Emulation von Rechnernetzen zur Leistungsanalyse von verteilten Anwendungen und Netzprotokollen

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Daniel J. Herrscher

aus Esslingen am Neckar

Hauptberichter:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
1. Mitberichter:	Prof. Dr.-Ing. Bernd Freisleben
2. Mitberichter:	Prof. Dr.-Ing. habil. Dr. h. c. mult. Paul J. Kühn
Tag der mündlichen Prüfung:	7. Dezember 2005

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart

2005

Danksagung

An erster Stelle möchte ich meinem Doktorvater Prof. Rothermel danken, der mir das Arbeiten in einem idealen Umfeld ermöglicht hat und mich mit seiner kritischen und stets konstruktiven Art begleitete. Prof. Freisleben und Prof. Kühn danke ich für die bereitwillige Übernahme des Mitberichts und ihre wertvollen Kommentare.

Großen Dank schulde ich auch allen Kollegen der Abteilung Verteilte Systeme für die zahllosen Diskussionen in einer unvergleichlichen Atmosphäre, in der ich mich immer wohl gefühlt habe. Insbesondere möchte ich Steffen Maier danken, der maßgeblich am Erfolg des NET-Projekts beteiligt war und es auch in Zukunft weiterführen wird. Mein Dank gilt auch den mehr als einem Dutzend Studierenden, die als wissenschaftliche Hilfskräfte, durch ihre studentischen Arbeiten oder im Rahmen von Praktika ihren Teil zum NET-Projekt beigetragen haben.

Das Land Baden-Württemberg, die Deutsche Forschungsgemeinschaft und der Deutsche Akademische Austauschdienst haben das NET-Projekt und damit diese Dissertation finanziell unterstützt. Dafür möchte ich mich ebenfalls herzlich bedanken.

Meinen Eltern danke ich für die großzügige Unterstützung während meines Informatikstudiums und die Ermutigung, die Promotion anzustreben. Nicht zuletzt möchte ich mich bei meiner Lebensgefährtin Friederike Bornträger bedanken, die mich auch in schwierigen Phasen stets unterstützt hat.

Stuttgart, im Dezember 2005

Daniel Herrscher

Kurzfassung

Um die Leistung von verteilten Anwendungen und Netzprotokollen in Abhängigkeit von den Eigenschaften der verwendeten Rechnernetze zu analysieren, wird eine Testumgebung benötigt, die Netzeigenschaften zuverlässig nachbilden („emulieren“) kann. Eine solche Testumgebung wird *Emulationssystem* genannt. Bisher existierende Emulationssysteme sind aufgrund ihrer Architektur entweder nur für sehr kleine Szenarien geeignet, oder sie können nur unabhängige Netzverbindungen nachbilden, und schließen damit alle Netztechnologien mit gemeinsamen Medien aus.

In dieser Arbeit werden zunächst verschiedene Architekturvarianten für die Realisierung eines Emulationssystems vorgestellt und bewertet. Für die Variante mit zentraler Steuerung und verteilten Emulationswerkzeugen wird dann detailliert die Funktionalität eines Emulationssystems mit seinen wesentlichen Komponenten beschrieben.

Das in dieser Arbeit entwickelte Emulationsverfahren greift auf der logischen Ebene der Sