timeliness, reliable delivery
Herald	partially distributed	type	timeliness, reliable delivery, persistence
Gryphon	partially distributed	subject, content	order, persistence
JEDI	partially distributed	content	order, persistence
READY	partially distributed	content	priority, live time, persistence
XMLBlaster	partially distributed	subject	persistence, (timeliness, security)
STEAM	distributed	subject, content	none

Table 3.1.: Publish/subscribe, event notification system overview

For wireless networks experience with publish/subscribe communication is still limited. Systems like ToPSS and SIENA have been extended to support mobile clients connected to a fixed backbone network. STEAM is the first well known system that is explicitly targeted for mobile ad hoc networks and with extensions it is able to provide real-time communication for local groups. But nevertheless we see a broad range of research focused on basic aspects of the communication in a MANET – ranging from a deterministic medium access to routing protocols with multicast and/or QoS support.

4. Wireless Communication in an IEEE 802.11 Network

In this chapter we will take a closer look on the unique nature of wireless networks – especially WLAN as defined in the IEEE 802.11 standards [Ins99]. The chapter starts with a description of the standard followed by a discussion of different ways to model the behavior of a WLAN.

4.1. The IEEE 802.11 Standard

The term 802.11 or *WI-FI* denotes a series of standards for Wireless LAN developed by the working group 11 of the IEEE LAN/MAN Standards Committee (IEEE 802). The working group 11 has several task groups that develop different aspects of the WLAN standard – each denoted by a single letter. Table 4.1 depicts the currently known task groups. Within this thesis only the standards 802.11a/b/g are relevant.

In the following some important aspects of the relevant standards will be presented.

IEEE 802.11 describes two basic network setups – the *infrastructure* and the *ad hoc* mode. In the infrastructure mode we have wireless nodes connected to base stations with a (usually wired) interconnection between each other. In the opposite we do not have such a backbone network when working in ad-hoc mode. Here every node directly communicates with other nodes in its transmission range. A communication with nodes outside of the direct transmission range is only possible by multi-hop transmissions, i.e. intermediate nodes have to forward packets using an arbitrary routing protocol.

The IEEE 802.11 standard specifies the physical as well as the MAC layer (in the sense of the ISO/OSI layer model). The relevant detail will be discussed in the following. WLAN used two frequency band at 2.4 GHz and 5 GHz. Both can be used license free with some restrictions – most noticeable, the transmission power has to be at most 100 mW.

The standard defines two basic access schemes for the physical layer – the *Point Coordination Function* (PCF) and the *Distributed Coordination Function* (DCF). PCF defines a polling scheme using a central coordinator but it is actually not used by any hardware implementation. So only the DCF will be regarded in the following. The DCF basically is a *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA) mechanism. It can be briefly described as follows: a node desiring to transmit first senses the physical medium. If the medium is sensed as free (idle) the node waits for a random time (back-off time) and then transmits its frame if

Task group	Finished	Description
802.11	1999	The original 1 Mbit/s and 2 Mbit/s, 2.4 GHz RF and IR standard
802.11a	1999	54 Mbit/s, 5 GHz standard
802.11b	1999	Enhancements to 802.11 to support 5.5 and 11 Mbit/s
802.11c	2001	Bridge operation procedures; included in the 802.1d standard
802.11d	2001	International (country-to-country) roaming extensions
802.11e	2005	Enhancements: QoS, including packet bursting
802.11F	2003	Inter-access point protocol
802.11g	2003	54 Mbit/s, 2.4 GHz standard (backwards compatible with 802.11b)
802.11h	2004	Spectrum managed 802.11a (5 GHz) for European compatibility
802.11i	2004	Enhanced security
802.11j	2004	Extensions for Japan
802.11k		Radio resource measurement
802.11m		Standard maintenance
802.11n		High throughput
802.11p		Wireless access for the vehicular environment
802.11r		Fast roaming
802.11s		ESS mesh networking
802.11t		Wireless performance prediction (WPP)
802.11u		Interworking with external (non-802) networks
802.11v		Wireless network management
802.11w		Protected management frames

Table 4.1.: IEEE 802.11 task groups

no other node started to transmit in the meantime. To detect collisions a positive acknowledgment scheme is used and the MAC layer tries to correct packet loss using a certain number of automatic retransmissions.

WLAN uses two different mechanisms for the carrier sense, a physical and a virtual carrier sense. The physical carrier sense detects activity on the medium if the signal level is above a given threshold (s. section 4.2 for more information). The virtual carrier sense in contrast is implemented as part of the MAC protocol. Each packet contains a field called *Network Allocation Vector* (NAV). This field tells other stations how long the sender will use the medium. During this time they will assume the medium to be busy. The NAV locks the medium for a certain time (specified in μs). This time can be longer than the actual transmission of the packet containing the NAV to allocated the medium for a following packet.

The 802.11 MAC layer defines different timing constants (*inter frame spaces (IFS)*) ([Ins99], section 9.2.3) for the transmission of frames on the medium, and a timing granularity denoted as *Slot Time*. The different IFSs are independent of the stations' bit rate even if the transmitter is multi-rate capable, i.e. can switch the transmission bit rate dynamically. They define the gaps on the medium between frames of different types. Figure 4.1 depicts the IFS timing.

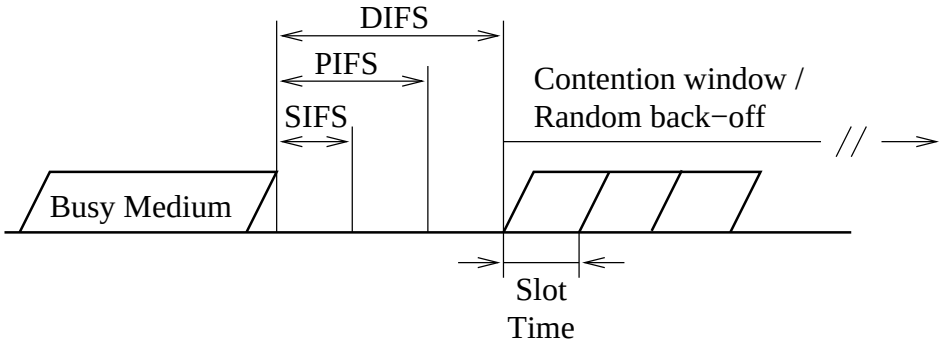

Figure 4.1.: 802.11 MAC timing

The following IFSs are defined:

- **Short interframe space (SIFS)**: The SIFS is used for ACK (acknowledgment) frames, CTS (clear to send) frames, and subsequent fragments of a frame burst. Because the SIFS is the shortest IFS, those packets are prioritized over all other packets, i.e. are send before them.
- **PCF interframe space (PIFS)**: The PIFS is used by stations operating in PCF mode to gain access to the medium at higher priority then stations operating in DCF mode. The PIFS is defined as $PIFS = SIFS + SlotTime$.
- **DCF interframe space (DIFS)**: The DIFS is used by stations operating in DCF mode to transmit data frames and management frames that are not subject to the SIFS. The DIFS is defined as $DIFS = SIFS + 2 * SlotTime$.

- **Extended interframe space (EIFS):** The EIFS is used by stations if they receive an erroneous frame to resynchronize with the virtual carrier sense scheme. The EIFS is defined as $EIFS = 2 * SIFS + 2 * SlotTime + t_{ACK}$ with t_{ACK} beeing the time required to transmit an ACK frame.

The table 4.2 shows timing values for different 802.11 standards.

Standard	SIFS [μs]	SlotTime [μs]
802.11b	10	20
802.11b/g (mixed mode)	10	20
802.11g (g only)	10	9
802.11a	16	9

Table 4.2.: 802.11 timing values

If a node senses the medium to be idle it is allowed to send. But it will not start immediately, instead it waits and continues to sense the medium for a random time (back-off time) and then starts to transmit if the medium is still idle. This back-off time is necessary to reduce the chance of a collision because it can be assumed that other stations are waiting too for the medium to become free and would start to transmit in the same moment. In contrast to an Ethernet device a WLAN device cannot sense the medium while it is transmitting to detect a collision. The reason is that due to cost savings WLAN devices are only equipped with a half-duplex transmitter. So, in order to detect a collision, WLAN uses a positive acknowledgment scheme. For unicast transmissions, that are packets intended for a single receiver, the receiver sends a small acknowledgment frame (ACK). If the sender does not receive this acknowledgment frame within a given time it will retransmit the packet several times. If all retransmissions (usually 4) fail, the node will drop the packet. It additionally increases the back-off window size, i.e. the maximum range the random back-off time is chosen from. Obviously this retransmission scheme will not be used for broadcast transmissions, i.e. packets intended for all nodes within transmission range. The back-off time is only chosen once for each packet to transmit. If the node has to defer its transmission because another node started earlier, it will use the remaining back-off time when the medium become idle again. This ensures a fair access to the medium.

4.2. Modelling the WLAN Propagation

This section discusses how a WLAN system can be modelled. The development of communication protocols for a WLAN system cannot be solely done using practical experiments. We additionally need a model to predict the behavior a priori. If we have a model that closely predicts the real behavior, we can test the protocol behavior in a lot of different environments – even in those that have very uncommon properties. But we also cannot completely perform the protocol development without experiments in the real world, although this is a common approach (or better to say mistake) within the research community (the resulting problems will be discussed

in section 4.5). We need experiments to verify our model and to determine the effects that the model cannot predict. Only this way we can create confidence in the results that we create.

Wireless LAN uses electromagnetic waves to transport energy and information. This propagation can be formally described using *Maxwell's equations* (s. figure 4.2) that describe the behavior of electric and magnetic fields and their interaction with matter. They provide a highly abstract way to describe the propagation of electromagnetic waves and so are impossible to use in most cases. The main reason is that usually one does not have enough knowledge to describe the environment in which we try to calculate the propagation. So we are required to find much simpler ways to describe a WLAN system. To do this we have to abstract from the real world by placing assumptions about the environment we are considering. This way be create a *model* of the real world.

$$\begin{aligned} \nabla \cdot \mathbf{D} &= \rho \\ \nabla \cdot \mathbf{B} &= 0 \\ \nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\ \nabla \times \mathbf{H} &= \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} \end{aligned}$$

Figure 4.2.: Maxwell's equation

The starting point is a model of the wireless communication system that we are using (s. figure 4.3). Our communication system model includes two *transceivers* (a combination of a *transmitter* and a *receiver* component) with one antenna each, and a wireless communication channel.

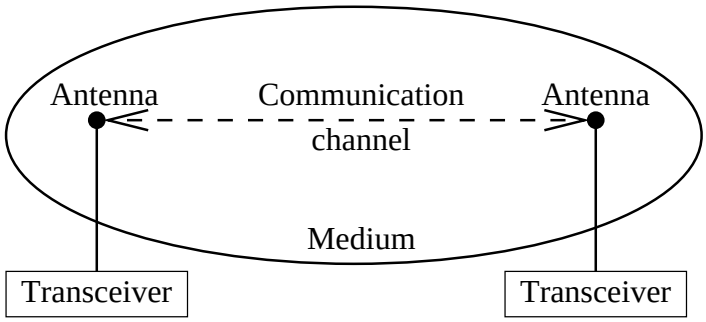

Figure 4.3.: Simplified WLAN communication system

The transmitter creates a signal with a given transmission power P_T. This signal is than emitted using an antenna that can amplify (positive amplification) or attenuate (negative amplification) it. This amplification is given as a constant factor (*antenna gain*) G with different values for the transmitter (G_T) and the receiver (G_R). The emitted signal propagates the information trough the communication medium and reaches the receiver antenna with the signal power P_R. The

medium causes an attenuation A of the signal due to various effects that will be explained later. If the signal power is above a given threshold (P_{CS}) the receiver detects the wireless medium as *busy*, i.e. currently in use by a neighboring transmitter. This detection of other stations is called *Physical Carrier Sense*. Additionally the receiver defines a second power threshold P_{RX} (with $P_{RX} > P_{CS}$) for the signal reception. If the received signal power is above P_{RX} the receiver starts to demodulate the signal to extract the transmitted information. To explain the reason for the existence of these two separated power thresholds we have to consider the *noise* at the receiver side. In addition to the signal of the sender the receiver also receives various other signals on the same part of the electromagnetic spectrum from natural and artificial sources. The sum of all these signals that do not belong to the sender is called noise (with the power P_N). The ratio of signal and noise is called *Signal-to-Noise Ratio* (SNR). The higher this ratio is, the better the signal can be distinguished from the noise. This directly relates to the probability that the information on the sender and receiver side are the same. As we use a digital system our smallest information unit is a bit and we define the probability of a wrong reception as *Bit Error Rate* (BER).

$$BER(P_R) = f_{Modulation}(SNR) = f_{Modulation}\left(\frac{P_R}{P_N}\right) \tag{4.1}$$

The exact BER function depends on the modulation scheme used for the transmission (s. [Lan05] for details). But independent from the modulation scheme this function is monotonic and the BER decreases with increasing SNR. That means $P_{RX} > P_{CS} \Rightarrow BER(P_{RX}) < BER(P_{CS})$. Usually P_{CS} is chosen so that we can distinguish a transmitting station from the background noise with a high probability but $BER(P_{CS})$ is too high to extract the original information. Thats way we set the reception threshold to a value that results in a BER low enough to extract the original information.

For better readability amplification factors are specified in *Decibel* (dB) and absolute power values as dBm – the power relative to the reference value of 1mW. We define

$$\frac{P_1}{P_2} = 10 * log\left(\frac{P_1}{P_2}\right)[dB] \tag{4.2}$$

$$P[dBm] = 10 * log\left(\frac{P}{1mW}\right) \tag{4.3}$$

In the following we will discuss how to model the different components of the WLAN communication system.

Modulation scheme model

Current WLAN adapters use a variety of modulation schemes to transmit information with different bandwidths. BPSK (Binary Phase Shift Keying) for 1 Mbps, QPSK (Quaternary Phase Shift

Modulation	Speed [Mbps]	Sensitivity [dBm]
BPSK	1	-94
QPSK	2	-91
CCK-DBPSK	5.5	-89
CCK-DQPSK	11	-85
OFDM-BPSK	6	-85
OFDM-BPSK	9	-84
OFDM-QPSK	12	-82
OFDM-QPSK	18	-80
OFDM-16QAM	24	-77
OFDM-16QAM	36	-73
OFDM-64QAM	48	-69
OFDM-64QAM	54	-68

Table 4.3.: Cisco Aironet350 Receiver Sensitivity

Keying) for 2 Mbps, CCK (Complementary Code Keying) for 5.5 and 11 Mbps, and OFDM (Orthogonal Frequency Division Multiplexing) for bandwidths from 6 up to 54 Mbps. For the model of the communication system the function of the modulation scheme is not important – only its properties. The most important for us is the *sensitivity* of the modulation, i.e. the power level required for a successful reception of a transmission with a given bit rate. The table 4.3 gives some example values for the receiver sensitivity of the Cisco Aironet 350 Card [Cis05]. Other vendors and cards have similar values ([3Co04, Fou05]).

Antenna model

The second component of the communication system that we will discuss, is the antenna. There is a wide range of different antenna types. In the following we will reduce this diversity to two generic properties – *antenna gain* and *directivity*.

The antenna gain is the ratio of the intensity of the antenna's radiation pattern in a particular direction (usually the direction of the strongest intensity) to that of a reference antenna. So the antenna gain describes the amplification of the transmitted or received signal due to the antenna design. The directivity describes the direction in which the antenna focusses the transmitted power or the direction from what it best receives a transmission. In general, or if we do not have specific knowledge about a used antenna, we assume an *omni directional antenna*, i.e. an antenna that radiates an equal amount of energy into each direction. The opposite are *directional antennas* like Yagi, patch, or parabolic dish antennas, that have one or more preferred directions. Parabolic dish antennas have one preferred radiation direction and are often used to set up wireless point-to-point connections. Other directional antennas are for instance used to improve radio coverage of wireless base stations (access points) in indoor installations. Obviously they are a suboptimal choice for MANETs because in general we do not know the placements of the wireless nodes

and so cannot optimize the radio coverage beforehand. Instead they could impede connectivity due to wrong antenna orientation of transmitting and receiving nodes.

So we assume to use omni directional antennas for wireless nodes in a MANET. It should be mentioned that an optimal omni directional antenna (*isotropic antenna*) is a theoretical construct with no exact physical implementation. But it serves as the perfect reference to determine the gain of an antenna. A close approximation can be constructed by a set of dipol antennas. In practice this is not really necessary because under the assumption that all transceivers are (approximately) in one plane, other antennas can be considered omni directional. Examples are classical dipol antennas (see [Ben00], page 44f.) as well as vertically placed whip or discone antennas. If we look at the cross-section along the assumed plane of the radiation pattern, all show an equivalent pattern to that of the isotropic antenna. The only difference to the isotropic antenna is the higher gain. A standard dipol antenna for instance has a gain of 2.15 dBi (decibel over isotropic). In our model we will only consider the antenna gain over an isotropic antenna.

Figure 4.4 shows the directional gain of an Hertzian dipol antenna. The figures have been created using the "Electromagnetic Waves & Antennas" toolbox for MATLAB as described in [Orf97]. It shows that we have zero radiation in vertical direction ($\Theta = 0^o$) along the antenna element and an equal antenna gain in horizontal direction. For comparison, the field diagram for an isotropic antenna is just the unit circle.

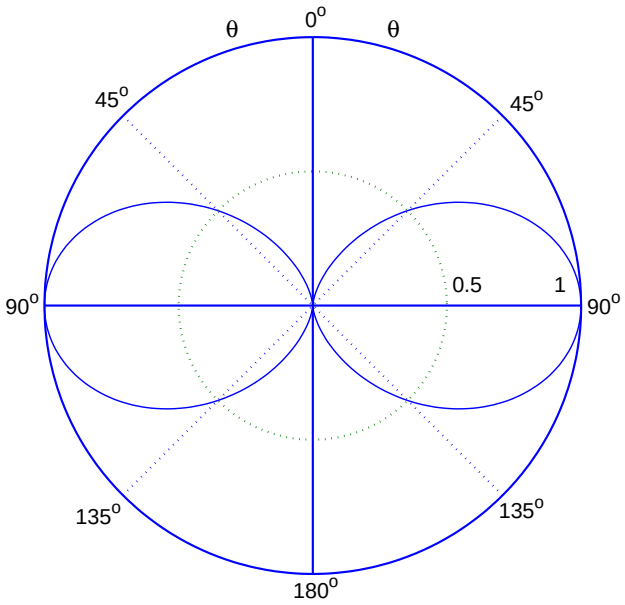

Figure 4.4.: Hertzian dipol gain in absolute units

Communication channel model

The most important part of our communication system model is the model of the wireless channel. This model describes the signal at the receiver as a function of the signal at the transmitter, the positions of the nodes, their hardware, and the surrounding environment. Their are a lot of propagation models ([Cra03, Ben00]) that can be used to describe the propagation of signals used in a WLAN system. In the following some selected models will be presented and discussed with respect to their relevance for this work.

Free space model

The *Free Space Model* assumes an empty environment where electromagnetic waves can propagate without any disturbance. Equation 4.4 (also called Friis transmission equation) calculates the power of a signal as a function of the distance from the transmitter:

$$P_{R_{Friis}}(d) = P_T * \frac{G_T * G_R * \lambda^2}{(4\pi d)^2} * g_T(\Theta, \phi) * g_R(\Theta, \phi) \tag{4.4}$$

where P_T is the transmission power; G_T and G_R the transmitter and receiver antenna gain; λ the wavelength of the signal; $g_T(\Theta, \phi)$ and $g_R(\Theta, \phi)$ the relative directive gains at spherical angles (Θ, ϕ) measured from the pointing direction of each antenna with a convenient reference direction for ϕ. As explained before, we assume omni directional antennas and nodes that approximately lie with the same horizontal plane. So we can set $g_T(\Theta, \phi)$ and $g_R(\Theta, \phi)$ to 1. The wavelength can be computed as:

$$\lambda = \frac{c}{f_T} = \frac{c_0}{F_T * n} \tag{4.5}$$

where c is the speed of light within the communication medium and f_T the frequency used by the transmitter. It should be noted that the speed of light in general is slower than the speed of light in a perfect vacuum (c_0). The slowdown (refractive index is given as a factor depending on the material). For air it is 1.003 and so can be omitted. So we get the simplified Friis equation:

$$P_{R_{Friis}}(d) = P_T * \frac{G_T * G_R * c_0^2}{(4\pi d * f_T)^2} \tag{4.6}$$

The free-space model is of limited use for modelling of a WLAN system. It can only be used to describe the direct neighborhood of the sender.

Two-ray ground reflection model

The two-ray ground reflection (TRG) model assumes not only a direct communication (line of sight). Instead it additionally considers interference with reflections from the ground and stations with different heights above ground. The power of the receiver can be computed as:

$$P_{R_{TRG}}(d) = P_T * \frac{G_T * G_R * h_T^2 * h_R^2}{d^4} \tag{4.7}$$

where G_T and G_R are the antenna gains; h_T and h_R are the antenna heights above ground. The TRG model is widely used to simulate wireless propagation in network simulations of communication protocols developed for MANETs. It is known to overestimate the receiver power for nodes that are very close to the sender. For longer ranges it delivers more useable results. So it is usually complemented by the free-space model to calculate the receiver power for nodes in close range. The major problem of the TRG model is that it does not considers effects like refraction, absorption, and dynamic fluctuations of the communication medium. The consequences of such a model will be discussed in the following (s. section 4.5).

Shadowing model

To overcome the limitations of the free-space model and the two-ray ground reflection model a third model will be discussed – the *shadowing model*. It uses a different approach to calculate the receiver power. The main difference is that the shadowing model does not try to compute the exact radio propagation but instead provides a model that allows to experience the same effects that can be experienced in the real environment. As said before, radio propagation is effected by a variety of physical effects. And in most cases we have insufficient knowledge to describe the environment exact enough to compute them. So we assume a *generalized environment*, e.g. outdoor downtown, outdoor landscape, office, indoor obstructed, etc. Every such environment is characterized by two properties – *path loss* and *disturbance*. The path loss describes the average signal attenuation cause by effects such as fading, reflection, multi-path interference, refraction, absorption etc. With disturbance we describe the dynamics in the environment. For example, it makes a significant difference if we measure connectivity in an office environment at night or at day when a lot of people move around in the building (s. [WJL03] for measurement results). The shadowing model defines the receiver power as a function of the distance as:

$$P_{R_{Shadow}}(d) = P_0 * 10^{-L_{Pe} * log\left(\frac{d}{d_0}\right) + \frac{\mathcal{X}}{10}} \tag{4.8}$$

where L_P is called the *path loss exponent* that specifies the degree of the path attenuation; P_0 is a reference power measured at a reference distance d_0 from the transmitter; $\mathcal{X} \sim \mathcal{N}(0, \sigma^2)$ a normal distributed error variable that describes the disturbance of the environment. It should be noted that $-L_P$ is sometimes also called *path gain exponent* to be consistent with the term

antenna gain. But it is also misleading because the communication path never causes an amplification. Table 4.4 shows values for typical areas. Typical values for σ^2 are between 2 and 10 dB.

Environment	**Path Loss Exponent** L_{P_e}
Free Space	2
Open field (long range)	4
Urban area	2.7 – 4
Shadowed urban area	5 – 6
In-building line of sight	1.6 – 1.8
In bulding, obstructed	4 – 6

Table 4.4.: Path Loss Exponents for typical environments (adapted from [Gib99])

As the shadowing path model is based on observations (measurements) in real environments it is well suited to study sets of randomly distributed nodes inside such an environment to gain statistical results about their connectivity, communication behavior etc. It cannot be used to describe the behavior of nodes placed at pre-defined positions in a known environment. Mullen et al. [MMAR04, Mul04] showed that fading models containing a stochastic element like the shadowing model are well suited to explain real-world behavior of wireless networks. Their measurements indicate a signal strength variation of up to 20dB.

Advanced models

There are a lot more models to calculate radio propagation in different environment. As explained before, to create a highly precise model we need an appropriate description of the environment. And depending on the available information we get different, usually more detailed models. One of the most detailed models uses a ray-tracing approach to predict the radio propagation. Despite the fact that this approach requires a highly detailed 3D model of the environment it is also highly computation expensive. An extensive summary of different propagation prediction models can be found in [IY02] and very extensive background information in [Gib99, Cra03]. But this shall not be discussed further here because it is out of the scope of this work.

But there is one work that should be mentioned here. In [IS05] Stepanov et. al. discuss the impact of the propagation model on wireless network simulation. They demonstrate the combination of a ray-tracing based propagation model with a network simulator. The most noticeable aspect of their work is that they build a bridge between the computational expensive ray-tracing model and a network simulator that needs to compute a large number of transmissions to analyze the behavior of a MANET. Their approach separates an outdoor environment (specified as a 2.5D model) into a 5m x 5m grid and calculates the radio propagation for a representative point of this grid element. The result is a propagation map that is later used for a fast lookup. This way they manage to speed up the computation of the propagation to that of the previously presented

analytical models. But it still has a major drawback – the pre-computed lookup maps do not consider disturbance of the environment.

4.3. Implications of the Propagation Models

In the previous section we have discussed some models that can be used to predict the propagation of wireless transmissions. This section will discuss the implications of the different propagation models and compare them with real-world measurements to determine their relevance and pitfalls. We will start with a discussion of possible interactions between different wireless devices (nodes).

We already discussed that the free-space model and the two-ray ground reflection model are only of limited use to predict the propagation of WLAN systems. Nevertheless, we will include them in the following sections because they are representative for all models that do not include a random component like the lookup-based ray-racing model (s. section 4.2). We will see that these models conceal some of the serious problems but can create artificial problems that can hardly be experienced in the real world. So, if we test a protocol development with models with and without a random component and make it insensitive to all possible problems it will perform better under real conditions.

4.3.1. Communication, Interference and Collisions

A node transmitting a message can affect another node in three different ways:

1. Communication: the second node receives the message transmitted
2. Interference: the second node is blocked to transmit due to the physical carrier sense
3. Collisions: the second node cannot correctly receive a message from a third node due to the transmission of the first node

A transmission is successful if the receiver signal is above the receiver threshold (P_{RX}) and if the signal-to-noise ratio is high enough. In the previous section we only requested $P_R > P_{RX}$ and $P_N < P_{CS}$ for a successful transmission. For only two nodes this requirement is sufficient because P_{RX} and P_{CS} are set to values that guarantee an adequate SNR. If we have more then two nodes this is not longer sufficient because we have to consider the additional noise caused by other nodes. To consider a network of more then two nodes we have to extend our equations. For a transmission from node n_a to node n_b we get:

$$P_{R_{a\to b}} = P_{Modulation}(d_{a\to b}); P_T := P_{T_a} \quad (4.9)$$

$$P_{N_b} = P_{N_{Floor}} + \sum_i P_{R_{i\to b}}; \forall i : i \neq a \vee i \neq b \quad (4.10)$$

$$SNR_b = \frac{P_{R_{a\to b}}}{P_{N_b}} \quad (4.11)$$

$$SNR_{min} = \frac{P_{RX}}{P_{CS}} \quad (4.12)$$

where $P_{R_{a\to b}}$ is the power of a signal transmitted from node n_a and received at node n_b in a distance $d_{a\to b}$ from n_a; $P_{N_{Floor}}$ is the power of *noise floor* of the environment, i.e. the background noise caused by other sources than the nodes in our model. We now replace the requirement for a successful signal reception $P_R > P_{RX}$ by $SNR_b > SNR_{min}$.

If $P_{N_b} < P_{CS}$ we get $SNR_b > SNR_{min}$ and can compute the maximum transmission range d_t from:

$$P_{RX} = P_{Modulation}(d_t) \quad (4.13)$$

In a similar way we can compute the minimum interference range d_i, i.e. the minimum range for that $P_R \geq P_{CS}$ applies to every node which this range as:

$$P_{CS} = P_{Modulation}(d_i) \quad (4.14)$$

Figure 4.5 depicts the two ranges. In the following we will refer to the inner circle as the *transmission area* and the outer annulus as the *interference area*. That means we will distinguish the area where communication is possible from that where only a physical carrier sense is possible.

The interference with other nodes can cause *collisions* of messages. Figure 4.6 depicts this problem. If node A sends a message to B and C at the same time to D, only D will successfully receive a message. As C is out of the interference range of A it cannot detect the transmission of A. But B is within the interference range of C and so B causes noise above the carrier sense threshold. This does not necessary means a collision as B is close to the maximum transmission range of A. $P_R > P_{RX}$ is still valid, but $P_N > P_{CS}$ which means that $SNR_R > SNR_{min}$ is not guaranteed anymore. Depending on the exact distance of B to A and C this requirement can become invalid which means that the message cannot be received correctly. As an example for all propagation models without a random element, the figure 4.7 shows the conditions at which a collision can occur. The yellow region depicts all constellations where node C is outside of the interference range of node A but can still prevent the node B to receive a packet send from A.

The circular border of the interference range in figure 4.5 is not the maximum range in which a node effects other nodes. The electromagnetic waves propagate far beyond this point and add to the noise level of nodes further away. If this noise from multiple nodes adds up, it can also cause a collision.

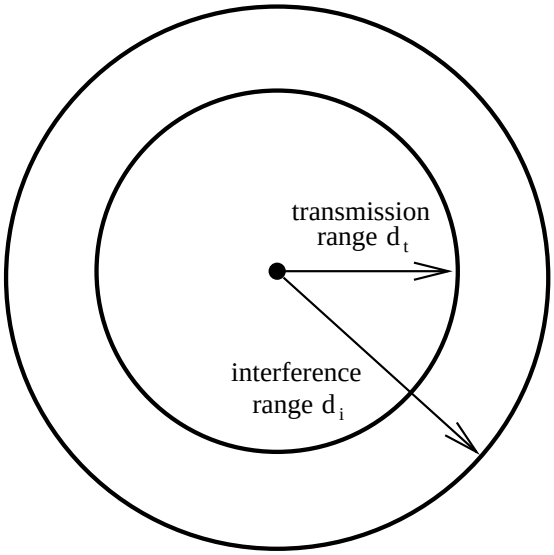

Figure 4.5.: WLAN transmission and interference range

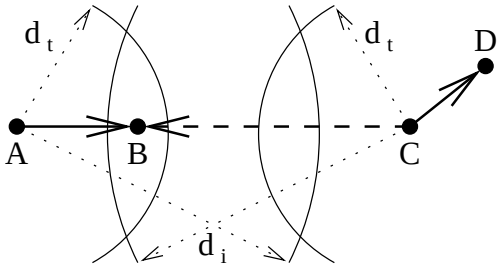

Figure 4.6.: Message collision

4.3.2. Transmission-to-Interference Range Ratio

A critical parameter for the WLAN communication model is the ratio of the transmission to the interference range. It defines the probability that a node can prevent another one from transmitting a message due to the physical carrier sense. It furthermore affects the amount of noise that is caused beyond the interference range. The reason is that a smaller ratio indicates a stronger signal attenuation.

Figure 4.8 depicts a comparison of the signal power as a function of the node distance for the three different propagation models. The following numbers have been used to compute the signal propagation:

- Transmitter: $P_T = -5dbm$ (free space, two-ray ground reflection); $P_T = 20dbm$ (shadowing) at $f_t = 2.412GHz$ (channel 1)
- Antenna: omni directional with $G_T = G_R = 0dbi$

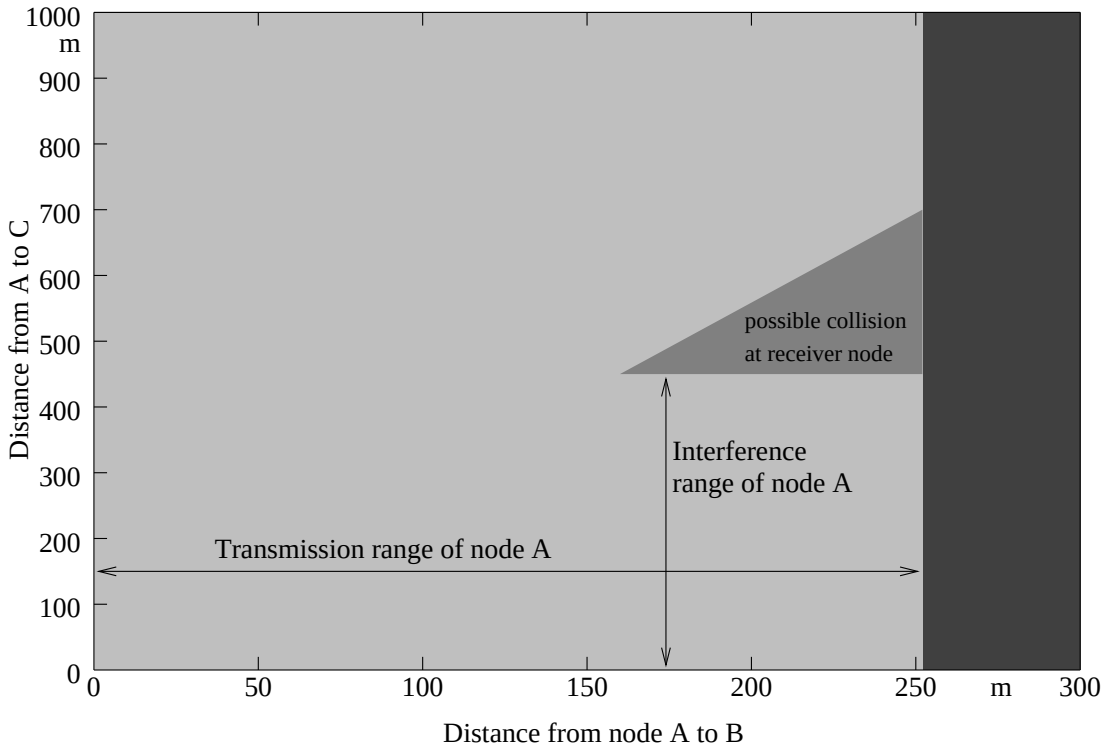

Figure 4.7.: Node placement for collisions

- Shadowing: $L_{P_e} = 3.6$ (urban area), $\sigma^2 = 4db$
- Modulation: BPSK ($P_{RX} = -94dbm$), Carrier Sense at $P_{CS} = -104dbm$

Because the shadowing model includes a random component, the diagram shows the median and the lower and upper bound of the possible values. The parameter setting shows a widely used but crude misuse of the first two propagation models. If setup with the parameters for the shadowing model, both models deliver much too high values for the signal power and so the maximum transmission range (>1000m). So, to bring both models into the range of values measured in the real world, the transmitter power is significant lowered to the unrealistic value of -5dbm (0.3mW) or the receiver sensitivity and carrier sense threshold are risen by 25db.

The diagram shows some other effects. First, that the two-ray ground reflection model overestimates the signal strength in the near range (it is impossible to reach a signal power above the free-space value). Second, that the models result in different transmission-to-interference range ratios (TIRR). The exact values will be discussed in the following.

The TIRR is defined by:

$$TIRR = \frac{d_t}{d_i} \tag{4.15}$$

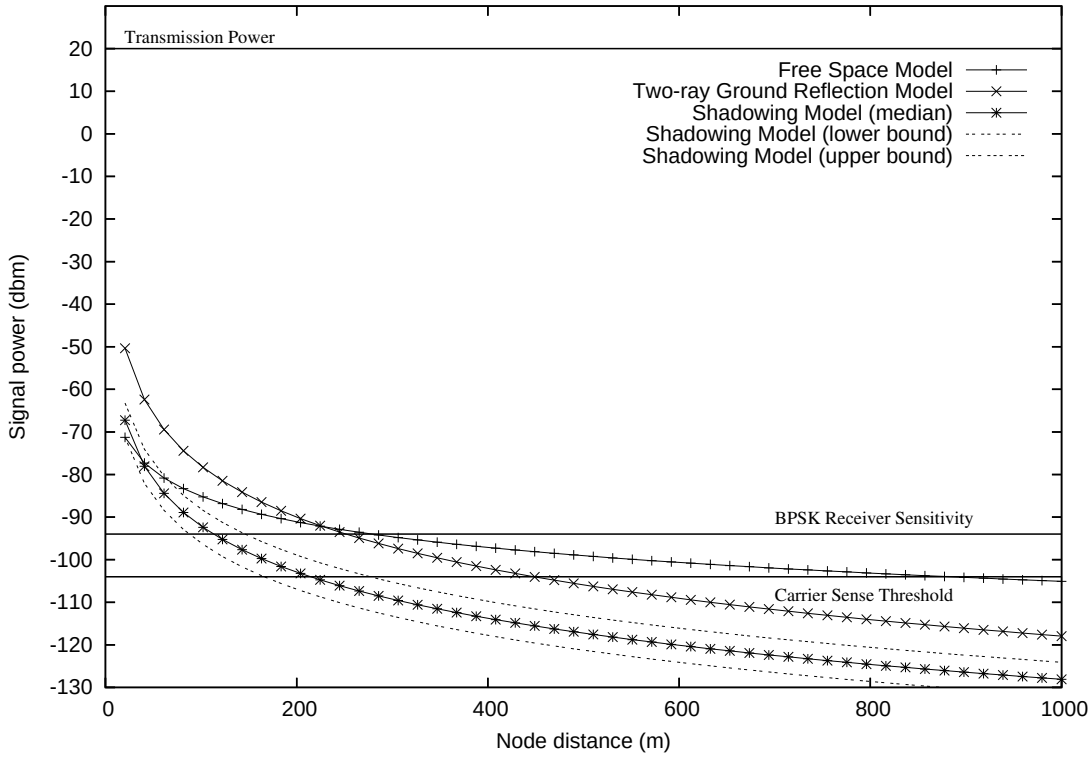

Figure 4.8.: Propagation model comparison

Free-Space Model

For a given signal power P the distance d can be computed by reversing equation 4.6:

$$d = \frac{\sqrt{\frac{P_T}{P_{R_{Friss}}(d)} * G_T * G_R} * c_0}{4\pi f_T} \tag{4.16}$$

If we set P_{CS} as the power at distance d_i and P_{RX} for d_t we can compute:

$$TIRR_{Friis} = \sqrt{\frac{P_{RX}}{P_{CS}}} \tag{4.17}$$

For the given values of -94dBm and -104dBm we get a $TIRR_{Friis} \approx 0.316$.

Two-Ray Ground Reflection Model

To compute the TIRR for the two-ray ground reflection model we perform a similar computation by reversing equation 4.7:

$$d = \sqrt[4]{\frac{P_T}{P_{R_{TRG}}(d)} * G_T * G_R * h_T^2 * h_R^2} \tag{4.18}$$

Again we set P_{CS} as the power at distance d_i and P_{RX} for d_t to compute the TIRR as:

$$TIRR_{TRG} = \sqrt[4]{\frac{P_{RX}}{P_{CS}}} \tag{4.19}$$

For our threshold values we get a $TIRR_{TRG} \approx 0.562$.

Shadowing Model

To determine the TIRR for the shadowing model we use the mean value of the propagation function, i.e. we remove the error variable. As for the previous models we reverse the propagation function to calculate the distance for a given signal power by:

$$d = 10^{\left(\frac{log\left(\frac{P_{R_{Shadow}}(d)}{P_T}\right)}{-L_p}\right)} * d_0 \tag{4.20}$$

and the TIRR by:

$$TIRR_{Shadow} = 10^{\left(\frac{log\left(\frac{P_{RX}}{P_{CS}}\right)}{-L_{P_e}}\right)} \tag{4.21}$$

For our threshold values and the given path loss exponent we get a $TIRR_{Shadow} \approx 0.527$. Figure 4.9 depicts $TIRR_{Shadow}$ as a function of the path loss exponent L_{P_e} for different ratios of carrier sense and receive thresholds. We see that $TIRR_{Shadow}$ is above 0.5 for all relevant cases (s. table 4.4) and is significant higher if we have only small differences between P_{RX} and P_{CS}.

4.3.3. Connectivity

After discussing how nodes can effect each other, we will now discuss the effect of the radio propagation on the connectivity of the notes in the network. As connectivity between two notes we define the ability to communicate between them.

Figure 4.10 shows the results of a simulation experiment to determine the probability of a node to receive a packet in a certain distance from the sender. The simulation used the same parameters as specified in section 4.3.2. The receive probability is the relative number of packets received in this distance to the number of send packets.

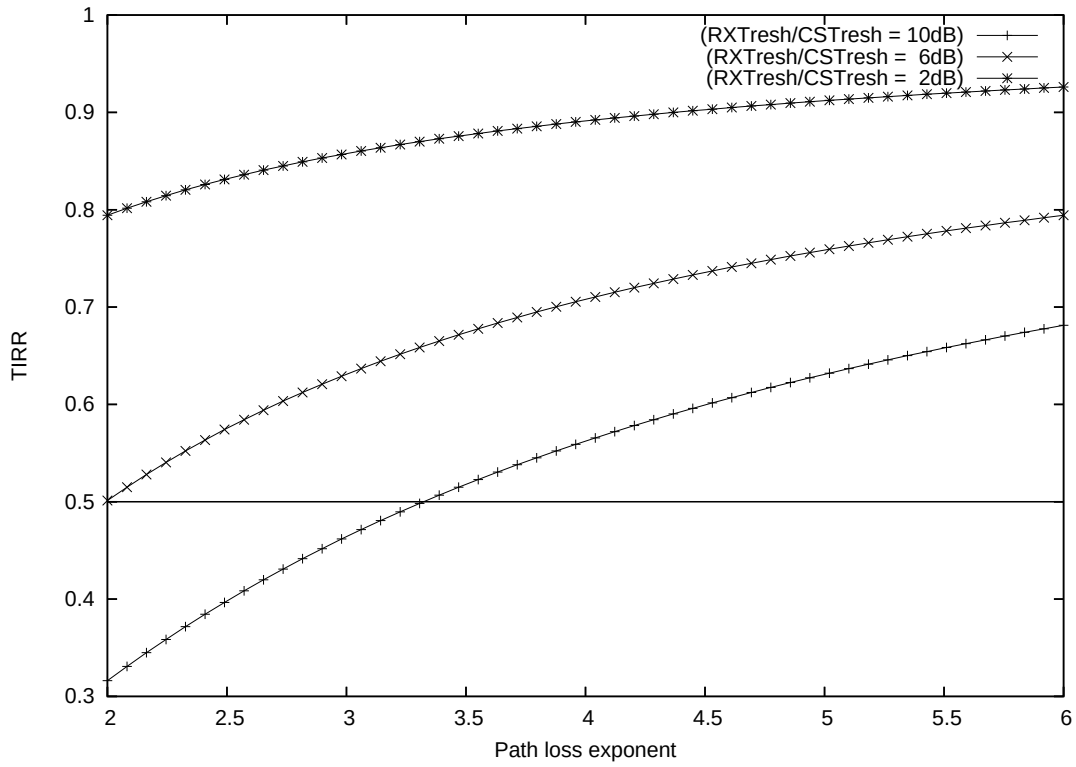

Figure 4.9.: TIRR in the shadowing model for different threshold differences

From the diagram we clearly see that the free-space as well as the two-ray ground reflection model result in a 100% or 0% probability. The only thing that could cause a packet loss is a collision. This will be discussed later as we do have only a single sender in this experiment. At this point we will not care about the fact that the setup uses unrealistic setting for the transmitter power (as explained before). The relevant result is that these models deliver a packet every time or never to a receiver in a fixed range. In such a model it is easy to decide if two nodes can communicate with each other and so have connectivity.

For the shadowing model we see a range with 100% receiver probability and than a transition range with decreasing probability – an effect that can be experienced in the real-world. The most noticeable point is that the range for 100% probability in this case is about 70m, but we receive individual packet still at a range of 400m. With such a model it is more difficult so define the connectivity between nodes. So, in the following we will use the term *link quality* as a synonym for the receive probability, and connectivity between two nodes as a link with a quality above a required threshold. This threshold depends on the communication that should happen on a link. This should be explained by example. If we have a link and use a communication protocol that uses retries (like the IEEE 802.11 MAC protocol for unicast transmissions), we can tolerate a burst of n transmission failures. This reduces the available bandwidth but increases the packet loss rate on the link. If we do not have such a retransmission scheme the effective communication range is smaller. As *effective communication range* we denote the maximum distance over that

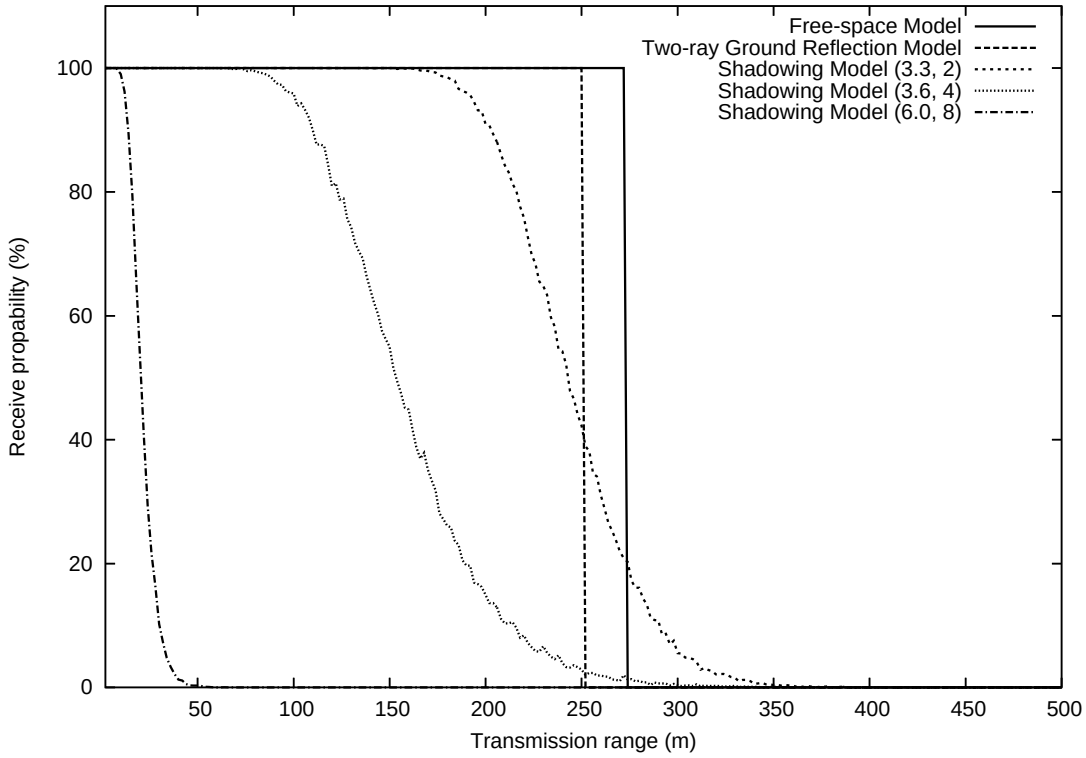

Figure 4.10.: Receive probability

two node can communicate with an acceptable packet loss rate. The actual value depends on the requirements of higher software layers.

Figure 4.11 again depicts the results from the simulation experiment. This time the results of the shadowing model and additionally the ratio of 1-, 2-, and 3-packet burst failures are shown in the diagram. If we assume a maximum of three retransmissions, the packet receive probability is the sum of the receive probability and the three burst ratios because these errors would be compensated by the retransmission scheme. As we see, now the effective communication range increases from 70m to 100m (approximately a 50% increase).

So far we studied the effects one node can have on other nodes. In the following we will focus on the interactions of a set of nodes in a network. We will study the connectivity and link quality in the network. For this study we will use the two-ray ground reflection model and the shadowing model and a grid as well as a random topology with different node densities.

The two propagation models have been chosen because they show the effect of a deterministic and a probabilistic transmission/interference range. The grid topology with its regular structure is unlikely to be found in the real world but is ideal to study the problems caused by the interference. Under the assumptions that the horizontal/vertical distance between neighboring nodes is less the transmission range and $TIRR \leq 1/\sqrt{(2)}$ (which is the case for the presented models) every node in the grid has at least two nodes within transmission range and one within interference

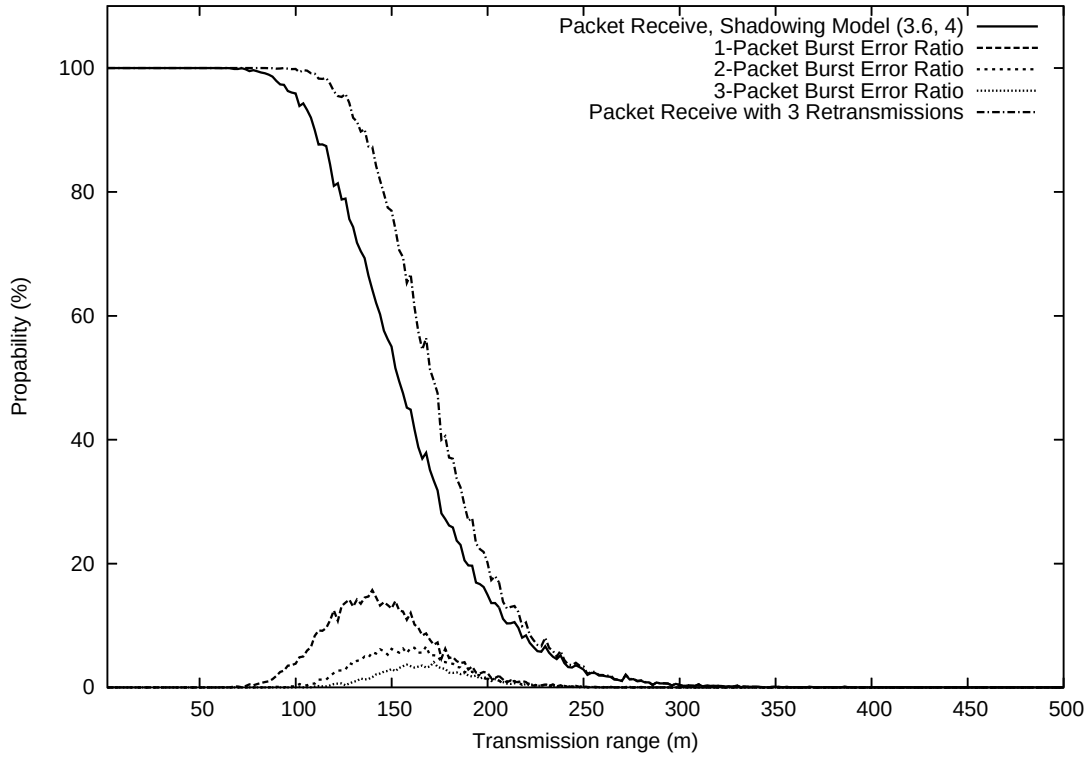

Figure 4.11.: Receive probability and burst error ratio

range. So every node can possibly be effected by collisions (s. section 4.3.1). Additionally the grid topology serves well to study the effect of noise because we have regions with different noise levels. Nodes in the middle receive higher noise as nodes on the borders of the grid.

The random topology in contrast much more represents a real-world topology. It allows to study the problems of dense and sparse regions. This cannot be done in the grid topology because here we have a uniform node density. So a random topology will mainly be used to investigate the link quality and connectivity within the whole network.

To study the influence of the propagation model on the different topologies with different node densities the following experiment has been conducted. 100 nodes where placed in a square region either in a grid or a random topology. The size of the region has been chosen so that these nodes result in a desired node density. Each of the nodes sends a low frequency beacon (1 second interval) to discover its neighborhood. The number of beacons received from neighboring nodes has been used to determine the link quality. To detect burst errors each beacon carries a sequence number.

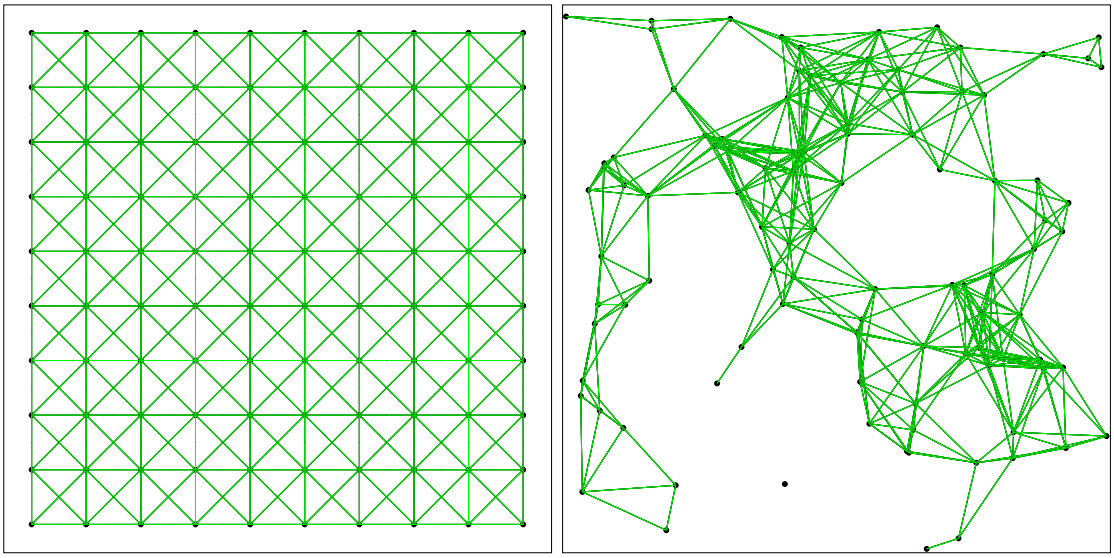

Figure 4.12.: Grid and random topology with two-ray ground model (50 nodes/km^2)

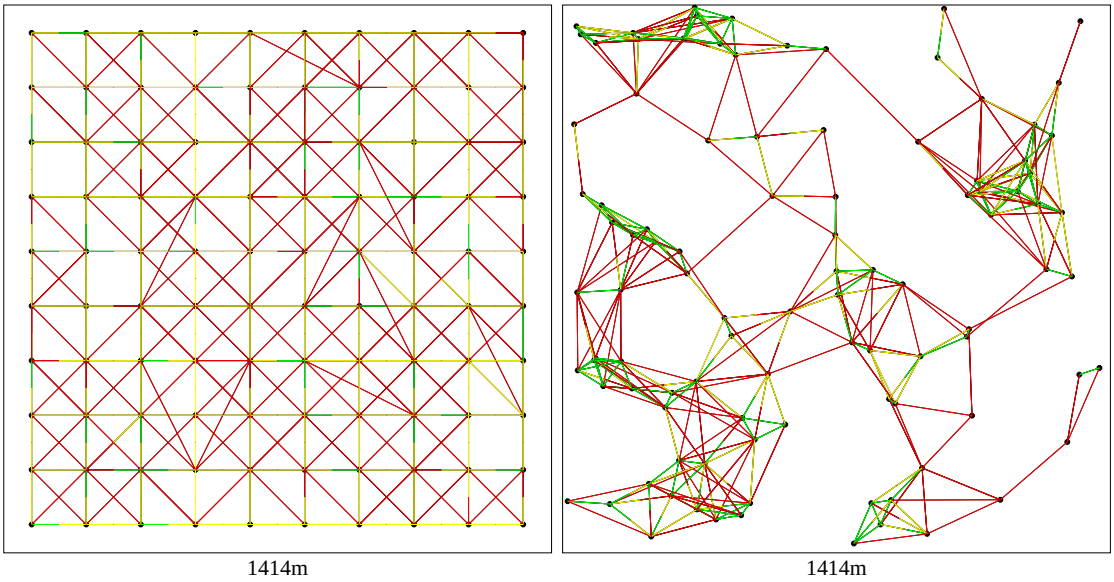

Figure 4.13.: Grid and random topology with shadowing model (50 nodes/km^2)

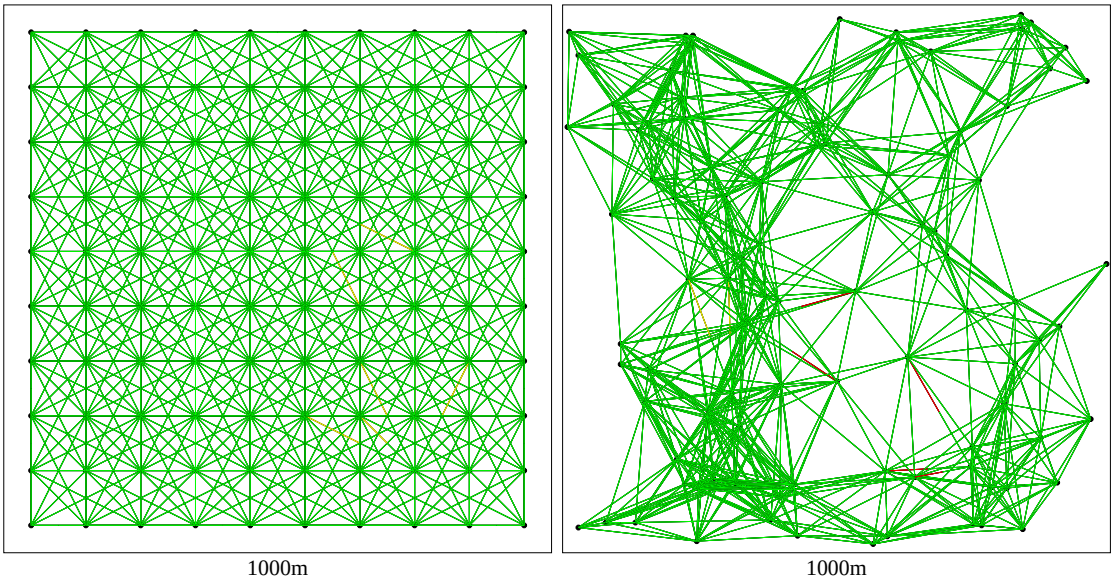

Figure 4.14.: Grid and random topology with two-ray ground model (100 nodes/km^2)

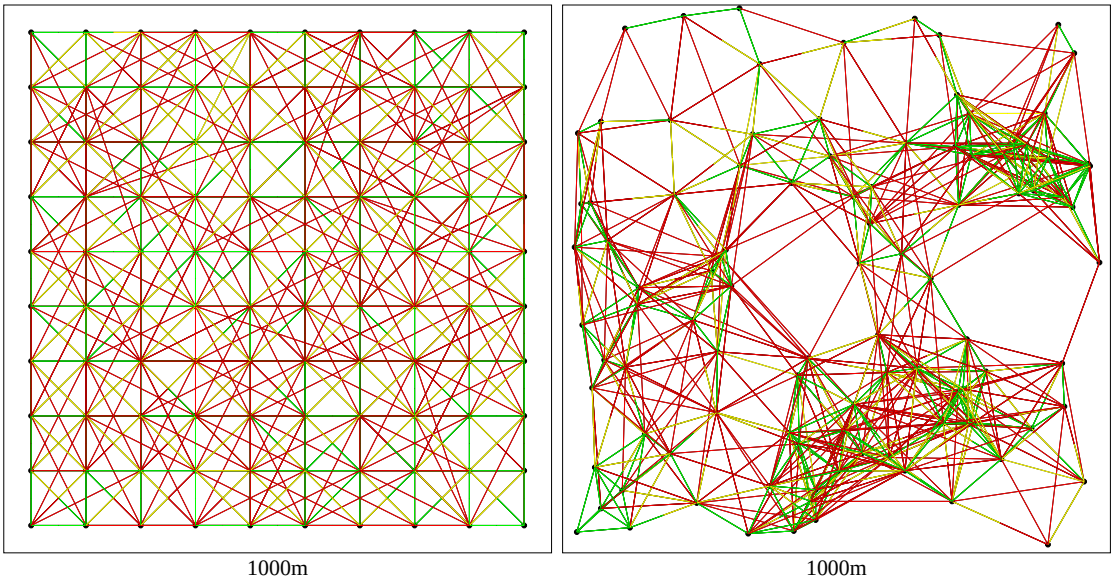

Figure 4.15.: Grid and random topology with shadowing model (100 nodes/km^2)

The figures 4.12, 4.13, 4.14, and 4.15 depict the link qualities for the grid and a random topology with two different node densities using the two-ray ground reflection (TRG) as well as the shadowing model. The link quality is represented by the color of the links. Green links have a quality (receive probability) of at least 80%, yellow of 50%–80%, and red for qualities of less than 50%. If the two nodes on the link have a different perception of the link quality, the links

are drawn with two colors – the side attached to the node shows its perceived link quality. In the following we will denote such links as *asymmetric links*.

If we look at figure 4.12 it seems that we do not experience any problems if we use the TRG model. But if we double the node density (s. figure 4.14), we see that we start to get links with lower quality as well as asymmetric links – especially for the random topology. As the propagation model does not include a random component, this can only be caused by collisions or an increased noise level. In contrast the figures 4.13 and 4.15 show that the random component of the shadowing model causes a lot of links with medium or low quality and also a significant number of asymmetric links.

The given figure only show two example densities and only allow for a subjective impression of the number of high, medium, or low quality links. To get an objective result we run multiple scenarios for a wide range of node densities and counted the number of high, medium, and low quality links as well as the number of asymmetric links and their maximum asymmetry, i.e. the difference of the quality measured at both sides of the link. A special case of asymmetry are one-way links where only one side knows about the other side. The ratio of those links is given as well.

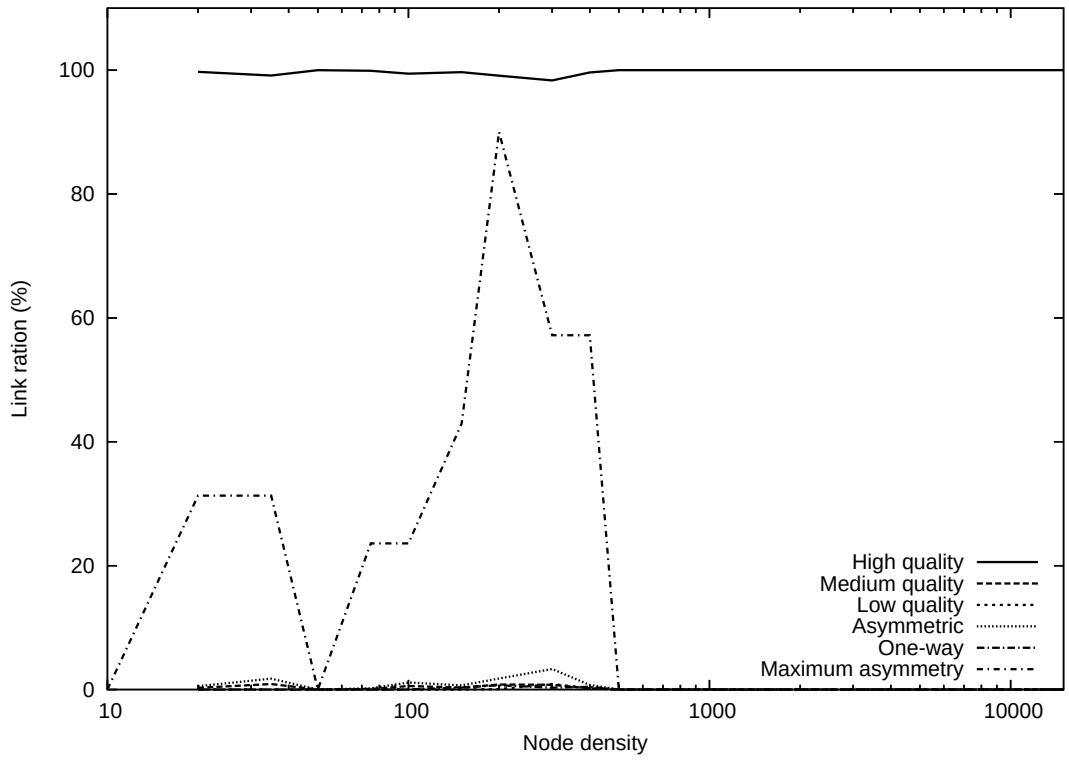

Figure 4.16.: Link quality vs. node density in grid topology with TRG model

Figure 4.16 and 4.17 depict the results for the grid and random topology using the TRG model. The results for the random topology are averaged over a number of independent replications (s.

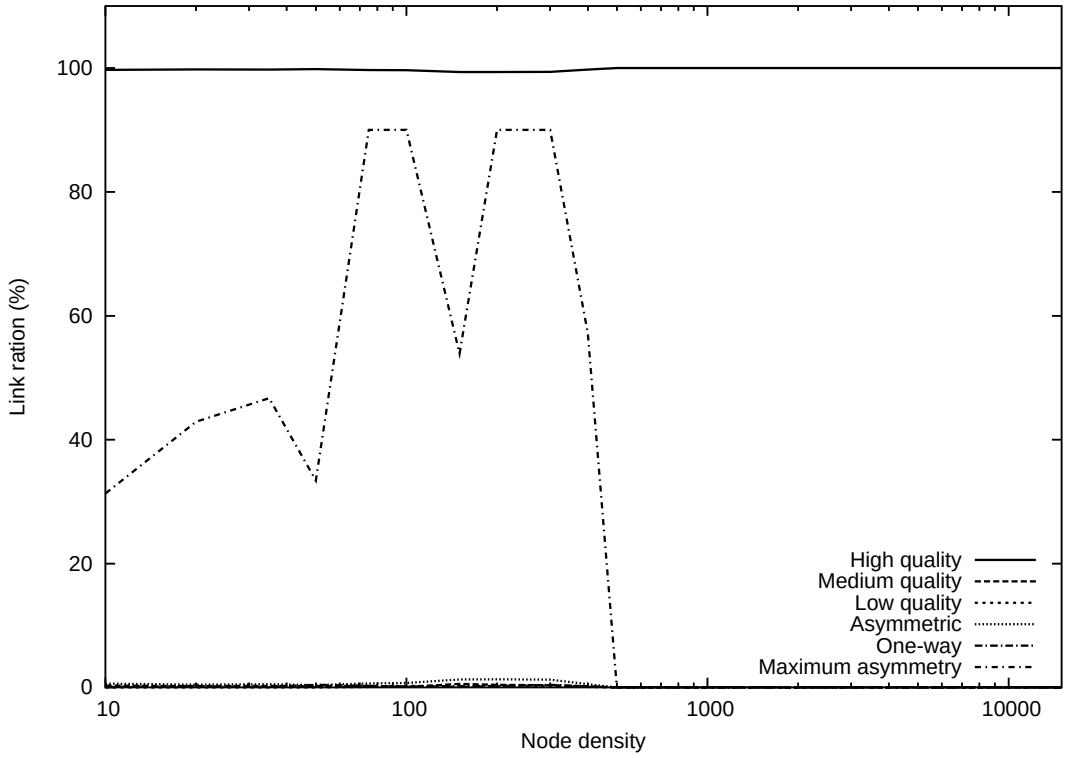

Figure 4.17.: Link quality vs. node density in random topology with TRG model

section 4.7.4 for more information on this topic). We see that both topologies deliver similar results. As expected, we get a ratio of high quality links of about 100% and only very small ratios for asymmetric or one-way links. But we can state that such links can happen even in a model without a random transmission range component. Furthermore we see that if we get asymmetric links, the asymmetry can be very high. That means that two neighboring nodes can have a completely different view of the connectivity to each other.

The figures 4.18 and 4.19 depict the results using the shadowing model. Here we get a significant different result. We see that all types of links can occur in the network. The results for both topologies look very similar. With one major exception – in the grid topology the results for node densities below 50 nodes/km^2 show a significant divergence. But this can easily be explained. Due to the regular placement in the grid topology the nodes are simply too far away from each other to create high quality links and with greater distance even medium quality links. For densities up to 1000 nodes/km^2 we see an almost constant ratio of high quality links that highly increases for higher densities. This seams to be a very interesting result in the first place but should not be overrated. The reason is that the number of overall nodes used for the experiment is always 100. That means that the number of nodes in direct communication range increases and so the probability for collisions decreases. At a density of around 20.000 nodes/km^2 we get a fully connected network where every node can communicate with every other node. And so there are no nodes left to cause interference and collisions. So the following discussions will only refer to

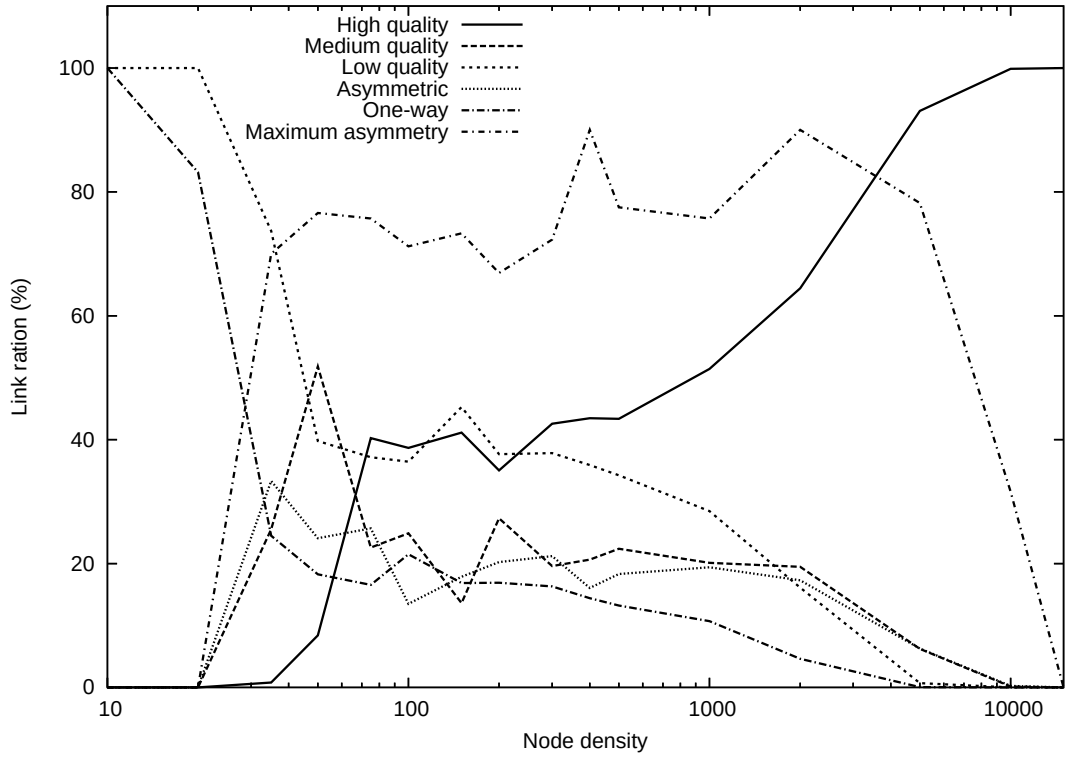

Figure 4.18.: Link quality vs. node density in grid topology with shadowing model

the values for node densities up to 1000 nodes/km^2. In this region we see an approximately equal ration of high and low quality links ($\approx$ 40%) – twice the ration of medium quality links ($\approx$ 20%). This means that most links are very likely not useful for unicast communication. We additionally see that we have a significant number of one-way and asymmetric links. The consequences of such asymmetries for the communication in a multi-hop network will be discussed later (s. section 5.4.1).

At last we will take a closer look at the high quality links. As we can see in figure 4.18 and 4.19, the maximum asymmetry reaches values of 80%–90%. That means that we have asymmetric links that are of high quality in one direction and of (very) low in the opposite. This value shows only the maximum. So the interesting question remains how much of the high quality links are effected by an asymmetry. The most useful links are those that have a high quality in both directions. We will denote such links as *perfect links*. The most awful are those that are high quality but only one-way. The figures 4.20 and 4.21 depict the overall ratios of high quality links as well as the ratio of perfect and one-way links. We see that about 80%–90% of all high quality links are also perfect links and that the ratio of high quality one-way links as about 0%.

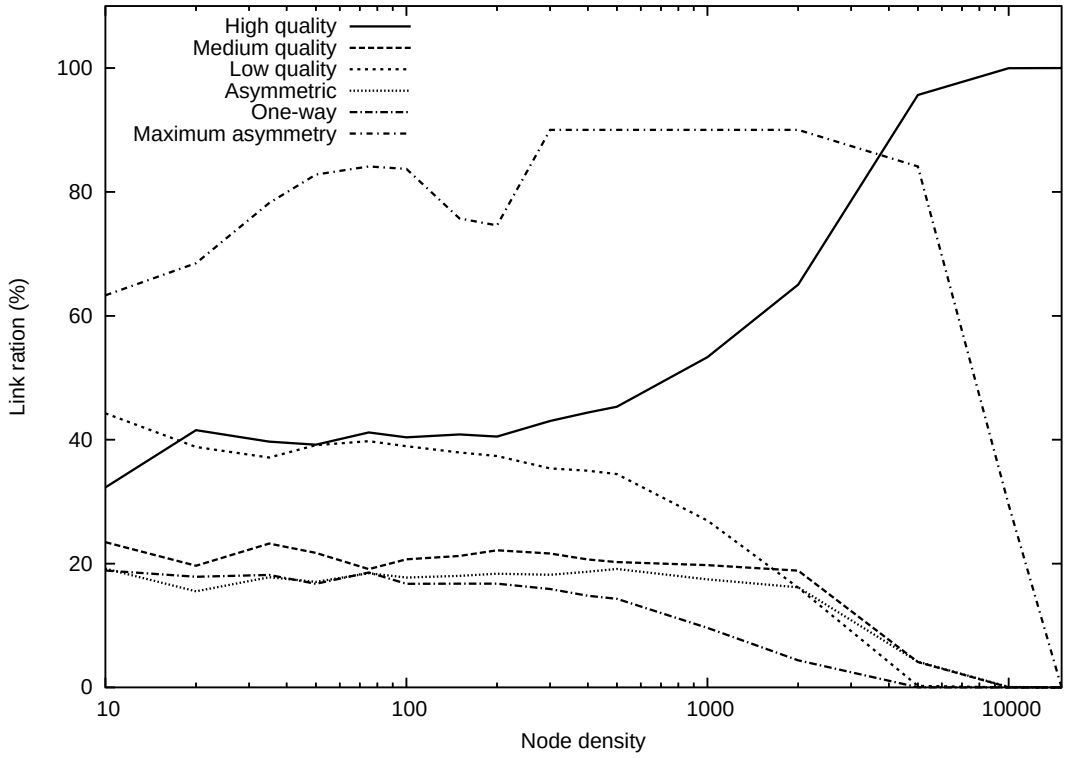

Figure 4.19.: Link quality vs. node density in random topology with shadowing model

4.4. Real-World Measurements

In the previous sections we discussed the behavior of WLAN devices on a theoretical basis. In the following we will discuss different experiments to compare the theoretical results with real-world results.

4.4.1. Antenna Characteristic

At first it should be noted that the following experiment was originally intended to analyze the loss probability as a function of the distance of two nodes in an open park area. Due to the test conditions it was a huge failure but provided us with insightful results about the characteristics of the used WLAN devices. Additionally it showed us some pitfalls that have to be avoided to perform successful measurements in the real world. So the original test conditions, results, and lessons learned will be discussed at this point.

For the experiment we took two identical laptops equipped with PCMCIA WLAN adapters (Netgear WAG511, a Prism54-based IEEE 802.11b/g compliant card). The first laptop had been stationary placed about 50cm above ground and transmitted small beacon frames at a rate of 10 per second. The second laptop functions as a receiver and was carried around by a person (about

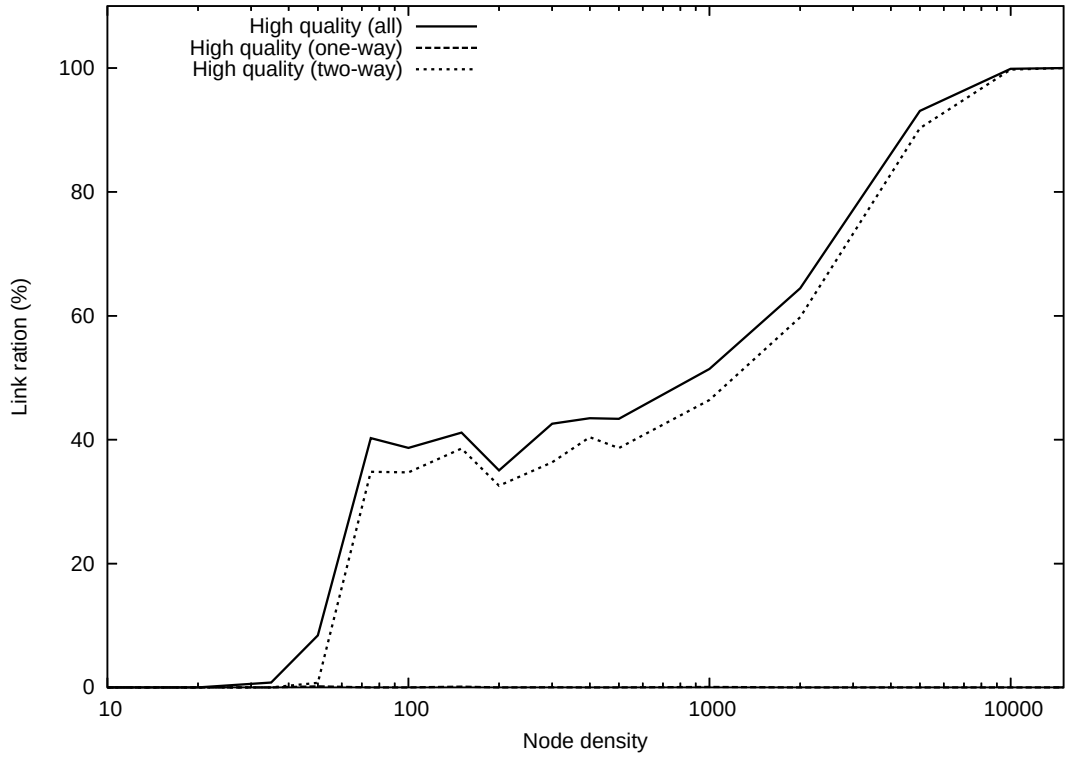

Figure 4.20.: High quality / perfect links vs. node density in grid topology with shadowing model

120cm above ground). During a measurement period of 60 seconds we counted the received and lost beacons. The second laptop remains at a fixed position during the measurement but we took no special care of the relative orientation of both laptops towards each other. We only made sure that we had a clear line of sight, although there were some obstacles (trees, small bushes) within the first fresnel zone (s. [Cra03] section 1.4.1.7 for details).

The results of the measurement were very surprising. In the near range of the transmitter (up to 70m) we noticed a packet loss rate of 0–5% – as expected. Above this range we noticed a higher loss rate. But we also noticed that small changes in the orientation of the second laptop, as small as 10^o could change the loss rate between 0 and 100%. This puts all measured loss rates into question because we could never ensure that we perform a measurement with an identical relative orientation. So we will not discuss the concrete results of this setup here. We came to the conclusion that the assumption of a nearly isotropic transmitter (s. section 4.2 for details) is not valid for this specific WLAN card. We skipped the initial goals of the experiment and instead investigated the directional gain of the antenna build into the WLAN card. It turned out that the antenna gain is not equal in all directions but has some preferred directions. That means that the radiation field does not look like a sphere, more like a combination of different ellipsoids along the preferred directions. Figure 4.22 depicts the preferred directions and the relative gain (longer arrows mean a higher gain). The figure should be considered a sketch but it gives an

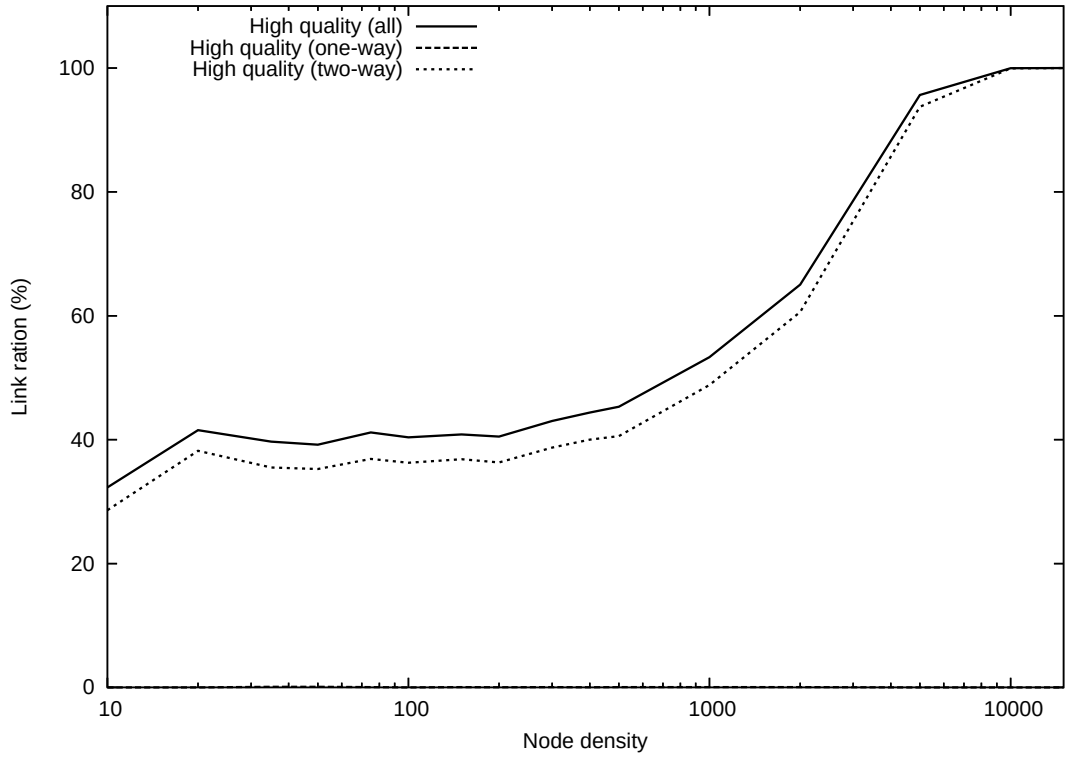

Figure 4.21.: High quality / perfect links vs. node density in random topology with shadowing model

impression of the card's behavior. Additionally it is only a single example – other cards will very likely show a different behavior. In [BP00] the authors discuss the practical problems of the directional antenna gain for a RF-based location tracking system. They conclude that it is necessary to measure the signal strength at least in four horizontal directions. Our observations show that even this will not be enough. But it clearly shows two things:

1. We have to consider a directional antenna gain,
2. a circular radio propagation cannot be assumed.

Usually we are not able to calculate the directional antenna gain because of missing information (e.g. antenna specification, relative positioning/orientation of nodes). So we have to include this uncertainty in our WLAN model. The shadowing model allows us to include such a probabilistic component.

4.4.2. Near-Range Loss Rate

A result from all discussed propagation models is that two nodes that are located very close together (e.g. with 2m distance) will never experience packet loss. This theoretical result will be

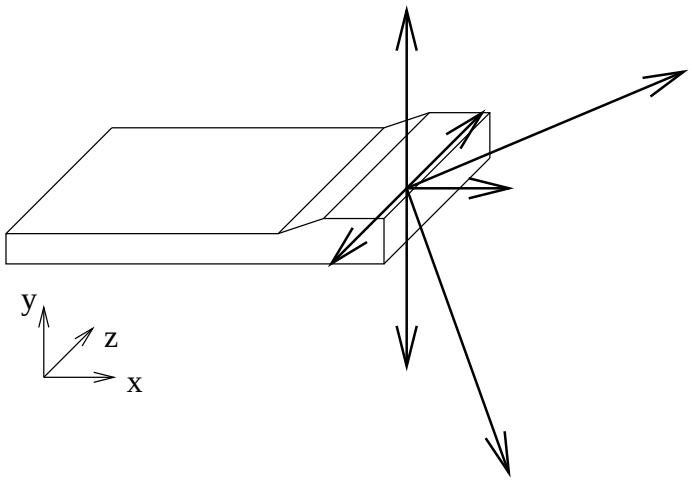

Figure 4.22.: Estimated directional gain of a WG511 WLAN card

tested with the following experiment. It had been originally performed to calibrate our emulation (real-time simulation) setup [MI04c, MI05] to reproduce the physical conditions measured in a test environment.

The goal of the experiment is to measure the raw packet loss rate between two laptops. As we are using standard WLAN hardware we have to ensure that the measurement is not falsified by automated retransmissions of the MAC layer. To avoid this we use broadcast packets. The application used to measure the packet loss rate is the `ping` command that sends ICMP (*Internet Control Message Protocol*) messages. It usually sends unicast packets. But if we statically map the IP addresses of the two laptops to the broadcast MAC address (`FF:FF:FF:FF:FF:FF`) we can perform the measurement without any automatic retransmissions. As the direct result of the measurement we get the ICMP success rate $p_{ICMP} = \frac{\#Received}{\#Send}$. We only receive an ICMP response for an ICMP request if both transmissions are successful. So we can calculate the raw packet loss rate (p_{raw}) as:

$$p_{ICMP} = \frac{\#Received}{\#Send} = 1 - (1 - p_{raw})^2 \tag{4.22}$$

$$p_{raw} = 1 - \sqrt{(1 - p_{ICMP})} \tag{4.23}$$

We ran the test in different environments with small (8 byte payload) and large (1000 byte payload) ICMP packets. For each test we send 100.000 packets at a rate of 100 per second. The results of the experiment are listed in table 4.5. In the office environment we had a number of active WLAN and other electronic devices. In the undisturbed environment there were no such devices nearby.

The results of the experiment show that although there are other disturbing devices present, they do not have a significant influence on the packet transmissions. If the propagation model

Environment	p_{ICMP}	p_{raw}
Office; small packets	99.954%	0.022%
Office; large packets	99.400%	0.300%
undisturbed; small packets	100%	0%
undisturbed; large packets	100%	0%

Table 4.5.: Close-up range packet loss rate

already includes the possibility to create partial packet losses, this will be sufficient. But if the environment is filled with electrical and electronic devices that cause a stronger disturbance, this will not be covered by the propagation models and has to be considered separately.

4.4.3. Transmission and Interference Range

In section 4.3.2 we already discussed the transmission-to-interference range. Because this is essential for the developed communication protocols (s. section 5.4.1, 5.8) we will test the actual behavior in an experiment.

The interference cannot be measured directly with standard WLAN hardware. So we need an indirect measurement. We have the additional problem that we need to know when a node actually transmits a packet. Because standard WLAN hardware uses a half-duplex transmitter we cannot lister for our own transmissions and we do not get any feedback from the operating system about when a packet has been transmitted. To solve both problems we decided for an experimental setup that comprises of four laptops – two active transmitters and two passive observers. The passive observers solve the problem of the half-duplex transmitters as they listen all the time. We place an observer directly beside each transmitter. So they receive all packets that the active node receives as well as all packets that it transmits. In the previous section we have already seen that the loss rate between two nodes that are located close together is negligible low. So this will not falsify the results. The second problem is to indirectly measure the interference effect. We know that a node is able to transmit after it senses the medium as free. So if two nodes are within interference range they will prevent each other from transmitting and so share the available bandwidth even if they cannot communicate. We will exploit this behavior in the following way. Both nodes try to transmit enough packets to saturate the medium. If a node has to share the medium with the other one, it will not be able to transmit as much packets as it tries. The passive nodes beside the transmitters will count the packets they receive. If it receives packets at a full rate from only one node that means that the other node is outside the interference range. If the rate decreases, this indicates that the node is affected by the other one through the physical carrier sense. Finally, if it receives packets from both nodes, this means that they are within transmission range.

Figure 4.23 shows results from two experiments and two simulations. The diagram shows the relative overall bandwidth utilization, that is the sum of all packets received by the passive observers. We conducted two real-world measurements – one indoor and one outdoor with two

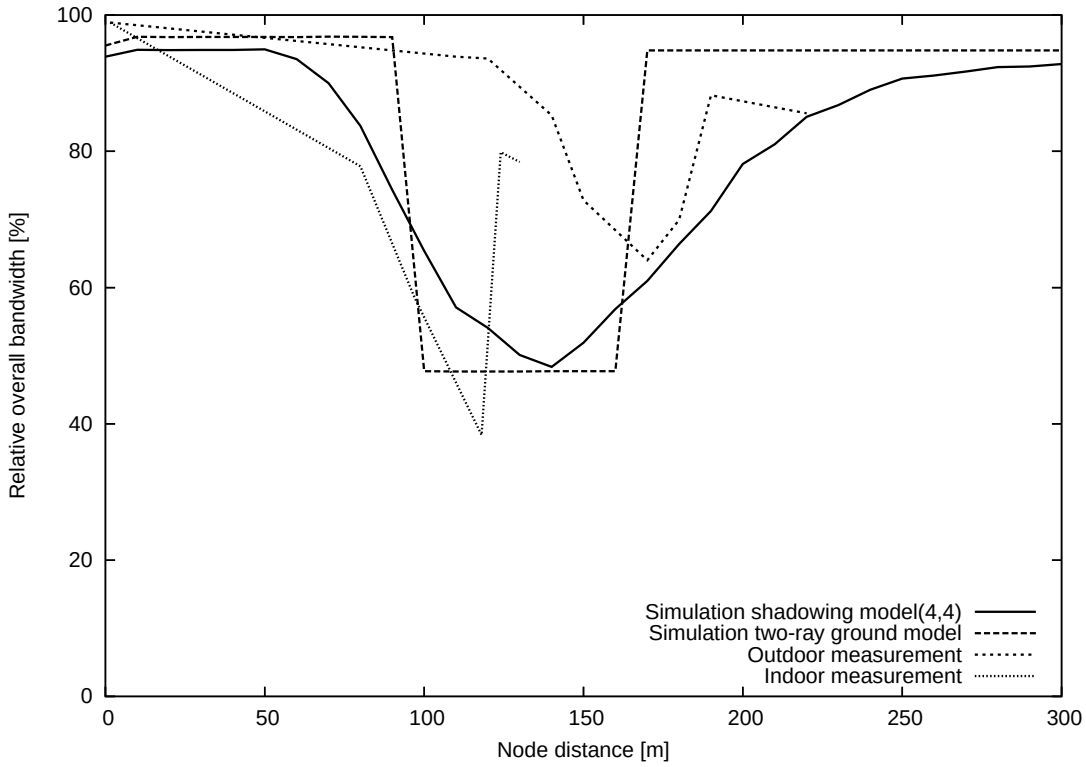

Figure 4.23.: Real-world and simulation results

different sets of wireless cards. Additionally the figure includes two simulation results for comparison – one using the shadowing and another using the two-ray ground reflection model. We see that the graph for the two measurements have a similar shape. The overall bandwidth starts to decrease at some point, reaching a minimum at a midrange, and rises again at higher range. This clearly indicates that we are affected by the physical carrier sense but cannot receive a packet at some distances. We also see that we do not have an all-or-nothing situation where we either receive a packet, detect only the physical carrier, or are not affected by the other node. Instead we have a gradual transition between these areas. If we compare the two measurements we notice that they show a different range for the minimum, different width and depth of the through in the chart. The reasons for this will be discussed later in section 4.7.4.

If we compare the measurements with the simulation results we see that we also get a range with a decreased overall bandwidth utilization. But only the shadowing model shows a gradual transition. The reason is that only the shadowing model includes a probabilistic component and so does not creates the all-or-nothing case for the reception of packets at a specific distance.

4.5. Summary

The main goal of this chapter was the modelling of a wireless LAN communication system. From this discussion we will draw some conclusions about the effects and properties of a WLAN that have to be considered during the design and testing of a protocol, especially if we strive to provide quality of service guarantees. The design, development, and testing of a new communication protocol is usually done with the help of simulations. It seems obvious that a simulation-based testing requires at least a careful parameter setup and evaluation of the simulation results. Optimally, a cross-checking of the simulations with real experiments is performed. Only this way we can create a high confidence in the results we obtain from simulations. Although it seems obvious, many protocol designers do not care much about the parameters they are using for their simulations and how they perform the simulation and the evaluation of their results.

There are several reasons why simulations are performed improperly – ignorance, laziness, or insufficient resources. To perform a cross-checking of a simulation with a real experiment usually a lot of hardware, software, and manpower is required. In most cases it will be impossible to perform such an experiment. Nevertheless, if it is impossible to perform a full experiment, at least certain aspects should be tested. Another serious problem is that most users that use simulation for network protocol development and testing use it without knowing its fundamental principles and assumptions. They never question its correctness and ability to produce valid results. Additionally, using simplified simulation models is very attractive because it has the (virtual) advantage to produce better results. These claims shall be supported by some evidence.

Kotz et al. [KNE03] list six common axioms used in wireless network research (that will be discussed later in this section) and compares them with the results of real measurements. They also performed a survey of papers published in the MobiCom proceedings in the years 1995 to 2002 to determine the radio propagation models used in network simulations. Their survey shows that 90% of all papers presented simulation results based on very simple propagation models like the free-space or two-ray ground reflection model. The remaining 10% papers, which mostly came from researchers interested in the actual radio propagation or from the cellular telephone community, used more realistic propagation models. That means that the majority of all presented simulation results are very likely flawed. An example to illustrate the problems arising from simulations using simplified propagation models can be found in [IS05]. Here the authors compare the performance of the AODV routing protocol [PBRD04] using different propagation models including the two-ray ground reflection model, a ray-tracing based model, and propagation model similar to the shadowing model but without the random component. The results show two things. First, the performance numbers are overall better if we use a simple propagation model, that means higher throughput, smaller hop counts, and smaller protocol overhead. And second, that a shadowing-like propagation model reproduces the topological properties like connectivity quite well for a set of randomly placed nodes, mainly because the shadowing model computes comparable transmission ranges. In comparison with the ray-tracing based model it only fails to model the effect of large obstacles like houses in the path. Takai et al. [TMB01] performed a similar study to identify the effects of the physical modelling on the measured performance of the AODV and DSR routing protocol in a MANET simulation. They measured metrics like the

packet delivery ratio, end-to-end delay, and management overhead of the protocols for different physical models. Their results clearly show that the protocols behave better if we use simplified models, i.e. if we ignore signal fading, noise, or the overhead of the MAC framing.

In [PJL02] Pawlikowski et al. highlight another serious source of problems. Many researchers (not limited to communication networks) that are using stochastic discrete-event simulations seam to ignore the functional basics of such simulations – mainly how *randomness* is modelled. Network simulations include random events at many places, starting with random back-off timers and ending with random motions of nodes in a network. The authors emphasis that a simulation-based evaluation does not only requires to build a valid model but also to use it in a valid simulation experiment. This includes an appropriate use of random-number sources and analysis of the simulation data output. As we heavily rely on random number generators, an improper use will very likely falsify the simulation results. A basic principle of such simulations is that we can use a single simulation run only to perform a functional check of a communication protocol. We cannot derive statistical significant results from it. To do this we have to apply a method called *independent replications*. This means that we have to run the simulation several times with independent random-number sequences. As a guideline, a minimum number of 30 replications is considered sufficient for most network simulations to receive significant results. The authors also conducted a survey of 2246 papers published in the Proceedings of the IEEE INFOCOM, the IEEE Transactions on Communications, the IEEE/ACM Transactions on Networking, and the Performance Evaluation Journal. The survey showed that more than 50% of all papers presented results based on stochastic simulations but around 75% of them seamed not to be concerned with the random nature of the results obtained from their simulations.

A last (but less representative) example for how less many network researchers seam to care about propagation models can be found on the ns-2 users mailing list. A common question asked is how to setup the transmission range of wireless network devices. This shows that this value is often improperly treated as a property of the transmitter and not as a result of the radio propagation and the interaction of transmitter, receiver, and the medium.

As mentioned before, there are six axioms commonly used in network research:

1. The world is flat.
2. A radio's transmission area is circular.
3. All radios have equal range.
4. If I can hear you, you can hear me.
5. If I can hear you at all, I can hear you perfectly.
6. Signal strength is a simple function of distance.

The assumption of a flat world has already been discussed in section 4.2 because it is a requirement to use the model of an isotropic antenna to calculate the signal propagation. The assumption

of a flat world is valid under the following conditions: we have a flat terrain and the network extends not to far. Otherwise we are subject to the earth curvature, the directional antenna gain, or additional ground reflections. Just as an example for the earth curvature – if we place two antennas each 1m above ground we get a maximum distance with clear line of sight of 5km. This is much more than the usual transmission or interference range. So we have a flat terrain or nodes on the same floor inside a building, we can assume a flat world for a MANET simulation. But to be generic we should not rely on it.

The second axiom is that we have a circular transmission area. This is what all distance-based propagation models in combination with an isotropic antenna tell us. But as we showed in section 4.4.1 already the antenna of a ordinary WLAN card behaves completely different. The results from the ray-tracing based propagation model [IS05] discussed in section 4.2 clearly show that this is not valid if we consider obstacles – something everyone can easily experience just by walking around with a laptop inside a building. As a result we cannot and should not assume a circular transmission area.

The third axiom, that all radios have an equal range, is only valid if we use comparable hardware. The transmitters used in current available WLAN devices are quite equal. But the antennas used in laptops, PDAs, and access points significantly differ. Together with the awareness that the second axiom is invalid we can conclude that we should not rely on a transmitter determined range. For a simulation we have to consider the heterogeneity of the used hardware (if we have no concrete knowledge) in a more general way. As we have different parameters that influence the propagation range, we can add the antenna heterogeneity to them by either increasing the random signal variance or by choosing node configurations (transmitter and antenna parameters) randomly from a set a predefined profiles.

The assumption given in the forth axiom that we have symmetric links has been discussed already in section 4.3.3. There we saw that symmetric links are very likely if the propagation model does not include a random component. But even than we can get asymmetric links because of collisions. So we cannot rely on the symmetry axiom. Field tests (s. section 7.1 for an example) support this theoretical result.

The fifth axiom tells that we have an all-or-nothing receive probability. We already investigated this in section 4.3.3 and 4.4. The results show that this assumption is completely wrong and it is easy to experience this problem in the everyday life. As a consequence we have to cope with gradual link qualities (loss rates) that additional are very likely to change dynamically.

The last axiom tells that the signal strength is a simple function of the distance between sender and receiver. We already discussed that this is only true if we do not have to consider obstacles, reflection, refraction, multi-path propagation, and so on. We also saw that we can approximate the effect of different environmental distortions using the shadowing model and dynamic changes in the environment using the random signal variance included in the model.

For this work we have to consider the following facts:

- we are able to reproduce a lot of effects in a simulation but without a cross-check we cannot be sure if we simulated something useful,

- the shadowing propagation model can be used to reproduce various effects for a specific type of environment but can not be used to calculate the signal propagation in a specific environment,
- communication links can be asymmetric, e.g. have different receive probabilities in both directions (especially if they have a low quality),
- communication links have a gradual loss probability that increases with the distance,
- we can have variable transmission ranges between any pair of nodes,
- distance and signal strength/quality are not strongly correlated and cannot be used to calculate one out of the other,
- we do not have circular coverage areas,
- we can only trust a measured connectivity.

4.6. Testing and Evaluation of Applications in Wireless Networks

An essential element in the development cycle of every software product and hence for a communication protocol and its implementation is the testing and evaluation phase. It will show if the implementation functions as specified and if the protocol provides the intended properties. The most important thing that we have seen in the previous sections is that a WLAN and the different effects experienced in a real-world environment can be modelled (simulated) quite accurate. But we are not able to simulate the actual behavior in a concrete environment and can not live without real-world experiments.

The major problem with real-world experiments is that we are mostly unable to perform full-scale test because of hardware and personal resource constraints. Sometimes they are not even possible because the required hardware is not available at that moment (e.g. still under development). So we need to combine different test methods to get a best possible coverage of the target environment. We will use real-world experiments to perform small-scale tests and to test basic components of the system. This is supplemented by simulation-based experiments to test the system under various, sometimes uncommon, extreme conditions and to perform full-scale experiments. If we manage to reproduce the results of a real small-scale experiment in a simulation we are able to get a good approximation of the properties of a full-scale system.

Most academic projects that develop new communication protocols, especially for wireless networks, share the same problem – they lack a practical evaluation of their protocols. The main reasons for this are often the same – the lack of hardware and human resources to perform experiments of adequate scale. So most academic developers use simulation as the only source for measurements of the behavior of a protocol. From an academic point of view this can be

sufficient (depending on the simulation methodology) but for the acceptance in practice it is not. Furthermore not only the plain numbers of the network properties like end-to-end delay, bandwidth utilization, etc. are important. For an end-user the most important question to be answered is, how the applications he is using will perform on a network. With other words, what his usage experience will be like – will the application run smoothly or slow with long response times. The negative side-effect of the limited testing possibilities is that promising developments do not find their way into practice and so do not get the chance to mature which slows down the technological progress.

In this section we will therefore discuss different techniques that can be used to measure the plain performance metrics as well as to test the user experience of an arbitrary application communicating on the network.

The testing and evaluation includes the following components:

- a set of formal or informal assumptions about the environment,
- the protocol specification,
- one or more protocol implementations,
- the test setup or simulation environment,
- the test applications,
- and metrics that define the properties of the protocol.

Functional test

The first step of the testing is a functional test, i.e. we test if the implementation is executable in the test environment and if it includes obvious implementation errors. This point should not be underestimated – common implementation mistakes like missing variable initializations, improper memory addressing, improper type conversions, just to name a few, can cause any kind of strange protocol behavior. If we use both real-world and simulation experiments we need an implementation for each of the test environments. We additionally need a way to prove that all implementations work equivalent. Otherwise we will hardly be able to compare the results obtained from the different implementations or even to use the simulation results to extrapolate the results in the real-world. Section 6.2 presents GEA, a thin portability layer that allows to develop event-based applications and protocols that can be used unmodified in either environment. This solves the equivalence problem of multiple concurrent implementations. We will not discuss the problem of implementation errors as it is a common problem for every software development. But we will show an interesting by-product that can be used to test and debug distributed software implemented using GEA on a single host.

Protocol testing and performance evaluation

If we assured the principle operability we have to test if the protocol implementation works as specified. In the next step we have to measure its performance with respect to some predefined performance metrics. This is mainly done using network simulation complemented by some real-world experiments. To test single components and features synthetic applications like traffic generator are commonly used instead of real applications like a web browser or a video conference system. There are two reasons for this decision. First, traffic generators are able to create arbitrary traffic patterns. And second, traffic generators are commonly available inside a simulation environment whereas other applications are not. In the context as this work all simulations are done using the network simulator ns-2 [ns2b] (s. section 4.7.2 for detail).

Application testing

As said before, users are more interested in the actual behavior of an application than in the plain performance numbers mostly gained by measuring synthetic applications. For a small-scale network it is often possible to test an application in a life environment. But this way it is almost impossible to test an actual application in a full-scale network or in a network that operates in an environment that is not accessible for the developers. Here a possible solution is to use network emulation (real-time network simulation) that integrates real applications with a network existing only inside a simulation. Section 4.7.3 presents a solution that bases on an improved version of the ns-2. Network emulation provides a second solution to the problem of concurrent implementations but has a limited scalability.

Test methodology

We can summarize that we need to include real applications into our testing, that we have to avoid separated implementations for testing and deployment, and that we need simulation at least to test for scalability. Cavin et al. [CSS02] presented a comparison of different network simulators with an actual test-bed. Their results showed a significant divergence from the results obtained from the test-bed. So they concluded that network simulation lacks the credibility to be used for serious protocol development. A look on the used simulation parameters, mainly the use of the free-space propagation, shows that they better should blame the simulation setup for the bad results. But apart from this misinterpretation of the results their work clearly showed that a simulation without a careful simulation setup and at least a cross-checking of the basic elements of the system very likely leads to totally flawed simulation results.

As a consequence we propose the following methodology to solve most problems:

1. Use simulation for small- and large-scale testing of various network topologies. Complement the simulation with real-world experiments (with the largest scale possible) and compare with the same setup inside the simulation to verify the simulation results. Extrapolate the real-world results of a large-scale setup from the simulation.

2. Use a unified, event-based implementation to prevent concurrent implementations and the need to proof their equivalence.

3. Use network emulation to connect legacy applications with a simulated network and an interface-bridge (s. section 6.6) to connect the application with the deployment version of the protocol. For newly developed application, check if they can be implemented in an event-based manner to be integrated directly with the implementation of the communication system.

This methodology cannot solve the following problems: It is not an replacement for a full-scale test of the whole system in the real world. It only provides an approximation with some uncertainty. This uncertainty has to be minimized by a precise comparison of some small-scale real-world experiments with comparable simulation setups. It also does not completely removes the need to proof an equivalent behavior of two implementation, but it reduces it to the event-based runtime system (s. section 6.2 for detail) that is quite slim and shared by all possible implementations. This eases this problem significantly.

4.7. Network Simulation and Emulation

In this section we will take a look onto the principles of network simulation and the difference between simulation and emulation.

4.7.1. Definitions

> A *simulation* is an imitation of some real device or state of affairs. Simulation attempts to represent certain features of the behavior of a physical or abstract system by the behavior of another system. [Wikipedia]

Simulation is a widely used technique to examine systems and processes that are hard or impossible to observe. Systems can be hard to observe because they change their state very fast (i.e. some chemical reactions) or very slow (i.e. the evolution of galaxies). So a simulation is used to change the time scale of such a process. Simulation is also used to perform experiments that are otherwise impossible in reality. This can be because it is either to dangerous to carry out an experiment or because it would require an unreasonable high effort.

In the context of this thesis we use simulation as a technique to experiment with communication networks that are hard to realize because of various reasons. To distinguish those experiments from other possible meanings of the term simulation we will refer to it as *network simulation*.

We define *network simulation* as a technique where entities of a real network, such as nodes, routers, switches, links, are modeled by simulation components that imitate the behavior of there

real-world counterparts, i.e. produce the same sequence of state changes when provided with the same input data.

When we speak of network simulation we require that it imitates the internal behavior of the simulated network entities as close as possible. We do not make any assumptions about how many time is consumes to compute the sequence of state changes. As special case where we require a special timing behavior ist *network emulation*.

> An *emulator*, in the most general sense, duplicates (provide an emulation of) the functions of one system with a different system, so that the second system appears to behave like the first system. [Wikipedia]

In the field of simulation, emulation is often referred to as *real-time simulation*.

We define *real-time simulation* as a modeling technique where components of the simulation (simulator objects) reproduce a timing behavior similar or equal to the timing behavior of the simulated targets (simulated entities).

The major difference between simulation and emulation is that we do not require an emulator to precisely model and reproduce the state of the device or process the we simulate, only its behavior observed at a defined interface.

We define *network emulation* as a technique to reproduce the functional properties of the entities of a network and the timing behavior observed at the network interfaces of the nodes of the network.

In practise this means that we strive to replace an arbitrary, real network with an emulation of it, so that applications communicating over the network can interact with the emulation exactly the same way they would with the real network. One important prerequisite for a simulation system in order to become an emulator is that it does not only reproduces the timing behavior of the network, it also has to deliver the data the same way, the real network would. That means that the emulator has to carry the whole payload of a packet and additionally has to perform any modification on the network packet headers that would happen in the real network.

4.7.2. The Network Simulator ns-2

The network simulator ns-2 [ns2b] is a widely used simulator for various kinds of communication networks. It is a discrete-event simulator that usually uses a discrete-time model but also supports continuous-time models. NS-2 was originally developed as part of the NFS/DARPA founded *Virtual InterNetworking Testbed* (VINT) project. It is a successor of the REAL [rea] network simulator that has been developed at Cornell University until 1997. The simulator is developed as an open-source project and freely available, the main reason why it became a defacto-standard in the network research community. During the last several years a large number of volunteers has contributed improvements, extensions, and helper tools to the ns-2 project. Since 2005 the project has changed into a community controlled open source project [ns2a]. The source code

(about 400.000 lines of code without external contributions) is now licenced under the GPL [Fre91].

ns-2 provides a wide range of simulation models for different technologies and protocols for wired, wireless, and optical transmission systems, routing protocols, and some application models. Overall, a clear focus on the lower network layers can be seen. So it fits very well for the simulation needs in this work.

There are other network simulators, most notable OPNET's ITGuru/Modeller [itg], a commercial simulator mainly targeted for the enterprise market and hence specially focused on the simulation of higher protocol layers and applications. Of cause it supports a wide range of technologies and protocols on the lower layers. Other well known simulators are GloMoSim [ZBG98, ea99] and OMNeT++ [Var02, Var01] that can be used for MANET simulation. Only little data exists that compares the different systems with each other. At least for the most widely used (ns-2 and OPNET) there are some studies [LPFJ+03, Mal04] that compare their performance for several network types, the design and usability. They conclude that both produce almost equivalent results that match the result gained from test-bed experiments, although they have different weaknesses simulating certain protocol aspects.

4.7.3. Network Emulation with ns-2

Real-time simulation is a modeling technique where components (simulator objects) reproduce a timing behavior similar or equal to the timing behavior of the simulated targets (simulated entities). During the development of an application it interacts with the simulated environment in the same way it would interact with a real one. This allows to test it in different environments with relatively small effort before.

Network emulation systems have been used for many years. Examples are EmuNET (first version) [Men97], DummyNet [Riz97], x-Sim [BP96], the hitbox pseudo-device [DLY95], and NistNET [CS03]. These are all systems that do not emulate the physical properties of the communication but instead reproduce effects like packet delay, reordering, jitter, bandwidth limitation on a statistical basis which is sufficient for many tasks, especially for the emulation of wired networks. For MANET simulations these systems are not sufficient. Systems that target the area of mobile communication are the newly developed EmuNET [KEHM04] and Seawind [KGM+01] primarily designed for GPRS networks, as well as MarNET that primarily addresses the problems of mobility management and client integration [EFMS04]. It should be noted that the MarNET emulation environment is also part of the special focus program SPP1140 to which the results of this work belong. It will be integrated with the emulation solution presented in the following that primarily addresses the problem of an exact emulation of the properties of the wireless communication.

The network simulator ns-2 already provides an *emulation* feature [Fal99], i.e. the ability to introduce the simulator into a live network using a soft real-time scheduler which tries to tie the event execution within the simulator with the real-time. This feature has not widely been used and maintained for years because it was intended for wired network research where more efficient

solutions are available. It gained new interest in recent years for wireless network research but proved to be unsuitable. The main reason is that event execution can hardly be tied to real-time without deviation. There are several factors that increase this deviation. First, if the execution of an event takes longer than the corresponding action in the real world. This often happens in the simulation of hardware devices or physical media properties. Second, by the nature of network simulation, the simulator has to execute multiple events at different nodes at the *same* time. This can hardly be achieved even with a distributed ns-2 version [RFA99]. So they have to be serialized. And third, the simulator heavily interacts with the operating system during the execution of events which causes unpredictable additional delays. So, normally the scheduler executes events with a delay from the real-time which is crucial especially for the performance and behavior of wireless network protocols. The following example will illustrate this.

Consider a RTS/CTS frame-exchange sequence in the IEEE 802.11 protocol [Ins99]. At the beginning a *transmit* event at the initiating node starts a transmission timer. When this timer expires and still a CTS frame has not been received, the node concludes that the RTS/CTS exchange has failed. However, due to delays in the event execution during this exchange, the node receives the CTS frame after the timer has expired, even if no transmission errors have occurred.

We see that the timing of the event-processing becomes an essential component of an emulation. But this is only one component of an emulator. The second major component is the integration of legacy applications with the simulator engine. The developed emulation environment consists of an improved version of the ns-2 simulator core combined with a special network setup that can integrate native clients running inside unmodified operating systems either using a virtual machine (e.g. UML [citc]) or on separated hosts. Figure 4.24 depicts the principle setup.

The modifications of the ns-2 include a number of performance enhancements to increase the overall event throughput but more important architectural modifications and a modified handling of time in the simulation models. A detailed description of the modifications can be found in [MI04c, MI05] and an in-deep explanation of the practical setup in [MI04b, MI04a].

4.7.4. Simulation Model Setup

Based on the discussions and observations in the previous sections, this section will detail the simulation environment used for all simulation-based experiments throughout this thesis. The main goal of this section is to define a simulation setup that is as close to a real-world setup as possible. This means that the simulation has to produce all effects that can be observed in the real world. It furthermore should allow to adjust a simulation to a specific hardware setup to mimic the transmission and propagation properties of the used hardware – if known. This allows to cross-check simulation results with real experiments to increase the confidence in the results gained from a simulation.

We already discussed that the shadowing propagation model is the only one that suits our needs and is applicable even with little knowledge about the environment. If we need to include the effect of large obstacles in the simulation, ns-2 provides an extended shadowing model that uses a 2D ground view to determine obstacles between transmitter and receiver. Two open problems

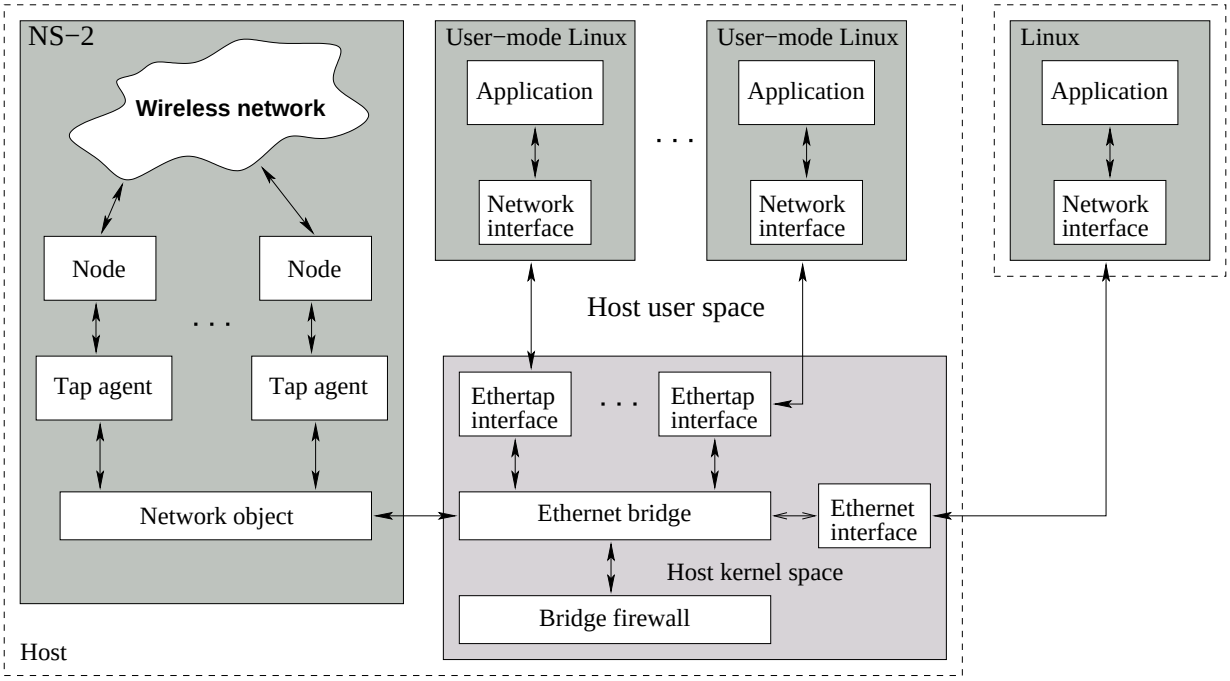

Figure 4.24.: Network emulation setup

remain. The first is to setup the parameter of the shadowing propagation model based on real-world measurements, and second to model the near-range packet loss.

All simulation are done using the network simulator ns-2 [ns2b] version 2.28 including the real-time improvements developed together with Svilen Ivanov [nse, MI04c, MI05]. Appendix A.1 lists the full set of simulation parameters. The most important parameters will be discussed in the following and we will investigate their effect on the results of the interference experiment. In the opposite direction this knowledge helps to setup a simulation based on the results measured in a real-world environment.

The most important parameters are the transceiver properties and the parameters of the propagation model. The parameters and their variable names inside the ns-2 are listed in table 4.6. The transmitter power is often reported by the WLAN device. The actual values for the carrier sense threshold and the receiver threshold are usually unknown. In some cases (s. [Cis05, 3Co04] for example) the receiver threshold and rarely the carrier sense threshold can be found in the data sheets of the device manufacturer. The receiver threshold is often denoted as *receiver sensitivity*.

At first we will investigate the influence of different values for the carrier sense and receiver thresholds. All following experiments use the values from the interference experiment explained in section 4.4.3 as a starting point. Figure 4.25 depicts the results from the interference with different carrier sense to receiver threshold ratios. In this series of experiments we set the parameters of the shadowing model to 3.6 for the path loss exponent and 4dB for the standard signal variance. We set the receiver threshold to -94dBm and vary the carrier sense threshold from -110dBm to -94dBm. We see that the shape of the chart is similar in all cases. With a growing

Variable name	Description
Pt_	transmitter power
CSThresh_	carrier sense threshold
RXThresh_	receive threshold
pathlossExp_	Path attenuation exponent
std_db_	standard distributed signal variance

Table 4.6.: Important ns-2 simulation parameters

difference between both thresholds the interference effect also grows. That means that the range in which a node is blocked from transmitting by the physical carrier sense but cannot receive the transmission of the other node increases – visible as the width of the range with decreased relative bandwidth. The starting distance of this effect is constant in all cases but the distance where we find the minimum relative bandwidth increases with the threshold difference. The minimum relative bandwidth value decreases with an increasing threshold difference. But the minimum will not drop below 50%. The reason is obvious, even if we have an infinite carrier sense and nearly zero transmission range, the 802.11 MAC layer will ensure a fair bandwidth sharing between the two nodes, but one node will not see packets from the other one. That means that we will measure a relative bandwidth utilization of 50%.

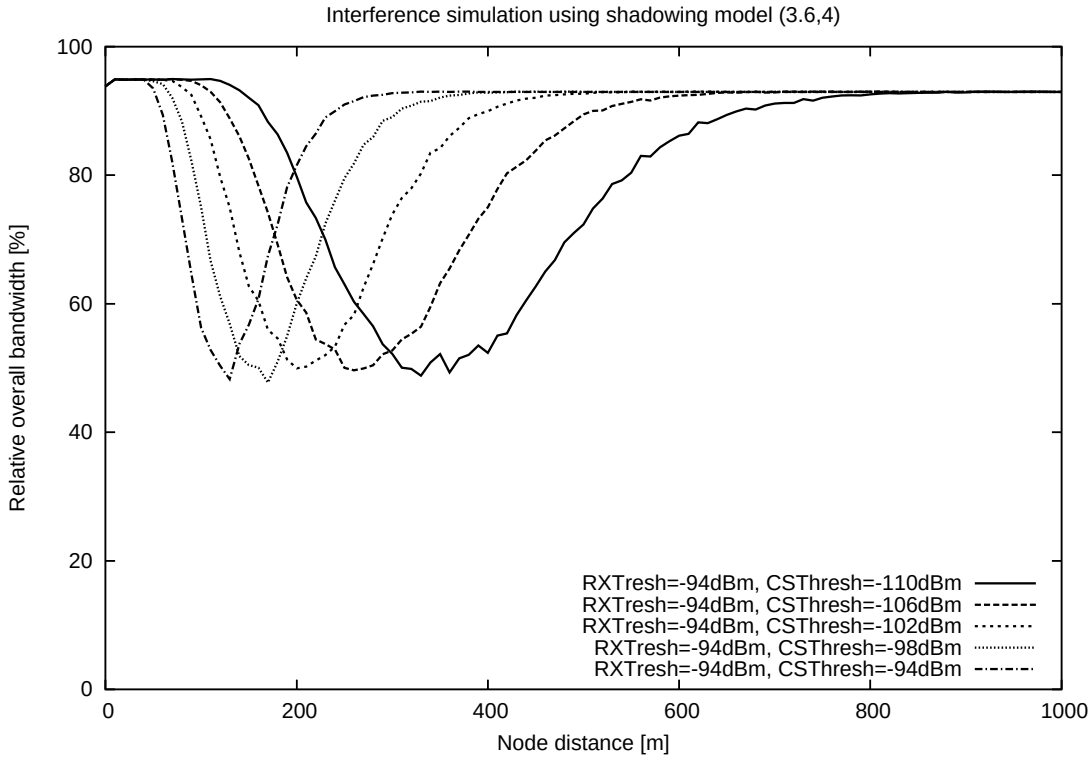

Figure 4.25.: Interference experiment with varying carrier sense to receive thresholds

The second series of simulations varies both thresholds at the same time and keeps their difference and all other parameters constant. Usually these thresholds are fixed in a present hardware. But changing the thresholds and keeping their difference constant is equivalent to changing the antenna gain, e.g. by replacing the antenna with a different one. Figure 4.26 shows the results from these simulations. We see that higher thresholds result in smaller transmission and interference ranges. As a consequence the range at which the relative bandwidth utilization starts to decrease grows with smaller thresholds. The same holds for the range where the bandwidth utilization reaches a minimum. The minimum value is not affected by the chosen thresholds.

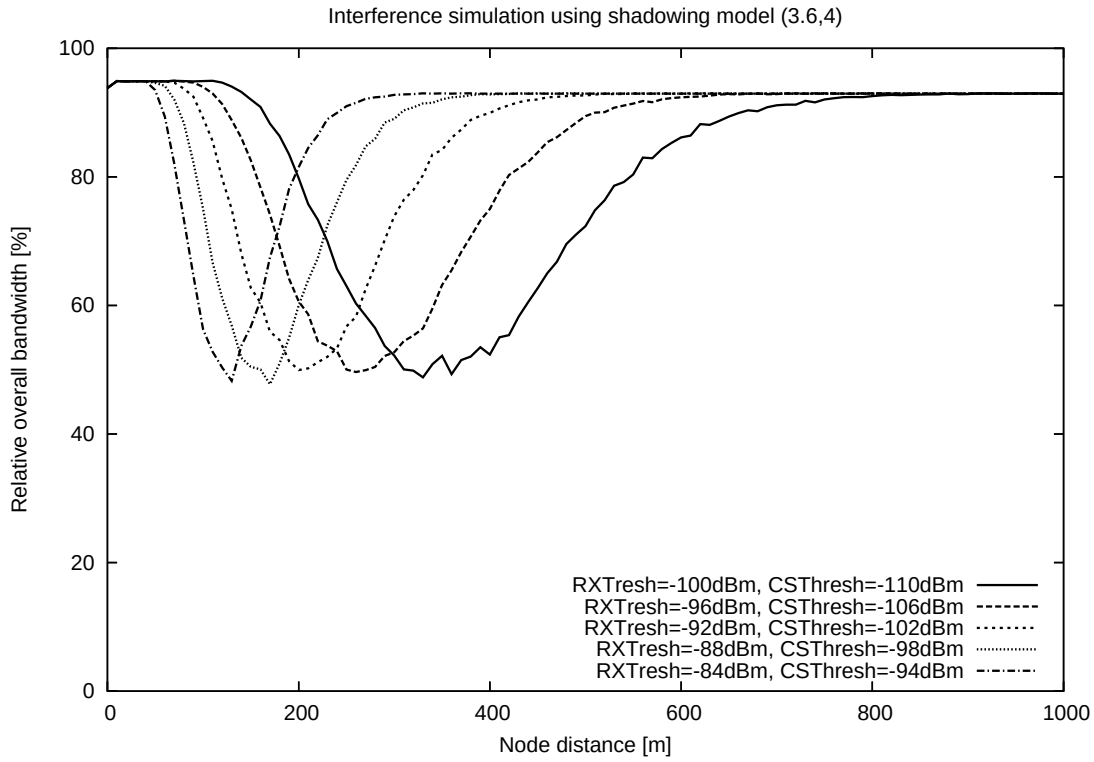

Figure 4.26.: Interference experiment with constant carrier sense to receive thresholds

The third series of simulations investigates the influence of the parameter for the signal variance. The thresholds are set to -104dBm for the carrier sense and -94dBm for the receiver. All other parameters are not modified. For the signal variance we choose values from 0dB to 10dB. From the results shown in figure 4.27 we notice that this parameter also affects the ranges at which the bandwidth utilization starts to decrease, where we have the minimum utilization, as well as the minimum value itself. But we notice two special points that we can use to calibrate the simulation parameters based on real measurements. All charts intersect in two points left and right from the minimum. The left intersection (with the lower range) is the most interesting point because it lies exactly in the middle between the minimum and the point where the bandwidth utilization starts to decrease. That this point is not just a coincidence can be explained with equation 4.8. The propagation model includes a standard distributed random variable. The mean value is constant

regardless of the chosen parameter for the signal variance which results in an equal number of events with a transmission range of the intersection point.

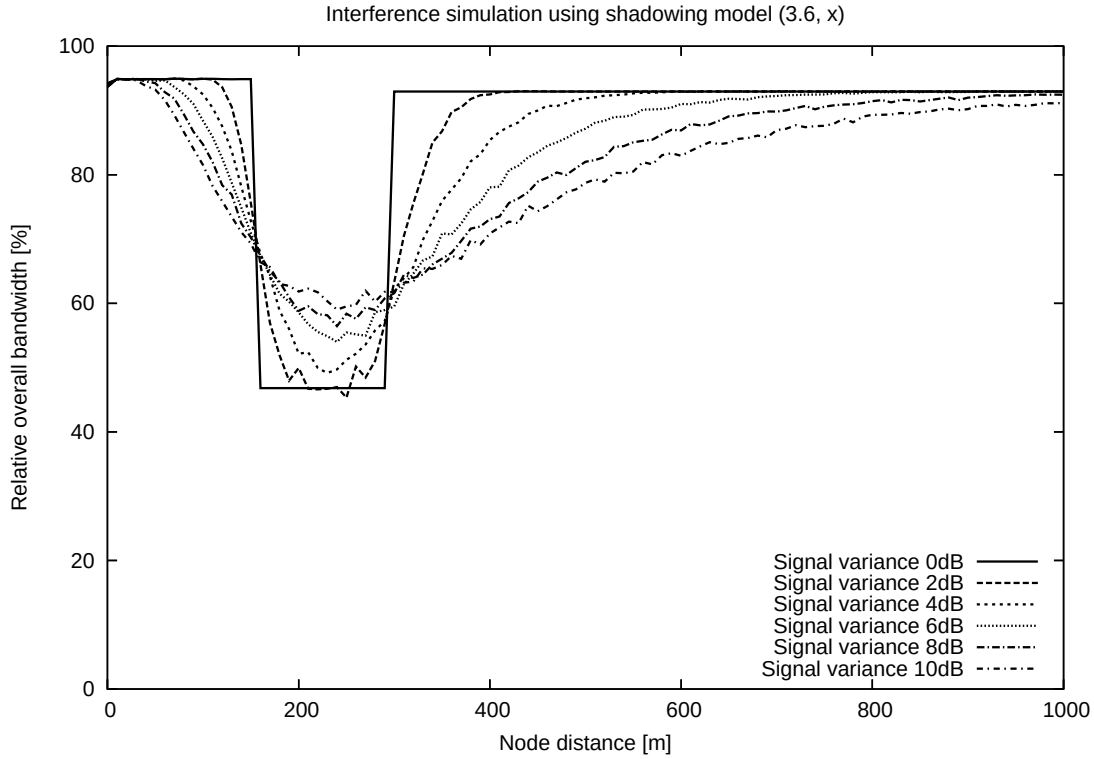

Figure 4.27.: Interference experiment with varying signal variance

The last series of simulations investigates the influence of the pathloss exponent parameter. Figure 4.28 depicts the results from this series. We see that the pathloss exponent affects the range where the bandwidth utilization starts to decrease as well as the range where it reaches the minimum. But it has no affect on the minimum value of the relative bandwidth utilization.

The previous observations showed that the simulation can be fined-tuned in several ways. But there is no exact solution to choose the parameters based on an actual measurement. The main reason is that all parameters have an effect on most of the significant points in the chart and so create some ambiguities. But with the help of information about the actual setup we can resolve them. If we know the environment type, we can restrict the range of possible values for the path loss exponent. If we have access to the specification of the used WLAN cards we can fix the power thresholds.

In section 4.4.2 we saw that even in the direct neighborhood of a transmitter we can experience packet loss – although the loss rate is very low. This effect cannot be modelled by the shadowing model so we have to add it on-top. To do this we use the simulators ability to drop packets based on a separated error model. NS-2 allows to apply an arbitrary error model to each transmitter and receiver. Dropping a packet at the transmitter is a bad choice because there is no reason why

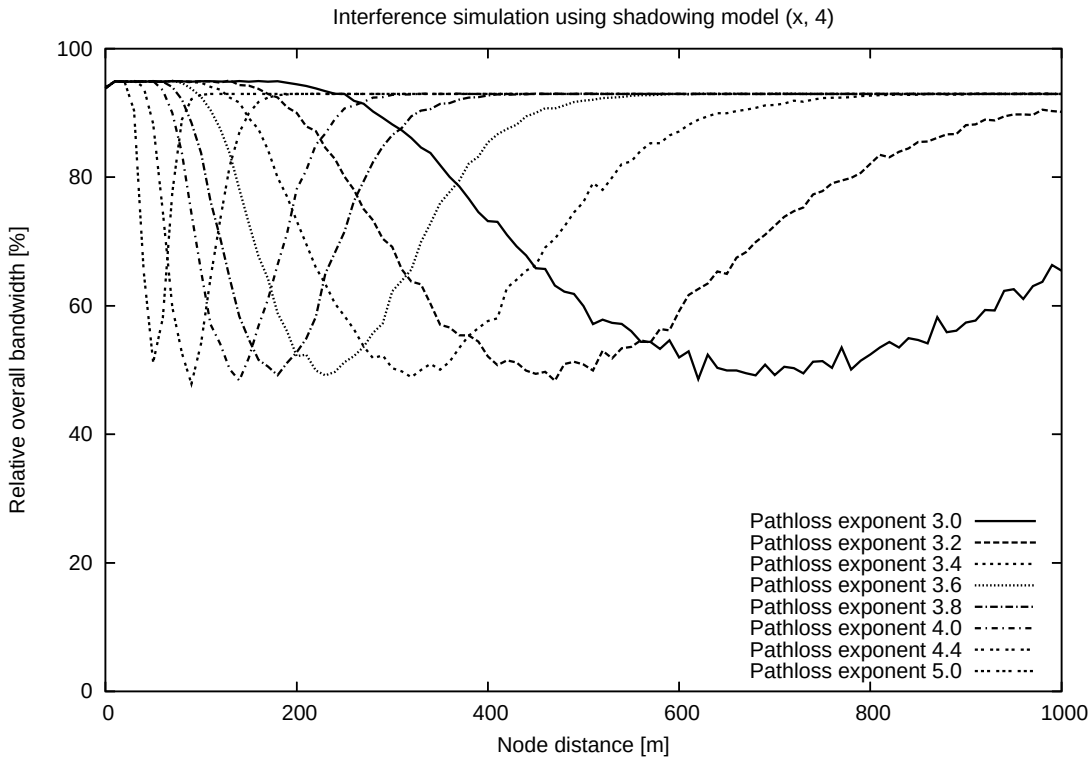

Figure 4.28.: Interference experiment with varying pathloss exponent

a packet should not reach all transceivers within transmission range. Additionally, each packet dropped at a transmitter would not occupy the wireless channel and so falsify the bandwidth usage. Appendix A.1 shows the source code to add an error model that creates a uniformly distributed packet loss with an adjustable loss rate. A uniform distribution has been chosen because we can only determine a statistical mean value and not the concrete distribution.

4.7.5. Practical Problems

> In theory there is no difference between theory and practice. In practice there is.
> [Lawrence Peter “Yogi” Berra]

Limited transmission range and interference are not the only sources of message loss and disconnected nodes in practice. There are more things that can happen and should be mentioned in the following. From a pure academic point of view these problems are irrelevant but in practice one has to be aware of them and consider them in an productive system.

The most serious problem is the implementation of the ad hoc mode in current WLAN cards. If a card is switched to the ad hoc mode it should start to send small beacons to announce itself to other cards. The beacon contains a *CellID*, a unique identifier for the ad hoc network, and

the service set identifier (SSID). If two nodes with the same SSID meet, they should agree on a single CellID (merge). Here the problems start – some cards do not start to send beacons and fall back to the managed (access point) mode, some are not willing to merge their CellIDs, some periodically choose a new CellID if they do not received one from other hosts at startup which can confuse other nodes that try to merge. These problems usually cause the network to break apart.

In practice we experienced a wide range of other strange, mostly driver related problems with current WLAN cards. Just to name a few examples – some drivers reset the card under heavy load, some silently drop packets for unknown reasons, some even send non standard compliant frames.

5. Providing QoS for Publish/Subscribe Communication

In this chapter a publish/subscribe communication system will we developed that provides end-to-end quality of service guarantees. The chapter starts with a definition of the intended system, its abilities and limitations, and the assumptions that where made about the operation environment. This is followed by a discussion of the basic functional blocks that are required to provide publish/subscribe communication. The chapter continues with a discussion of the fundamental problems that have to be solved to provide quality of service in a MANET with a special focus on the problem of bandwidth sharing and management on the wireless medium. This problem discussion is based on the studies and results given in chapter 4.

5.1. Objectives

In section 2.2 we saw several possible P/S models. Not all are equally well suited to provide QoS guarantees. A basic principle is that we need a kind of determinism and computability to enforce the quality guarantees we give to an application. Additionally not all QoS properties are equally easy to enforce, some are conflictive. For example, we cannot strive for high reliability and throughput at the same time. Reliability usually requires some kind of redundancy which decreases the capacity of the communication channel. So there has been a tradeoff between them – the more reliability we require, the less bandwidth we can offer.

5.1.1. Publish/Subscribe Communication

But first we have to decide for one of the P/S models. A subject-based P/S system, sometimes also referred to as *first generation* P/S system, provides less flexibility in terms of the abilities to select interesting events compared to a content-based P/S (also referred to as *second generation* P/S system). An increased flexibility of course results in an increased uncertainty about which events are to be delivered to which subscribers. If we allow only one subject to be assigned to each event, all events created by a publisher have to be delivered to the same set of subscribers. This set will only change if subscribers on the subject enter or leave the system. So we get a comparatively static relationship between publishers and subscribers. All other P/S models are more flexible and hence more dynamic. So the subject-based P/S model will we chosen. This seems to be a very serious limitation of the P/S communication scheme, but only in the first place.

We will not sacrifice the flexibility and expressiveness of a hierarchical subject-based or even content-based P/S system to provide QoS guarantees. We will only sacrifice the optimization potential in terms of the overall network traffic of those models. This statement will be explained in more detail in the following.

First we have to discuss how we can map a hierarchical subject-based or content-based P/S to a simple subject-based P/S system. And second, what overhead we introduce by such a mapping. The basic principle for the mapping of a complex to the basic P/S system is to aggregate subscriptions on a single subject and perform the fine-grained filtering locally on the receiving node before passing the events to the subscriber. For a hierarchical subject-based P/S system this mapping is straight forward – we only regard the top-level subject in the event service and filter the sub-level subjects at the destination (as in the STEAM system [MC02]). For a content-based P/S there is no such trivial mapping. We need to aggregate subscriptions – how this can be done, highly depends on the actual subscription mechanism. For systems using for instance XPath-based [W3C99, W3C03] subscriptions like XNet [CF04b] such an aggregation is done at run-time. But for a mapping to a simple subject-based P/S system we need a static aggregation, i.e. a function that transforms content-based subscription into a subject, or a distributed service that performs this aggregation. If we use a key-value-pair structured content we can disregard the value part from a subscription and use the key as the subject. As a last resort, e.g. for systems using binary predicate objects, we can map all subscriptions to the same object, those transforming the P/S system into a broadcast overlay network between all publishers and subscribers.

A more flexible P/S model not only provides a more fine-grained event filter mechanism, for the system it allows for more traffic optimization. If a subscriber can select events in a more specific way, the event service probably has to forward less events. If we map a complex subscription scheme to a simple subject-based and perform a fine-grained filtering at the destination, we obviously waste bandwidth on the network – every event that has to be discarded at the destination has been transmitted unnecessarily. In the contrary such an optimization does not come at zero costs. In order to filter events closer to the source we need to communicate subscriptions backward in the network, e.g. to update the aggregated filters along the distribution path. How this is actually done depends on the concrete system, but it is obvious that it has to be done. This means that the overhead for the mapping from a complex to the simple subscription scheme is smaller as thought in the first place. Another aspect should be mentioned at this point. So far we only looked at the network traffic. But the processing of events on intermediate nodes also requires processing power. This is especially true if we consider a content-based P/S system. For example, if we encode events as XML documents and use XPath-based subscriptions, we have to perform XML processing on each node. This obviously requires more computing power than a simple subject matching. There is little information about the actual computation overhead of the event matching. An example can be found an the XMLBlaster website [Ruf00]. They measured a throughput of 672 messages per second on a 600 MHz CPU. Such a CPU seams quite slow compared to current desktop CPUs but is still faster than most mobile CPUs. So this event throughput is smaller than the possible number of messages a mobile device can receive per second which makes it unsuitable for such devices. We have to consider the processing time for another reason – more processing time also increases the delay of events transmitted

over multiple hops – something we have to be very carefully of if we are going to provide QoS guarantees.

We can conclude that the decision for a subject-based P/S system does not expel complex subscription schemes – we can map every complex subscription scheme, more or less efficient, to a simple subject-based P/S system. By doing this we acknowledge that we potentially waste bandwidth on the network. At the same time we significantly decrease the processing overhead on all nodes that have to perform event matching. By choosing a complex subscription model we would definitely waste scarce processing power and rise the lower bounds for possible end-to-end delays. That means we have to decide whether we accept a possible bandwidth degradation or a definite delay increase. Because the later would inhibit certain delay-sensitive applications like VoIP we have another argument to decide for the subject-based approach.

5.1.2. Quality of Service

In section 2.1 we already defined some QoS properties that could be relevant for a P/S system. In the following we will discuss the QoS properties that will actually be regarded in the developed system and give reasons for the decisions made.

As a first group we have *latency*, *bandwidth*, and *jitter* that relate to the timing of the event delivery.

Latency is measured as the time from creating an event until its reception. The problem with this measurement is that it requires synchronized clocks on all nodes. Such a synchronization is possible up to some degree, but it requires additional communication. We therefore replace latency by *age*. In an ideal case, where we have synchronized clocks, the age of an event would be the same as the latency. But this is something we cannot assume. So we will measure the age of an event in a distributed fashion (s. section 5.7).

The end-to-end bandwidth that will be guaranteed by the system is specified on a packet base, not as a byte per second value. A data stream is determined as a flow of packets each with a maximum given size and interval. Individual packets of the flow can be smaller but not larger than the specified size. The reason for not choosing a byte-per-second approach is that it would produce an indeterministic number of packets. But such a deterministic flow of packets will be needed to coordinate the bandwidth usage on the medium (s. section 5.5) and to detect QoS violations.

Jitter will not explicitly be handled in the developed system. It is an implicit result of the traffic flow definition. Each flow has a given period. The system will guarantee that a packet will arrive within this time period. So the jitter can become not larger than twice the period time without causing a QoS violation. The actual jitter can be smaller, but this depends on the actual implementation

The second group of QoS properties, that more relates to the P/S rather then the network part, includes *reliability* and *persistence*.

Reliability is very difficult to enforce in an wireless communication system. Here it will be handled an a static basis. That means the system will allocated an appropriate amount of redundant resources to increase the reliability. There will be no direct reliability QoS property that can be controlled by an application. Instead an application can control how much overhead (e.g. multiple transmissions) the system will spend or which links (e.g. specified by a minimum link quality) it will use to increase the reliability. Unnecessary allocated resources will be used for non-QoS communication but not for other communication with QoS guarantees. The reason for this is simple, non-QoS traffic can be skipped at any time without notice. But if we dynamically reallocate resources from one QoS stream to another we may violate our initial promises later. If there is optimization potential, the system will use it. But it is not guaranteed that it will spend more resources to react to bad conditions. Because there are only statistical or empirical relations between the spend resources and the obtained reliability we will not speak of reliability guarantees. But there is a guarantee associated with reliability, the guarantee to detect violations of a required reliability and to give a notification in time. This can easily be done using traffic monitors. Based on the flow specification they can count received, lost, and delayed events. These measurements are also required to detect other QoS violations and so introduce no additional overhead. The measurement results can be simply compared with threshold values (e.g. average loss rate, burst loss count) to trigger a notification about its violation.

Finally, we have to discuss the persistence property. In P/S systems that do not provide QoS guarantees for bandwidth, latency etc. it is a common feature to assign events a lifetime and to store the events for at least this time. If a subscriber joins the system during the lifetime of the event, the P/S will deliver the event also to this new subscriber. Such a persistence does not make much sense if we concurrently have a latency requirement for each event. The reason is that the lifetime can only be as high as the maximum latency. If a subscriber joins the system after the event has been created its existence first has to be announce, its subscription has to handled and then the event has to be forwarded to the subscriber. In most cases this requires to allocate a route that supports the required QoS properties. All this activities take some time, usually much more than the actual event forwarding. So it is very likely that we will not have enough time left to successfully deliver the event to the subscriber. As a consequence we will not provide such a persistence property in the system. Events will only be delivered to subscribers that already have established a connection to the publisher. A node in the network will only store an event to forward it to a number of other nodes or during the time of a route repair.

5.2. Communication Environment Assumptions

In this section we will discuss the assumptions we make about the environment in which we will operate. These assumptions provide a basis for the discussion of the developed protocols. Some are direct consequences of the observations made in the previous chapter (s. section 4.5 for a summary).

What assumptions do we make about the communication environment:

- *Heterogeneity*: Mobile stations are equipped with heterogeneous transceivers (transmission power, thresholds), antennas, and system resources (processor, memory).
- *Interoperability*: All mobile stations use IEEE 802.11 compliant communication devices.
- *Link Quality*: A uni-directional link between two nodes can have any quality (loss probability) between 0%–100%.
- *Asymmetry*: If node A can receive information from node B, that does not mean that B can receive information from node A.
- *Cooperation*: All nodes run the same software stack and we do not have malicious nodes, i.e. nodes that deliberately violate the protocol specification.
- *Scalability*: The developed protocol is intended for networks up to 1000 nodes.
- *Node Identification*: Nodes are identified by a fixed, unique number that has to be assigned locally.

Heterogeneity is a direct consequence of the application scenarios. As we assume a variety of different devices (laptops, PDAs, ...) in an open environment we cannot make any assumptions about the hardware like byte order, processing speed, reliability of the hardware and so on. But what we require is that they run the same protocol stack in a cooperative manner. That means that we require that all nodes use IEEE 802.11 compliant WLAN devices and compatible versions of the P/S communication stack. Compatible does not necessary means identical binaries but we have to make sure that they all use the same message formats. The cooperation is a very important requirement because the developed protocols will not focus on the problem of malicious behavior. The protocol will handle all problems that arise from the communication medium (message loss, delays, reordering) but not problems caused by a deliberate misuse, e.g. invalid modification of parameters in forwarded messages. The middleware will ensure that it only generates valid messages but will not perform any measures to check or enforce the integrity afterwards – this is out of scope. Cooperation is also very important to provide QoS. Just as a quick counter-example. If we have one or more malicious nodes that constantly pollute the medium with uncontrolled messages we have little chance to perform any bandwidth reservation.

5.3. Protocol Layers

The publish/subscribe communication system can be divided into three layers (s. figure 5.1).

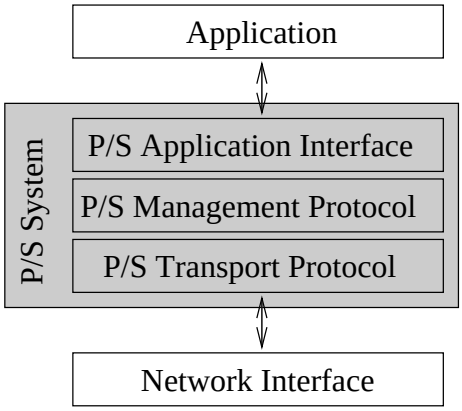

Figure 5.1.: P/S System layers

The application interface defines how an application accesses the P/S communication system. This includes the definition of the communication endpoint, i.e. how an application can send or receive events, the method to specify the content of an event, as well as the method to express interest in events and the associated QoS requirements. The management protocol describes how the system handles publishers, subscribers and their subscriptions. As we are going to provide QoS guarantees the management protocol is also responsible to perform the necessary resource allocation and admission control. Finally, at the lowest level, the transport protocol describes how an event is delivered from a publisher to all interested subscribers. The transport protocol is responsible to enforce the quality of service guarantees. In the following we will take a closer look on an abstract version of a P/S system to explain the peculiarities of the developed system that are necessary to provide QoS guarantees. The concrete mechanisms and implementations will be discussed afterwards.

5.3.1. P/S Application Interface

To provide a publish/subscribe communication the application interface at least needs to provide primitives to send events, to express interest in certain events, and to receive events. Additionally we need to define how to describe the content of an event. This last point we already discussed with respect to the QoS constraints in section 5.1.2. The decision was made to use a subject-based P/S system because of its comparable good predictability. That means that we will describe the content of an event by a flat subject. If necessary this can be used to build a more fine-grained P/S system on-top of it.

As predictability is a key element to provide QoS the application will have a significant difference to other P/S systems. To fulfill the P/S communication scheme it is sufficient that a subscriber

explicitly expresses its interest in various events and that a publisher just creates an event. The P/S middleware will route the events to the subscribers based on their content. To deliver events within a given time-bound it is necessary to know the routing before the event is transmitted. The reason is that we have to find a route that ensures a timely delivery beforehand. Usually this includes a reservation of some bandwidth along the path. In order to allow the planning and reservation of the routing path the application interface requires an explicit registration of a publisher. So publishers and subscribers are treated the same way. An application that wishes to publish or to subscribe has first to register with the P/S middleware. This registration creates a one-way communication end-point that is bound to the transmission of events with a given subject. For the middleware software, especially the P/S management protocol, this registration allows to determine the sources and destinations and hence the necessary delivery paths for all events related to a certain subject before they are actually created. Other P/S systems (s. section 3.2) like SIENA and Elvin use an explicit publisher registration to filter events as close to the publisher as possible.

After an application has successfully created a communication end-point (`Publisher` or `Subscriber`) it has to wait until it becomes active. This means that the registration creates an interface object and concurrently calls the middleware to connect it to all necessary peers. This can take some time. So the application will be notified when this setup process is finished. Afterwards an application can start to publish events on the subject. A subscribing application does not need to listen for the activation notification because it will be notified about incoming events anyway. If the middleware detects an error, e.g. a broken connection or QoS violation, it will send a notification to the application. If an application does not longer wants to publish events or is not longer interested in a subject it has to destroy the interface object. This will instruct to middleware to release the internal connections and corresponding resources. If a communication end-point is not destroyed, e.g. because the application using it crashes, the middleware will later destroy it because of buffer overruns or QoS violations. Figure 5.2 depicts the whole life-cycle of a P/S communication end-point.

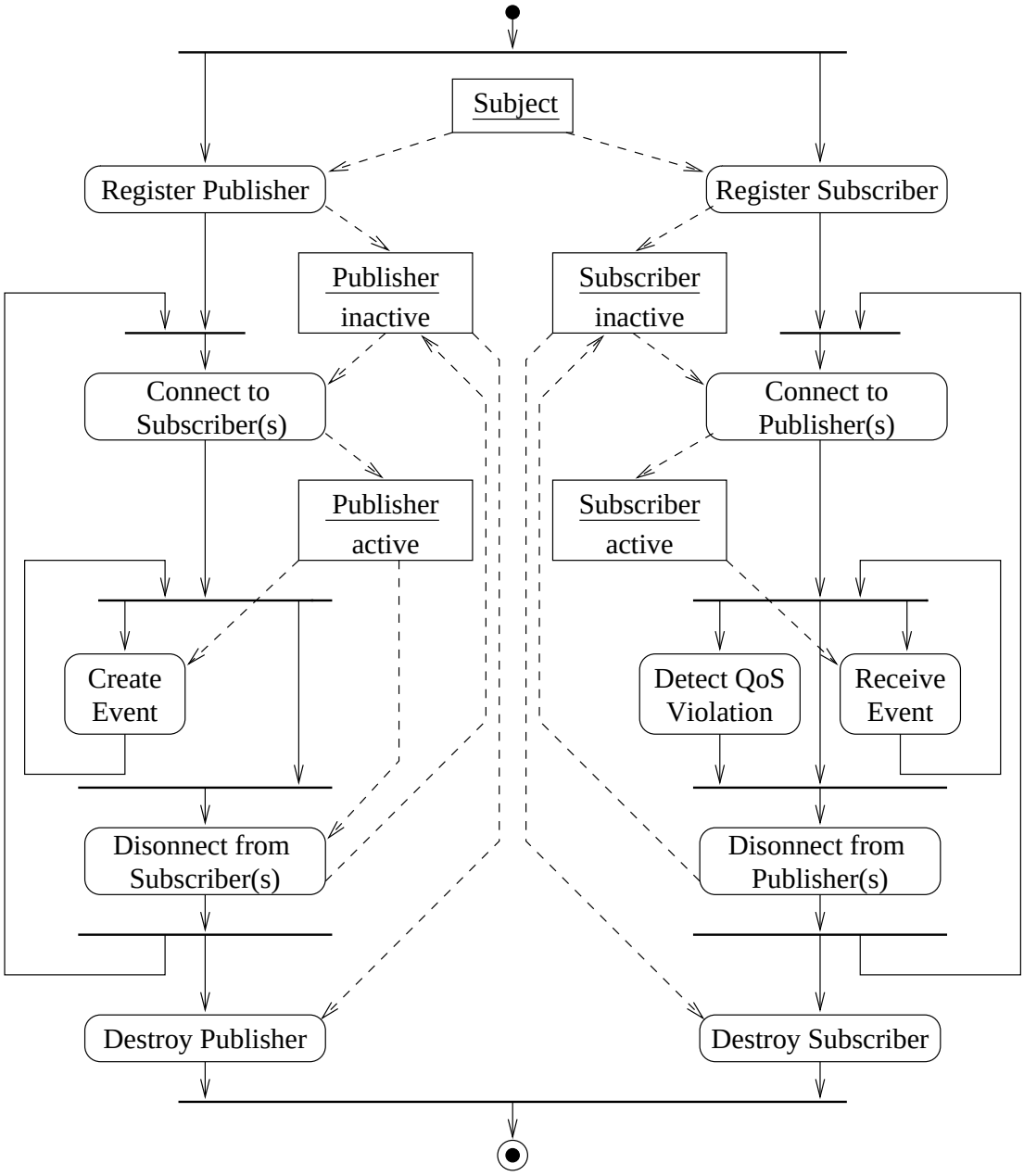

Figure 5.2.: P/S communication end-pont lifecycle

The registrations of a publisher or a subscriber requires the specification of a `Subject` object. This object encapsulate the subject description and the QoS specification. We have to distinguish between an *internal* subject and an *external* subject. An external subject is used by applications to describe its interest or the events it creates. An external subject can have multiple or even no QoS specifications. This way a subscriber can specify different QoS sets that would be sufficient. This feature only makes sense if used together with different application QoS requirements. As an example, it makes virtually no sense to accept the same event with different maximum delays

because the larger value would supersede the smaller one. But it makes a lot of sense if we can select events containing e.g. audio data encoded in different ways. Than each maximum delay would relate to the coding scheme used. An internal subject, as the name implies, is only used internally to match and route events. In contrast to the external subjects an internal subject contains only a single QoS specification. So, an internal subject can be seen as an instantiation of the template provided by an external subject. The external subject additionally provides a conversion between the external and internal representation of a subject. For the user or an application developer it is very convenient to be able to use human-readable subject descriptions, e.g. in XML format. But for the system it is not. It is obvious that matching between strings takes a lot more time than matching between e.g. integer numbers. A P/S system like XNet [CF04b] that uses XML throughout the whole system including the routing [CF03], event filtering [CFGR02], and subscription management [CF04a] introduces high resource demands for the run-time system, especially if we consider to use embedded devices with very restricted resources. But even if we provide sufficient resources we have the problem that the processing of an arbitrary XML document takes an unpredictable time – something that is not real-time friendly. Additionally, this processing time adds to the end-to-end delay for the delivery of an event which is very undesirable. To avoid such processing induced delays we will use human-readable subject descriptions only externally. Internally we will map them to simple numeric identifiers. This can easily be done by applying a hash function to the subject description and a binary storage of all QoS parameters. As explained before, an internal subject includes only one set of QoS parameters.

To supply events with different application QoS properties (e.g. encodings) we have to solve another problem. We need a way to distinguish them on both sides. The reason is that a client probably will needed different processing functions for events with different application QoS properties. Furthermore we need a way to avoid the creation of unused events. If there is no subscriber for a subject, it makes little sense to create the event in the first place – from an efficiency point of view because the P/S system should actually hide details such as if there are subscribers to a publisher. As a tradeoff we will hide the actual list of subscribers but disclose the used QoS profiles to the publisher. For publishers that only provide events with one set of application QoS properties this will retain the original publish/subscribe scheme. If it uses multiple profiles it assigns them to individual event flows that are multiplexed at the receiver side. The received event is augmented with the event flow identifier so that the subscriber can distinguish them. Subscribers that are not interested in different application QoS properties can safely ignore this extra information.

5.3.2. P/S Management Protocol

The P/S management protocol is responsible to interconnect the publishers and subscribers registered by the applications. To do so it has to perform the following tasks:

- Search for matching publishers and subscribers
- Allocate routes between publishers and subscribers to deliver events with QoS

- Monitor route usage for broken links
- Monitor end-points to determine dead publishers or subscribers

As explained before, we need to find and allocated routes from all publishers to all matching subscribers to be able to give QoS guarantees. But before we can start to create any routes we first have to find the matching publishers or subscribers. As a result we get a list of matching peers. The next step is to find a route from each publisher to each matching subscriber, using a multicast mechanism if available. When we have established these routes we notify the publishers so that they can start to transmit events (`NewSubscriber` signal). In the following time we have to monitor the established links for activity and QoS violations. When we detect such a QoS violation we need to notify the peers on both ends (`QoSViolation` message and signal). Additionally we have to try a local error correction, that includes a local route repair or alternative packet forwarding schemes. If this local repair is not successful the publisher has to reestablish the complete route. If a publisher quits (or dies) we have to release all routes originating at it. Similar, if a subscriber quits we have to release all routes to it that are only used to deliver events to it.

The search for matching peers is a distributed process that runs without any real-time constrains. That means that it can take an arbitrary time to finish a search. The concrete implementation of the search process is explained in section 5.4. The search process runs asynchronously. That means we do not start a search process and wait until it is finish, instead we initiate the distributed search process and react on each returning result. There are several reasons for this decision. First, because it is very difficult to define what “finished” means for a distributed search. From a global point of view a search would be complete and finished if the initiator received information about all matches. From a nodes point of view it is impossible to decide if the search is complete if we have no a priori knowledge about the number of nodes in the network. So we could define a timeout and declare the search finished at this point. Then we had the problem to define the timeout. If we define it too short we would miss some results. If it is too long we slow down the following protocol steps. Additionally we do not know how long it will take to search through the whole network. Another reason to work asynchronously is that we can utilize this behavior for announcements. If a new publisher enters the system it can announce itself to all subscribers. If a subscriber acts as soon as it receives an answer for its search, a new publisher can just send out an answer without being searched. Clients not interested can drop this answer, all other have an active search and will react by connecting to the new publisher.

For every result that we receive we try to build a route from the subscriber to the publisher that provides the requested QoS properties. This is subject to the P/S transport protocol. If the route is established successfully we start to monitor its activity and healthiness. A route is defined to be active if we regularly receive events that comply with the QoS specification of the publisher. This specification includes a value for the event period. So, if we do not receive an event within twice this time we are sure that there are missing events which indicates a QoS violation. There can be two reasons for this. First, the publisher has stopped to create events without informing the subscribers about this. Or second, we lost an event on the route to the node, either because of

a single transmission error or a link break. Each QoS violation is reported to the locally running applications and the transport protocol. The transport protocol will then try to correct the error. If it is not able to correct the error it will report back to the management protocol. This will than notify the application and start to reinitiate the connection, i.e. performs the same steps it would for a new publisher.

The second part of the monitoring controls the activity of the application. A publisher that does not create events as specified in its QoS specification can be considered dead. That means, if we do not get an event from an application registered as a publisher on the local node within twice the time of the specified event interval we can be sure that the publisher missed to create an event. If we miss a specified number of consecutive events we assume that the application has finished without performing a de-registration. In this case we notify the connected subscribers and intermediate nodes that the publisher has finished.

5.3.3. P/S Transport Protocol

The P/S transport protocol is the most important part of the system because it has to ensure the compliance with QoS guarantees. Only if the lowest layer is able to provide QoS guarantees we can gives these guarantees to higher layers and the applications on top.

The main task of the transport protocol is to provide the higher layers with a kind of multicast tree that is able to deliver events from one publisher to all its connected subscribers. This task can be divided into the following subtasks:

- Manage medium access and bandwidth allocation on the physical medium,
- enforce bandwidth limits on all nodes,
- multiplex QoS and best-effort traffic,
- establish uni-directional routes between two given nodes that provide a certain QoS,
- build a multicast tree out of the point-to-point links that provides end-to-end QoS properties,
- provide multicast packet routing,
- monitor routes for activity and QoS compliance,
- monitor neighborhood for local route repairs / optimization,
- monitor neighborhood for link quality,
- provide network broadcast optimized for reliability,
- perform node / low level service discovery on the network.

The wireless medium is a shared medium and we have to control the access to it. Only this way we are able to give any QoS guarantees. All stations have to run the same protocol stack and to behave cooperatively. So the first step is to control the medium access and to implement a coordination scheme among stations that share the medium in the same region. We will solve this problem by a combination of neighborhood discovery, admission control, and locally prioritized traffic shaping (s. section 5.5 for details). Upon this bandwidth management scheme we will build multi-hop point-to-point links with defined QoS properties and arrange them in a tree structure (s. section 5.8 for details). This tree provides us with a multicast feature that is used to deliver events from one publisher to all its connected subscribers. Each such tree will carry one flow of events. After the tree has been constructed we have to monitor and maintain it. Due to changes in the environment (e.g. moving, leaving, joining nodes) we will experience a different connectivity between nodes over time. This can require us to adapt the tree to retain the end-to-end connectivity of the tree. In order to detect environmental changes we monitor the received packets for new or vanished neighbors, changes in the quality of links to neighbors, and the bandwidth utilization. At this level we do not monitor the compliance with QoS requirements, this is done on the management level. Instead we focus on the tree and the forwarding of the data packets. If the link quality to a neighbor that is part of a route across the node degrades or if the neighbor vanishes at all, we start a local route repair and notify the management layer about the problem.

Another important service that the transport protocol provides is a network broadcast service. This is a flooding primitive that is primarily used to discover routes to other nodes or low level network services running on remote nodes. There are several ways to implement such a broadcast primitive. But because we use it to discover reliable routes, we will use a protocol that considers link qualities in the network (s. section 5.4.1 for details).

5.4. Searching for Peers

The problem to find a matching peer (publisher or subscriber) is the same like any other information or service discovery in a mobile environment. Such mobile environments have to be divided into systems with a wired backbone, e.g. a classical office environment with mobile laptops and PDAs, or a mobile wireless telecommunication system, and systems without such a backbone like a MANET. In systems with an almost static backbone we mostly find centralized or partially distributed systems. Such systems comprise of one or more central search servers that are known to all clients and that perform a coordination and synchronization among each other. A well-known example is the *Domain Name System* (DNS). A mobile user that connects to an access point, wireless hotspot, or something similar only needs to know the IP address of a reachable DNS server to be able to lookup the IP address of any host based on its name.

Systems with stationary, centralized event servers like the Toronto Publish/Subscribe System (ToPSS) [ALJ02] or LeSubscribe [PLF+00, PFLS00] that use a single server on a wired backbone do not need for any search facility. They just need a way to inform new peers of the central servers. All nodes register at this server that in turn handles the event matching and distribution.

Systems that use a partially distributed (i.e. on some reliable, static hosts) event service have to cope with the search problem. Examples for such systems are Siena [CRW01], READY [GKP99], Elvin4 [SAB+00], and JEDI [CNF01]. All systems use different implementations but they follow a similar scheme. The nodes that run the distributed event service (also denoted as *dispatching server*) build an overlay network to communicate with each other. Within this overlay network each node acts like a super-peer that is a subscriber and publisher at the same time. This super-peer works as a proxy for all connected publishers and subscribers. If possible it aggregates the subscriptions from the connected subscribers to reduce the overall number of subscriptions that have to be handled within the overlay network. Each publisher forwards all its events to the connected event server that republishes them within the overlay. At the opposite side another event server receives the events and forwards them to the subscribers behind it. In our system we will have a fully distributed event service, that means all nodes are equal and each one participates in the event forwarding.

In a MANET we need a fully distributed event service because we cannot rely on the existence and reachability of some central nodes to perform the event forwarding. To find matching peers we have to compare the query with the information on every node. Usually that means that we search for publishers of a given topic. We can generalize this problem as follows. We have a query $\mathcal{Q}$ and a predicate $\mathcal{P}_n$ (with n being a unique node number) that defines whether a node matches a search or not.

$$\mathcal{P}_n(\mathcal{Q}) = \begin{cases} 1 & \text{if } \mathcal{Q} \text{ matches;} \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

If we perform a search we expect an answer $\mathcal{A} = \{n_1, n_2, \ldots n_k\}$ with $\mathcal{P}_{n_i} = 1 \forall 1 \leq i \leq k$. We start the search with a request ($\mathcal{Q}$) that we send into the network. Every node that matches the query answers with a reply message that contains a partial answer $\mathcal{A}_j \subset \mathcal{A}$. If we do not have any caches each answer will contain only a single element. Together all partial answers should result in the complete answer ($\mathcal{A} = \bigcup \mathcal{A}_j$).

To fulfill this goal we have to ensure three points. First, we have to make sure that the query reaches every node. Or if we have caches, the query has to reach enough nodes to give an answer for every node. Second, we must ensure that the result reaches the node that started the search. And last, we have to strive for a best possible freshness of the results. We always have the problem that an answer that we receive gives information about the past. The older this information is, the higher is the chance that the target node does not longer match (a *false positive*) or more seriously, we do not get an answer for a node that contains a match. Because we only get positive answers we will not get *false negatives*, i.e. the out-dated information that a node does not contain a match.

In a P/S system we have another difficulty. We do not have a single search, instead we have a continuous search process. That means once we start to search for matching peers we would also like to know changes to this search result in the future. This brings us back to the problem to build a global view of the state of all nodes. We will solve the problem in the following way. We

will investigate how to realize a reliable broadcast protocol. This broadcast protocol will be used to send messages to every reachable node in the network, e.g. search queries or status updates. Additionally we will use this protocol to discover routes for the answers to the search requests. The details of the protocol will be discussed in the following.

We have to consider the following cases that change the global state:

1. A subscriber joins the network,
2. a subscriber leaves the network,
3. a publisher joins the network,
4. a publisher leaves the network,
5. the network splits into pieces,
6. two or more networks rejoin.

Although we will strive to prevent message loss in the first place we have to consider it. That means we have to cope with the problem of incomplete results.

5.4.1. Efficient, Reliable Flooding

An essential element for the search protocol is the broadcast protocol (also denoted as flooding) to send messages to all reachable nodes in the network. This broadcast protocol will later be used not only for the search for publishers and subscribers but for all search tasks.

Broadcasting has a primary goal – to deliver a message to every node in the network. But it is also a very expensive operation (in terms of bandwidth consumption) and so the secondary goal is to keep the required number of messages respectively the consumed bandwidth as low as possible. As we intend to use our broadcasting protocol not only for information dissemination, i.e. a one-way communication, but instead expect an answer, we have to make sure that we discover a reliable route at the same time. This route then will be used to deliver the answer back to the origin. We will exploit this restriction also to limit the bandwidth utilization, i.e. we will not waste messages to reach nodes that very likely are not able to answer. This last requirement need further explanation as it is really uncommon. There are several different broadcasting protocols and all seam to care about the bandwidth problem. Often broadcast protocols are used to discover a target and a route to it. A common approach is to record the route (sequence of nodes passed on the way) to the destination and use it in reverse direction to route the answer. But it seams that all protocols that use this approach do not care about the problem of the reliability of the way back. The reliability problem will be discussed in the following.

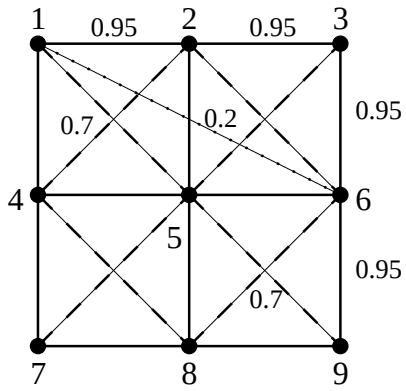

Figure 5.3.: Link quality in a small scale grid topology

To illustrate the problem we have to look back to the results from section 4.3.3 about the quality of links. Figure 5.3 is an extract from the link quality measurements depicted in figure 4.15. The picture shows nine nodes from the grid with the links of different qualities between them – the link qualities are denoted as numerical values. If we now use broadcasting to discover a route from node 1 to node 9 we have several possibilities. We will consider only three of them:

1. $1 \rightarrow 5 \rightarrow 9$
2. $1 \rightarrow 6 \rightarrow 9$
3. $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9$

At this point we do not discuss how we came to these routes, just what they mean for us. If node 9 tries to answer on the discovered routes we get the following probabilities that the answer reaches node 1:

1. $p_{1\rightarrow5\rightarrow9} = 0.95 \cdot 0.20 = 0.19$
2. $p_{1\rightarrow6\rightarrow9} = 0.70 \cdot 0.70 = 0.49$
3. $p_{1\rightarrow2\rightarrow3\rightarrow6\rightarrow9} = 0.95^4 \approx 0.81$

We see that for the short routes it is more likely that we do not get the answer back to the origin which is an undesirable situation. In section 4.3.3 we already saw that retransmission can have a significant effect on the receive probability of a packet. Now, if we consider retransmissions (a maximum of 4) on the way back, we get the following receive probabilities:

1. $p_{1\rightarrow5\rightarrow9} = (1 - (1 - 0.95)^4) \cdot (1 - (1 - 0.20)^4) \approx 0.59$
2. $p_{1\rightarrow6\rightarrow9} = (1 - (1 - 0.70)^4)^2 \approx 0.98$

3. $p_{1\to2\to3\to6\to9} = (1-(1-0.95)^4)^4 \approx 1$

From these results we learn several things. First, it can be favorably to use a longer route to increase the overall (end-to-end) reliability. Second, retransmissions significantly increase the reliability so that we can accept weaker links (with lower quality) on the route. Third, the overall reliability with retransmissions is much better than the best result without. And last, even with retransmissions we should avoid links with a very low quality.

Now we have to investigate the different strategies used to implement broadcasting / flooding. There are basically three strategies to solve the problem: pure flooding, virtual overlay networks [OBP05a], and ring search [CH03]. The first two are intended to reach every node in the network. The ring search only tries to locate a target with minimal effort.

Pure flooding

Pure flooding works as follows. The starting nodes sends a packet via WLAN broadcast. Each node that receives such a broadcast rebroadcasts it. If we perform a search and the node is a match, it additionally sends back an answer. To limit the number of messages each node performs a duplicate suppression. Depending on the task there are two techniques for duplicate suppression. First, if we only need to reach each node once, the node stores a backlog of flooding packets identified by the originator address and a sequence number and matches incoming packets against this list. The other alternative is to record the path the packet takes through the network and to check whether the packet already contains passed the node. This method discovers all possible path in a network. In a wired network it works quite well but in a MANET where we have much more interconnections between nodes, it can cause an enormous number of messages. The first method performs better because each node will only send the packet once. The overall number of transmissions grows linear with the network size. There are a number of other optimizations that try to limit the overall number of messages further [OBP05a, CH03, SCS02, YGK03, NTCS99] using some heuristics (*probabilistic flooding*). The individual approaches will not be discusses but they share a common idea. All assume that using physical broadcast transmissions to forward flood packets results in overlapping coverage areas (s. figure 5.4). Obviously all these optimizations assume a circular coverage area.

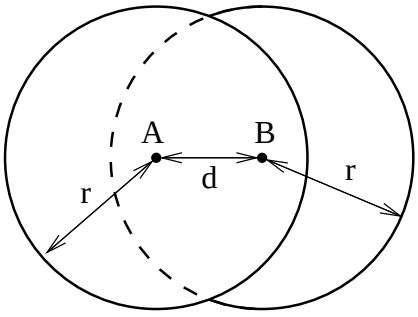

Figure 5.4.: Overlapping coverage area (from [NTCS99] figure 2)

Under the assumption that the coverage area is circular the additional coverage area (A_d) can be calculated as the coverage area of the first node (A_A) minus the intersection area (A_{A-B}):

$$A_d(d) = \pi r^2 - 4 \int_{d/2}^{r} \sqrt{r^2 - x^2} dx \tag{5.2}$$

According to [NTCS99] we get the maximum additional coverage area for $d = r$ as $A_d \approx 0.61\pi r^2$. That means the second node increases the overall coverage area only by 0..61%. Furthermore, for a node randomly placed inside the coverage area of A we get an average A_d of:

$$\overline{A_d} = \int_{0}^{r} \frac{2\pi x \cdot (\pi r^2 - A_d(x))}{\pi r^2} dx \approx 0.41\pi r^2 \tag{5.3}$$

The authors further explain that $\overline{A_d}$ decreases if we consider more nodes in the network. This is obvious as the coverage area of a third node C within transmission range of node B will probably overlap with A. By simulation they claim that the expected additional coverage area falls below 5% for node that are within range of 4 or more other nodes. Based on this observation several optimizations have been proposed. For instance to forward a broadcast packet only if we have not received it by more then 4 neighbors, to forward it only with a certain probability, to forward it based on location information, or even based on the estimated (from the signal strength) distance between nodes. The problem with all these optimizations is that they base on unrealistic assumptions (already discussed in section 4.3.3). Even if we consider an open terrain without major obstacles where we have an equal signal propagation into all directions, we do not get the assumed circular coverage area. We get circular regions with an equal receive probability. One could argue that this is not a problem, but it is. The reason is the implicit assumption that defines the redundancy of the coverage areas. All optimizations base on the assumption that the coverage area has a discrete border and that if one node within this region receives a packet all other do as well. But this would only happen if we have a 100% receive probability within the coverage area – which is not the case as explained before.

Now, if we consider obstacles within our communication environment, things get more complicated. Obviously we do not get circular coverage areas in this case. Even if we neglect the signal fading for a moment we can find topologies where they are contra-productive, i.e. are the cause for a serious fault that would not happen without it. Just as an example we can consider the topology in figure 5.5. We start a flooding at node 1 and apply a counter-based optimization with a forward threshold of 3, i.e. if a node receives 3 packets within a certain period of time it will not continue to forward because the expected additional coverage area would be too small. On the left we have seven nodes depicted with there assumed coverage area and connectivity. On the right we inserted a wall that avoids a communication between node 7 and 3, 4, 5. Without the wall the forwarding scheme would produce the expected result. But if we consider the wall and node 5 receives packets from 2,3, and 4 before it decides to forward the packet, the forwarding will stop at this point and 6 and 7 will not get the information.

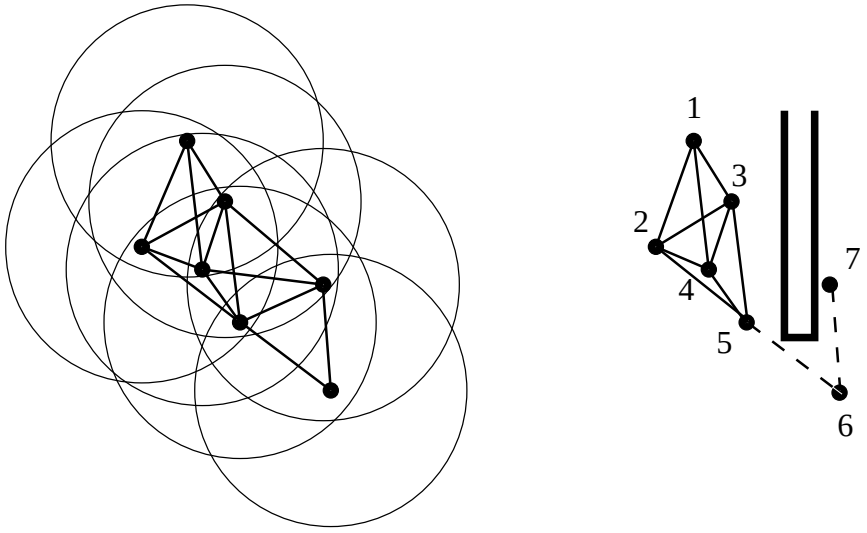

Figure 5.5.: Indoor topology

We see that there are several problems related to the simplified assumptions made in the development of the proposed optimizations. With a special focus on a reliable behavior of the protocol these optimizations are contra-productive. Additionally there is no indication that any of the proposed optimizations has been tested in a real-environment – another reason better not to rely on their effectiveness.

Overlay networks

Another class of broadcasting schemes use virtual overlay networks (VON) to control the flow of packets. Overlay networks are a common technique to abstract from the physical topology the create an application friendly topology, e.g. to route information more efficient. In a MANET it is not desirable to abstract from the physical topology because it is a scarce resource that should not be wasted. In a MANET we need a overlay network that respects the physical topology. Oliveira et al. propose an overlay network for resource discovery in MANETs [OBP05c, OBP05b]. This overlay network is based on dynamically determined clusters. That means a group of nodes within broadcast range elect a head node (broadcast group leader, BGL) that coordinates the forwarding within the broadcast group. This cluster structure is adjusted according to changing connectivity conditions. The authors present two algorithms. The first, considering a 1.5-hop environment, is an improved version of the algorithms presented by Choi et al. [CP02] and Peng et al. [PL00]. The second, considering a 2.5-hop environment is an improvement of the algorithm presented by We et al. [WL03].

We will not discuss the protocol details, instead we will focus on the most noticeable properties of the protocols and their contribution for this work. The clustering scheme is based on a beaconing protocol. Every node periodically broadcasts a beacon frame containing one or more of the following information: only local information (1-hop), the BGL (1.5-hop), a list of neighbor

nodes (2-hop), or the neighbor nodes with their BGLs (2.5-hop). The beacons are used to perform the BGL election (which will not be detailed further) and a link quality measurement at the same time. This is noticeable as most protocols completely ignore the link quality and stability. The link stability is defined as the number of consecutive received beacons. If one is lost, it is set to zero. The search algorithm now works as follows: BGLs rebroadcast a packet as soon as the receive it – the packets are marked by the BGLs they have visited. Non-BGL nodes back-off for a constant plus a random time while they listen for packets from other nodes. They monitor packets for visited BGLs. If the timer goes off and they did not received packets containing all adjacent BGLs (and the neighbor's BGLs in case of the 2.5-hop version) they rebroadcast the packet to cover the missing BGLs otherwise it is dropped because it is redundant. The authors show that the 1.5-hop version performs better in cases of higher mobility because this mobility cases a lot of outdated information about the 2.5-hop neighborhood. It should be emphasized that this protocol is an advancement over most other protocols because it only relies on the detected connectivity and does not assumes anything like a circular coverage area. But it still has a major problem, it assumes that packet loss is only caused by collisions. In an environment where we have links with gradual link qualities, the stability metric will deliver a lot of links with a very short lifetime. A second consequence of this assumption is that the protocol will rebroadcast packets unnecessarily if a non-BGL node just misses a packet from an adjacent BGL. This will decrease the efficiency of the protocol. There is a simple reason why this problem did not occur during the evaluation of the protocol – the evaluation has been done in a simulation using the two-ray-ground reflection model that is unable to produce gradual link qualities.

Ring Search

Ring search is intended to find a fixed (a priori known) number of targets. The algorithm works quite simple. A searching node starts a pure flooding like broadcast with a very limited maximum hop count (usually 2). If the target is not within this region it starts a second search with an increased hop count and so on. There are a number of variants that increase the maximum hop count in different ways or try to utilize already gained knowledge for the subsequence searches. One example is to get a feedback from the nodes where the search reaches the maximum hop count. The searching node then instructs those nodes by a unicast packet to continue the search without flooding the already searched area. As we do not have an a priori known number of targets, we can not utilize this technique and will not discuss it in detail.

Link Quality Overlay Flooding

In the following a new flooding scheme will be explained that combines the overlay principle with a link quality measurement. It does not uses a clustering scheme or other methods to minimize the protocol overhead in order to improve reliability. As we intend to use the protocol for target discovery and not only for information dissemination we have an important constraint: the reliability property is limited to those nodes where we have a high probability for a successful answer. With other words, we do not try to reach nodes that are very likely not able to answer

using unicast communication. The need for such a link quality constraint has been impressively demonstrated by the OLSR experiment discussed in section 3.1.

It should be emphasized that the presented protocol does not make assumptions such as a circular coverage area and considers gradual link qualities. It only relies on a measured connectivity and link quality between nodes. The measured link quality is a byproduct of the beacon-based neighborhood discovery and bandwidth coordination protocol that will presented in section 5.6. At this point we only need to know that we have a link quality value $q_{i \to j}$ ($0 \leq q_{i \to j} \leq 1$) that specifies the packet loss probability for a broadcast packet from node i to node j. Additionally we have a forward threshold t_{fdw} ($0 \leq t_{fdw} \leq 1$) that specifies what we consider as a high quality link. The value of t_{fwd} depends on the requirements for the loss probability of a unicast transmission (including a given maximum number of retransmissions). If not otherwise specified (e.g. by a QoS requirement) it has a value of 0.8. If we consider a maximum of four retransmissions on the way back, a 80% link quality means a 99.8% success probability of the answer packet on the link.

It should be noted that we did not discussed the effect of the packet size on its receive probability. The reason is that the packets used by the beaconing service carry a comparable amount of payload as the packets used for searches. So we already measure the probability of packets of a size we considering at this point. Additionally we have to mention that the answer packets will be send as unicast transmissions, i.e. are subject to the RTS/CTS mechanism to minimize the risk of collisions. The RTS/CTS packets itself are always smaller than the beacon packets and so have a smaller loss probability.

The flooding protocol now works quit simple. The initiating node creates a packet and than acts like any other node that receives such a packet. A flood packet is identified by the station ID of the originator and a unique sequence number assigned by the originator. The packet additionally includes a list of nodes (station IDs) that it passed already (path history) and a time-to-live (TTL) value. Every node that receives a flood packet first decreases the TTL value and checks whether it has already seen the packet. This is done using a FIFO queue that stores a fixed number of recently seen flood packets. If the packet is new and the TTL value is greater than zero, the node will store it in the FIFO queue and forward it. The forwarding is done in the following way: the node looks at the list of neighbors and the measured link quality. If the link quality is above the forwarding threshold (t_{fwd}) and the target node is not already in the path history of the packet, it will forward the packet using unicast transmission.

It may surprise to use unicast communication to implement flooding in a MANET that bases on a broadcast medium. But there are two major reasons for this decision. First, on standard 802.11 hardware, broadcast transmissions were done using the basic rate of 2 Mbps whereas unicast communication is done at higher bit rates (depending on the link quality) of up to 11 Mbps (802.11b) or 54 Mbps (802.11a/g). This means that a single unicast packet takes less time to transmit than a broadcast packet of the same size. To increase the reliability of the forwarding we have to transmit a broadcast packet several times. Unicast packets are subject to the automatic retransmissions performed by the WLAN device. Overall, if we have a small number of target nodes (≤ 3) and do not utilize the maximum number of automatic retransmissions we hereby

reduce the medium utilization. The second reason to choose unicast transmissions is that we intend to use the flooding mechanism for target discovery und thus expect an answer that will be send by unicast transmissions. So it does not make much sense to discover a path to the target that is not appropriate for unicast transmissions. But if we successfully flood packets using unicast transmissions we have a high probability that the link is appropriate to deliver an answer back. At least we are sure that it is not unidirectional because we would not choose it if we were not able to at least receive beacons from the other side.

Modifications

There are two modifications to the basic flooding protocol that shall be presented in the following: flooding with high speed broadcast transmissions, and flooding with end-to-end constraints.

One reason to choose unicast transmissions to forward packets is that broadcast transmissions are done with a smaller bit rate. If we have a network where we can ensure that all participating nodes are able to transmit broadcast packets at a higher bit rate, we cannot longer benefit from unicast transmissions. In this case the forwarding will be modified in the following way: a node will store the list of neighbors that should receive the packet (have a link quality above the threshold) inside the packet. Each node that receives a flood packet then additionally checks if the packet is intended for itself. To increase the reliability of the transmission the flood packet is transmitted several times (maximum number of unicast transmissions).

If the flooding protocol is used to discover paths with special properties (e.g. end-to-end reliability) we need another modification. In this case it can be possible that a node receives a duplicate flood packet later that used an alternative path with better properties. In the basic version of the protocol it would be dropped because the node has already seen it. So we modify the protocol in the following way: if a node receives a packet, it processes the packet if it has not seen it before or if the new packet used a path that provides better properties than the one of the packet stored in the FIFO history. In this case the node will forward the packet as usual, remove the old packet from the history, and insert the new one. It should be noted that using this modification can significantly increase the overall number of transmitted messages as we do not longer limit the number of forwards per node to one.

5.4.2. Search and Update Protocol

In this section we will discuss the protocol used to search matching peers and to update their relationships. It is based on the link quality overlay flooding presented in the previous section. In the parent section we already listed the six cases that we have to consider in the following.

A subscriber joins

If a new subscriber joins it needs to know the list of matching publishers in order to receive the events they create. In order to discover active publishers it initiates a broadcast search

(`SearchPublisher` message) including the machine-readable subject, and optional application QoS requirements. Each node that has a matching publisher replies with an answer (`FoundPublisher`) that includes its concrete application QoS properties so that the subscriber can adapt to them after the connection process (in case it is able to do so). The usage of application QoS requires that both sides use the same way to specify there QoS parameters. They can embed these as an opaque QoS property in the network QoS specification (s. section 6.4.4). If a publishers has multiple matching QoS profiles it will create multiple event flows. Therefore it has to include multiple entries in the `FoundPublisher` message. That means that it actually is treated as multiple publishers. The answer additionally includes a list of all active subscribers. All nodes that have already subscribed and connected to a matching publisher will reply with a full list of publishers (`FoundPublisher` message). The searching node will collect all answers and store them for some time to detect inconsistencies. Additionally the subscriber builds a list of all matching publishers and starts the connection process immediately.

The inconsistency detection needs further explanations. In an ideal case each answer from a subscriber should include the same set of publishers and this set should match the set of publishers that answered. Similarly each publisher should answer with the same set of connected subscribers that should match the set of subscribers that reply to our search. But due to lost discovery packets, broken routes, the inability to establish a route, or other reasons there can be inconsistencies in the returned results. To correct the detected inconsistencies the node sends a message (`FoundPublisher`) to each affected subscriber node to announce the missing publishers as new publishers. Each of the subscriber nodes will than try to connect to the missing publishers. This message also includes the set of known (from their answers) subscribers.

A subscriber leaves

If a subscriber leaves this can be because of three reasons:

1. The application deregisters the subscriber,
2. the application crashes,
3. the node running the application crashes or shuts done unexpectedly.

The first case is the easiest one. If an application deregisters or destroys the subscriber we send a `StopSubscriber` message backward the event distribution trees to the publishers. These remove the subscriber from their lists of active subscribers. Additionally the P/S transport layer on the intermediate nodes is notified to update the forward tables.

If the application crashes, this will be detected by the P/S transport layer (s. section 5.3.1). It will then perform the deregistration process itself and notify the publishers accordingly. The last case will be detected by the node that is one hop closer to the publisher using its forwarding table and neighborhood information (s. section 5.8 for detail). It will than issue the `StopSubscriber` message if a local repair fails.

A publisher joins

If a new publisher joins the network it will announce its presence by a `FoundPublisher` message that is flooded through the network. All nodes that have a matching subscriber will in turn try to connect to the new publisher.

A publisher leaves

If a publisher leaves this can have the same three reason as for a subscriber:

1. The application deregisters the publisher,
2. the application crashes,
3. the node running the application crashes or shuts done unexpectedly.

In the first case the publisher will send a `StopPublisher` message along the event distribution tree to inform all subscribers and intermediate nodes. The subscribers will thereupon delete the publisher from their lists of known publishers.

If the application crashes this will be recognized by the run-time system because of the mission events (a QoS violation). The run-time system will then send the `StopPublisher` message to the subscribers. If the node itself crashes this will be recognized by all nodes on the distribution tree in a 1-hop distance because of there neighborhood information and forwarding tables. They will then send the `StopPublisher` message along their branches of the tree. It should be noted that all other intermediate nodes as well as the subscriber nodes will recognize the missing publisher as a QoS violation but cannot detect that the publisher has gone because it is not in a direct neighborhood. They will assume a link break and start a local repair that will be stopped when the `StopPublisher` message is received.

The network splits / rejoins

As there is a network split we do not have a direct way to detect this. If the separated part of the network does not contain any active publishers or subscribers we will not even notice a difference because nothing is missing. Otherwise we will detect a series of broken connections. As the traffic monitoring will already detect and report link breaks and lost peers (s. section 5.8) we need a heuristic to distinguish dead nodes from split networks. We will assume a dead node if a neighbor (1-hop distance) that is on the multicast tree reports the missing node. Otherwise we will assume a link break in between.

If we assume a network split, publishers and subscribers will react in the following way. A publisher will remove the subscribers from its list of connected subscribers and waits to be re-discovered. A subscriber will actively search with increasing random intervals. To reduce the

overall number of messages a subscriber will include a list of connected publishers in its search request. Other subscribers that receive this search will defer their own search for a new random time. But they will perform the inconsistency check. If the search includes unknown publishers they will try to connect to them. If they are connected to publishers that are not included in the search they will report back to the originator. A publisher that receives the search request checks if it is on the included publisher list. If not it will immediately respond with an announcement. This way all other subscribers will get informed and then reconnected to the publisher.

5.5. Bandwidth Management in a MANET

Now that we are able to locate peers in the network, we have to discuss the problem of bandwidth management in a MANET. It has already been explained that the wireless medium is a shared medium and therefore we have to control the medium access and bandwidth usage in order to be able to ensure QoS properties. In this section we will start with a general discussion of the problem to answer the questions with whom we have to share the medium and how much bandwidth we can utilize. Afterwards we will present a beacon-based coordination scheme.

5.5.1. Capacity of a Wireless Network

As the first question we will discuss the capacity of a wireless network. With capacity we denote the number of information that can be delivered end-to-end. An important question related to the maximum capacity of a MANET is how such a network scales. We could also ask, if it really make sense to build large-scale ad hoc networks.

Gupta and Kumar [GK00] presented a very formal paper that analyzes this problem. They assumed to have a network consisting of n nodes each able to communicate at a maximum possible bandwidth of W bits per second. The paper further assumes that all nodes have the same circular coverage area and that we are communicating using a non-interference protocol. The central result of their work is that each node will only reach a throughput of $\Theta(\frac{W}{\sqrt{n}})$ for the communication with a randomly chosen destination even under optimal circumstances. If the nodes are randomly placed the throughput is bounded by $\Theta(\frac{W}{\sqrt{nlogn}})$. They further explain that it does not matter if we use a single channel or dived it into several sub-channels. There are basically two reasons for this result. The first reason is the shared medium. Each node has to share its bandwidth with all other nodes in its local neighborhood. Even if we consider optimal placed nodes and their coverage areas, we see that with a growing number of nodes the probability to overlap with another node grows. If we consider randomly placed nodes and keep the density of nodes constant, the overall coverage area (extension of the network) grows with the number of nodes. This obviously means that for larger networks and randomly chosen communication pairs the average path length increases. As a consequence we notice that the same amount of information transmitted between the two nodes results in an overall larger bandwidth consumption if the path length grows. From these results it seams very pointless to strive for large scale MANETs because their efficiency

will tend to zero. But the authors also point out that the main problem in this calculation are the randomly chosen communication pairs. If we can limit communication to short ranges we will benefit from the extended range of the network as we get independent areas of communication.

In [LBD+01] Li et al. investigated the problem further. They put a special focus on different traffic patterns and their influence on the overall capacity. They argue that the assumption of randomly chosen communication pairs is not necessarily valid in large scale MANETs. In a larger network it is more likely that a user will connect to a physically nearby node, e.g. students in a lecture working on the same task. Another example are redundant services. If we have a network that is intended to support a large user base we usually find redundant systems to ensure service levels and availability. The authors perform an extended discussion of the capacity of a chain of nodes. In a multi-hop wireless network a packet has to be transmitted over multiple hops that are at least pairwise in transmission range. Together with the influence of interference a single packet will consume bandwidth multiple times at each node. This effect shall be explained using figure 5.6. The picture depicts six nodes in a chain. The question now is, how these nodes effect each other. If node 1 transmits a packet, node 2 will not send because it senses to medium to be busy. If we assume a physical carrier sense range to be twice the transmission range, node 3 will be blocked by the physical carrier sense. Node 4 would sense the medium to be idle and could start to transmit. But this could cause a collision at node 2. So, the capacity on the path would reduce by a factor of 4.

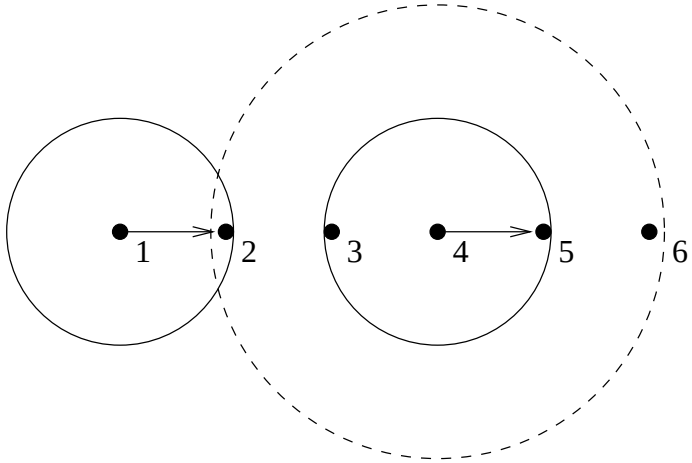

Figure 5.6.: MAC interference among a chain of nodes (from [LBD+01])

An important result of this work is that the authors verified the results of the chain capacity in a real world experiment. As a benchmark they used the direct communication (unicast with RTS/CTS) between two nodes and determined a throughput of 1.7 Mbps at a data rate of 2 Mbps. This value complies with the theoretical maximum. In the simulation as well as the real experiment they were able to achieve an end-to-end throughput of about 0.41 Mbps which is 97% of the expected maximum. But they also noticed two other important things. First, that the 802.11 MAC layer is able to achieve the optimal chain schedule but it is not able to discover

it on its own. That means, if node 1 would be a greedy sender the end-to-end throughput falls from $\frac{1}{4}$ to $\frac{1}{7}$ of the maximum. The second result, which is more a side node in the paper, is important with respect to the discussion about the modelling of a WLAN (s. section 4.7.4). The authors noticed that it was not that easy to setup the real experiment to mimic the behavior of the simulation because they experienced a gradual loss probability that was not present in the simulation. The reason was the same as in most other papers – the use of the two-ray-ground propagation model with all its problems (s. section 4.5). So they placed all physical nodes in a way that they had a negligible loss probability on the links to their respective neighbors.

5.5.2. Bandwidth Coordination Area

The second question that we are going to investigate is, with whom we have to share the wireless medium. Additionally we have to discuss the problem with whom we can coordinate the bandwidth through direct communication.

In section 4.3.1 we already discussed how one node can affect another one. And in the previous section we saw how nodes affect each other in a chain. If we now consider the bandwidth sharing in the area we have the same effects. For a unicast transmission we directly block the nodes that are in transmission range. Additionally we block nodes further away through the physical carrier sense. Beyond this range we still can affect other nodes by adding to their experienced noise level. For a unicast transmission we have to consider ACK packets and probably RTS/CTS packets. An RTS is transmitted by the node itself so it does not differ from a short broadcast packet. But the ACK and CTS packets are send from the destination and so probably affect nodes one hop further away. This means that if we plan to forward traffic to another node it is not sufficient to reserve some bandwidth in the own neighborhood. We additionally need to reserve some bandwidth on the next node and in its neighborhood.

We still have to answer the question how far the affected area reaches. In section 4.3.2 we calculated the transmission-to-interference range ratio. The higher this ratio is, the shorter is the interference range compared to the transmission range. We can calculate a minimum TIRR value if we assume a flat environment without significant obstacles by using the equations 4.19 and 4.20. We calculated $TIRR_{TRG} \approx 0.562 > TIRR_{Shadow} \approx 0.527 > 0.5$. That means that the interference range is less than twice the transmission range (the maximum 2-hop range) in both cases. If we assume an indoor environment or more sensitive receivers (smaller P_{RX}/P_{CS} ratio) the affected area shrinks further (s. figure 4.9). As a result we see that we have to coordinate our bandwidth usage with all nodes within a 2-hop range, i.e. with all nodes that we can communicate with and all their respective neighbors. This 2-hop range will be larger than the affected area of the sender.

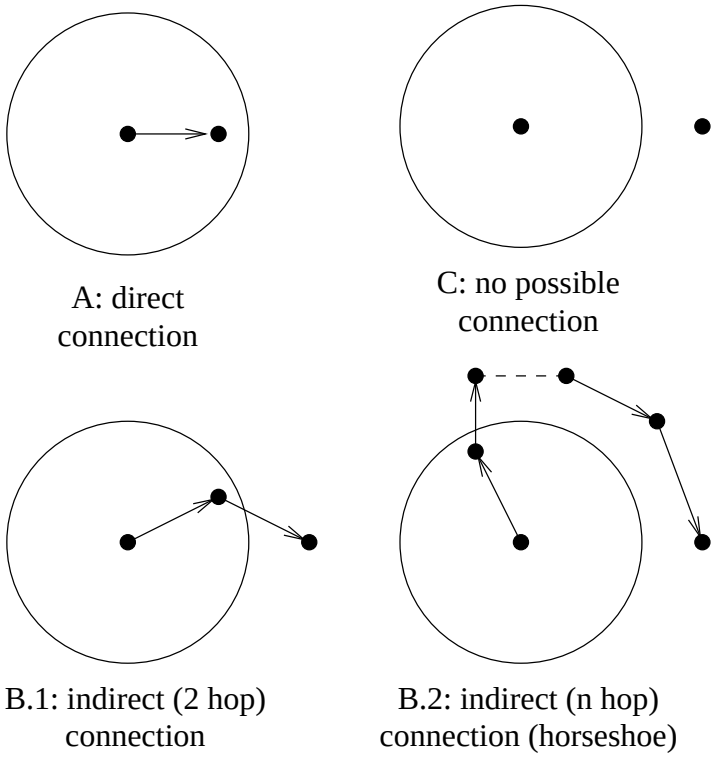

Figure 5.7.: Example topologies for bandwidth coordination

We know that we have to coordinate with all nodes within a 2-hop range. The remaining question is, how to realize it. A theoretical solutions will be discussed in section 5.5.3, and the actual solution developed in this work in section 5.5.4.

At this point we have to discuss a general problem. If we have to coordinate with a node in the affected area we can A) communicate with it directly, B) communicate with it using an arbitrary number of intermediate nodes, or C) cannot communicate with it at all (s. figure 5.7 for examples). Case A is without any problems. Case C is very critical because we do not have a chance for a communication and have to solve the problem in another way. Case B can be critical as well. If we have a limited number of intermediate nodes, especially exactly one (case B.1), we can still handle the problem. But if we do not know how much intermediate nodes are involved (case B.2), it is virtually the same problem as in case C. Additionally we have to keep in mind that it becomes significantly more difficult to coordinate nodes over a growing number of hops because of the increasing packet loss probability and end-to-end latency.

If we are not able to perform a direct coordination because we have no connection or it has a long distance (hop count) we can still try to detect it. As a result of the physical carrier sense we still have to share the available bandwidth with nodes outside of the transmission range. Simulation and real-world measurements about this effect have already been presented in section 4.4.3. If a node tries to utilize more bandwidth than it is available on the channel it can recognize this by a growing output queue. It than has to adapt by reducing what it regards as the maximum channel

capacity. If we had two nodes that transmit at full speed the available bandwidth for both would drop to about 50%.

When we remember the results in figure 4.11 in section 4.3.3 we see that the problem is only that acute if we assume a fixed transmission range. But as we have a variance in the transmission range and hence a gradual loss probability the effect is not that abrupt. Just as an estimation we look at figure 4.4.3 and the results of equation 4.21. In the figure we see the larges distance with almost 100% receive probability at about 75m. With a $TIRR_{Shadow} \approx 0.527$ we get a interference range of about 140m. But at this range we still have a receive probability of more than 50%. That means, if we assume a node that fully effects us by its carrier signal, we have a good chance to communicate with it. Nodes further away will have a smaller effect on us. We will introduce a second threshold (t_{beacon}) that defines with which neighbors we will perform a bandwidth coordination. This threshold has to be lower then the forward threshold (t_{fwd}), e.g. 0.5 for an indoor area and 0.3 for an outdoor area as we have a smaller TIIR and hence larger interference area there.

In practice, if two active nodes move towards each other they will first notice a small degradation of the channel capacity. This degradation will continue at an increasing rate. But at the same time the probability for a communication with the other node increases at a growing speed. We will react to such environmental changes dynamically (s. section 5.5.4.3).

5.5.3. Complexity of the Bandwidth Coordination Problem

In this section we will discuss the problem of the complexity of the bandwidth coordination problem. With other words, how much effort we have to spend to perform an exact or best possible bandwidth coordination. Allard et al. [AGJM04] discussed the problem with respect to the existence of interference in a WLAN. They assume a system that uses admission control and resource reservation to provide QoS guarantees and model a wireless network as an undirected graph. The authors explain that it is very important to remember that it is necessary to reserve bandwidth not only on the sender node and in its coverage area but also at the receiver node and in its coverage area. They proved that the problem is NP-complete and hence cannot be solved exactly in an actual setup. The authors proposed three heuristics that assume that all nodes have global knowledge of the network connectivity. This is a very critical requirement in a dynamic ad hoc network. Furthermore, a look on the proposed algorithms shows that the authors have another implicit assumption – that every node in an n-hop environment of a node is reachable via n intermediate nodes. This assumption seams very common for all graph-based algorithms. But as we have seen in the previous section, we can create topologies where a communication between nodes that need to perform a bandwidth coordination is impossible. Two nodes within interference range that cannot communicate with each other are a perfect counter-example for all graph-based algorithm. But nevertheless the paper provides some valuable results. First, that even if we can ensure a way to communicate with all nodes in the interference range, the problem of an exact solution is NP-complete. Second, a bandwidth planning is possible using the local knowledge about the used and remaining capacity of a node. To cope with the horseshoe and

similar problems we have to extend the idea to include a dynamic node capacity measured online (s. section 5.5.4.3). Finally, the simulation results (gained in a very simple simulation model) show that routes that support QoS are usually longer than best-effort routes. This supports the investigations in section 5.4.1.

5.5.4. Beacon-based Bandwidth Coordination

In section 5.5.2 we saw that it is necessary to perform a 2-hop bandwidth coordination that also dynamically adapts to a changing channel capacity. In section 5.5.3 we discussed that it is practical impossible to perform an exact coordination and hence we have to use a heuristics. In this section we will present a heuristics that uses a beaconing scheme to perform a neighborhood discovery and bandwidth coordination at the same time.

The basis for the coordination protocol is a periodic beacon sender running on each node. Using this beaconing service nodes perform a distributed bandwidth allocation. This allocation is enforced on the medium using a local traffic shaping and prioritization in combination with the standard IEEE 802.11 MAC protocol (DCF).

5.5.4.1. Beaconing Service

The periodical beacon includes the following information:

- The identity of the sender,
- the beacon interval time,
- the airtime allocated by the sender,
- the estimated bandwidth utilization in the nodes neighborhood,
- a list of all known neighbors and their allocated airtimes, and estimated bandwidth utilization.

Airtime

We will not use account transmitted as consumed bandwidth as bytes per second, instead we account the channel utilization of a node as the percentage of airtime it consumes. There are several reasons to do so. Most important the fact that the same amount of data transmitted in several small packets requires more airtime compared with large packets because of the prepended headers. Furthermore the same amount of data requires different airtimes when send with different modulations (transmission rates). But one part of the header (the preamble) is always transmitted using the lowest transmission rate ($PLCPrate = 1\frac{Mbit}{s}$). The airtime required for a broadcast packet is calculated as:

$$
\begin{aligned}
AT_{BC}(P) &= \frac{payload(P) + headers(P)}{datarate(P)} + \frac{preamblesize}{PLCPrate} \\
&= \frac{payload(P) + headers(P)}{datarate(P)} + overhead_{BC} \qquad (5.4)
\end{aligned}
$$

If we use unicast communication from node i to j we have to calculate the required airtime for both nodes:

$$
\begin{aligned}
AT_i(P) &= \sum_{i=1}^{retry(P)} \left(\frac{payload(P) + headers(P) + RTSretry_{i,P} \cdot RTSsize}{datarate_{i,P}} \right) \\
&\quad + retry(P) \cdot \frac{preamblesize(1 + RTSretry_{i,P})}{PLCPrate} \qquad (5.5) \\
AT_j(P) &= \sum_{i=1}^{retry(P)} \left(\frac{ACKsize + RTSretry_{i,P} \cdot CTSsize}{datarate_{i,P}} \right) \\
&\quad + retry(P) \cdot \frac{preamblesize(1 + RTSretry_{i,P})}{PLCPrate} \qquad (5.6)
\end{aligned}
$$

Each unicast transmission can cause several automatic retransmissions of a packet ($retry(P)$). If we use the RTS/CTS mechanism we first need to transmit a RTS packet (with size $RTSsize$) and will try it several times ($RTStry_{i,P}$). The destination node will answer each RTS with a CTS packet (with size $CTSsize$) and the payload packet with an ACK packet (with size $ACKsize$).

As we do not know how many times an unicast packet will be transmitted we have to assume the worst case ($\overline{retry}$). Usually this value is set to 4. For the number of retransmissions of RTS packets we also have to assume the worst case ($\overline{RTSretry}$) which usually is set to 7.

In practice we will simplify the calculation by assuming an average number of retransmissions ($retry$) and RTS packets per try ($RTSretry$). Additionally we will assume that the transmission data rate ($datarate(P)$) will be constant for one packet, i.e. will not change for possible retransmissions. This simplifies the airtime calculation to a payload size dependent and an independent part.

$$
\begin{aligned}
AT_i(P) &= retry \cdot \frac{payload(P) + headers(P) + RTSretry \cdot RTSsize}{datarate(P)} \\
&\quad + retry \cdot \frac{preamblesize(1 + RTSretry)}{PLCPrate} \\
&= retry \left(\frac{payload(P) + headers(P)}{datarate(P)} + overhead_{send} \right) \qquad (5.7) \\
AT_j(P) &= retry \cdot \frac{ACKsize + RTSretry \cdot CTSsize}{datarate(P)} \\
&\quad + retry \cdot \frac{preamblesize(1 + RTSretry)}{PLCPrate} \\
&= retry \cdot overhead_{receive} \qquad (5.8)
\end{aligned}
$$

Neighborhood discovery

Each node periodically transmits a beacon frame to announce its existence, bandwidth utilization, and known neighbors including their bandwidth utilization. The period of the beacon can be chosen arbitrary. A common value is 1s, but for fast moving nodes it can be favorable to choose a shorter period so that neighbors notice topology changes faster. This approach resembles the neighborhood discovery used by the OLSR protocol as well as the adaptable beacon frequency found in Fast-OLSR (s. section 3.1).

Each node that receives a beacon from a neighbor node stores it in its own neighborhood table. This table contains for every neighbor the information transmitted with the beacon and the time-stamp of the last received beacon. The lifetime of such an entry is defined in a number of beacon intervals and so can have a different value for each neighbor. If the nodes does not receive a subsequent beacon within the defined lifetime, the entry will be deleted and the neighbor regarded as lost.

Neighborhood bandwidth utilization estimation

As explained before, we are not able to perform an exact calculation in all cases because of the interference problem and gradual link quality. We will perform an estimation. Furthermore it is only an estimation because a node can calculate the bandwidth utilization only with the data provided by the neighbors but this information arrives with a certain delay. This is especially true as we also need information for nodes that are in 2-hop distance.

From the neighborhood list of node i we get the following data:

- $\mathcal{N}_i$ the list of direct neighbors,
- q_j the link quality for the neighbor j (s. section 5.6),

- $\mathcal{N}_j$ the neighbor list of node j,
- a_j the allocated bandwidth of node j reported by itself,
- b_j the estimated bandwidth utilization in the neighborhood of node j reported by itself,
- $a_{k,j}$, the allocated bandwidth of node k reported by node j,
- $b_{k,j}$, the estimated bandwidth utilization of node k reported by node j,
- t_j, the time-stamp of the last update,

with $j \neq i$, $j \in \mathcal{N}_i$, $k \neq i$.

A node allocates bandwidth for QoS flows, the best-effort management traffic (beacons, search queries, ...), and to compensate for the measured difference between the theoretical and the actual channel capacity (s. section 5.5.4.3).

As we can see, the neighborhood list contains redundant information because multiple nodes can report the allocated bandwidth and bandwidth utilization for the same node independently. As the first step we compensate update inconsistencies in a conservative way. This means that we will use the maximum value reported for the allocated bandwidth of a node and the estimated bandwidth utilization.

In the next step we perform a local update of the reported bandwidth allocations, i.e. we make sure that all entries relating to the same node have the same value. If we update a node's allocation entry we have to update the corresponding bandwidth utilization estimation in the beacon entry too. This is simply done be increasing it by the difference of the old and the new value.

After this preparation we compute the bandwidth utilization value (b_i) as the sum of the updated bandwidth allocations of the node itself and of all known 1-hop neighbors with $q_j > t_{beacon}$ and corresponding 2-hop neighbors. This value will be stored and propagated in subsequent beacons.

Bandwidth allocation

If we need to allocate bandwidth on a node we first have to compute the required airtime a. We than have to check if we are able to allocate that amount of airtime. This is possible if there is enough airtime left in our own neighborhood and in the neighborhood of our 1-hop and 2-hop neighbors. We compute the available free airtime as the minimum of the remaining free airtime estimated on the local node and all its 1-hop and 2-hop neighbors. If this remaining airtime is less than a we can accept the allocation (s. figure 5.8).

The new allocation will be stored in a local allocation table and the modified airtime estimation propagated using the beaconing service. To speed up the propagation of the allocation we propagate the locally updated beacons of our 1-hop neighbors too. Figure 5.9 depicts the update process. To make it more readable it shows only those beacons that carry new information for one of the neighbors. We see that without propagation of pre-calculated neighbor beacons we

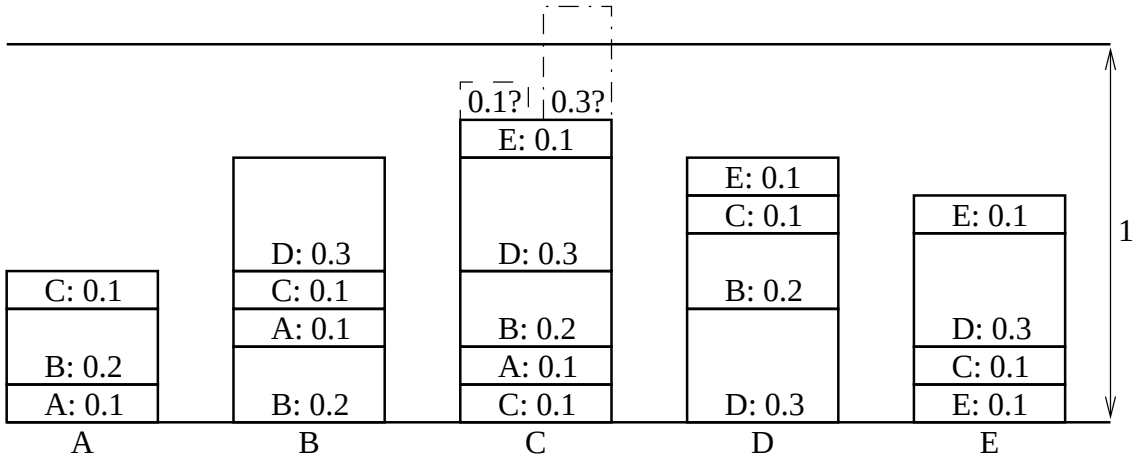

Figure 5.8.: Beacon-based bandwidth coordination

need four steps to complete the neighborhood update instead of two. We additionally see that the 1-hop neighbors take longer to come into an consistent state without the optimization. Together with the conservative merge of inconsistent values we get a much more reliable bandwidth coordination. The reason is that if the update is incomplete and the conservative merge does not correct this we have the possibility to make a *false admission*. So it is important to propagate allocation updates as fast as possible to avoid this.

Method	standard beaconing					beacon with pre-calculated updates				
Step/Node	A	B	C	D	E	A	B	C	D	E
0	abc	$abcd$	$ab\underline{c'}cd$ $\longleftrightarrow$ $\{bc'd\}$	$bcde$	cde	abc	$abcd$	$\underline{a'b'c'd'e'}$ $\longleftrightarrow$ $\{b'c'd'\}$	$bcde$	cde
1	abc	$a\underline{b'}c'd$ $\longleftrightarrow$ $\{ab'c'\}$	$abc'cd$	$bc'\underline{d'}e$ $\longleftrightarrow$ $\{c'd'e\}$	cde	abc	$\underline{a'}b'c'd'$ $\longleftrightarrow$ $\{a'b'c'\}$	$\underline{a'}b'c'd'\underline{e'}$	$b'c'd'\underline{e'}$ $\longleftrightarrow$ $\{c'd'e'\}$	cde
2	$\underline{a'}b'c'$ $\longleftrightarrow$ $\{a'b'\}$	$ab'c'd$	$ab'c'c'd$ $\longleftrightarrow$ $\{b'c'd'\}$	$bc'd'e$	$c'd'\underline{e'}$ $\longleftrightarrow$ $\{d'e'\}$	$a'b'c'$	$a'b'c'd'$	$a'b'c'd'e'$	$b'c'd'e'$	$c'd'e'$
3	$a'b'c'$	$a'b'c'd'$ $\longleftrightarrow$ $\{a'b'c'\}$	$ab'c'c'd$	$b'c'd'e'$ $\longleftrightarrow$ $\{b'c'd'\}$	$c'd'e'$					
4	$a'b'c'$	$a'b'c'd'$	$a'b'c'c'd'$	$b'c'd'e'$	$c'd'e'$					
x: original value; x': updated value; $\underline{x'}$: calculated update										

Figure 5.9.: Bandwidth estimation propagation

Releasing allocated bandwidth

To release allocated bandwidth it is only necessary to delete the corresponding entry in the allocation table. Afterwards the node will propagate the reduced bandwidth allocation in its beacons.

Because of the conservative strategy to resolve inconsistencies the reduced bandwidth allocation will be recognizes by the 1-hop and 2-hop neighbors with a delay. This causes a temporary over-estimation of the allocated bandwidth. To speed-up this process we can apply the same optimization as for the bandwidth allocation – to update the stored beacon data based on local knowledge and propagate their value. In this case it is not that critical to delay the update but it probably will cause a new bandwidth request to be rejected although it would be possible.

Handling bandwidth over-commitment

There are two possible reasons that can lead to a bandwidth over-commitment. First, due to the propagation delay of allocation updates it can happen that two node within 2-hop distance independently accept different bandwidth requests. Second, due to topology changes, e.g. a merge of two networks, we can get into situations where the allocated airtime exceeds the available airtime. In these situations we have no other chance than to sacrifices one or more local allocations in order to save the remaining.

In most cases an over-commitment will be caused by two parallel bandwidth requests. If two networks join it is more likely that we will experience QoS violations because of the interference before we start to receive beacons. As a result we will probably release one or more QoS flows with solves the over-commitment situation at a higher level.

A special case where we have parallel bandwidth requests is the reservation of routes. In this case we will use the address information from the reservation packet to pre-calculate the changes in the neighborhood and store them locally until we receive an updated beacon.

5.5.4.2. Traffic Shaping and Prioritization

To enforce the allocated bandwidths we have to perform a local traffic prioritization and shaping. The IEEE 802.11 MAC protocol will ensure an almost fair medium access, i.e. if all senders that have to share the medium try to saturate the bandwidth each will be able to transmit an equal amount of data. To shift the bandwidth distribution in favor of one of the senders we have to ensure that the others will not try to occupy the medium at all time. This can be done using a local traffic shaper. The prioritization is necessary to favor some packets over others (s. appendix A.2.2 for defined classes). Primary this is needed to transmit packets belonging to QoS flows before any best-effort packets. But it can also be used to optimize the end-to-end delay of certain flows, e.g. by assigning a higher priority to late packets.

The goal of a traffic shaper is to limit the transmission rate to a certain amount. This limit can be specified either as a number of bytes per second or as a number of packets per second. As we do not use packets of fixed size we decided to use the bytes per second versions but specified as the required airtime to account for headers and different transmission rates. We cannot control the transmission of individual bits so we have to make sure that at least at the beginning of each packet the bandwidth limit is not exceeded.

The traffic shaper consists of two parts – the time triggered gate and an input queue. The time triggered gate is basically a temporal lock variable. If we transmit a packet of size s and have a bandwidth/airtime limit of at_{max} the gate will block further transmissions for t_{lock} calculated as:

$$t_{lock}[s] = \frac{airtime(s[byte])}{at_{max}} \tag{5.9}$$

The current implementation (s. section 6.4.2) will notify the upper layer (usually a priority queue) whenever the lock becomes free to request the next packet send. If the sender utilizes the allocated bandwidth the queue should never run dry, i.e. it always should contain at least one packet to send.

The function of the traffic shaper in combination with the IEEE 802.11 MAC protocol shall be explained in the following. We will omit the airtime transformation for better readability in the example. We assume that we have tree nodes that each want wo transmit a packet with the sizes $s_1 = 1000B$, $s_2 = 1500B$, and $s_3 = 2000B$ each with a traffic limit of $b_1 = b_2 = b_3 = 500\frac{kbit}{s}$. If all three nodes try to send at the same time, two things will happen. First, the traffic shapers on each node will block the node to transmit a second packet for $t_{lock_1} = 16ms$, $t_{lock_2} = 24ms$, and $t_{lock_3} = 32ms$. If the wireless medium has a transmission bit rate of $2\frac{Mbit}{s}$ the actual transmissions will take approximately $t_{tx_1} = 4ms$, $t_{tx_2} = 6ms$, and $t_{tx_3} = 8ms$. As all nodes are subject to the random back-off mechanism we cannot say which node will win the contention for the medium access – as an example, lets assume node 2 wins. It will start to transmit its packet and then has an empty transmitter buffer as the traffic shaper blocks further packet transmissions. The remaining two node still compete for the medium access. Lets assume, node 1 gains the medium access next. It will transmit its packet in the time from $6ms$–$10ms$ and then has an empty transmitter buffer as well. Last, node 3 can access the medium in the time $10ms$–$18ms$. During this time the traffic shaper of node 1 passes the next packet to the transmitter that can be transmitted immediately ($18ms$–$22ms$). After this transmission the medium will be busy for $2ms$ because all nodes are blocked by their traffic shapers. The medium access is depicted in figure 5.10. We see that regardless of the random order caused by the MAC layer we get all packets transmitted because of the traffic shaper. It should be noted that we left out the time required for the packet headers and possible retransmissions to simplify the example. In the actual system they will be included in the calculation as well.

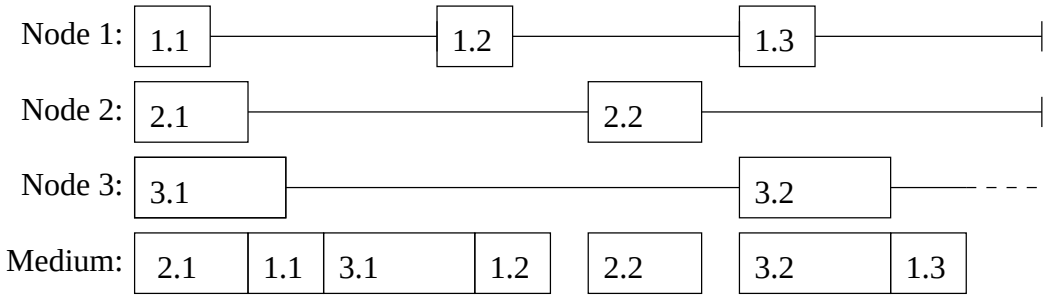

Figure 5.10.: Medium access with traffic shaping

5.5.4.3. Dynamic Adaptation

In section 5.5.2 we already discussed the problem of interference outside of the bandwidth coordination area, i.e. the problem that two nodes have to share the bandwidth but are unable to perform communication. So we have to solve the problem using a reactive approach. We start with the assumption that we do not have the interference problem. Than the beaconing service and traffic shaping are sufficient to coordinate the bandwidth utilization among nodes. If this assumption is false we can detect this using the traffic shaper and especially its input queue.

We assume that applications will transmit packets periodically at a given rate. Applications that use burst transmissions can be converted to a periodic sender using a leaky bucket filter. Now, if the applications fill the traffic shaper queue at the same rate it transmits the packets we should get a constant average queue length. Additionally, if we look on the lock times and the transmit buffer (one element) of the traffic shaper we should see a constant average idle time, i.e. the time the transmit buffer is empty but the lock variable prevents another packet to be send. It should be noted that it requires a feedback from the network driver to determine this value which maybe not available (s. section 4.7.5). We now can see different effects. First, if the queue length grows or the idle time shrinks we know that the actual channel utilization is higher than expected. In the opposite, if the idle time grows we know that the channel utilization is lower than expected. The queue length will not shrink as long as the traffic shaper has the same transmission rate as the applications. Both values, the queue length as well as the idle time have to be used in combination. The idle time provides us with an indication of weak changes whereas the queue length indicates rapid and significant changes. If a new node would start to transmit at full speed in close range this would cause an almost instant significant increase in the queue lengths. The idle time of the transmitter would fall to zero but the calculated average would take some time to indicate this rapid change.

When we detect an over-utilization of the channel we can have two reasons: a false admission, or an interfering node. In both cases we need to reduce our own transmission rate which obviously means to sacrifices some packets. The surplus bandwidth (b_s) is equivalent to the growth of the average queue length. A decreased idle time is just an indication that the channel capacity started to degrade. If the surplus bandwidth is low we will reduce the bandwidth allocation for best-effort data and send a warning message to the local applications. Although we do not know who causes the channel degradation we have to adapt to it. So we will store b_s as a local bandwidth allocation without using it, and report it to other nodes using the beaconing service.

In the opposite direction we have to adapt to an under-utilization of the channel. This can happen in two cases: the channel degradation caused by an interfering node vanishes, or we spend less time on retransmissions than we have planned – which is a common case. We will recognize this situation by an increased idle time of the transmitter. The first thing we have to do is to revert any compensation for an over-utilization. Second, we will try to empty the input queue of the traffic shaper. This is done by an immediate reset of the temporal lock so that the next packet can pass through. This can cause the input queue to run dry, i.e. we do not have any packets left to send. This will be signaled to the higher layers that react e.g. by creating on-demand best-effort traffic. We can also decide to do nothing. In this case we will provide other nodes with an increased

chance to send their traffic.

There is another possible usage of the under-utilization detection. If we have a mostly static network, e.g. in an office environment, or if we mostly use multi-media applications that can tolerate packet loss of some degree at the application level we can also perform an optimistic air-time calculation. In section 5.5.4.1 we used a $retry$ factor to calculate the bandwidth required for unicast transmissions. This is usually set to the maximum possible number of retransmissions. But if we recognize a permanent under-utilization we can reduce this factor too. This would allow to admit more QoS bandwidth. It should be noted that this will not be done by default as it could lead to a significant number of QoS violations in a dynamic environment. So this feature requires *expert knowledge*, i.e. a manual activation depending on the application area.

5.6. Link Quality Monitoring

As explained before, link monitoring is an essential part of our system as we have to cope with gradual link qualities. The link monitoring algorithm presented in this section has the following objectives:

- estimate the probability for a successful broadcast transmission to a certain neighbor,
- estimate the usability of a link for unicast communication,
- include link stability in quality calculation,
- perform conservative estimation (avoid overestimation of quality),
- consider dynamic of the network.

The link quality monitoring is based on the received packets from neighboring nodes that give a clear statement about the quality of a link. That means, if we receive a single broadcast packet from a neighbor this only gives us the information that such a communication is possible. But it does not help us to determine the loss probability of a broadcast packet. In contrast, if we know that nodes are sending periodical broadcast packets that include a sequence number or something similar (e.g. a beacon) we can count received and lost packets to determine the loss probability. From the loss probability of broadcast packets we can estimate the usability for unicast communication. Unicast communication includes an automatic retransmission scheme to increase the reliability. But there is another source of information to determine the usability for unicast communication. If we receive a unicast message from a neighbor we can take this as a very reliable indication that our neighbor is able to transmit messages to us via unicast. This does not necessary means that we will be able to do the same – this is a rare situation but cannot completely be neglected.

When we calculate the link quality we have to keep the link stability in mind. If a new neighbor starts to transmit a beacon, we will receive a first packet and could calculate a link probability

of 100% because we received all packets so far. Of cause this does not make much sense. So we will limit the maximum possible quality by the time we know about a neighbor. A neighbor first needs to successfully transmit beacons over a continues time before we assume the link to be stable. After the time t_{mq} a link can become a medium quality link and after t_{hq} a high quality link, based on the loss statistics. There is another reason to require a certain number of frames before defining a link to be stable. We only count received frames but we will estimate the success probability for packets transmitted in the opposite direction. In section 4.3.3 we saw that we have to consider asymmetric links, i.e. links that have different quality values in both directions. We additionally saw that high quality links are very likely to be of high or at least medium quality in the opposite direction. So, before we define a link to be of high quality, we have to be confident about this, which is only possible with a number of probes. In contrast we have to keep the dynamics of the network in mind. For example, if we have two nodes close together for a long time, we could for instance count 997 successfully received out of 1000 send beacons. Now, if one node starts to move away it would take a long time and hence a large number of lost beacons to reduce the estimate loss probability significantly. To cope with this effect and to be able to detect link degradations or breaks within a short time we will introduce the following measures: a bounded time of acceptable link silence ($t_{silence}$), and a moving average calculation of the link quality value. If we loss a certain number of consecutive probe frames we define the link to be broken regardless of the number of successfully received packets earlier. To limit the computational overhead of the link quality and moving average computation we will only count received and lost packets and compute the quality value on demand. For the moving average calculation we will halve the counted numbers of received and missed packets in a fixed interval (t_{avg}).

For the quality of a link from node i to j we define:

$$q_{i\to j} = \begin{cases} 0.1 & \text{if } t_{i\to j} < t_{mq}; \\ min(0.5, \frac{n_{recv}}{n_{recv}+n_{lost}}) & \text{if } t_{mq} \leq t_{i\to j} < t_{hq}; \\ \frac{n_{recv}}{n_{recv}+n_{lost}} & \text{if } t_{i\to j} \geq t_{hq} \end{cases} \tag{5.10}$$

The number of received packets (n_{recv}) is initialized as 1 if we receive the first packet from a new neighbor. The number of lost packets (n_{lost}) is determined based on the sequence numbers included in periodic transmissions – beacons in most cases. We store the sequence number of the last beacon and calculate the difference to the next received sequence number. If it is more than 1 we know that we lost some packets in the meantime. Each time we receive a packet from a neighbor we store a time-stamp. We periodically check the list of known neighbors if we have received a packet within $t_{silence}$. If not we remove the entry from the list and report a link break. If we consider periodic beacons (interval t_{beacon}) as the source for our measurement we usually use the following values: $t_{silence} = 5 \cdot t_{beacon}$, $t_{mq} = 3s$, $t_{hq} = 5s$, and $t_{avg} = 10 \cdot t_{beacon}$.

Received unicast packets get a special treatment. In general, if we receive a unicast packet we count it as one successfully received packet. If the packet is part of a flow we additionally set $n_{loss} = 0$ (equivalent to $L_{i\to j} = 1$) if we receive subsequent packets because establishing a flow requires to successfully transmit packets in both directions (s. section 5.8). If we lose packets of

a flow we increase n_{lost}. For active links that periodically transport data (e.g. of a QoS flow) this means a more detailed monitoring of the link.

For links that are not actively used we can improve the link monitoring by an active neighbor probing – we simple send a *ProbeRequest* via unicast to a neighbor that has to answer with a *ProbeReply*. If we receive the reply we can be sure that the link is fully functional – we increase n_{recv} and set $n_{loss} = 0$.

5.7. Packet Age Estimation

One of the supported QoS properties is the end-to-end delay. As we use the IEEE 802.11 MAC protocol we have little chance to influence the medium access and hence the end-to-end delay (s. section 5.5.4.2 for details). So, aside from the local prioritization this is mostly a monitored property, i.e. we will measure the packet delay and report violations of QoS requirements.

The delay (d) of a packet n is defined as

$$d_n = t_{now} - t_{send_n} \tag{5.11}$$

If we had synchronized clocks on all nodes the delay calculation would be straight forward – store a time-stamp when a packet has been created and calculate the difference to the current time. But unfortunately there is nothing as a synchronized clock in our system – this is a standard problem in almost every distributed system. As it is a standard problem, there are also various solutions to synchronize the clocks but they require a lot of time, specially if the number of nodes grows. Additionally they cannot provide an arbitrary precision. So we will not try to improve the quality of the clock synchronization in this work, instead we are going to estimate the delay with least possible effort.

The proposed solution uses the notion of the *age* of a packet. The age (a) is the sum of the times the packet n spend on the different nodes:

$$a_n = \sum_{i=k}^{l} (t_{received_n} - t_{send_n}) \tag{5.12}$$

with $t_{received_0}$ being the creating time of the packet and t_{send_l} being the time the packet is handed over to the application. The age estimation works as follows: whenever a packet enters a node (is created or received) it gets a local time-stamp. When it leaves the node (transmitted or handed over to the application) the difference of the current time and the stored time-stamp is added to the age stored in the packet. Obviously we introduce an inaccuracy each time we transmit a packet to a neighbor node. This time includes the actual propagation time on the medium which can be neglected and the time between the moment we update the age and the actual transmission on the sender side, and the moment we receive the packet and store the local time-stamp on the other side. To reduce the introduced error we need the support of the network driver. Current

IEEE 802.11 compatible network cards include a local timer used for the back-off timers that can be accessed by the driver. Currently this is often used for the *monitoring mode* of the cards, that is a special mode of operation specially designed to monitor all packets received by the network card. Using this timer can significantly reduce the measurement error. Otherwise we can only compensate the error by adding the average (or more conservative the maximum) delay that we cannot measure.

If we compare the age measurement with synchronized clocks it is preferable for two reasons:

1. we do not need any clock synchronization and therefore save some communication – we can neglect the local clock skew during the time the event resides on the node,

2. we are free to perform an optimistic (e.g. average case) or pessimistic (worst case) estimation of the transmission time and so can satisfy different requirements to the compliance with a maximum event age. If we always assume a worst case transmission time we will never deliver an event too late but possibly report some false-positive QoS violations.

5.8. QoS Multicast Tree Construction and Maintenance

In this section we will discuss the actual delivery of messages from a publisher to its subscribers. Basically we have two possibilities: multiple point-to-point connections, or a single multicast tree. As we discussed in section 5.5.1 a MANET will only scale if we limit the length of the routes between peers. So the obvious choice is to use the multicast tree option. With a growing number of subscribers for a publisher the probability to find a node on the tree in short distance grows. Additionally, if we have to use a long route this in turn results in a higher number of intermediate nodes that all can be used as the source of a new branch. Figure 5.11 depicts 10 randomly placed nodes in a 10x10 grid. We have one publisher (marked by a square) and nine subscribers (marked by a circle). On the left we see a possible topology created by using a shortest path algorithm. On the right we see a multicast tree created using shortest paths to other nodes on the tree. In practice we possibly use longer routes but here we are going to compare the minimum traffic caused by this node placement. For the point-to-point connections we count 38 messages that need to be transmitted for each event created by the publisher. For the multicast tree we count 21 individual message and this without any optimization like broadcasting to multiple neighbors at once. We see that using a multicast tree can significantly reduce the overall bandwidth utilization. In some cases when we have a bottleneck in the network, a multicast tree can be the only chance to deliver the event to all subscribers behind because we will use the bottleneck link only once. Of cause we have a drawback – a multicast tree can be more error-prone as a single node failure can potentially disconnect more nodes from the publisher as in the point-to-point scenario. Nevertheless we will use the multicast tree version as it is the only chance to build a scalable system.

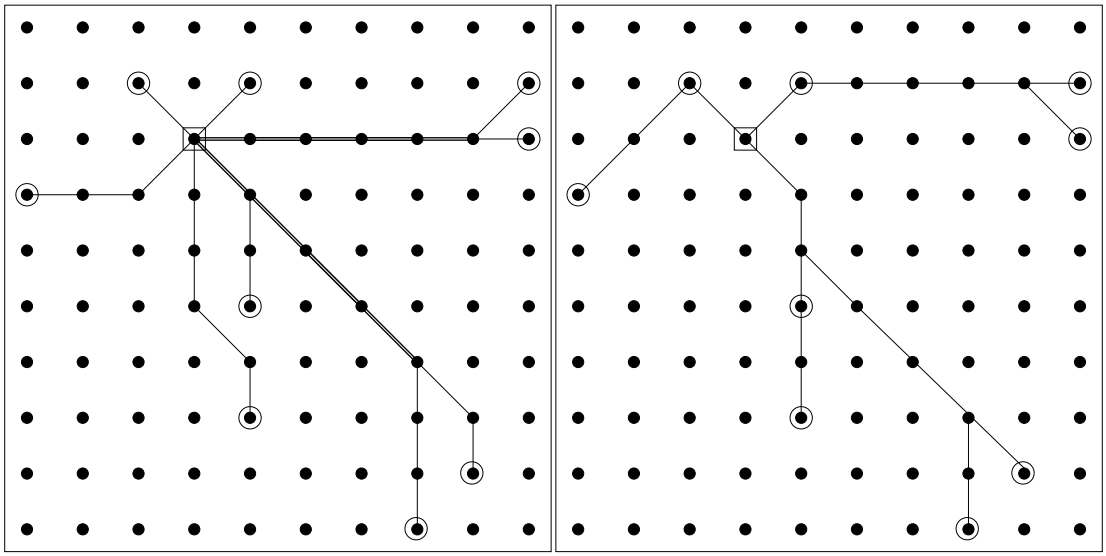

Figure 5.11.: Point-to-point connections vs. multicast tree

5.8.1. Tree Construction

Each node of the tree can either be the root, a leaf, or an intermediate node. Each tree carries exactly one flow, e.g. events with one specific subject. So, every node that is part of the tree can function as the source of the event for node that is going to join the tree. But the nodes differ in the following parameters: the distance to the root, and the age of the events arriving at the node. To forward messages along the tree each node has a *forward table* that stores the identities of the leaf nodes further away from the root on the same branch together with the identity of the neighbor that leads to the leaf (*downstream neighbor*). Additionally the forward table includes the serial number of the message that last updates the entry (explained in the following). Each node will also store the identity of the successor node in the tree (*upstream neighbor*).

Due to the IEEE 802.11 MAC protocol we have no direct control over the order of transmissions on the medium and so cannot directly influence the delay of a packet. We can only try to improve it by a local order, but this does not affect other nodes. So we use a monitoring-based approach to provide end-to-end delay properties. Each node n stores for the flow the distance (hop count) to the root (h_n) of the tree, the minimal ($\underline{d}_n$), average (d_n), and maximum ($\overline{d}_n$) delay of events of the flow, as well as the number of received and missed events (determined based on the sequence number).

The multicast tree is the result on an evolutionary process – every node that we connect to the tree creates a new branch. This works in the following steps:

1. Route discovery and resource pre-allocation,
2. route selection among alternatives,

3. branch activation.

The route discovery searches for possible routes to one node of the existing tree that provides the required QoS properties. This search process includes a pre-allocation of the required bandwidth. As a result we get one or more possible routes for the new branch that will be reported to the root to select the most suitable. To create a new branch we will notify all nodes along the route about the new upstream node. All these nodes will update their forward tables accordingly. Nodes that do not have been part of the tree before become new intermediate nodes. Together with the route activation any pre-allocation converts to a permanent allocation. This means that from the moment of the activation the route is subject to the QoS monitoring and tree maintenance process. The individual steps shall be detailed in the following.

Figure 5.12 depicts the single steps to create a new branch. The search process uses a modified version of the link quality overlay flooding (s. section 5.4.1) to ensure reliable path. It is initiated by the new leaf node and uses the same forwarding mechanism but will forward search requests only if the following additional conditions are true:

- The node is not currently part of the tree,
- the node could allocate enough airtime for the flow,
- the route so far satisfies the QoS requirements.

The search request (`FlowREQ`) includes the following information (the message formats can be found in appendix A.2.1):

- The searched flow (specified by the root node and its local flow identifier),
- the identity of the new leaf node,
- the bandwidth specification (packet size and period),
- the allocation timeout,
- further QoS requirements (e.g. minimum/maximum/average delay, event loss rate).

Each node that forwards the search request pre-allocates the required bandwidth and sets a timeout (*allocation timeout*). If the route will not be activated within this time the allocation is automatically removed. This soft-state approach for the bandwidth allocation is similar to the soft-state of routes in AODV (s. section 3.1). The decision to forward a packets depends upon if we can allocated enough airtime in the neighborhood. For this it is not sufficient to check whether the local node could allocate enough airtime but also if all other nodes in the environment can do it as well. Search requests where prioritized (s. appendix A.2.2) over the beacons. This means the stored beacons represent an old state that will not include the new reservation. The estimated

airtime utilization will increase by the airtime required for the new flow multiplied by the number of nodes that will forward it in our 2-hop neighborhood. Without any further knowledge we have to assume that we have five nodes. If we know the source and it is only one hop away we can decrement it by one. If we additionally know the destination and it is within the 1-hop or 2-hop neighborhood we can decrement the multiplier by two respectively one. Unfortunately we will usually not know how far the destination is and so perform a too pessimistic airtime check at the end of a route. But we will know the distance later in the maintenance phase (s. section 5.8). We have the option to reduce the maximum number of assumed forwarding nodes. In the general case this will include some optimism (and possible lead to QoS violations) but would be completely valid if we have a small-scale network with a maximum known extension of up to five hops.

If we reach a node (*Hit*) that is already part of the tree it is a possible source for the new branch. This node will send a `BranchREQ` message backward to the root. In the example we see three such messages (the root reacts like a intermediate node and send a message to itself). Actually there will be much more but have been left out for better readability. These messages arrive at the root at different times. The messages contain the following information:

- Identity of the new leaf node,
- the estimated age of the packet at the branch node,
- hop count for the route from the new leaf to the branch node,
- the latest flow statistics (delays, loss rate) at the branch node,
- the QoS requirements of the new leaf.

The root will defer the first message for a small amount of time to collect other messages before making a decision. After that it selects the most suitable branch node and route to the new leaf. If we have end-to-end delay requirements (maximum or average), we will estimate an upper bound of the delay to the new leaf from the delay measured at the branch node plus the packet age, and a lower bound by scaling the measured per-hop delay to the new branch length. As search packets have a lower priority than QoS traffic the age from the leaf to the branch node is an upper bound for the delay on that path.

It responds with a `BranchConnect` message that includes the following information:

- identity of the branch node,
- identity of the leaf node,
- route from the branch to the leaf node,
- a serial number to identify the message.

This message will be forwarded along the tree to the branch node using the normal forward tables. Each node along the path to the branch node extends the forward table by the new leaf node. The branch node forwards the message along the route stored in the message to the leaf. The following nodes (except the leaf node itself) become intermediate nodes. That means that they create a new forward table for the flow and start to monitor the flow and to keep the flow statistics. The leaf node itself will be notified of the successful connection and starts to monitor the flow too.

As the `BranchREQ` messages arrive at the root node at different times it can happen that the it finds a better branch. In this case it creates two messages, a `BranchConnect` message to activate the new branch and a `BranchDisconnect` message to deactivate the old branch. To allow a seamless branch migration the `BranchConnect` message gets the higher serial number but is send out first. The `BranchDisconnect` message with a lower serial number follows thereafter. A node that receives a `BranchDisconnect` message checks if the serial number stored in the forward table is less than the serial number of the message. If so, it removes the entry from the forward table and deallocates the associated bandwidth. If not, it just forwards the message. A node that receives the `BranchConnect` message will add a new entry to the forward table or updates it if it is already present from a previous `BranchConnect` message. We see that sending the new `BranchConnect` message before the `BranchDisconnect` message will result in an update of the entry and note a deallocation followed by a reallocation of the entry. Although we use unicast transmissions and hence retransmissions for the tree maintenance messages it is possible that one gets lost. In this case we will forward packets to a node that is not the destination and has no forward entry for the flow. If it still has an active pre-allocation from the search process, it synthesis a `BranchConnect` message from the stored information, otherwise it sends a `BranchReject` message back to the root. After a certain time the destination will start to reestablish the route.

As soon as the forward entries are included in the forward table the new routes will be used to deliver events. If the flow monitoring receives an event it refreshes the timeout to detect a QoS violation. This way the initial allocation timeout that is usually set to a much higher value gets overwritten by the actual event timeout. For all nodes that forwarded the search request this means that they allocate the necessary bandwidth to forward the flow but if they do not get the first event within the allocation timeout they will simply drop the allocation. After that moment they will also answer each `BranchConnect` message with a `BranchReject` message to the root to indicate the error condition.

5.8.2. Tree Maintenance

After the tree (the first branch) has been established it has to be maintained. The maintenance has the following objectives:

1. Detect link breaks,
2. perform a local link repair,

3. notify peers about irreparable link failures.

The tree maintenance focuses on the detection and correction of link failures and will try to correct them in a local area (2 hops). The reason is that most link breaks are caused by the movement of nodes. As nodes cannot jump from one position to a completely different one, we assume that if a node disappears from the 1-hop neighborhood it has moved to the 2-hop range. Probably we will have a 2-hop route to the missing node. If yes we will try to rearrange the local routes. Otherwise it will be necessary to perform a complete reconstruction of the tree. The tree maintenance will further try to optimize the structure of the tree by shortening the path lengths if possible.

For the tree maintenance we will use the information from the forward and the neighborhood tables.

Repairing broken links

A broken link will be noticed either by the sender or by the receiver side. The sender notices a link break if the neighborhood discovery reports a vanished neighbor and this neighbor is a downstream neighbor. The receiver will notice a link break by the missing upstream neighbor link, or by a series of missing events (QoS violations). The repair process is similar for both sides.

If the receiver notices a link break (s. figure 5.13 step 1) it initiates a local search for the missing neighbor (with a TTL of 2) using the link quality overlay flooding (step 2) as used for the `FlowREQ` search. The search request (`ReconnectREQ`) includes the identity of the missing neighbor, the flow identification, and all QoS related information that are required for the bandwidth pre-allocation. It additionally sends a `Standby` message instead of the missing event along the branch to prevent other downstream nodes to start their own repair processes because of a QoS violation, they will only count the lost events. If the missing neighbor receives the search request (step 3a) it will reply with a `BranchRepair` message to the downstream node to update the bandwidth allocations and forward tables on the new (1-hop or 2-hop) link. If another node on the branch receives the `ReconnectREQ` (step 3b) it will answer with a `BranchOffer` message. The searching node will pick the first answer and will accept the new link with a `BranchAccept` message. In both cases the source of the repaired link will send a `LinkChangeInfo` message back to the root of the flow. If the source of the repaired link differs from the previous source, all nodes on the way back can update their forwarding tables, the root will send a `BranchDisconnect` message including a list of all migrated leafs to the previous source so that it can stop searching for the missing neighbor. Additionally all nodes that receive this message will remove the migrated leafs from their forwarding tables and free the allocated bandwidth resources if applicable.

In the opposite direction, if the source of a link is missing its neighbor will send a `BranchOffer` message as local broadcast (2-hop flooding). The missing node will reply with an `BranchAccept` message to reroute the link. The source node will than send a `LinkChangeInfo` message back to the root.

Branch shortening

The second important task of the tree maintenance is the shortening of branches. This way we try to reduce the complexity of the tree. This leads to a better utilization of the medium, shorter end-to-end delays, and increases the reliability as we get fewer links that can fail.

The link shortening uses the same information as the link repair – the neighborhood and forwarding tables. It works very simple. If a node discovers a new neighbor it locks into its forward table to test if the new neighbor is included in the branch but more than one hop away. If yes, it tries to create a direct link to it. The following message sequence is the same as in figure 5.13 picture 4: the node sends a `BranchOffer` message that the target node can accept with a `BranchAccept` message. If the new neighbor accepts the offer, a `LinkChangeInfo` will be send to the root of the flow. For this task it is essential to know the list of downstream nodes at every node of the tree. Otherwise the link shortening would require a coordination with the root node of the flow which would save some memory on the intermediate nodes but significantly increases the tree maintenance traffic.

Report irreparable link breaks

If a node repair fails we have to notify the root and leaf nodes about this so that the higher level P/S protocol can handle the problem. To do so we set a timeout on each local repair attempt. After this time the receiver node of a broken links will send a `BranchLost` message to all downstream nodes. These node will clear their forward tables and free the allocated bandwidth. A leaf node will additionally notify the P/S management protocol of the problem. The sender node of a link will send a `BranchLost` message upstream so that all nodes in-between can remove the lost leafs from their forward tables and free the allocated bandwidth resources if applicable.

5.8.3. Tree Deconstruction

At last we need a way to disconnect nodes from the free intentionally. If an application at one of the leafs finishes it sends a `BranchDisconnect` message including its own identifier upstream. All nodes on the way to the root will update their forward tables and bandwidth allocations accordingly. If the root node finishes, it sends a `BranchDisconnect` message including all leaf nodes downstream.

A more difficult situation is when an intermediate node intends to leaf. In this case it will send a `ForceRepair` message to all its direct downstream neighbors. When they receive the message they will start the usual repair process (s. figure 5.13 no. 2, 4). The node that intends to leave will additionally stop to forward any messages related to the tree maintenance. It will continue to forward events until it shouts down. But it will not longer monitor the QoS properties to avoid to initiate a QoS related repair process. If the repair process is successful, it will stop to receive

further events. At any point in the future (usually the repair timeout) it can decide to drop the forward table and bandwidth allocation for the flow.

5.8.4. QoS Monitoring Management

We previously mentioned that missing neighbors can be detected by QoS violations. But we have only three messages (`FlowREQ`, `BranchREQ`, `ReconnectREQ`) that include QoS parameters. All messages that are used to migrate branches do not carry any QoS parameters. The reason is very simple – storing a list of all downstream nodes at every node is essential to maintain the connectivity of the tree. But to store and monitor the QoS parameters of every leaf at every intermediate nodes probably requires a lot of memory to store and transfer the parameters on a per-node basis, and significant CPU resources to compare every event against every requirement. So we will limit the processing in the following ways. First, we will only store a maximum of one QoS parameters set (the least common dominator) at a node. So we will detect every event that is not sufficient for any of the leaf nodes. The threshold parameters are setup by the root. It will use a `SetQoSThreshold` message to set QoS parameters on one or more nodes. An intermediate nodes is free to accept the monitor settings or the ignore it to save CPU resources. Only the leaf nodes are required to perform a QoS monitoring, but will determine the QoS threshold parameters from the local communication end-points. The message uses a special address format to perform an efficient sub-tree addressing. If a node identifier is followed by a `BroadcastID` this means the node and all its downstream nodes should use the set of QoS thresholds. Several such sub-tree addresses can be separated by a `NoStationID` identifier. As nodes do not exactly know within what sub-tree they are located, each node that forwards such a sub-tree addressed message has to rewrite the address entry to point to the downstream neighbor and its sub-tree.

5.9. Mixing QoS with Best-Effort Traffic

The whole system bases on the reservation of bandwidth (airtime) on individual nodes. The amount of airtime to reserve is determined by the specification of QoS flows. But we additionally have best-effort communication (beacons, search queries, P/S management information, multicast tree management information, ...). At least for the beacons we know how much airtime a node will use. But for all the remaining best-effort communication we simply do not know. To avoid an over-utilization of the medium we will use a combination of a dynamically allocated airtime contingent and the recycling of unused airtime. As explained in section 5.5.4.3 it is possible to measure the actual airtime consumption to detect planned but unused retransmissions. These can safely be used for best-effort traffic. If this is not sufficient we need to increase the allocated amount best-effort traffic. This can be done but only in small steps. Each node is allowed to increase its allocated airtime by the amount of free airtime divided by the number of nodes in the 2-hop neighborhood within the time of an beacon update cycle in the 2-hop neighborhood. But a node is allowed to reduce its best-effort allocation at any time.

We see that the airtime allocation scheme is very focused on the support of QoS flows. It is able to handle a significant amount of best-effort traffic but only under some constraints. If we already have allocated a lot of QoS flows and have a good link quality we will recognize a large percentage of unused retransmissions. So we could forward a large amount of best-effort traffic along the QoS paths. If we try to forward best-effort traffic along nodes that do not carry QoS flows we have to adapt their allocated best-effort contingent which is a slow process, especially in a dense neighborhood. This adaption works fine to handle the management traffic which is routed along the QoS multicast trees and continuous flows but is contra-productive for random point-to-point bursts like file transmissions.

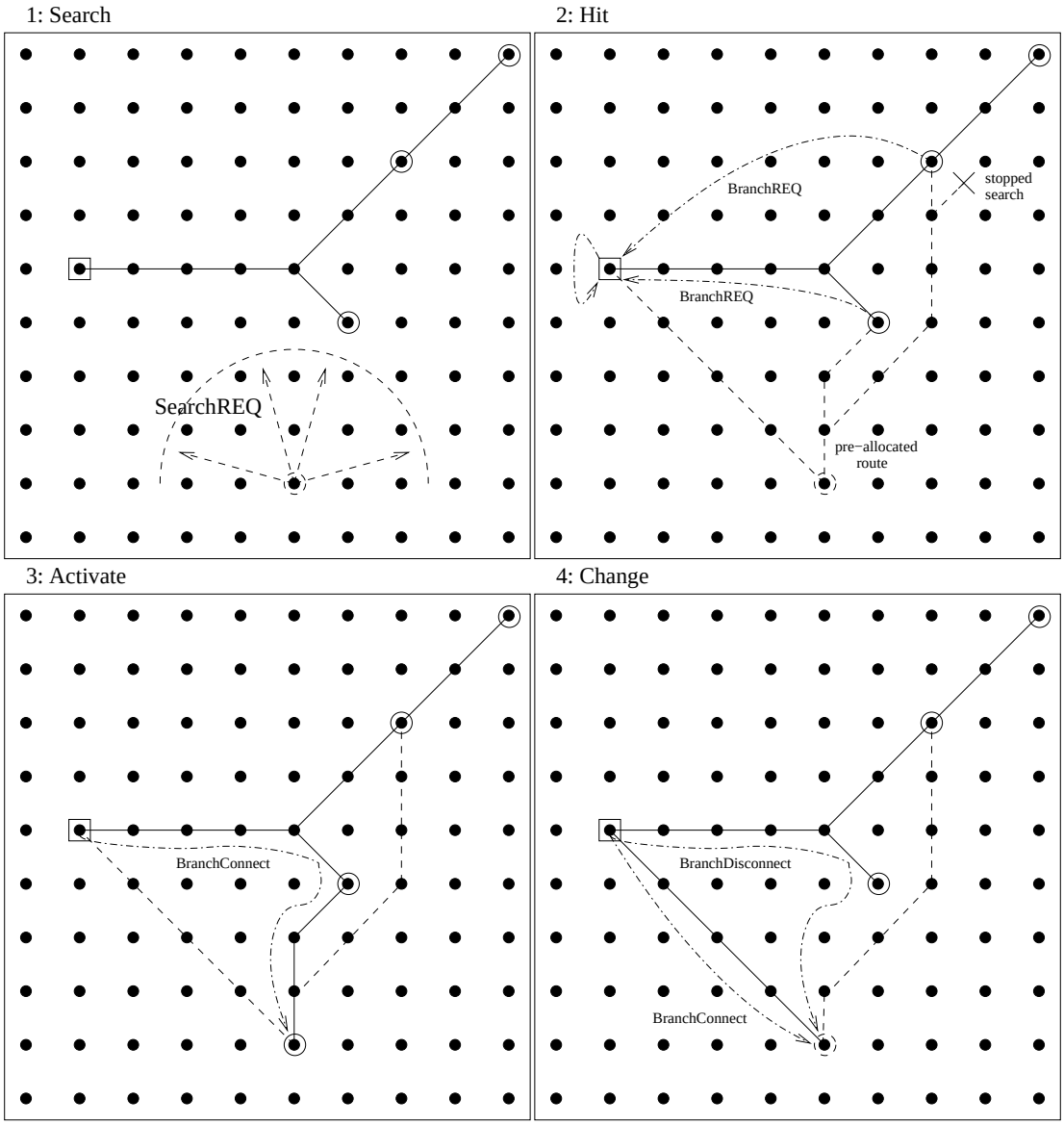

Figure 5.12.: Creating a new branch

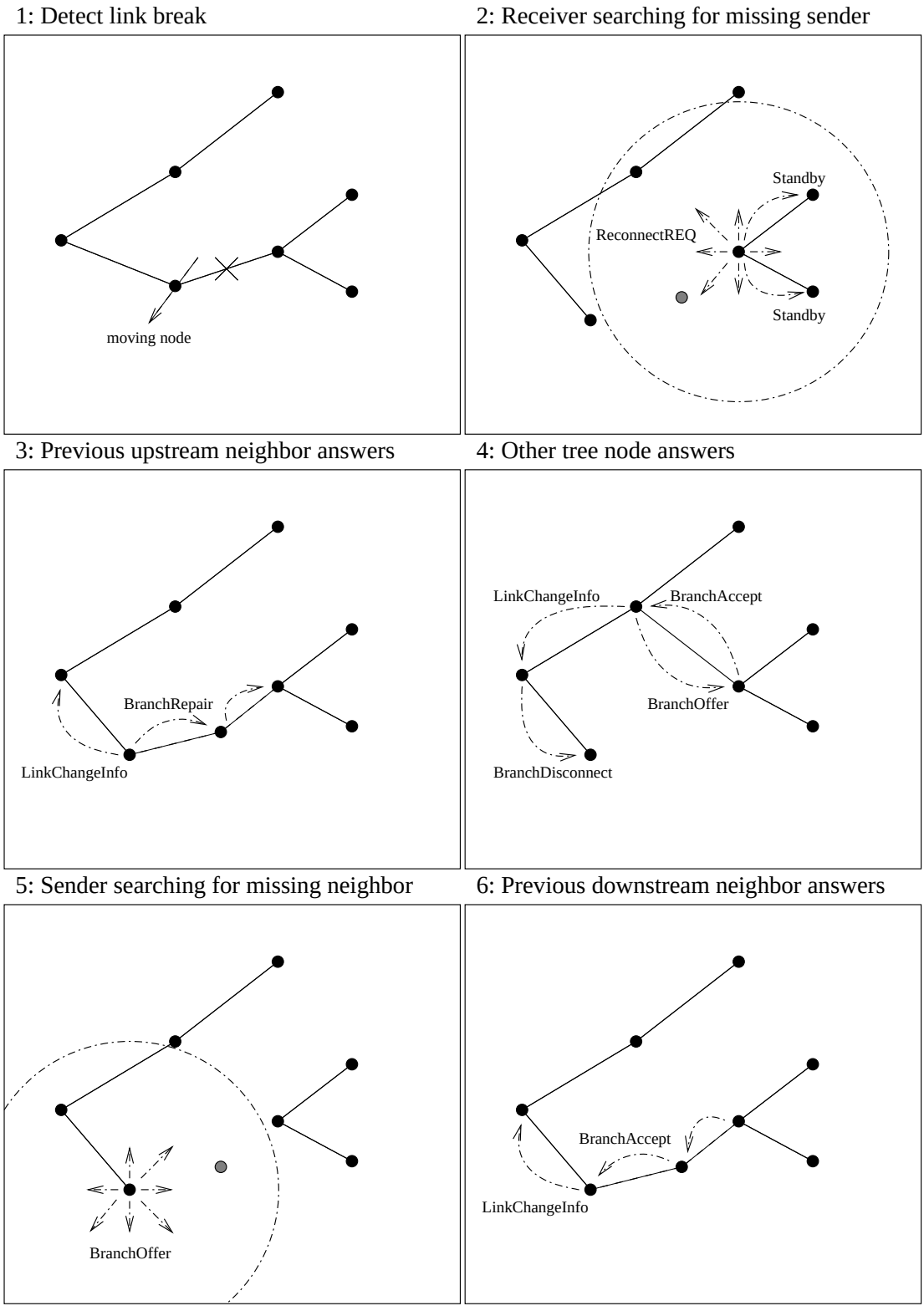

Figure 5.13.: Repairing a broken link

6. Middleware Implementation

In this chapter we will take a closer look on the actual implementation of the communication middleware. The requirements have been specified in section 2.4.1. The functional have already been discussed in the main chapters of this work. At this point we will put a special focus on the non-functional properties of the system.

6.1. Component Overview

The middleware is structured into several components that are logically grouped into several subsystems. Figure 6.1 depicts the component structure. At the bottom we have the hardware and operating system that we handle as black-boxes. On top of the operating system we have a thin runtime-system that provides an abstract interface to the underlying system. It is the only platform-specific part of the whole middleware. Most of this functionality is provided by GEA that will be described in section 6.2. Above this abstraction layer we have the *communication core* that encapsulates basic communication functions and shared information, an arbitrary number of *services* that encapsulate active processes in the communication layer and the *P/S* system that includes all extensions to allow for publish/subscribe communication. Beside the communication part we have the monitoring component that can be access from any other component to store and retrieve global state information (s. section 6.5).

6.2. Event-based Implementation using GEA

In the design phase of the middleware development two fundamental questions had to be answered. First, how to perform an efficient testing of the developed network protocols and second, how to handle concurrencies in the implementation. From the very beginning the necessity for a simulation based testing of the developed protocols was clear (s. section 4.6 for details). But actually our goal was to develop a system that can be used on native systems because simulated protocols (with the exception of network emulation) are of little use in practice. So we had at least two different target platforms for our development – certainly more as we intended to deploy our system on heterogeneous platforms. Previous developments in our research group showed that concurrent implementations of the same protocol were hard to main, and even harder to extend, especially if the development involves a number of people. This leads to the obvious desire to find a way to use a single, unified implementation on all target platforms. At the same time it was

evident that the developed middleware would include several concurrent control flows that need to be handled, and more important synchronized. Such control flows are for instance different protocols, monitoring activities, services, and of course applications that communicate using the middleware.

An investigation of the system interfaces of the various platforms and the way protocols are usually described lead to the decision for an event-based implementation. The network simulator ns-2 required us to use an event-based implementation whereas the usual communication API on POSIX-like system are sockets. We decided to build an abstraction layer that provides the event-based interface on both systems – called GEA (*Generic Event API*) [HM05]. This decision answered the problem how to handle concurrencies at the same time. As events (and the actions they trigger) are atomic units in an event-based system we do not longer need to care about synchronization. We only have to make sure that shared data is in a consistent state at the end of an event action.

GEA is public available [gea] as a native (POSIX) version and as an extension for ns-2. In addition to the event processing interface GEA provides the ability to use communication devices, dynamic loading of event-based applications, and an *object repository* to share objects between applications.

The idea of an unified protocol implementation is not new and there are already several solutions for this problem. Nsclick [NJG02] is a combination of the popular network simulator ns-2 and the Click modular router [KMC+00], a software library that enables to construct routing software from basic building blocks. Nsclick use the Click language to specify routing protocols and executes it on top of ns-2 or the native click execution environment. The main difference to our approach is their additional abstraction layer which restricts the user to certain protocol primitives. Another approach is to simulate the system interface used by an existing implementation. This approach has been used to run existing user-level protocol implementations [LYN+04] or actual network stack implementations [JM05] inside the simulation.

6.2.1. Event API

GEA ist based on the concept of *event notifications*. There is one central unit – the *event handler*. The application or protocol can register for events like receiving a messages or the occurrence of a timeout. The event handler will notify the protocol when something important happens. We decided to use simple callback functions for this – implemented as C function pointers. The resulting mechanism is efficient, as the protocol code only has to work when there is something to do. The only drawback is that it must be ensured that the event handlers do not spend too much time. Otherwise the delivery of other events is delayed.

Handles and Callbacks

The central system primitives in GEA are *handles*. A handle is normally associated with an input our output resources. It provides functions for sending and receiving. But they are also used for

event notification. Therefore the event handler provides a function `waitFor(Handle *h, AbsTime t, Event e, void *data)` which allows to register callbacks. The parameter `e` is a function pointer like `event(Handle *h, AbsTime t, void *data)`. The parameter `t` defines a timeout.

For receiving data a handle is created. After that a callback is registered with `waitFor`. When data on the handle arrives the callback is executed. The occurrence time and the data pointer specified on the registration are given as parameter. The callback can then retrieve the packet, process it and maybe register other callbacks. The timeout parameter defines when the handle should stop to wait for data. If this happens the callback will also be called. The protocol has to check the status of the handle to detect if a timeout happened.

A special case is waiting without an I/O handle. This can be for a self-generated event or for a certain period of time. For the first one GEA provides so called `DependHandles`. These can be used like others, but are triggered internally by calling `complied` on it. The second functionality is achieved using a pseudo handle of type `Blocker`. A `waitFor` always returns a timeout after the specified time. The event handler called by the timeout can then execute the delayed commands.

As we use GEA for the development of real-time protocols, the handling of time is very important. This involves that the external communication meets timing restrictions. GEA aids to fulfil this by its handling of absolute time values.

Time Representation

Many runtime environments use relative values for time representation. Common are calls like `sleep(2)` for stopping the process execution for 2 seconds from the current time. Critical for real-time systems is that the current time is not always determined. Calling `sleep(1)` ten times does not result in sleeping 10 seconds. There is always a delay resulting from the processing in between. Theses delays accumulate and can lead to not intended times. We avoid this by using absolute values for time representation. A data type called `AbsTime` is used for this. A second time type, `Duration` is used for representing differences between absolute times. The handling of this time representation is very restrictive. This ensures a correct usage. There are three possible ways of creating an absolute time value. The function `AbsTime::now` returns the current time. When an I/O event occurs, the time is given as parameter to the callback function. The actual `AbsTime` value has no defined meaning for the application, only for the event processing core. A `Duration` value instead is defined to represent a time in seconds. It can be added to `AbsTime` values to specify future points in time. So the creation of time values does not depend on the internal processing. Normally only external events create time points. Variable processing times of event handlers cannot influence the absolute time values.

6.2.2. Event Processing in C++

To ease the development of C++ applications, there is an object-oriented extension to the basic C API of GEA. The following base classes can be used to define objects that are activated if an event occurs:

- `SingleShootEvent`: for single I/O events
- `SingleShootTimerEvent`: for single timer events
- `MultiShootEvent`: for recurring I/O events
- `MultiShootTimeEvent`: for periodic timer events

All classes provide the virtual method `handle(Handle *h, AbsTime t)` that has to be overloaded. It is called whenever the event is triggered. Multi-shoot events additionally provide the methode `void waitagain()` that can be called to wait again for the next event of the same type. Object of these classes need to be allocated dynamically and will destroy itself if not longer required. The following code example should illustrate how the classic *Hello World* program looks like using this interface.

Event-based *Hello World* program

```
1  #include <gea/utils/SingleShootTimerEvent.h>
2  #include <iostream>
3  using namespace std;
4  using namespace gea;

6  SSE_HelloWorld : SingleShootTimerEvent {
7   public:
8    SSE_HelloWorld(Duration t): SingleShootTimerEvent(t) {}
9   protected:
10   void handle(Handle *h, AbsTime t) {
11     cout << "Hello World" << endl;
12   }
13 };

15 extern "C" {
16   int gea_main(int argc, char * const *argv) {
17     new SSE_HelloWorld(2); // activate event in 2 seconds
18     return 0;
19   }
20 }
```

6.2.3. Dynamic Loading and Execution

Event-based applications or protocol implementations to be used with GEA have to be compiled as shared libraries. At least for the ns-2 and POSIX system we can use the same binary for simulation and a native system as long as both run on the same CPU type. Obviously it is not possible to run an Intel binary on an ARM CPU. As an example the start of the Hello World example in the simulator looks like:
`$node_(0) gea_start ./helloworld.so`.
The counterpart in POSIX is done by:
`gea_start ./helloworld.so`.
Each GEA instance is able to load and run an arbitrary number of those libraries. When loaded, GEA will call the `gea_main(...)` function to create some initial events. After it returns, the normal event processing is resumed. All GEA application share the same address space. Furthermore, if we load multiple instances of the same software into the simulation, they share the same code and only require additional data per instance. As an example, a simulation that started 10000 instances of the middleware used about 120 KB of memory per instance.

6.2.4. Object Repository

To share objects between different application the *object repository* can be used. It stores triples that include the name, the type (given as a string), and the pointer to the object. The object repository provides the following methods:

- `bool insertObj(const char *name, const char *type, void *obj)` to insert a new object into the repository; it will report an error if an object with the same name is already in the repository,
- `void* getObj(const char *name)` to query the pointer of an object specified by its name,
- `const char* getType(const char *name)` to query the type of an object specified by its name,
- `bool removeObject(const char *name)` to remove a specific object from the repository.

An application that stores a pointer to an object in the repository has to ensure that it removes the object before the pointer becomes invalid.

6.2.5. Debugging of Distributed Applications

In this section an interesting byproduct of the development work should be briefly presented. The combination of GEA and the ns-2 proved to be very efficient for the protocol testing and evaluation. In a combination with *valgrind*, another widely used debugging tool it became even more useful. Valgrind [NM03, Net04, SN05] is a binary code analysis tool that allows to determine common mistakes like invalid memory access, use of uninitialized values, improper memory management. It is available under the GPL license.

Usually, if we debug a protocol implementation we have several protocol instances running on separated nodes or within separated virtual machines on one node, if we use network emulation. In such a environment we are for instance not able to decide if a header field in a received message has a random value because it was not initialized properly at the transmitter side. Such a mistake can lead to a strange protocol behavior that can be very difficult to analyze. But as the simulator, GEA and all GEA-based applications share the same address space and run in a single process in the simulation we are able to perform an end-to-end debugging of a distributed system. This setup showed its usefulness numerous time during the development of the middleware.

6.3. Component Interaction

In this section we will explain how individual components interact. We have to distinguish interactions of components inside GEA and with external applications.

Components that run inside the same GEA instance can interact using events, shared data, and direct method calls. GEA provides a special handle `DependHandle` that can be used to wait for self-generated (no I/O or timer) events and to create them. As all components share the same address space they potentially have full access to the memory of other components. To find the addresses of other components, the object repository can be used. It basically stores any pointer value, but is intended to store pointers to objects. These pointers than can be used for direct method calls. To share data and to react upon changes, the *Monitoring* subsystem of the middleware is more appropriate (s. section 6.5).

For the communication between external components and GEA we have to use normal interprocess communication (IPC), i.e. local sockets, named pipes, shared memory segments, signals.

To interact with the middleware we have designed two mechanisms on top of the basic IPC mechanisms – a *bridge*, and blocking-free ring buffers stored in shared memory segments. The bride (s. section 6.6) enables legacy applications to use the basic routing functions transparently, i.e. without knowing that there is a special middleware. It additionally allows applications that are aware of the middleware to control it using special UDP packets that are translated into method calls inside the bridge. For a shared memory based communication with the middleware an application first has to register using a remote procedure call (RPC). In return it will receive a reference to a shared memory segment that stores two ring buffers – one for sending and one for receiving messages. The application can read and write into these ring buffers without the need

for synchronization. If the application specified that it will send periodic data, the middleware will periodically check the ring buffer for new messages to transmit. Otherwise the application has to notify the middleware of new messages using signals. The same holds for the opposite direction.

6.4. Communication APIs

In this section we will take a look on the various APIs of the communication system and discuss some of the important design decisions. The different layers will be discussed bottom-up.

6.4.1. Packet API

An essential requirement was *configurability* of the system and the network packets – the system should only contain the components required for a task, and network packets should contain only necessary headers.

The *PacketPool* together with the *header access classes* provide a solution for the second part. The PacketPool allows to define the structure of network packets at run-time – usually at start up time. The access classes provide transparent access to the header fields – an application need not to know details about the internal structure of the whole packet. Figure 6.2 defines the basic structure. We have a *System Header Area* that stores information that are only valid on the local host an are not transmitted. It includes at least the *Base* header that stores various management informations like the size of the whole packet, or a pointer to the queue the packet is currently stored in. The *Network Header Area* stores the actual packet as it will be transmitted. It includes at least the header that stores the routing / address information. It has the lowest priority and so will be placed directly before the payload.

Figure 6.2.: Packet structure

Packet configuration

The PacketPool provides the following methods for configuration:

- `addNetworkHeader` to add a header in the network header area,
- `addSystemHeader` to add a header in the system header area,
- `removeNetworkHeader` to remove a network header,

- `removeSystemHeader` to remove a system header.

To add a header a name, the size in bytes, a priority value, a pointer to an initializer function, a pointer to a variable that will store the relative offset of the header in the packet, and a pointer to a type state variable have to be provided. The name is used to identify the header. The priority value defines the order of headers so that the resulting packet structure is independent from the order in which the headers are added. The initializer function is required to initialize the header fields when a new packet is created. The offset variable points to a static class variable in the access class of the header and will be updated whenever the header's relative position changes. The state variable is also a static member of the access class and will be set to a non-zero value when the header is currently registered in the system. This state variable can be used to check if access to the corresponding header fields will be possible. A header can be removed by its name. This will probability change the positions of other headers. In this case their offset values will be updated. The state variable of the removed header class is set to zero.

Packet creation

The PacketPool furthermore provides the following methods for the packet handling:

- `newPacket` to allocate and initialize a new packet,
- `clonePacket` to create an exact, independent copy,
- `deletepacket` to delete a packet.

If a packet is deleted the memory allocated is not necessary freed at the same time. The PacketPool stores a fixed number of packets so that they can be reused.

Header access

The header fields can be accessed through *access classes*. The code snippet in figure 6.4.1 gives an example.

packet header access example

```
1 BasePacket *p = packetpool.newPacket();
2 {
3   HDR_StaticPriority sphdr(*p);
4   sphdr.setPriority(10);
5 }
```

All access classes are derived from `BasePacket` that only stores a pointer to the packet structure. Each access class contains the offset for the specific header that will be updated by the PacketPool. To access a header we have to instantiate an access object (line 3). This takes the pointer and adds the current header offset and stores the result in the object. All subsequent read/write operations will be addressed relative to this header pointer using a constant value defined in the access class. These operations can be very well optimized by the compiler. Just as an example – an Intel CPU will need two instructions to store the packet pointer and the header offset into two registers. Each subsequent access than requires exactly one instruction. The temporary offset on the stack will be optimized out.

Dynamic reconfiguration

It is important to node that currently a dynamic reconfiguration of the packets is possible but should be avoided because it requires a migration of existing packets. To do this, the following improvements are necessary:

- Addition of a special header that includes a unique identifier for the composed packet structure,
- a repository that stores known packet configurations together with their unique identifiers,
- a conversation function that rearranges headers to match the current local default configuration and marks them as incomplete if essential header fields are not available after the conversation,
- a protocol to exchange packet configuration data so that nodes can retrieve unknown configurations from the neighbor that send the packet.

6.4.2. Extended MAC API

The extended MAC (EMAC) layer provides additional functionality on top of the IEEE 802.11 MAC. It consists of an arbitrary number of sub-layers that provide individual functions like traffic shaping, prioritization, queueing. The EMAC API is not intended to be used by applications or services, only by the routing layer.

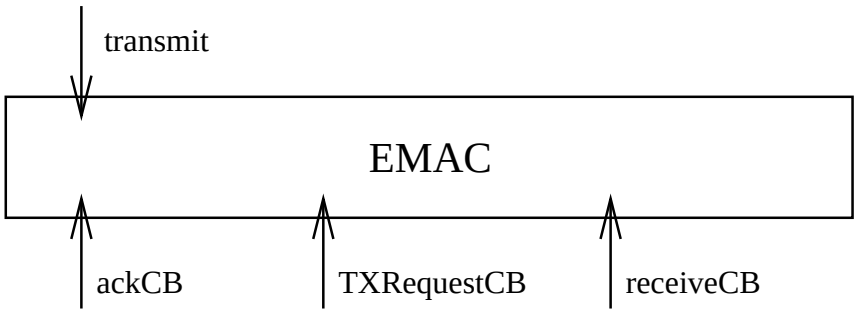

Figure 6.3.: EMAC Interface

Each EMAC module has the same communication interface (s. figure 6.3) that consists of the `transmit` method to hand over a packet to a EMAC module for transmission, and three optional callbacks – `ackCB` that will be called by the lower layer to acknowledge a transmission, `TXRequestCB` to ask for a packet to transmit, and `receiveCB` if the lower layer wants to hand over a received packet. All callbacks are optional, that means they are not required to be defined and an EMAC module first has to register them with the next lower layer. The routing layer will register with the top most layer. That results in a chain of EMAC modules that can be reconfigured at any time during runtime.

In addition to the communication interface an EMAC module can have a configuration interface that depends on the actual task of the module, e.g. the bandwidth limit of the traffic shaper, or the size of the queues.

6.4.3. Communication Interface / Routing API

The routing API (RAPI) provides various packet-based communication primitives and object-oriented communication end-points. Additionally it defines the possible addressing schemes and their semantics that have to be fulfilled by the actual routing implementation. It is intended to be used by a wide range of applications, not only by the P/S layer.

Routing header structure

Figure 6.4 depicts the default layout of the routing header that is present in every packet. It is the default layout because the `Source`, `Service`, and `ServiceKey` fields can be configured to an arbitrary length. The `AddressType` defines the addressing scheme to be used. This will be detailed in the following. The actual length of the `Address Data` area is stored in the `AddressLength` field. `Source` specifies the sender of the packet, `Service` and `ServiceKey` the service that this data belongs to (e.g. the *beaconing* or the *P/S* service). The `SeqNumber` is a unique number assigned to every transmitted packet. The `UserDataSize` specifies the length of the payload. After this set of fixed size fields we have the address area that can be of different size. It is followed by an array of `RecPathLen` element size that is used

to store the path a packet is actually forwarded through the network. It can have a size of zero elements.

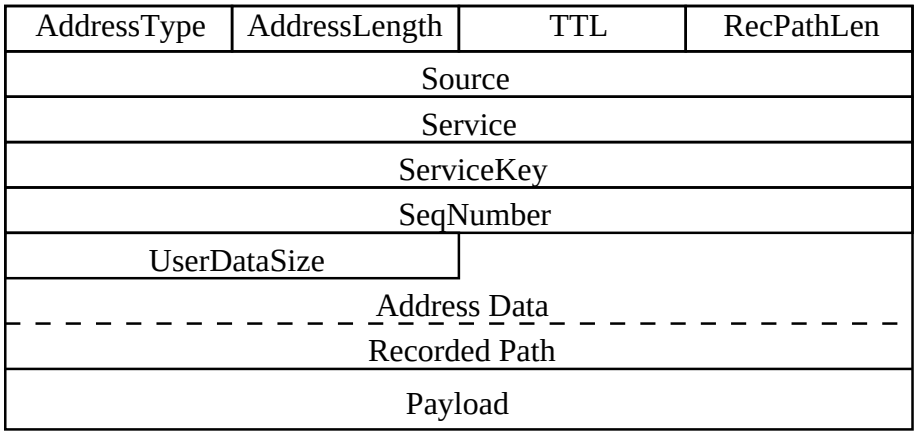

Figure 6.4.: Routing header structure

Addressing schemes

The RAPI defines two mandatory routing primitives (`Flooding`, `SourceRouting`), two optional routing primitives (`DynamicRouting`, `FlowRouting`), as well as three routing flags (`Neighborhood`, `Strict`, `Multicast`) that give additional information about the intended forwarding.

`Flooding` is used to reach any possible node in the network, the address field is empty in this case. `SourceRouting` is used when the sender already knows a path to the destination which will be stored in the address area. `DynamicRouting` requires to specify a destination in the address field and leaves the forwarding decisions to the routing implementation. Usually it will use a route discovery service to find routes and a routing table to store them. `FlowRouting` is used for periodic QoS traffic on multicast trees. The address field contains a unique `FlowID` that can be used to query the next hop from a `FlowTable` that has to be provided to the routing model via the object repository. The creation and maintenance of the table entries has to be done externally, i.e. the routing module has not to care about it. If a routing module does not implement one of these optional primitives it can substitute it using `Flooding`. The RAPI module at the target node will filter the received packets and drop needless ones.

The routing flags give additional information that the routing implementation can use. `Multicast` specifies that the packet is addressed to multiple receivers – the address field will contain multiple entries of the basic addressing scheme. The routing implementation can split the packet into multiple basic packets, or just flood it through the network if it cannot handle it in an optimized way. `Strict` tells that the routing implementation should not apply any optimizations, i.e. if we use `SourceRouting` and a single hop of the path is not possible it should not try to perform a local repair or something else. `Neighborhood` tells the routing that the packet is

only useful in the neighborhood (however that will be defined). Usually this will be equivalent to a 1-hop flooding, but a position based routing implementation could for instance limit forwarding to nodes in the same room.

Communication end-points

All communication end-points are one-way, i.e. they are either used to send (`Server`) or to receive data (`Client`). Each end-point has two queues, one to store ready-to-send or received packets, and a second to receive management and status information (a `Signal`). An application can register a notification callback on the queues to get notified about new elements or just poll it. A Signal can carry a reference to an arbitrary object that includes the actual information. This can also be used to request an answer from the receiver – this will be given using a method call of the information object.

A `Client` is created using the API function `newClient` and can be used in three different modes – *connected*, *broadcast*, or *promiscuous*. A connected client receives data only from a single server. It is connect by calling the `connect` method with a target address given as a `StationID:ServiceID` pair. The target will be asked to accept the connection on the service and can reject it. But this should not be confused with a TCP-style connection. This interface does not define an acknowledgement protocol or something alike. The routing layer is required to perform a monitoring of the connection and to inform both sides if it breaks. It is not required to actively maintain it. A client is switched to the broadcast mode using the method `enable_broadcast`. It will now receive all packets addressed to a certain `ServiceID` regardless where they come from as long as the local node is included in the specified destination set. To receive all packets that the local node sees (even its own packets) a client can be switched to promiscuous mode using `enable_promiscuous`. The reception of packets can be disabled using `disconnect`, `disable_broadcast`, and `disable_promiscuous`. To read packets from the queue an application has to call `BasePacket *next()` that will return the next received packet or zero if the queue is empty.

At the other side we have three different server class – `Server`, an end-point to be connected with a `Client`, the `BroadcastServer` to send all kinds of broadcast packets, and `RawPacketServer` to create arbitrary packets. These different server classes allow for more or less flexibility. The standard `Server` class performs the whole packet initialization and addressing automatically whereas the `RawPacketServer` provides only bare packets that have to be initialized completely be the application. All server objects have two basic functions – `BasePacket *next()` to select the next packet for transmission, and
`void send(BasePacket*)` to mark it read-to-send.

To allow a fine-grained configuration of the internal functionality and a dynamic reconfiguration the communication interface defines an extension mechanism. All active processes like route discovery, bandwidth reservation, connection handling are implemented as *Services*. The base class for all services is `NetworkService`. Each service is assigned a globally unique `ServiceID`.

When a service is registered it will receive all packets addressed for its `ServiceID` for processing. The route discovery service for instance will store its information in a central routing table that will be used by the routing module to determine paths to a destination. All three components (the routing module, the route discovery service, and the routing table) can be chosen individually for maximum flexibility. It is even possible to replace them at run-time or to run multiple route discovery services that update the same routing table.

As an example, in the current system we have three independent sources for route information – path information stored in flooded packets (gathered by a passive monitoring service), the P/S tree construction and maintenance service, and a periodic network discovery service to route best-effort traffic. All store their information in a central routing table to utilize as much information from received packets as possible.

6.4.4. Publish/Subscribe API

The P/S API is very simple compared to the RAPI. It provides two basic functions:

- `Publisher *Publish(ServiceDescription &service)` to create a publisher
- `Subscriber *Subscribe(ServiceDescription &service)` to create a subscriber

Both functions require a `ServiceDescription` that includes the subject description and a QoS specification. The returned objects (`Publisher`, `Subscriber`) function as the interface objects for the applications for the actual communication. The `ServiceDescription` can be initialized using a XML-encoded description or by setting all values in machine-readable format. This interface is provided to be able to avoid the dependency on a XML parser (s. requirements in section 2.4.2). The subject is stored in a `SubjectKey`, a machine-readable unique identifier. If the subject is specified in the XML document, it is converted using a hash function. The `QoSSpecification` is an array of key-value pairs that describe the QoS properties. The supported QoS properties are listed in table 6.1.

Name	**Type**	**Scale Unit**	**Meaning**
Period	integer	μs	interval between two packets
maxSize	integer	byte	maximum size of the payload per packet
maxDelay	integer	μs	maximum allowed end-to-end delay
maxBurstLoss	integer	packet	maximum acceptable number of consecutively lost events
minHopQuality	integer	‰	minimum link quality per hop
applicationQoS	string		opaque container for application QoS

Table 6.1.: Supported QoS properties

`Period` and `maxSize` are mandatory parameters as they are required to compute the bandwidth allocation. `maxBurstLoss` is also a mandatory parameters but has a default value of 3. `maxBurstLoss`, `minHopQuality`, and `maxDelay` are controlled by monitoring the flows and neighborhood. The last two are additionally considered during the tree construction (s. section 5.8). The `applicationQoS` parameter is a special parameter. It includes arbitrary data that specify application-specific QoS parameters that do not have a meaning on this layer. They will only be used during the peer matching process (s. section 5.4).

Both, `Subscriber` and `Publisher` objects have two queues to store events and signals (s. appendix A.2.3). The objects additionally store the `FlowID` of the event flow they use. When a `Publisher` is created, it automatically is assigned a default `FlowID` that is associated with the `SubjectKey` of the `ServiceDescription`. If a publisher is able to support different application QoS profiles it has to allocated additional `FlowID` using `FlowID allocFlow(QoSSpecification&)`. The function returns a new `FlowID` and associates it with the `QoSSpecification`. As the application part of the `QoSSpecification` is meaningless for the middleware, publishers that provide multiple application QoS profiles have to assist in the search process. If the node receives a `SearchPublisher` message for a subject that belongs to a publisher with multiple profiles, it will send a `SelectProfile` signal to the publisher. This signal requires an answer from the publisher. Therefore the signal includes a reference to an `SPAnswer` object. The publisher gives its answer by calling `select(FlowID)` on the answer object. This will cause the P/S management to send the appropriate `FoundPublisher` message. As the whole search process runs without time constraints the publishers is free to answer the signal at any time. But it is encouraged to answer as soon as possible.

An application does not need to end a subscription explicitly, this will automatically be done if the `Subscriber` object is destroyed. The same holds for the `Publisher` object.

6.5. Monitoring API

The monitoring subsystem is basically a named repository of dynamically allocated, global variables. It includes five elements – probes that gather data, `MonitorObjects` that represent the shared variables, the monitoring repository, observers that bind actions to state changes, and import/export converters.

The monitoring system includes no default probe implementation as this is very task specific. Probes, i.e. the code that determines values, can be hard wired into the code or added on demand. That means, if we know that a certain attribute or state change of a component will be of interest for another component we can introduce the probe code. An efficient way to do this is *Aspect-Oriented Programming* (AOP) that allows to separate the component code from the probe code. So we will not have the monitoring overhead if not required. Detailed examples for this can be found in [MSSP02, Mah00].

A `MonitorObject` encapsulates the access to the variable value. It stores the actual address,

the type, and a unique identifier. A static type checking is performed on each read or write access using template programming. This makes it a good target for compiler optimizations – the actual costs of accessing a `MonitorObject` are the same as for access to a pointer-addressed variable. For production systems it is also possible to remove the type checks using a static configuration. This ensures an efficient, resource saving code. Compared to a pointer-addressed variable, the only overhead is the check if a notification of the state change has to be produced. In the worst case (if we do not have any observer) this wastes a simple comparison per write access which is acceptable and sufficient for most cases. If not, it can be further optimized under certain conditions which is detailed in [Mah01]. A `MonitorObject` can currently handle variables of type `Integer`, `Float`, `String`, and `Event`. `Event` is a special type that is used to exchange structured data with other components – a simple, local P/S system.

The `Monitor` repository is the central management point for the monitoring subsystem. It provides the following methods:

- `MonitorObject *createExclusiveObject(key, valuetype)` to create a new `MonitorObject` or to report an error if it already exists,
- `MonitorObject *createSharedObject(key, valuetype)` to create a new `MonitorObject` or to return a pointer to the existing one,
- `void freeObject(MonitorObject *object)` to remove a `MonitorObject`; all attached observers will be noticed and deattached,
- `MonitorObject *findObject(key)` to find a `MonitorObject` by its key,
- `MOObserver *observeObject(key)` to attach an observer to a `MonitorObject` identified by its key.

If an observer is attached to a `MonitorObject` it will be notified of any state change (write or object destruction) and can react at will. The import/export functions are intended to allow an exchange of monitored values with other nodes or to ease the debugging process. The monitor repository provides a list of all monitored variables and creates a textual output on demand. Other conversion functions (e.g. to/from XML) can be added to the repository or as observers to individual variables on demand.

6.6. Legacy Application Integration

One of the major problems for every newly developed communication middleware are legacy applications. More generally spoken, we have a typical chicken-egg-problem – without applications using the new system it will not attract new users and without new users it will not of general interest for developers to use. Within this thesis the problem was to integrate existing applications into the system without the need for modifications. The solution includes two parts:

the transparent redirection of traffic through the middleware and the remote control of the middleware. On UNIX-like system, applications usually send messages using a network device (e.g. `wlan0`) that represents a physical device in the user space. All packets send are handled by the network stack and transmitted using a physical device that is controlled by the device driver. We now introduce a special bridge device that is able to connect different devices like a software switch [cita]. We create a virtual TUN/TAP device (`gea0`) [citb] and attach it to the bridge. The bridge interface of the middleware now relays all packets coming from the bridge or that are addressed for the local node. It therefor uses the raw Ethernet frames captured from the bridge device and encapsulates them in the payload of regular packets. The middleware itself uses the actual WLAN interface like an ordinary application and provides its additional service to the legacy application in a transparent way.

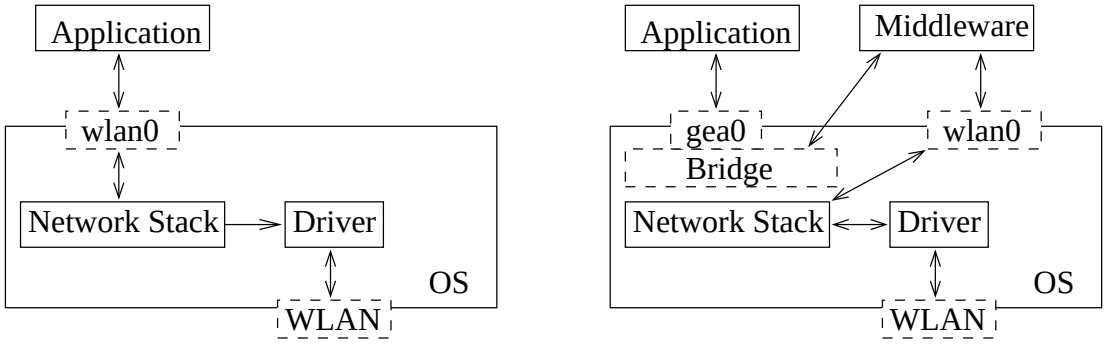

Figure 6.5.: Traffic redirection using bridge device

If we like to use or control the extended capabilities of the middleware (e.g. to reserve a QoS route) we need access to the middleware functions. To allow such an access for legacy applications the bridge module defines a special (freely configurable) UDP port to send RPC requests to. If the bridge module receives such a packet it extracts the calling parameters and executes the middleware function. The result is send back to the originator using an UDP packet.

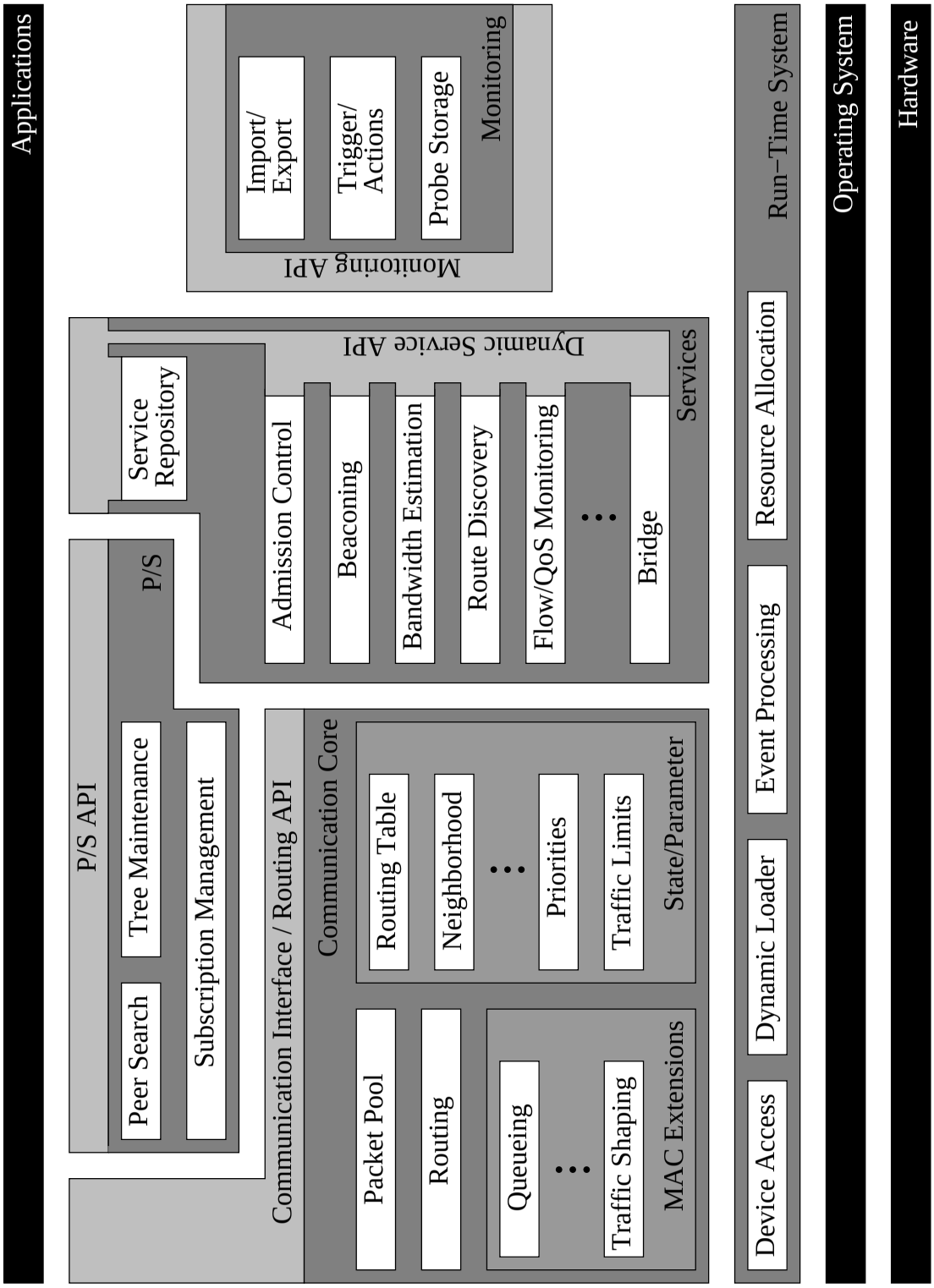

Figure 6.1.: Middleware Component Overview

7. Testing and Evaluation

In this chapter we will evaluate the performance of some key components of the system in different experiments. If possible we will run the experiments in the simulation as well as on a native setup. This will show two different thing – first, if these components work as intended, and second if the simulation is able to reproduce the results. The second point is important because it provides as with an evaluation of the developed WLAN model, and identifies possible sources for divergences that we have to keep in mind for further simulation-based experiments.

7.1. Link Quality Monitoring

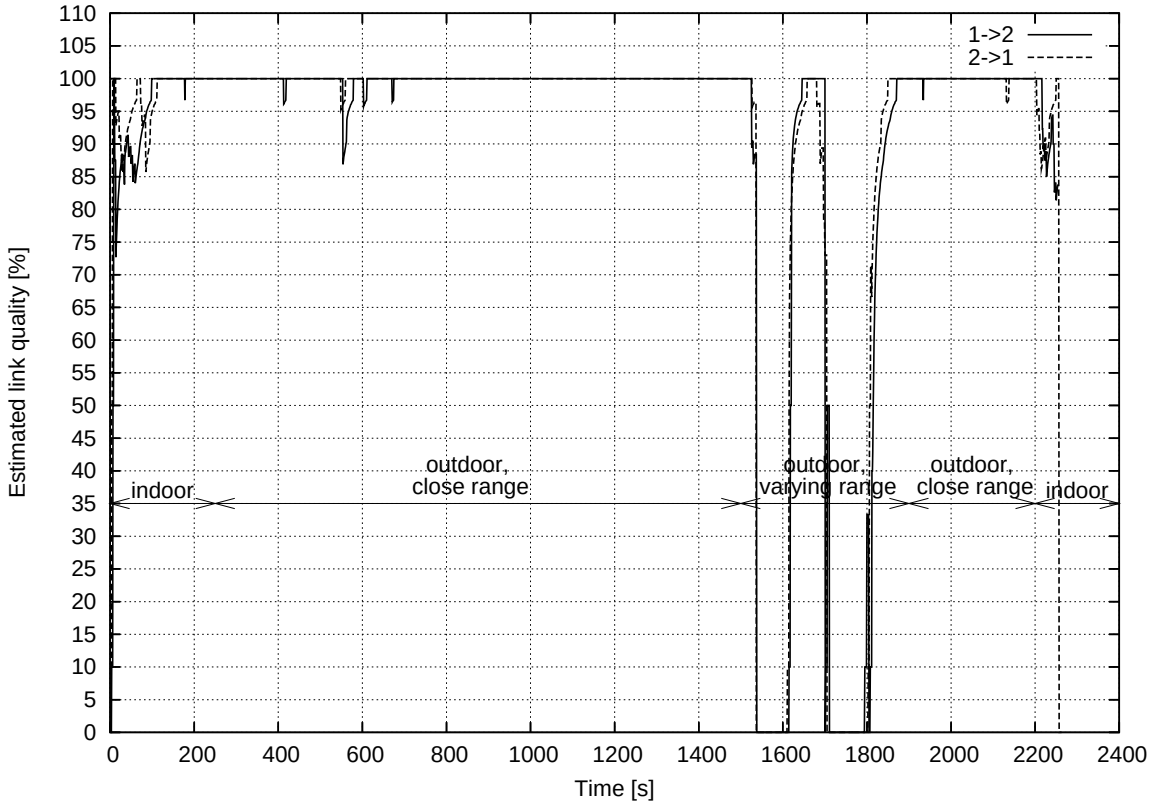

Figure 7.1.: Link quality estimation with two node

The first key element of the system is the link quality monitoring because it defines which links will be chosen to forward packets. In section 4.3.3 we already discussed the link quality under different propagation models. The following experiment will answer the question how serious the problem of asymmetry is in practise. To do this we used the following setup. Two persons, each carrying a laptop in a backpack move around in different environments. Both laptops run the beaconing service with a frequency of 10/s and stored the estimated link quality to a file (once per second). Additionally we run a standard `ping` using a interval time of 0.1 s and packets with 1000 byte payload to measure the round trip time (RTT) and packet loss rate. Figure 7.1 depicts the estimated link quality for both nodes. The experiment run for about 35 minutes – the time spend in the different environments is marked in the diagram. *Outdoor, close range* means that both persons do not moved away from each other more then 20 m – something we would encounter for a small number of people moving as a group. Later this maximum distance has been increased to 200 m – as for people that move independently from each other. Due to the attenuation of the backpacks the maximum transmission range reduced to about 100 m which leads to completely disconnected times. The results show that both nodes estimate comparable values for the link quality. Figure 7.2 depicts the calculated asymmetry. For better readability the estimate link quality from node 2 to 1 has been mirrored on the x-axis.

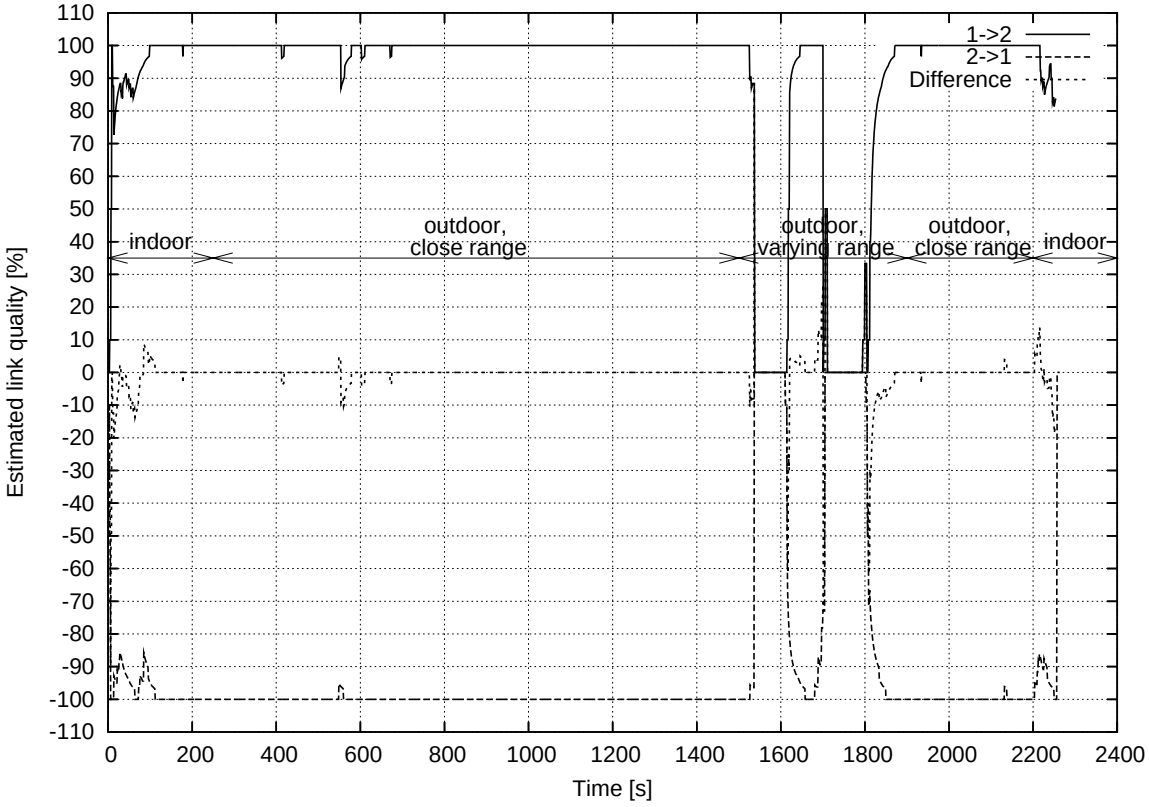

Figure 7.2.: Asymmetry in the link quality estimation

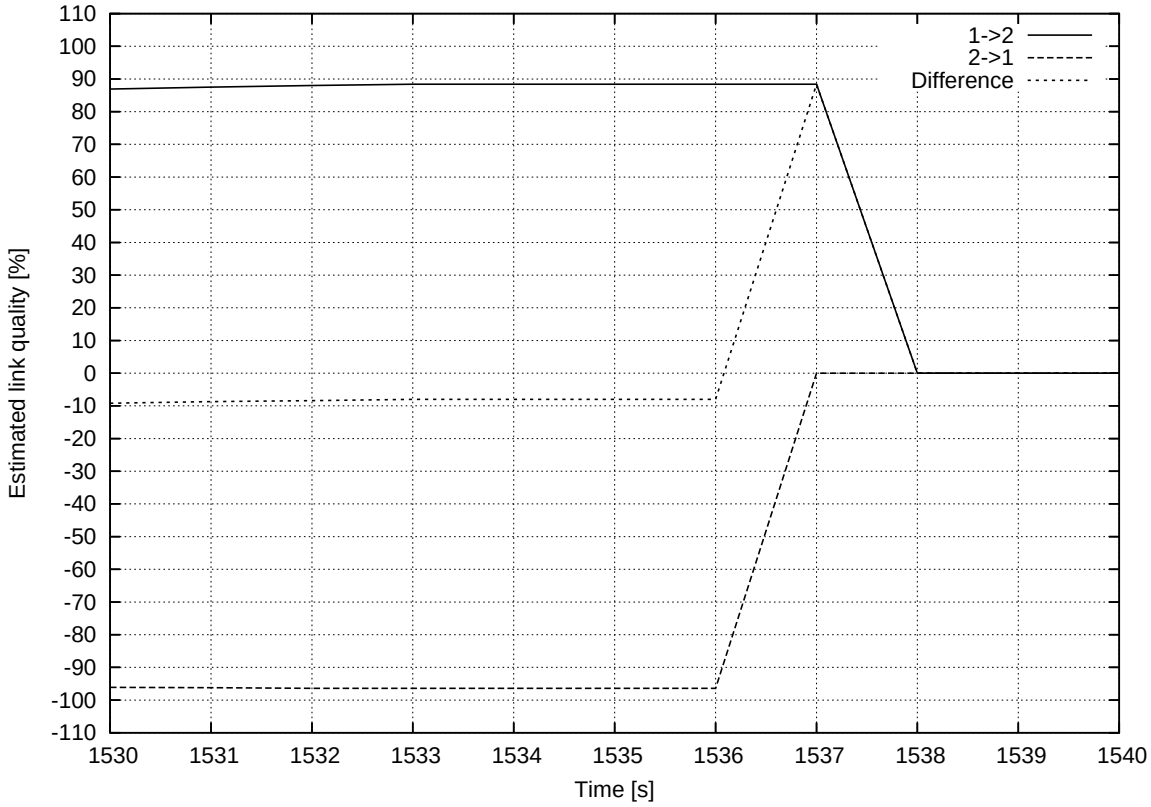

Figure 7.3.: Asymmetry in the link quality estimation (magnified)

Although the estimated link qualities seem to be almost equal in the first place, we see very large asymmetries. The link quality overlay would cope with this but from the discussion in section 4.3.3 this is unexpected. So we have to investigate this further. If we zoom into the chart (s. figure 7.3) we notice that the large asymmetry is the result of a temporal inconsistency. As explained in section 5.6 we cap the possible link quality value to consider link stability and have a maximum number of lost beacons we tolerate before we declare a link to be broken – both can result in an abrupt change of the value. In the depicted case both nodes loss the connection (the value drops from about 90% quality to 0%). But node 2 notices it short before time 1573 s and node 1 afterwards. So it reports it with 1 s delay. The same happens for all other cases and so this observation does not conflict with our model.

7.2. Traffic Shaping

The second key element is the local traffic shaper because it significantly affects the medium access. To test the traffic shaper the following experiment has been conducted. We use four laptops, three that will send data and a fourth that is placed between the three to capture all transmitted packets using a WLAN card in *monitor mode*.

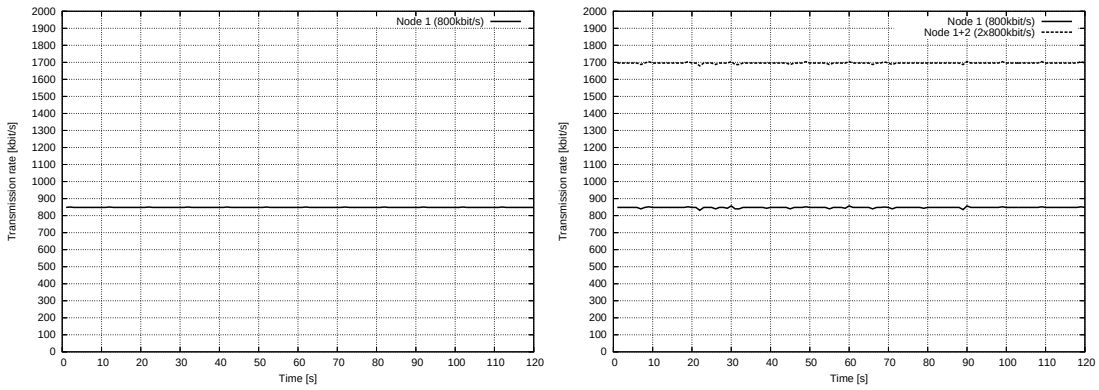

Figure 7.4.: Bandwidth utilization with one and two nodes

For the first test we disable the traffic shaper and use a traffic generator that sends packets of 1000 bytes each (without the WLAN header) every 10ms. This creates traffic at 800 kbit/s per node. For one and two nodes sending at the same time (s. figure 7.4) we see a steady bandwidth utilization. We see that the actual bandwidth utilization on the medium is about 850 kbit/s because this chart includes the WLAN header. Figure 7.5 depicts the transmission interval between the messages send by the nodes (the maximum of all packets in a second is shown). We see that most packets are actually send in the expected interval of 10ms with a very small number of runaways.

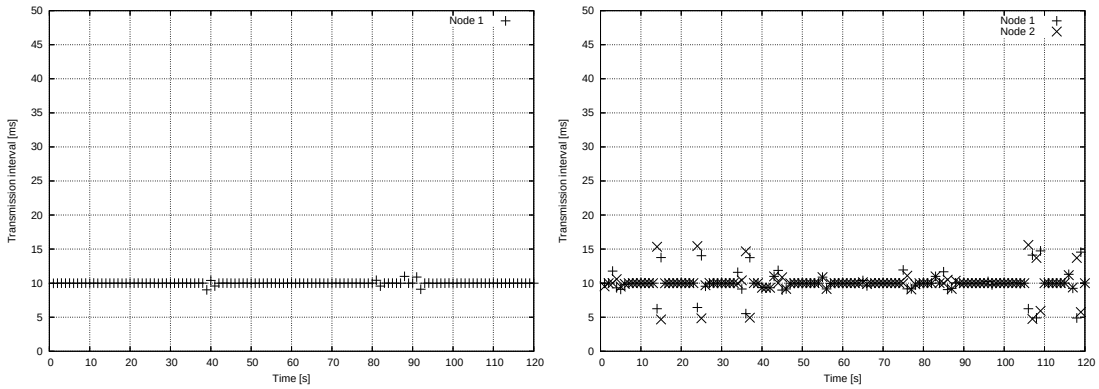

Figure 7.5.: Transmission interval with one and two nodes

When we start a third node we get an overload on the medium. Figure 7.6 depicts the bandwidth utilization and transmission intervals for this case. We see that we get an almost fair bandwidth sharing. But the transmission intervals vary significantly.

We now enable the traffic shaper and limit the bandwidth of node 1 to 800 kbit/s (including the WLAN header), to 400 kbit/s for node 2, and 160 kbit/s for node 3. This time we use the traffic generator to create packets of 1000 bytes size (including the WLAN header) in intervals of 10, 20, and 50 ms. In figure 7.7 we see that each node is able to transmit its packets. We additionally

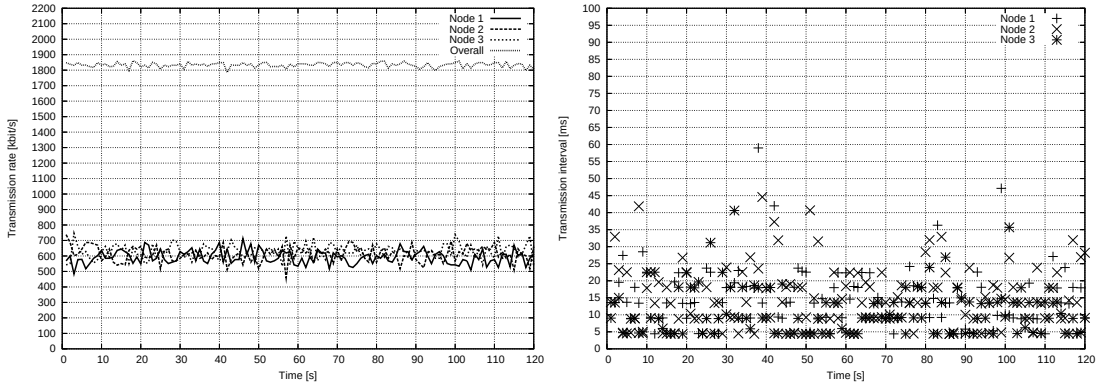

Figure 7.6.: Transmission interval and bandwidth utilization with three nodes

see that the transmission intervals on the medium match that of the traffic generator with a small number of runaways. We see that the jitter is independent from the transmission interval of the traffic generator – in all cases messages are not delayed more than 5 ms.

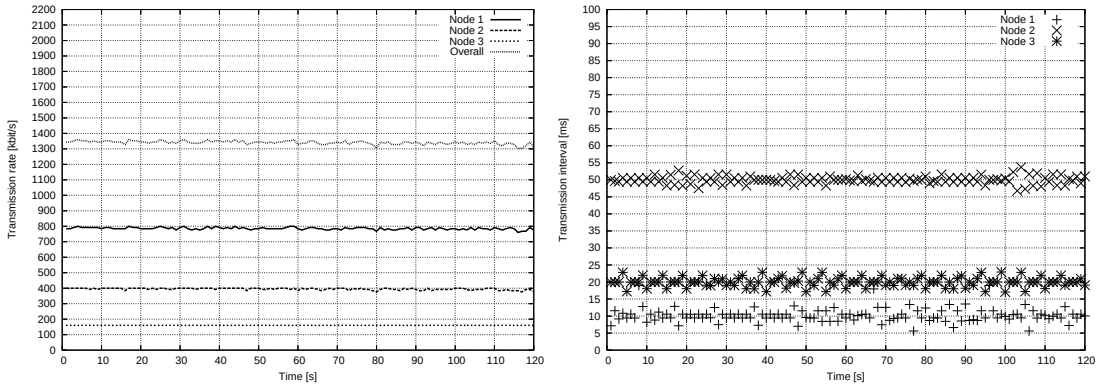

Figure 7.7.: Transmission interval and bandwidth utilization with traffic shaping (1)

As we can see in figure 7.7 we have an overall bandwidth utilization of about 1350 kbit/s and so still have some bandwidth left. We now increase the bandwidth limit for node 2 from 400 to 800 kbit/s and decrease the transmission interval from 20 to 10 ms (the same as for node 1). The results of this setup can be seen in figure 7.8. We notice that node 1 and 2 now perform almost exactly. But compared with the previous experiment we see some significant runaway values in the maximum transmission intervals. An in-deep investigation of the captured packets gives a possible reason. Although we have chosen a supposedly undisturbed environment for the experiment we have received some packets not originating from any of the test machines. This clearly shows that even though the traffic shaping amongst our own nodes works as desired, we have to be prepared for unexpected disturbance.

In the next step we are going to compare our measured results with results from various simulations. In the first simulation we test the fairness of the medium access in the simulation. For this we use different numbers of nodes positioned very close together – to ensure that they affect each

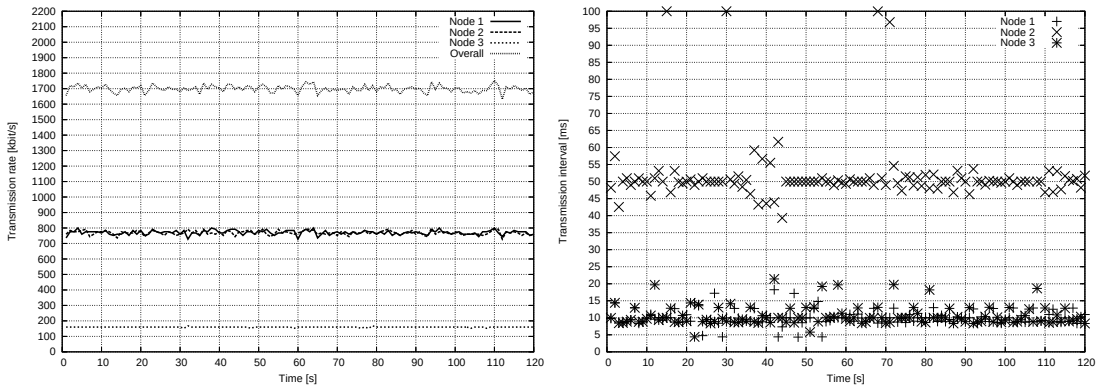

Figure 7.8.: Transmission interval and bandwidth utilization with traffic shaping (2)

other. On each node we ran the same code as in the experiment. The results of this simulation can be seen in figure 7.9.

Next we perform the experiment inside the simulation, that means we setup three node positioned close together, activate the traffic shaper with limits of 800 kbit/s, 400 kbit/s, 160 kbit/s for the first simulation and 800 kbit/s, 800 kbit/s, 160 kbit/s for the second. To exclude effects caused by the synchronized virtual clocks of the nodes inside the simulation the traffic generator creates an artificial inaccuracy of ± 1 ms in the transmission timer. Additionally, to avoid simulation startup artefacts, we ignore the data gathered in the first 30 seconds of the simulation. To compare with the experiment we measure the achieved transmission rate and messages interval. The results are depicted in figures 7.10 and 7.11. We see that the bandwidth utilization is constant over the time. This is not surprising as we do not have any disturbing effects in the simulation. So, the bandwidth utilization results in the simulation are minimal better than the measured results. The delays in the simulation show a similar behavior as in the experiment with two exceptions. First, we do not see extreme runaway values in the simulation because of the missing disturbances and second, we see a higher spread around the expected value. So, the simulation behaves slightly worse in this case. But overall we can summarize that the simulated results match the measured results very well.

7.3. Point-to-Point Communication

The next test compares several performance numbers for single-hop communication. We first compare a native transmission between two Linux hosts with communication over the bridge device to determine the overhead caused by the communication middleware. Afterwards we compare the communication of a simple application running on a native host and inside the simulation. Another example can be found in [HM05].

To measure the overhead caused by the middleware we use two tools, a standard `ping` to measure the transmission round trip time, and `iperf` [ipe] to measure the end-to-end throughput for

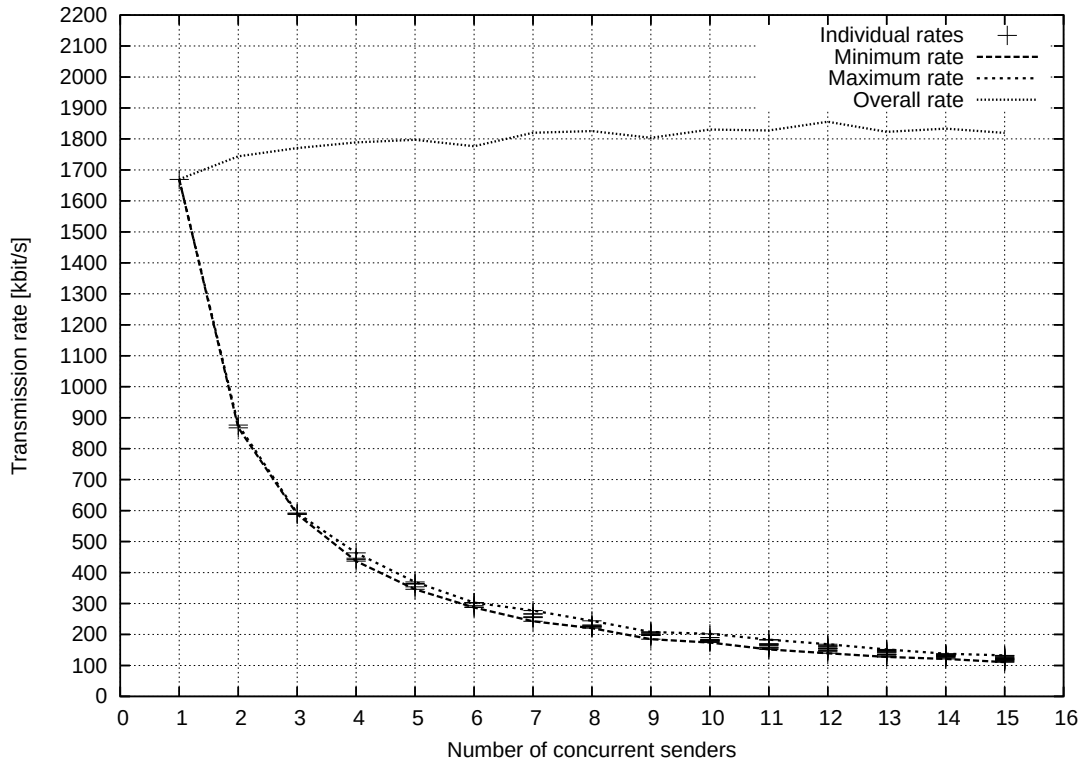

Figure 7.9.: Fairness in the simulated medium access

UDP and TCP. Table 7.1 lists the measured numbers. From the round trip times we see that the middleware introduces an overhead of about 0.6 ms. Most of this time accounts for the additional kernel-to-userspace communication that is need because the middleware runs as a user-level process. That means each packet send using the bridge device is relayed back to userspace where it is processed by the middleware that sends it back to the kernel for actual transmission (s. section 6.6. The throughput numbers are quite interesting. We see that the TCP throughput is 50% higher in the native version. This is not surprising if we consider the transmission delay in both setups which is about 50% higher for the bridge setup. If we calculate the bandwidth-delay product we get about the same value in both cases. But the UDP throughput values are somehow odd because they are significant higher in the bridge setup – and this is not the result of a confusion of the result sets. A reason for this behavior can be found in the technique used to measure the UDP throughput. As we do not get a feedback from the kernel about transmitted packets, or from the other host about received packets (`Iperf` only receives one packet per second containing the transmission statistics), `Iperf` has to guess the transmission speed. In this setup we try to send packets at a rate of 11 Mbit/s. It is possible (but cannot be verified) that some packets are dropped in the sending kernel because they arrive in a burst. In the bridge setup, the middleware adds a queue that would buffer such a burst and so could be the reason for the higher throughput.

The second experiment compares the traffic generated in a simulation and on the real host. For this experiment we use the same code in both cases and count the transmitted packets. The traffic

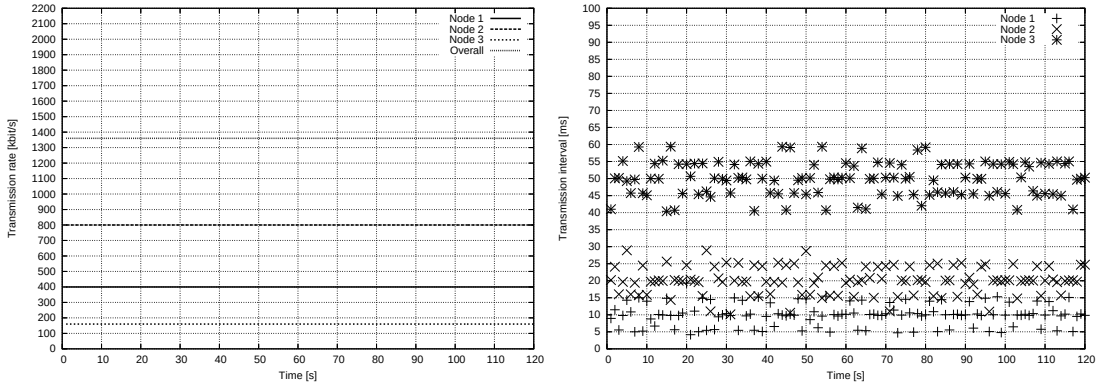

Figure 7.10.: Simulated traffic shaping (800/400/160 kbit/s)

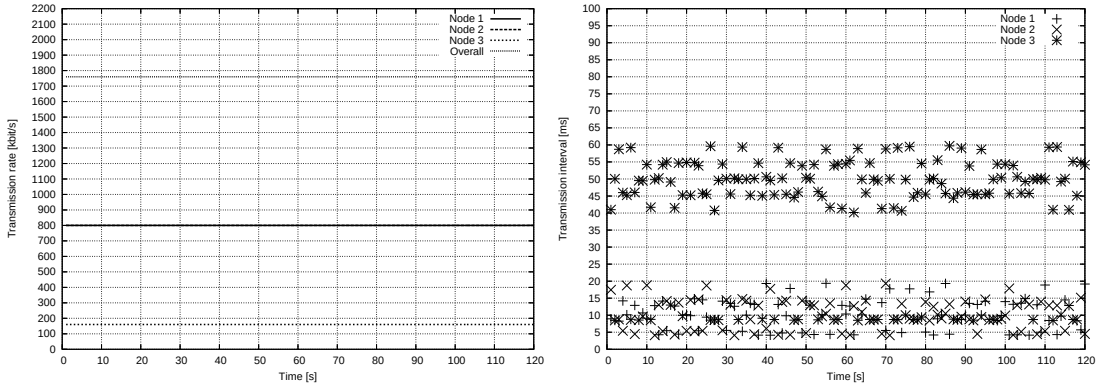

Figure 7.11.: Simulated traffic shaping (800/800/160 kbit/s)

generator tries to send packets of 1000 bytes (including all headers) as fast as possible. Table 7.2 gives the results of this experiment. The first thing that we notice is that the simulation does not include an beacons. Second, we see that the native packet throughput is about 15% higher than the simulated throughput. That means that simulations will produce more conservative results.

7.4. Multi-Hop Communication

The next series of experiments compares the behavior of a the real setup with the simulation for multi-hop communication. For this experiment we use four laptops und try to send packets as fast as possible from the first to the last over the two intermediate hops. All nodes are placed in close range, so they will affect each other. Usually a route will not use the intermediate node, but this test only should serve as a benchmark for the complex interactions of the nodes. Table 7.3 shows the result from this experiment. Again we see that the simulation results in a smaller throughput, but this time only about 10%.

Test		Native	Bridged
Round Trip Time	(min) [ms]	1.207	1.828
	(avg) [ms]	1.341	1.995
	(max) [ms]	1.862	2.881
Round Trip Time (adaptive)	(min) [ms]	1.203	1.822
	(avg) [ms]	1.245	1.893
	(max) [ms]	4.248	4.455
Throughput (TCP)	[Mbit/s]	5.98	3.98
Throughput (UDP)	[Mbit/s]	6.85	9.12

Table 7.1.: Implementation overhead

Packet type	Native		Simulation	
	[pkt/s]	[kbit/s]	[pkt/s]	[kbit/s]
Data	750.7	6006	662.4	5299
ACK	749.9	228	662.4	201
Beacon	9.7	7.4	0	0
Overall	1510.3	6241.4	1324.8	5500

Table 7.2.: Comparison of simulated and native unicast throughput

7.5. Idle Time Measurement

In section 5.5.4.3 we discussed how we can dynamically adapt to a degrading channel capacity even if we are not able to perform a coordination with the disturbing node. The main idea is to measure changes in the idle time of the transmitter. The following experiment will demonstrate this effect. Figure 7.12 depicts the simulation test setup. We have two stationary nodes (1,2) both configured with a bandwidth limit of 800 kbit/s (reserved air time of 0.4) that try to transmit packets of different sizes and in different intervals. We have a third node (3) that tries to send 50

Packet type	Native		Simulation	
	[pkt/s]	[kbit/s]	[pkt/s]	[kbit/s]
Data $1 \rightarrow 2$	293.3	2402	270.1	2213
Data $2 \rightarrow 3$	288.0	2359	253.1	2073
Data $3 \rightarrow 4$	261.8	2144	245.6	2012
ACK $1 \leftarrow 2$	286.1	87	243.6	74
ACK $2 \leftarrow 3$	286.0	87	238.9	73
ACK $3 \leftarrow 4$	282.5	86	238.2	73
Beacon	9.8	7.5	0	0
Overall	1707.5	7172.5	1489.5	6518

Table 7.3.: Comparison of simulated and native multi-hop throughput

packets of 1000 bytes per second. This node starts in a distance of 500 m and moves toward the first two nodes. The simulation uses the parameter set for an open park environment (shadowing model {3.6, 4}, s. appendix A.1 for details) because here we have the largest transmission-to-interference range and hence a far reaching disturbance (s. sections 4.7.4, 4.3.2 for details).

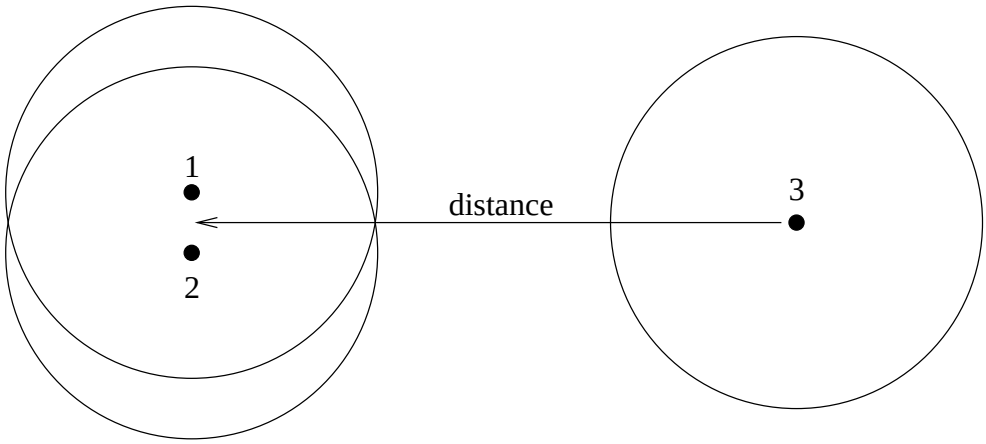

Figure 7.12.: Experiment configuration: transmitter idle time

Figure 7.13 depicts the results from the first experiment. In this experiment we use 2 fully synchronized flows (both have the same period and transmit at the same time, which is almost impossible in practise) of 100 packets of 1000 bytes per second for the nodes 1 and 2. The chart shows the relative idle time, that is the time the transmitter is idle divided by the time the temporal lock of the traffic shaper is closed. We see that without the disturber (or with it in a long distance) we get an average idle time of about 33%. In a distance of about 350m we start to see a reduction of the idle time that reaches zero in a distance of about 220m. If we compare it with figure 4.11 (that was created using the same propagation parameters) we notice that we have only a 10% probability to receive the beacon from node 3 at this point. If the node comes closer we now get a chance to coordinate with it. But we see that the idle time measurement provides us with a much earlier warning about a degrading channel capacity. If node 3 behaves cooperatively it will notice the presence of the nodes 1 and 2 too because of its own idle time measurement and will reduce its own bandwidth utilization until it is able to communicate/coordinate directly.

Figure 7.14 shows the results for the same experiment but with a constant offset between the transmitters. That means node 1 always tries to transmit its packet before node 2. We see that we get a very similar result but that the idle time of node 1 (left) is higher than of node 2 (right). This is not surprising as node 2 has to wait for the transmission of node 1 in all cases because of the time offset.

So far we used only flows with the same period and without any jitter. Figure 7.15 shows the results for flows with different periods and if we consider jitter. On the left we have the result for two harmonic flows of different sizes (node 1: 1000 bytes every 10ms, node 2: 2000 bytes every 20ms). On the right we see the result if we consider unsynchronized, non-harmonic flows and jitter (node 1: 100 bytes every 10 ms $\pm$5ms, node 2: 500 bytes every 7ms $\pm$ 3.5ms). A packet i

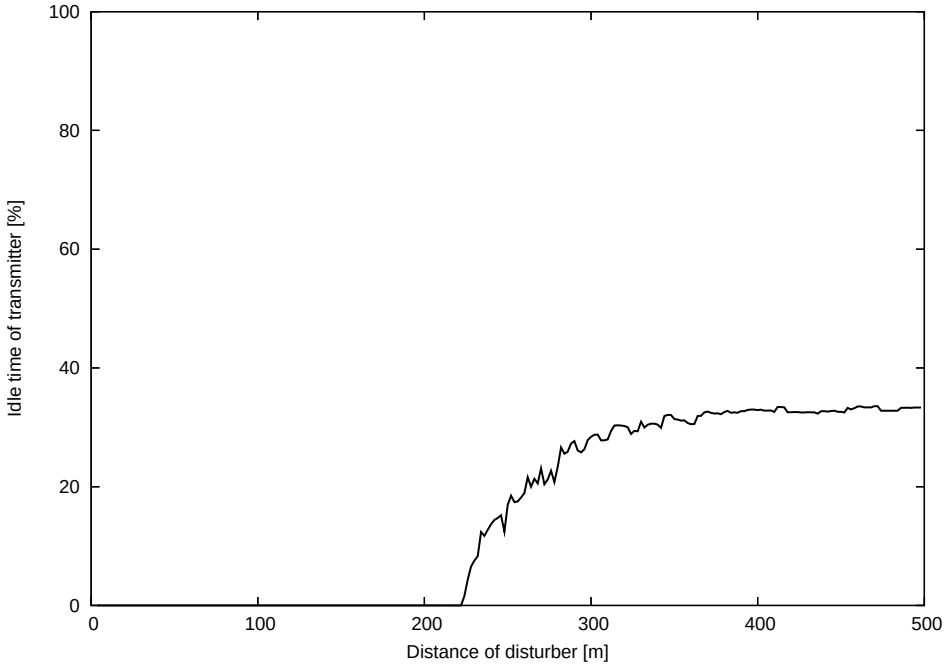

Figure 7.13.: Impact of disturbing node on transmitter idle time average (1)

of a flow with period t_p is now send in the interval $[t_0 + i \cdot t_p - \frac{t_p}{2}, t_0 + i \cdot t_p + \frac{t_p}{2}]$. We see that the last chart looks very similar to the first (figure 7.13) – we have the same maximum idle time and the same distance where it reaches zero. In the case of harmonic flows without jitter we see a similar behavior as on the left in figure 7.14 because if the disturbing node is near enough we have a scenario with three flows where node 1 always get access to the medium first. The high idle time value should not distract from the fact that we now reached an almost saturated channel – the idle times of node 2 are significant lower and around zero for node 3 as it gets access to the channel last.

Overall we see that the idle time measurement method provides an early-warning mechanism to detect a degrading channel capacity. But we also see that from the pure idle time value of a single node we cannot derive any information about the bandwidth utilization of the disturbing node. It remains an open question if it is possible to derive some more information from the combination of idle time information of the nodes in the neighborhood.

7.6. Route Discovery

In this experiment we will measure the time required to discover a path using the link overlay flooding. The measurement provides us with two information – the minimum time for the search and the link qualities along the discovered route. We will perform the experiment in the following way. First we create a random topology consisting of 50 nodes spread across an area

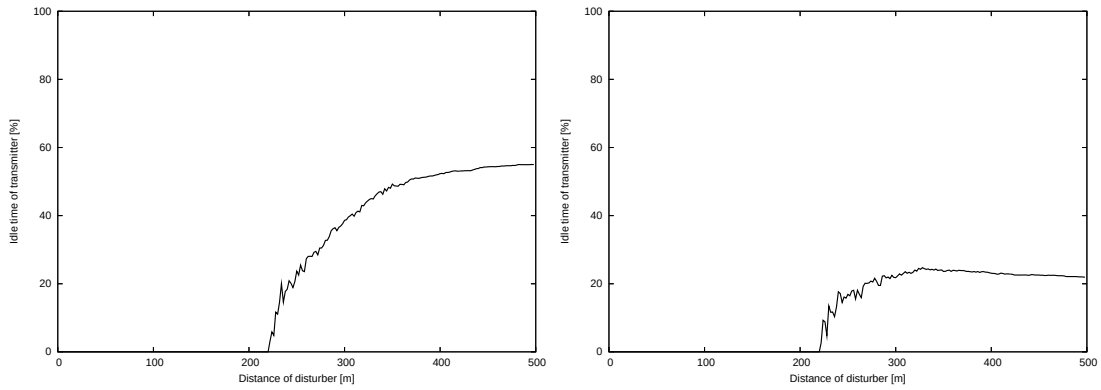

Figure 7.14.: Impact of disturbing node on transmitter idle time average (2)

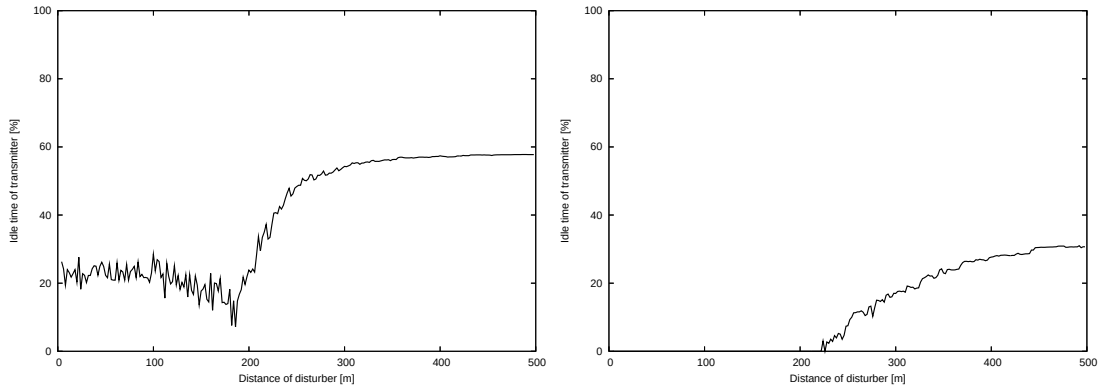

Figure 7.15.: Impact of disturbing node on transmitter idle time average (3)

of 1000x1000m. For each random topology we randomly select 15 unique pairs of nodes and start the discovery and bandwidth pre-allocation process with 1 second interval. We repeat the experiment for three different forward thresholds (0.75, 0.80, 0.90) each with the same topology and node pairs as input. To create comparable scenarios we deactivate the accounting of unicast transmissions on the link quality values. This way we avoid to change the link qualities for subsequent searches with the packets used in the previous search.

The results of the experiment are depicted in the tables 7.4, 7.5, and 7.6. Each table lists the results of the individual tests (topologies). They are not meant as a statistical evaluation, just to show some expected and some unexpected results. The given times provide a lower bound for the route discovery process we performed this test without any application traffic, i.e. only had the beacon and search traffic.

A quick look on the results clearly shows that the link quality overlay flooding functions as expected – we do not select any links that are below the set threshold. We also see that the search times show a wide range even for the same route length, e.g. 3–37ms for a single hop and up to about 125ms per hop if we calculate the per hop times. This is not surprising as the search requests are send with low priority. If we look on the successful search requests (those that

Test	successful	time (min/max)	link quality (min/avg/max)	route length (min/max)
1	8	0.014 / 0.212	750 / 949 / 1000	1 / 4
2	2	0.006 / 0.145	777 / 900 / 1000	1 / 3
3	6	0.005 / 0.339	842 / 977 / 1000	1 / 10
4	4	0.036 / 0.161	800 / 926 / 1000	1 / 5
5	11	0.031 / 0.516	769 / 934 / 1000	1 / 10
6	8	0.024 / 0.408	764 / 953 / 1000	1 / 6
7	3	0.144 / 0.491	800 / 950 / 1000	2 / 6
8	7	0.038 / 0.381	842 / 962 / 1000	2 / 11
9	6	0.037 / 0.510	764 / 937 / 1000	1 / 8
10	6	0.028 / 0.196	846 / 967 / 1000	1 / 5

Table 7.4.: Route discovery time, success rate, and link qualities; $t_{fwd} = 0.75$

Test	successful	time (min/max)	link quality (min/avg/max)	route length (min/max)
1	8	0.006 / 0.144	812 / 959 / 1000	1 / 4
2	1	0.037 / 0.0370	1000 / 1000 / 1000	1 / 1
3	8	0.037 / 0.384	823 / 967 / 1000	1 / 12
4	3	0.058 / 0.136	866 / 944 / 1000	2 / 5
5	3	0.077 / 0.301	857 / 969 / 1000	4 / 8
6	3	0.004 / 0.037	818 / 921 / 1000	1 / 4
7	4	0.114 / 0.372	818 / 970 / 1000	4 / 12
8	8	0.041 / 0.587	812 / 970 / 1000	2 / 11
9	4	0.0122 / 0.495	875 / 965 / 1000	1 / 4
10	4	0.0161 / 0.215	666 / 942 / 1000	1 / 5

Table 7.5.: Route discovery time, success rate, and link qualities; $t_{fwd} = 0.80$

Test	successful	time (min/max)	link quality (min/avg/max)	route length (min/max)
1	6	0.007 / 0.410	909 / 959 / 1000	1 / 6
2	1	0.023 / 0.023	1000 / 1000 / 1000	2 / 2
3	4	0.003 / 0.130	1000 / 1000 / 1000	1 / 4
4	1	0.056 / 0.056	933 / 966 / 1000	2 / 2
5	2	0.058 / 0.059	1000 / 1000 / 1000	4 / 4
6	2	0.013 / 0.378	916 / 978 / 1000	2 / 4
7	3	0.105 / 0.372	923 / 989 / 1000	4 / 5
8	5	0.040 / 0.148	909 / 977 / 1000	2 / 5
9	3	0.077 / 0.432	909 / 987 / 1000	2 / 7
10	2	0.012 / 0.056	923 / 961 / 1000	1 / 3

Table 7.6.: Route discovery time, success rate, and link qualities; $t_{fwd} = 0.90$

found a route and transmitted the answer back to the originator) we see that an increased forward threshold (f_{fwd}) results in a lower number, with three exceptions. If we compare the results from the tests 3,7,8 for forward thresholds of 0.75 and 0.80 we see that the higher thresholds results in a higher number of successful route searches. This can be explained as follows. If we use the lower threshold we have a slightly higher chance to loose the answer packet on the way back to the originator which in turn can lead to a reduced number of successful searches. We see that we get a trade-off. On the one hand we will reduce the number of possible paths that will be selected while choosing a higher threshold. On the other hand, choosing a too low threshold increases the risk to loose the answer packet. In practice this is not a serious problem as we will retry the search if we do not get an answer. That means we are free to choose a threshold as low as required to fulfil our reliability demands. A look on the path lengths shows that we also create longer routes for the higher threshold. This is an expected behavior as the number of possible links to build routes decreases with a higher forward threshold.

8. Conclusions and Future Work

The proliferation of mobile devices and growing use of wireless communication for mobile and stationary equipment gives rise to the ad hoc networking and interaction amongst them. The growing experiences with mobile devices and interest in mobile ad hoc networks (MANETs) causes an increased desire to explore the newly gained possibility while retaining communication services that we are used to. Currently wireless communication still relies on a wired backbone and provides only a solution for the “last mile”. But the ultimate goal is to become independent from this wired backbone. To allow this a wireless network has to be able to provide communication services that provide certain quality of service guarantees. Furthermore, to support and easy ad hoc collaboration of devices and hence people, new interaction schemes like publish/subscribe communication have to be explored. Within this thesis a middleware has been developed that provides a subject-based publish/subscribe communication with QoS guarantees – primarily end-to-end throughput, secondary delay and reliability. This thesis mainly contributes to the solution of the following challenges:

- A WLAN communication system model that allows to describe a variety of effects and problems related to the wireless transmission of information, most notable gradual interference and receive probability in different environments. An implementation in a simulation model, as well as a method to setup such a simulation based on actual measurements in a concrete environment are provided.
- The establishment of reliable communication links that provide communication with guaranteed throughput on the shared wireless medium in presents of concurrent senders and gradual link qualities based on the measured environmental conditions. The bandwidth coordination scheme incorporates coordination with nodes in communication range as well as the detection and adaption to interfering nodes. Both aspects, reliable links and the knowledge about interference are a prerequisite for many other communication protocols in MANETs.

The discussion of the developed model for the wireless communication systems showed that we have to consider a wide range of effects that influence the propagation of information. We learned that most of these effects are often impossible to model because it requires a lot of information about the environment that are usually not available. Some effects are known but still not fully understood by communication engineers. But we have also seen that although it is almost impossible to model a concrete environment, we are able to reproduce most effects experienced in a certain type of environment like an office building, or an open urban area. We

saw that we cannot rely on a uniform signal propagation and have to consider the fact that packet receive probability gradually changes with the distance between two nodes. The most important consequence of these observations is that we can ultimately trust only the connectivity that we measure in an actual setup. We furthermore saw that it is essential to consider the link quality to build routes that provide us with a minimum end-to-end throughput.

As a consequence of these observations we introduced a link quality measurement that provides the input for the definition of our neighborhood and the selection of links to neighbors. Upon this neighborhood we introduced a cooperative bandwidth coordination scheme that allows for an arbitrary distribution of the available bandwidth to individual nodes in a neighborhood. This coordination scheme bases on the 2-hop neighborhood discovered by a beacon service. Each individual node enforces the limitation of its allocated bandwidth using a local traffic shaper that is supplemented by a local traffic prioritization to favor QoS over best-effort traffic. We saw that due to the probabilistic medium access of the IEEE 802.11 MAC layer we have almost no control over the order in which nodes access the medium and hence the distribution of the delay of concurrent transmissions. But we recognized that the delay is upper bounded in most cases as long as we do not fully saturate the channel capacity – so the developed system is definitely not suited for hard real-time requirements, but will be sufficient for soft real-time systems like multimedia streaming that can tolerate jitter and individual deadline misses or lost packets.

One oft the most serious problems that we encounter are interfering nodes that are out of communication range. This case is commonly neglected in current research. The presented approach for this problem uses an online monitoring of the average input queue length and idle time of the traffic shaper. Both values provide an indication of the over/under utilization of the medium. For every transmission we calculate a number of retransmissions, if they are not uses, the actual channel utilization we be smaller than expected. This is a common case and will be exploited to transmit best-effort data. In the opposite, if we have interfering nodes that are not considered in the bandwidth planning, we will notice an over utilization and can adapt dynamically to it. We saw that as long as we do not have other ways to communicate and hence coordinate with interfering nodes we cannot forgo a dynamic adaptation.

Based on the link quality overlay we presented a protocol to construct and maintain a multicast tree that provides end-to-end QoS properties. This multicast tree is the underlying transport mechanism to disseminate events in the developed publish/subscribe system. But it can be used independently. Theoretical results from other works clearly show that it is essential to limit the length of communication paths to build scalable wireless networks. The event dissemination is design so that every node included in the underlying multicast tree can be the source of a new branch. With a growing number of subscribers to a publisher that means an increasing chance to find a node in short range that can be used as the source of the desired events. This supports the construction of a scalable P/S system.

The publish/subscribe protocol uses a subject-based approach to increase predictability that is essential to provide QoS guarantees, and to reduce the processing overhead on all nodes. The system distinguishes between communication QoS (bandwidth, delay, loss rate) and application QoS (e.g. coding schemes for multimedia data, sensory precision). Application QoS is handled

solely by the publishers and subscribers, and is treated as an opaque QoS property for the P/S system itself. It is mapped onto separated multicast trees that are rooted at the publisher nodes. Intermediate nodes therefore have no additional overhead for the dissemination of events compared with other multicast transmissions. This is essential as every processing overhead would increase the resulting end-to-end delay.

In addition to the function requirements we had a number of non-functional requirements on the developed middleware, most notably portability, efficiency, configurability, and extensibility. To fulfill these requirements several decisions have been made. To create an efficient system we decided for a subject-based P/S using machine-readable subjects. This reduces the event matching process to simple integer comparisons. Textual subjects are optionally supported on publisher or subscriber nodes but will be mapped locally to machine-readable subjects using a hash function. Extended P/S schemes can be implemented locally on top of the basis scheme. Furthermore, frequent operations like the calculation of link qualities and bandwidth utilization requires only simple integer operations. A key element to provide a portable system is the event-based implementation using GEA that provides a thin abstraction layer. Implementations currently exist for 32bit and 64bit Intel-compatible, ARM, and Mipsel CPUs, and for the network simulator ns-2. The implementation does not rely on the existence of threads as all concurrent activities are realized as event flows inside the same GEA instance. GEA in combination with its helper components allows for a highly modularized implementation that consists of a number of independent components. These components can be dynamically combined to a fully functional middleware. Central components like the routing layer are separated into independent sup-components like the packet forwarding, the routing table, and the route discovery that can be individually replaced by alternative implementation. The middleware additionally allows to configure the structure of network packets at run-time. Together these techniques allow a fine-grained selection of functions and services that should be present (and hence consume resources) in the system. It furthermore allows to replace individual elements for a rapid integration of new algorithms or techniques.

A prototype implementation of the protocols developed in this thesis has been created and used to evaluate the functional capabilities and performance of the key components of the system using simulation and a native testbed. A special focus has been placed to compare the behavior of the traffic shaping components and the interaction of multiple nodes in the native setup and the simulation because this additionally provides an evaluation of the developed simulation setup. We saw that both setups showed very similar results. This increases our confidence in the usability of simulation-based tests for larger networks. But we also noticed that there are measurable differences between bot setups, most notably the absence of beacons from the 802.11 MAC layer and a slightly worse throughput in the simulation, that have to be considered when extrapolating the native full-scale performance from simulation results.

As a summary, the resulting system provides guaranteed end-to-end throughput based on fixed allocation, reliability that is controlled by the per-link quality requirements and global setting of automatic retransmissions, timely ordering of messages from one publisher. There is no global ordering of messages. Furthermore, based on the flow monitoring and appropriate path selection it provides limited support for timeliness and a maximum average / burst loss rate. At least a

detection of QoS violations is provided for these properties by the flow monitor.

The work presented within this thesis will be continued in the future as part of the DFG founded special focus program 1140 where most of this work has already been conducted in. Future lines of development include the improvement of the simulation model, the detection and handling of interfering nodes, the bandwidth coordination, and multiplexing of QoS with best effort traffic.

A limitation of the current simulation model is that it can only be used to describe environmental classes, not actual environment for which we have detailed information. This does not impede the protocol development but its evaluation or troubleshooting for a concrete environment. For this improvement the current propagation model has to be replaced by a propagation model that is aware of the environmental properties and can calculate the appropriate signal propagation including all possible fluctuations.

We identified the detection and handling of interference nodes as an essential element of the system as we cannot assume a 2-hop communication with them in all cases. The presented dynamic adaption relies on the feedback of the network driver that is usually not available to the protocol. Future development in this direction has to focus on utilizing internal data available in the IEEE 802.11 MAC implementation for upper layers. An open question remains whether it is possible to derive other information then the indication of over/under utilization from the local data or by combining data available in the nodes neighborhood. It would be very valuable if it is possible to measure the bandwidth that an interfering node utilizes. This could allow for a more accurate estimation of the bandwidth utilization. As we saw, the current approach only allows to measure the idle time. It would be a significant improvement if it would be possible to calculate the expected value.

The bandwidth coordination so far is focused on the support of QoS flows. We are able to multiplex QoS with best-effort traffic but we also saw that this is not equally possible. Along the event dissemination trees we have a number of planned but unused retransmissions that can be utilized for best-effort communication. But apart from that we have to use the dynamic allocation of best-effort contingents which is not well suited to transmit large amounts of data in a burst. Future developments have to strive for a faster allocation protocol. If the number of allocations grows we have to consider a higher chance for parallel allocations that could lead to a bandwidth over utilization. The dynamic adaptation is able to handle this case but if it drops a QoS flow in favor of a best-effort flow this is contra-productive. The multiplexing has to extend this error handling to account for such cases. An interesting question will be, if the integration of WLAN devices using the 802.11e standard that provides different traffic classes will be of benefit for the multiplexing of QoS and best-effort traffic. Especially the prioritization on the medium in combination with the dynamic adaption could provide a possible solution to the temporary over utilization as it would most likely defer best-effort data which would accumulate in the traffic shaper queue without causing dropouts of QoS packets. Under the assumption that we are able to determine interfering nodes with a high probability an adoption of an interference-aware routing protocol like IQRouting [GJTW05] to route best-effort traffic would be of great value.

A. Appendix

A.1. Simulation Setup

Shadowing propagation model

ns-2 WLAN setup (1/5: helper functions)

```
# return a^x
proc pow {a x} {
    return [expr exp($x*log($a))]
}
# return absolute power specified in dBm notation
proc dbm {x} {
    return [expr [pow 10.0 [expr $x/10.0]] * 0.001]
}
```

ns-2 WLAN setup (2/5: WLAN system model)

```
# antenna model: omni-directional antenna
set val(ant)            Antenna/OmniAntenna
# physical network type
set val(netif)          Phy/WirelessPhy
# wireless propagation model: shadowing
set val(prop)           Propagation/Shadowing
# physical channel separation: wireless
set val(chan)           Channel/WirelessChannel
# MAC: 802.11
set val(mac)            Mac/802_11
# device interface: generic link layer; drop tail queue, length 10
set val(ll)             LL
set val(ifq)            Queue/DropTail/PriQueue
set val(ifqlen)         10
# routing layer: no build-in routing, use own
set val(rp)             DumbAgent
```

ns-2 WLAN setup (3/5: propagation model parameter)

```
25 # path loss exponent: 3.6 (open urban area)
26 Propagation/Shadowing set pathlossExp_ 3.6
27 # standard signal distribution (4dB - medium fluctuations)
28 Propagation/Shadowing set std_db_      4
29 # reference distance for transmitter power (1m)
30 Propagation/Shadowing set dist0_       1.0
```

ns-2 WLAN setup (4/5: WLAN device parameters)

```
31 # transmitter power: 25dBm
32 Phy/WirelessPhy set Pt_ [dbm 25]
33 # transmitter frequency: 2.472 GHz (channel 13)
34 Phy/WirelessPhy set freq_ 2.472e9
35 # signal bandwidth: 11 Mbit/s
36 Phy/WirelessPhy set bandwidth_ 11e6
37 # carrier sense treshold
38 Phy/WirelessPhy set CSThresh_ [dbm -104]
39 # receive threshold
40 Phy/WirelessPhy set RXThresh_ [dbm -94]
41 # minimum sinal-to-noise ratio for error free reception: 10dB
42 Phy/WirelessPhy set CPThresh_ 10.0
```

ns-2 WLAN setup (5/5: 802.11 MAC parameters)

```
43 # RTS/CTS threshold: 250 byte
44 Mac/802_11 set RTSThreshold_ 250
45 # data rate (used from unicast): 11 Mbit/s
46 Mac/802_11 set dataRate_ 11.0e6
47 # basic rate (used for broadcast): 2 Mbit/s
48 Mac/802_11 set basicRate_ 2.0e6
49 # PLCP (Physical Layer Convergence Procedure)
50 # used for preamble 1 Mbit/s
51 Mac/802_11 set PLCPDataRate_  1.0e6
52 # number of retries for short (management) frames
53 Mac/802_11 set ShortRetryLimit_       7
54 # number of retries for data frames
55 Mac/802_11 set LongRetryLimit_        4
```

Additional error model

ns-2 WLAN setup (Near-range error model)

```
proc UniformErr {} {
    set err [new ErrorModel]
    $err unit packet
    $err set rate_ 0.005 # 0.5%
    $err ranvar [new RandomVariable/Uniform]
    $err drop-target [new Agent/Null]
    return $err
}

$ns node-config -IncomingErrProc UniformErr
```

A.2. Message Formats

A.2.1. Multicast Tree Construction and Maintenance

The multicast tree construction and maintenance protocol uses the following messages. It should be noted that the originator of a message is always contained in the packet header and not the message itself.

Types

QoSPair: storage type for QoS parameters		
QoSKey	QoSID	unique identifier of the metric
value	<>	value; type depends on the value

QoSThresholdList: QoS monitoring thresholds specification for a group of nodes		
Field	Type	Comment
nodecount	uint8	number of node identifiers
nodes	StationID[]	array of node identifiers to apply parameters to
qoscount	uint8	number of QoS threshold parameters
qosparam	QoSPair	array of QoS threshold parameters

QoSState: Record to store current state of a QoS property		
Field	Type	Comment
property	QoSID	unique identifier of the QoS property
required	<>	required value; type depends on the property
current	<>	current value; type depends on the property

Messages

FlowREQ: search for a provider of a flow		
Field	Type	Comment
flowroot	StationID	originator of the flow
flowid	FlowID	unique (at the originator) identifier of the flow
pktsize	uint32	maximum packet size of events
period	uint32	average interval time of events [μs]
timeout	uint32	timeout [μs] for bandwidth pre-allocation
qosreqcount	uint8	number of additional QoS requirements
qosreqs	QoSPair[]	array of additional QoS requirements

BranchREQ: request flow root to admit a new branch		
Field	Type	Comment
leaf	StationID	identity of the new leaf node
leafhops	uint8	distance of the leaf from the branch node
age	uint32	age at the branch node
leafpath	StationID[]	path from leaf to branch node
qosstatcount	uint8	number of flow statistics values
qosstats	QoSPair[]	array of current flow statistics values
qosreqcount	uint8	number of leaf QoS requirements
qosreqs	QoSPair	additional QoS requirements of the leaf node

BranchConnect: instruct branch node to connect to a new leaf node		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the root)
branchnode	StationID	node that should create the new branch
leafnode	StationID	identity of the new leaf node
leafhops	uint8	distance of the leaf from the branch node
leafpath	StationID[]	path from branch to leaf node

BranchDisconnect: instruct the whole branch to disconnect from a number of leafs		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the root)
leafcount	uint8	number of leaf nodes to disconnect
leafnodes	StationID[]	array of identities of the leaf nodes

BranchReject: reject the instruction to create a new branch		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the root)
leafnode	StationID	identity of the leaf node

ReconnectREQ: search for a missing neighbor or other node of the flow		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the originator)
neighbor	StationID	identity of the missing neighbor
flowroot	StationID	originator of the flow
flowid	FlowID	unique identifier of the flow
pktsize	uint32	maximum packet size of events
period	uint32	average interval time of events [μs]
timeout	uint32	timeout [μs] for bandwidth pre-allocation
qosreqcount	uint8	number of additional QoS requirements
qosreqs	QoSPair[]	array of additional QoS requirements

Standby: prevent downstream nodes from starting own repair process		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the originator)
flowroot	StationID	originator of the flow
flowid	FlowID	unique identifier of the flow
timeout	uint32	standby time [μs]

BranchOffer: offer a new upstream event provider		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the originator)
flowroot	StationID	originator of the flow
flowid	FlowID	unique identifier of the flow
to	StationID	intended target node

BranchAccept: accept a new upstream event provider		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the originator)
flowroot	StationID	originator of the flow
flowid	FlowID	unique identifier of the flow
leafcount	uint8	number of migrated leaf nodes
leafnodes	StationID[]	array of identifiers of migrated leaf nodes

LinkChangeInfo: inform flow root about link reconfiguration		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the originator)
flowroot	StationID	originator of the flow
flowid	FlowID	unique identifier of the flow
to	StationID	downstream node of the link
from_old	StationID	old upstream node of the link
from_new	StationID	new upstream node of the link
intermediate	StationID	intermediate node for new upstream node (or zero if direct link)
leafcount	uint8	number of migrated leaf nodes
leafnodes	StationID[]	array of identifiers of migrated leaf nodes

BranchRepair: re-route a link over an intermediate node		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the originator)
flowroot	StationID	originator of the flow
flowid	FlowID	unique identifier of the flow
leafcount	uint8	number of migrated leaf nodes
leafnodes	StationID[]	array of identifiers of migrated leaf nodes

ForceRepair: instruct a node to start repair because of leaving upstream neighbor		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the originator)
flowroot	StationID	originator of the flow
flowid	FlowID	unique identifier of the flow

SetQoSThreshold: setup QoS monitoring thresholds on one or more nodes		
Field	Type	Comment
serial	uint32	serial number of the message (unique at the originator)
flowroot	StationID	originator of the flow
flowid	FlowID	unique identifier of the flow
qoslistcount	uint8	number of QoSThresholdList records
qoslists	QoSThresholdList[]	array of QoSThresholdList records

QoSViolation: Report a QoS violation		
Field	Type	Comment
source	StationID	node that reports the QoS violation
flow	FlowID	event flow for which we report a QoS violation
root	StationID	address of the root node; report to leafs if `NoStationID`
count	uint8	number of violated properties
violations	QoSState[]	list of individual QoS violations

A.2.2. Message Priorities

The system uses the following priorities for the various messages (with 0 as the lowest priority).

Class	Priority	Description
Bulk	0	any non-management best-effort data
Beacon	1	beacons for bandwidth coordination and neighborhood discovery
P/S	2	management data of the P/S protocol
MCast	3	multicast tree maintenance
QoS	4...	basic priority for QoS flows; higher values for delay optimization

A.2.3. P/S Search and Update Protocol

Messages

SearchPublisher: Search for matching publishers		
Field	Type	Comment
subject	SubjectKey	machine-readable subject
appqos	String	opaque container to transport application QoS requirements
numpub	uint8	number of known publishers
publishers	StationID[]	array of known publishers

FoundPublisher: report matching publishers		
Field	Type	Comment
numpub	uint8	number of records included in the message
publishers	StationID[]	array of nodes with a matching publisher
flows	FlowID[]	unique identifier of the event flow from a publisher
appqos	String[]	array with the corresponding opaque application QoS
numsub	uint8	number of known subscribers for these publishers
subscribers	StationID[]	array of nodes with active subscribers

StopSubscriber: Disconnect a subscriber from a publisher		
Field	Type	Comment
subscriber	StationID	node that unsubscribed, died, or moved away
publisher	StationID	the publisher it was connected to
flow	FlowID	the unique identifier of the event flow

StopPublisher: Disconnect all subscribers from a publisher		
Field	Type	Comment
publisher	StationID	node that stopped publishing
flow	FlowID	unique identifier for the event flow of the publisher

Signals

QoSViolation: Report a QoS violation to a subscriber		
Field	Type	Comment
count	uint8	number of violated properties
violations	QoSState[]	list of individual QoS violations

NewSubscriber: Report a new connected subscriber on an event flow		
Field	Type	Comment
flow	FlowID	the flow the subscriber connected to

SelectProfile: Require assistance in the search process		
Field	Type	Comment
appqos	String	the opaque QoS selection of the subscriber
answer	SPAnswer&	reference to answer object

List of Tables

List of Figures

Bibliography

[3Co04] 3Com Corporation. *3Com OfficeConnect Wireless 11g Access Point, PC Card, and USB Adapter Specification*, 2004.

[ADGS02] Emmanuelle Anceaume, Ajoy K. Datta, Maria Gradinariu, and Gwendal Simon. Publish/subscribe scheme for mobile networks. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 74–81, New York, NY, USA, 2002. ACM Press.

[AGJM04] Geraud Allard, Leonidas Georgiadis, Philippe Jacquet, and Bernard Mans. Bandwidth reservation in multihop wireless networks: Complexity, mechanisms and heuristics. In *Proceedings of 24th International Conference on Distributed Computing Systems Workshops – W6: WWAN (ICDCSW'04)*, pages 762–767, 2004.

[ALJ02] Ghazaleh Ashayer, Hubert Ka Yau Leung, and H.-Arno Jacobsen. Predicate matching and subscription matching in publish/subscribe systems. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 539–548, Washington, DC, USA, 2002. IEEE Computer Society.

[AS00] Marcos Kawazoe Aguilera and Robert E. Strom. Efficient atomic broadcast using deterministic merge. In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 209–218, New York, NY, USA, 2000. ACM Press.

[ASS+99] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching Events in a Content-based Subscription System. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC '99)*, Atlanta, GA, USA, May 1999.

[B+99] S Bajaj et al. Improving simulation for network research. Technical report, Department of Computer Science, USC, 1999.

[BA03] Hakim Badis and Khaldoun Al Agha. Scalable model for the simulation of OLSR and Fast-OLSR protocols. In *Proceedings of IFIP Med-Hoc-Net'03*, Mahdia, Tunisia, June 2003.

[BA04] Hakim Badis and Khaldoun Alagha. An Efficient QOLSR Extension Protocol For QoS in Ad hoc Networks. In *IEEE VTC'04-Fall: Vehicular Technology Conference*, Los Angeles, USA, September 2004.

[BA05] Hakim Badis and Khaldoun Al Agha. *Quality of Service for Ad hoc Optimized Link State Routing Protocol (QOLSR)*, April 2005. http://www.ietf.org/internet-drafts/draft-badis-manet-qolsr-01.txt.

[BCM+99] Guruduth Banavar, Tushar Deepak Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, Austin, TX, USA, June 1999.

[Ben00] Alan Bensky. *Short-range Wireless Communication – Fundamentals of RF System Design and Application*. Demystifying Technology Series. LLH Technology Publishing, 2000. ISBN 1-878707-58-28.

[BFC93] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (cbt). In *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*, pages 85–95, New York, NY, USA, 1993. ACM Press.

[BKS+99] Guruduth Banavar, Marc Kaplan, Kelly Shaw, Robert E. Strom, Daniel C. Sturman, , and Wei Tao. Information Flow Based Event Distribution Middleware. In *Proceedings of the Middleware Workshop at ICDCS'99*, 1999.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1, A System of Patterns*. Wiley, July 1996. ISBN: 0-471-95869-7.

[BP96] Lawrence S. Brakmo and Larry L. Peterson. Experiences with Network Simulation. In *Measurement and Modeling of Computer Systems*, pages 80–90, 1996.

[BP00] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An In-Building RF-based User Location and Tracking System. In *Proceedings of IEEE INFOCOM 2000*, 2000.

[CABM03] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proceedings of MOBICOM 2003*, 2003.

[CBR02] Ian D. Chakeres and Elizabeth M. Belding-Royer. The Utility of Hello Messages for Determining Link Connectivity. In *Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications (WPMC'02)*, 2002.

[CC02] Raymond Cunningham and Vinny Cahill. Time Bounded Medium Access Control for Ad Hoc Networks. In *Time Bounded Medium Access Control for Ad Hoc Networks. Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC'02)*, pages 1–8, Toulouse France, 2002. ACM Press.

[CF03] Raphael Chand and Pascal Felber. A scalable protocol for content-based routing in overlay networks. In *NCA '03: Proceedings of the Second IEEE International Symposium on Network Computing and Applications*, page 123, Washington, DC, USA, 2003. IEEE Computer Society.

[CF04a] Raphael Chand and Pascal Felb. Efficient Subscription Management in Content-based Networks. In *Proceedings of DEBS*, May 2004.

[CF04b] Raphael Chand and Pascal Felber. XNET: a reliable content-based publish/subscribe system. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, pages 264 – 273, October 2004.

[CFGR02] Chee-Yong Chan, Pascal Felber, Minos Garofalakis, and Rajeev Rastogii. Efficient Filtering of XML Documents with XPath Expressions. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, page 235, Washington, DC, USA, 2002. IEEE Computer Society.

[CH03] Zhao Cheng and Wendi B. Heinzelman. Flooding strategy for target discovery in wireless networks. In *Proceedings of the Sixth ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2003)*, 2003.

[Cis05] Cisco Systems, Inc. *Technical Specifications for Cisco Aironet 350 and CB20A Wireless LAN Client Adapters*, 2005.

[cita] Bridge – Linux Ethernet Bridging. http://bridge.sf.net/.

[citb] Universal TUN/TAP Driver. http://vtun.sf.net/tun/.

[citc] User-Mode-Linux (UML). http://user-mode-linux.sourceforge.net/.

[CJ02] Gianpaolo Cugola and H.-Arno Jacobsen. Using publish/subscribe middleware for mobile systems. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):25–33, 2002.

[CJ03] T. Clausen and P. Jacquet. *RFC 3626: Optimized Link State Routing Protocol (OLSR)*. Project Hipercom, INRIA, October 2003.

[CJT01] Luis Felipe Cabrera, Michael B. Jones, and Marvin Theimer. Herald: Achieving a Global Event Notification Service. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau, Gemany, May 2001.

[CN01] Gianpaolo Cugola and E. Di Nitto. Using a publish/subscribe middleware to support mobile computing. In *Proceedings of the Workshop on Middleware for Mobile Computing in association with the IFPI/ACM Middleware Conference 2001*, Heidelberg, Germany, November 2001.

[CNF01] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, 2001.

[cor] CORTEX project homepage. http://cortex.di.fc.ul.pt/index.htm.

[CP02] Y. Choi and D. Park. Associativity based clustering and query stride for on-demand routing protocoly. *Journal of Communications and Networks*, 4(1):4–13, 2002.

[CPM02] Gianpaolo Cugola, Gian Pietro Picco, and Amy L. Murphy. Towards Dynamic Reconfiguration of Distributed Publish-Subscribe Middleware. In *Proceedings of 3rd International Workshop on Software Engineering and Middleware (SEM 2002)*, Orlando, Florida, May 2002.

[Cra03] Robert K. Crane. *Propagation handbook for wireless communication system design*. Number 13 in Electrical engineering and applied signal processing series. CRC Press LLC, 2003. ISBN 0-8493-0820-8.

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19:332–383, 2001.

[CS03] Mark Carson and Darrin Santay. NIST Net: a Linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3):111–126, 2003.

[CSS02] David Cavin, Yoav Sasson, and André Schiper. On the accuracy of manet simulators. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 38–43, New York, NY, USA, 2002. ACM Press.

[DLBF+03] Hector A. Duran-Limon, Gordon S. Blair, Adrian Friday, Thirunavukkarasu Sivaharan, and George Samartzidis. A Resource and QoS Management Framework for a Real-Time Event System in Mobile Ad Hoc Environments. In *Proceedings of the Ninth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003)*, Capri Island, Italy, October 2003.

[DLY95] P. Danzig, Z. Liu, and L. Yan. An Evaluation of TCP Vegas by Live Emulation. In *Proceedings of ACM SIGMetrics '95*, 1995.

[DSYB90] Alexander Dupuy, Jed Schwartz, Yechiam Yemini, and David Bacon. Nest: a network simulation and prototyping testbed. *Commun. ACM*, 33(10):63–74, 1990.

[ea99] Lokesh Bajaj et al. GloMoSim: A Scalable Network Simulation Environment. Technical Report 990027, 13, 1999.

[ebt] Ethernet Bridge Tables (ebtables) homepage. http://ebtables.sf.net/.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Survey*, 2003.

[EFMS04] M. Engel, B. Freisleben, and S. Hanemann M. Smith. Wireless Ad-Hoc Network Emulation Using Microkernel-Based Virtual Linux Systems. In *Proceedings of the 5th EUROSIM Congress on Modeling and Simulation, Marne la Vallee, France*, pages 198–203. EUROSIM Publishers, 2004.

[elv] Elvin project homepage. http://elvin.dstc.com/index.html.

[Fal99] K. Fall. Network emulation in the VINT/NS simulator. *Proceedings of the fourth IEEE Symposium on Computers and Communications*, 1999.

[ff] freifunk.net OLSR-Experiment homepage. http://www.olsrexperiment.de/.

[FGKZ03] Ludger Fiege, Felix C. Gärtner, Oliver Kasten, and Andreas Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Middleware*, pages 103–122, 2003.

[FMK+99] Geraldine Fitzpatrick, Tim Mansfield, Simon Kaplan, David Arnold, Ted Phelps, and Bill Segall. Augmenting the Workaday World with Elvin. In *Proceedings of the 6th European Conference on Computer-Supported Cooperative Work (ECSCW'99)*, pages 431–451, Copenhagen, Denmark, September 1999. Kluver Academic Publishers.

[Fou05] Foundry Networks. *Ironpoint 200 WiFi Access Point Specification*, 2005. http://www.foundrynet.com/products/PDFs/IP200_DS_012604.pdf.

[Fre91] Free Software Foundation, Inc. *GNU General Public License, version 2*, 1991. http://www.gnu.org/copyleft/gpl.html.

[gea] GEA Homepage. http://www-ivs.cs.uni-magdeburg.de/EuK/forschung/projekte/gea/index.shtml.

[GGT00] Yang Guo, Weibo Gong, and Don Towsley. Time-stepped hybrid simulation (tshs) for large scale networks. In *Proceedings of IEEE Infocom (2000)*, 2000.

[Gib99] Jerry D. Gibson, editor. *The mobile communications handbook*. CRC Press LLC, 2nd edition edition, 1999. ISBN 0-8493-8597-0.

[GJTW05] Rajarshi Gupta, Zhanfeng Jia, Teresa Tung, and Jean Walrand. Interference-aware QoS Routing (IQRouting) for Ad-Hoc Networks. In *Proceedings of GlobeCom 2005*, St. Louis, Missouri, USA, November 2005.

[GK00] Piyush Gupta and P. R. Kumar. The Capacity of Wireless Networks. *IEEE Transactions on Information Theory*, 46(2), March 2000.

[GKP99] R. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the READY event notification service. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop*, 1999.

[gry] The Gryphon Project homepage. http://www.research.ibm.com/distributedmessaging/gryphon.html.

[Gup05] Rajarshi Gupta. *Quality of Service in Ad-Hoc Networks*. PhD thesis, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, 2005.

[HBA04] Ignacy Gawedzki Hakim Badis and Khaldoun Al Agha. QoS Routing in Ad hoc Networks Using QOLSR with no Need of Explicit Reservation. In *IEEE VTC'04-Fall: Vehicular Technology Conference*, Los Angeles, USA, September 2004.

[HC03a] Barbara Hughes and Vinny Cahill. Towards Real-time Event-based Communication in Mobile Ad Hoc Wireless Networks. In *Proceedings of 2nd International Workshop on Real-Time LANS in the Internet Age 2003 (ECRTS/RTLIA03)*, pages 77–80, Porto, Portugal, July 2003.

[HC03b] Barbara Hughes and Vinny Cahill. Achieving Real-time Guarantees in Mobile Ad Hoc Wireless Networks. In *Work in progress session, IEEE Real Time Systems Symposium 2003*, Cancun, Mexico, Dezember 2003.

[HGM04] Yongqiang Huang and Hector Garcia-Molina. Publish/Subscribe in a Mobile Environment. *Wireless Networks – Pervasive Computing and Communications*, 10(6):643–652, November 2004.

[HM05] André Herms and Daniel Mahrenholz. GEA: A Method for Implementing and Testing of Event- driven Protocols. In *Proceedings of MeshNets 2005*, Budapest, Hungary, July 2005.

[Ins99] Institute of Electrical and Electronics Engineers (IEEE), Inc. *ANSI/IEEE Std 802.11, 1999 Edition*, 1999. Available online at http://standards.ieee.org/getieee802/802.11.html.

[ION02] IONA Technologies PLC. *OrbixTalk Programmer's Guide, version 3.3*, 2002. http://www.iona.com/support/docs/manuals/orbix/33/html/orbixtalk33_pguide /orbixtalk33IX.html.

[ipe] Iperf bandwidth measurement tool – homepage. http://dast.nlanr.net/Projects/Iperf/.

[IS05] Kurt Rothermel Illya Stepanov, Daniel Herrscher. On the impact of radio propagation models on manet simulation results. In *Proceedings of the 7th IFIP International Conference on Mobile and Wireless Communications Networks (MWCN 2005)*, September 2005.

[itg] OPNET Tech Inc. Homepage. http://www.opnet.com/.

[IY02] Magdy F. Iskander and Zhengqing Yun. Propagation prediction models for wireless communication systems. *IEEE Transactions on Microwave Theory and Techniques*, 50(3), March 2002.

[Jab04a] Jabber Software Foundation. *RFC 3920: Extensible Messaging and Presence Protocol (XMPP) – Core*, 2004.

[Jab04b] Jabber Software Foundation. *RFC 3921: Extensible Messaging and Presence Protocol (XMPP) – Instant Messaging and Presence*, 2004.

[Jab04c] Jabber Software Foundation. *RFC 3922: Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)*, 2004.

[Jab04d] Jabber Software Foundation. *RFC 3923: End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)*, 2004.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, April 1991. ISBN: 0471503363.

[JGWV05] Zhanfeng Jia, Rajarshi Gupta, Jean Walrand, and Pravin Varaiya. Bandwidth Guaranteed Routing for Ad-Hoc Networks with Interference Consideration. In *Proceedings of ISCC 2005*, Cartagena, Spain, June 2005.

[jhp] Jabber.org homepage. http://www.jabber.org.

[JM05] Sam Jansen and Anthony McGregor. Simulation with Real World Network Stack. In *Proceedings of the 2005 Winter Simulation Conference (WSC 2005)*, Orland Florida, USA, Dezember 2005.

[JMB01] David B. Johnson, David A. Maltz, and Josh Broch. *Ad Hoc Networking*, chapter 5 – DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks, pages 139–172. Addison-Wesley, 2001.

[JMH03] Johnson, Maltz, and Hu. *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR))*, April 2003.
http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-09.txt.

[KCM+01] M. O. Killijian, Raymond Cunningham, R. Meier, L. Mazare, and Vinny Cahill. Towards Group Communication for Mobile Participants. In *Proceedings of Principles of Mobile Computing (POMC'2001)*, pages 75–82, Newport, Rhode Island, USA, 2001.

[KEHM04] Ayman Kayssi and Ali El-Haj-Mahmoud. EmuNET: a real-time network emulator. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 357–362, New York, NY, USA, 2004. ACM Press.

[KGM+01] M. Kojo, A. Gurtov, J. Mannner, P. Sarolahti, T. Alanko, and K. Raatikainen. Seawind: a wireless network emulator. In *Proceedings of 11th GI/ITG Conference on Measuring, Modelling and and Evaluation of Computer and Communication Systems (MMB 2001)*, Aachen, Germany, September 2001.

[Kle93] Jack P.C. Kleijnen. Theory and methodology – verification and validation of simulation models. *European Journal of Operational Research*, 1995(82):145–162, 1993.

[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, August 2000.

[KNE03] David Kotz, Calvin Newport, and Chip Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth College, July 2003.

[Lan05] Charan Langton. *Intuitive Guide to Principles of Communications*. Loral Space Systems, 2005. Available online at http://www.complextoreal.com/index.htm.

[LBD+01] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of Ad Hoc Wireless Networks. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking*, pages 61–69, Rome, Italy, July 2001.

[LFG+01] Benyuan Liu, Daniel R. Figueiredo, Yang Guo, Jim Kurose, and Don Towsley. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Procedings of IEEE Infocom (2001)*, 2001.

[LLP+01] R. Leung, Jilei Liu, E. Poon, A.-L.C. Chan, and Baochun Li. MP-DSR: a QoS-aware multi-path dynamic source routing protocol for wireless ad-hoc networks. In *Proceedings of 26th Annual IEEE Conference on Local Computer Networks (LCN 2001)*, pages 132–141, Tampa, FL, USA, November 2001.

[LM94] Averill M. Law and Michael G. McConuisc. Simulation of commun1cations networks. In *Proceedings of the 1994 Winter Simulation Conferencen*, 1994.

[LPFJ+03] Gilberto Flores Lucio, Marcos Paredes-Farrera, Emmanuel Jammeh, Martin Fleury, and Martin J. Reed. OPNET Modeler and Ns-2: Comparing the Accuracy Of Network Simulators for Packet-Level Analysis using a Network Testbed. *WSEAS Transactions on Computers*, 2003.

[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 eighteenth workshop on Parallel and distributed simulation*, pages 7–16, New York, NY, USA, 2004. ACM Press.

[Mac70] M. H. MacDougall. Computer system simulation: An introduction. *ACM Comput. Surv.*, 2(3):191–209, 1970.

[Mah00] Daniel Mahrenholz. Aspektorientierte Realisierung eines generischen Systemmonitors. Master's thesis, Otto-von-Guericke University, Magdeburg, 2000. (in German).

[Mah01] Daniel Mahrenholz. Minimal Invasive Monitoring. In *Proceedings of The Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001)*, Magdeburg, Germany, May 2001.

[Mal04] Marek Malowidzkio. Network Simulators: A Developer's Perspective. Technical report, Military Communication Institute, Zergze, Poland, 2004.

[MC02] René Meier and Vinny Cahill. Steam: Event-based middleware for wireless ad hoc network. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 639–644, Washington, DC, USA, 2002. IEEE Computer Society.

[Men97] Xiannong Meng. EMUNET: Design and Implementation - A Debugging Aid for Distributed Programs in TCP/IP Based Network. In *Proceedings of the IEEE International Performance Computing & Communications Conference*, pages 101–107, Phoenix, AZ; USA, 1997.

[MGPT02] Gabriel Montenegro, Benjamin Gaidioz, Pascale Primet, and Bernard Tourancheau. Equivalent differentiated services for AODVng. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(3):110–111, 2002.

[MI04a] Daniel Mahrenholz and Svilen Ivanov. *Howto: Wireless Network Emulation using ns-2 and Distributed Applications, Version 1.0*. Otto-vou-Guericke University of Magdeburg, Germany, December 2004. Available online at http://www-ivs.cs.uni-magdeburg.de/EuK/forschung/projekte/nse/howtos/ns2dsapp_userguide.pdf.

[MI04b] Daniel Mahrenholz and Svilen Ivanov. *Howto: Wireless Network Emulation using ns-2 and User-Mode-Linux (UML), Version 1.0*. Otto-vou-Guericke University of Magdeburg, Germany, December 2004. Available online at http://www-ivs.cs.uni-magdeburg.de/EuK/forschung/projekte/nse/howtos/ns2uml_userguide.pdf.

[MI04c] Daniel Mahrenholz and Svilen Ivanov. Real-Time Network Emulation with ns-2. In *Proceedings of the The 8-th IEEE International Symposium on Distributed*

Simulation and Real Time Applications (DS-RT 2004), Budapest, Hungary, October 2004.

[MI05] Daniel Mahrenholz and Svilen Ivanov. Adjusting the ns-2 Emulation Mode to a Live Network. In *Proceedings of KiVS05 – 14. Fachtagung Kommunikation in Verteilten Systemen*, Kaiserslautern, Germany, March 2005. Gesellschaft für Informatik (GI), Springer.

[MMAR04] J. Mullen, T. Matis, K. Adams, and S. Rangan. Achieving Robust Protocols for Mobile Ad Hoc Networks. In *Proceedings of IERC 2004 - the 2004 Industrial Engineering Research Conference*, Houston, Texas, USA, May 2004.

[MSAM05] Peter Millard, Peter Saint-Andre, and Ralph Meijer. *Jabber Enhancement Proposals: Publish-Subscribe (JEP-0060)*. Jabber Software Foundation (JSF), 2005. http://www.jabber.org/jeps/jep-0060.html.

[MSSP02] Daniel Mahrenholz, Olaf Spinczyk, and Wolfgang Schröder-Preikschat. Program Instrumentation for Debugging and Monitoring with AspectC++. In *Proceedings of The 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC 2002)*, Washington DC, USA, April 2002.

[Mul04] John P. Mullen. Efficient Models of Fine-Grain Variations in Signal Strength. In *OPNETWORK 04*, Washington DC., USA, September 2004.

[Net04] Nicholas Nethercote. *Dynamic Binary Analysis and Instrumentation*. PhD thesis, University of Cambridge, November 2004.

[NJG02] Michael Neufeld, Ashish Jain, and Dirk Grunwald. Nsclick: Bridging Network Simulation and Deployment. In *Proceedings of the 5th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pages 74–81. ACM Press, 2002.

[NM03] Nicholas Nethercote and Alan Mycroft. Redux: A Dynamic Dataflow Tracer. *Electronic Notes in Theoretical Computer Science*, 89(2), 2003.

[ns2a] ns and nam Sourceforge homepage. http://nsnam.sourceforge.net/.

[ns2b] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns/.

[nse] NS-2 Emulation Extensions Homepage. http://www-ivs.cs.uni-magdeburg.de/EuK/forschung/projekte/nse/index.shtml.

[NTCS99] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the Fifth Anual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 151–162, August 1999.

[Obj04a] Object Management Group (OMG). *CORBA Event Service Specification, version 1.2*, 2004. http://www.omg.org/technology/documents/formal/event_service.htm.

[Obj04b] Object Management Group (OMG). *CORBA/IIOP Specification, version 3*, 2004. http://www.omg.org/technology/documents/formal/corba_iiop.htm.

[Obj04c] Object Management Group (OMG). *Notification Service Specification, version 1.1*, 2004. http://www.omg.org/technology/documents/formal/notification_service.htm.

[OBP05a] Rodolfo Oliveira, Luis Bernardo, and Paulo Pinto. Flooding Techniques for Resource Discovery on High Mobility MANETs. In *Proceeding of the International Workshop on Wireless Ad-hoc Networks (IWWAN 2005)*, London, UK, 2005.

[OBP05b] Rodolfo Oliveira, Luis Bernardo, and Paulo Pinto. Searching for resources in MANETS: a cluster based flooding approach. In *Proceedings of the 2nd International Conference on E-business and Telecommunication Networks (ICETE'05)*, pages 105–111, Reading, UK, 2005. ISBN: 972-8865-33-3.

[OBP05c] Rodolfo Oliveira, Luis Bernardo, and Paulo Pinto. Searching for resources in MANETs: Improving flooding efficiency in 802.11. Technical report, Tele - DEE - FCT/Universidade Nova de Lisboa, March 2005. Telecommunications Section Internal Report TR-2005-01.

[Orf97] Sophocles J. Orfanidis. Electromagnetic waves and antennas. Available online at www.ece.rutgers.edu/ orfanidi/ewa, ECE Department, Rutgers University, 1997.

[PB02] Peter Pietzuch and Jean Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *In Proceedings of the 22nd International Conferenence on Distributed Computing Systems (ICDCS'02) Workshop*, pages 611–618, Vienna,Austria, July 2002.

[PBRD04] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir Das. *RFC 3561: Ad Hoc On Demand Distance Vector (AODV) Routing*. The Internet Society, 2004.

[pca] tcpdump/libpcap homepage. http://www.tcpdump.org/.

[PFLS00] Joso Pereira, Francoise Fabret, Francois Llirbat, and Dennis Shasha. Efficient matching for web-based publish/subscribe systems. In *Proceedings of the International Conference on Cooperative Information Systems (COOPIS)*, Eiat, Israel, September 2000.

[Pie04] Peter Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, Computer Laboratory, Queens' College, University of Cambridge, February 2004.

[PJL02] K. Pawlikowski, H.-D.J. Jeong, and J.-S.R. Lee. On credibility of simulation studies of telecommunication networks. *IEEE Communications Magazine*, 40(1):132–139, Januar 2002.

[PL00] Wei Peng and Xi-Cheng Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 129–130, Piscataway, NJ, USA, 2000. IEEE Press.

[PLF+00] Joso Pereira, Francois Llirbat, Francoise Fabret, Radu Preotiuc-Pietro, Dennis Shasha, and Kenneth Ross. Publish/Subscribe on the Web at Extreme Speed. In *Proceedings of ACM SIGMOD Conference on Management of Data*, Cairo, Egypt, 2000.

[Pow96] David Powell. Group communication. *Communication of the ACM*, 39(4):50–53, 1996.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.

[rea] The REAL Network Simulator. http://www.cs.cornell.edu/skeshav/real/overview.html.

[RFA99] George F. Riley, Richard Fujimoto, and Mostafa H. Ammar. A Generic Framework for Parallelization of Network Simulations. In *MASCOTS*, 1999.

[Riz97] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[Ruf00] Marcel Ruff. *XMLblaster – Message-oriented Middleware (MOM), whitepaper*. XMLBlaster.org, 2000. http://www.xmlblaster.org/xmlBlaster/doc/whitepaper/whitepaper.html.

[SA] Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG Technical Conference (AUUG'97)*.

[SAB+00] Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelpso. Content Based Routing with Elvin4. In *Proceedings of AUUG2K*, Canberra, Australia, June 2000.

[SBB05] Matthias Scheidegger, Florian Baumgartner, and Torsten Braun. An Integrated Simulator for Inter-domain Scenarios. In *Proceedings of KiVS05 – 14. Fachtagung Kommunikation in Verteilten Systemen*, pages 295 – 306, Kaiserslautern, Germany, March 2005. Gesellschaft für Informatik (GI), Springer.

[Sch78] H. D. Schwetman. Hybrid simulation models of computer systems. *Commun. ACM*, 21(9):718–723, 1978.

[SCS02] Yoav Sasson, David Cavin, and André Schiper. Probabilistic Broadcast for Flooding in Wireless Mobile Ad hoc Networks. Technical Report IC/2002/54, Swiss Federal Institute of Technology (EFPL), 2002.

[sie] SIENA Homepage. http://serl.cs.colorado.edu/ carzanig/siena/software/.

[Sie05] Richard Sietmann. Luftbrücken über Berlin. *c't*, (26), 2005.

[SN05] Julian Seward and Nicholas Nethercote. Using Valgrind to detect undefined value errors with bit-precision. In *Proceedings of the USENIX'05 Annual Technical Conference*, Anaheim, California, USA, April 2005.

[Sun95] Sun Microsystems. *RFC 1831: RPC: Remote Procedure Call Protocol Specification Version 2*, 1995.

[Sun02] Sun Microsystems, Inc. *JavaSpaces Service Specification, version 1.2.1*, 2002. http://www.sun.com/software/jini/specs/jini1.2html/js-title.html.

[Sun04] Sun Microsystems, Inc. *Java Remote Methode Invocation (RMI) Specification, version 1.5*, 2004. http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html.

[Sun05] Sun Microsystems, Inc. *Jini Network Technology - Specifications, version 2*, 2005. http://www.sun.com/software/jini/specs/.

[Tar98] Sasu Tarkoma. Scalable internet event notification architecture (siena). In *Proceedings of Workshop on Internet Scale Event Notification (WISEN'98)*, Irvine, California, USA, July 1998.

[tib06] TIBCO Rendezvous Overview, 2006. http://www.tibco.com/software/enterprise_backbone/rendezvous.jsp.

[TMB01] Mineo Takai, Jay Martin, and Rajive Bagrodia. Effects of wireless physical layer modeling in mobile ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 87–94, New York, NY, USA, 2001. ACM Press.

[top] Toronto Publish/Subscribe development projects index. http://www.msrg.toronto.edu/projects-index.mhtml.

[TvS02] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 1st edition, 2002. ISBN 0130888931.

[Var01] András Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic., June 2001.

[Var02] András Varga. OMNeT++. *IEEE Network Interactive*, 16(4), July 2002.

[W3C99] W3C. *XML Path Language (XPath) Version 1.0*, 1999. http://www.w3.org/TR/xpath.

[W3C03] W3C. *XML Path Language (XPath) Version 2.0*, 2003. http://www.w3.org/TR/xpath20.

[WJL03] Y. Wang, X. Jia, and H.K. Lee. An indoors wireless positioning system based on wireless local area network infrastructureo. In *Procedings of the 6th International Symposium on Satellite Navigation Technology Including Mobile Positioning & Location Services (SatNav 2003)*, 2003.

[WL03] Jie Wu and Wei Loud. Forward-Node-Set-Based Broadcast in Clustered Mobile Ad Hoc Networks. *Wireless Communications and Mobile Computing*, 3(2):155–173, 2003.

[wsm] IBM WebSphere MQ homepage. http://www-306.ibm.com/software/integration/wmq/.

[YGK03] Yunjung Yi, Mario Gerla, and Taek-Jin Kwon. Efficient Flooding in Ad hoc Networks: a Comparative Performance Study. In *Proceedings of the IEEE International Conference on Communication (ICC03)*, 2003.

[YGM99] Tak W. Yan and Hector Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.

[ZBG98] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

[zli] The zlib compression/decompression library. http://www.gzip.org/zlib/.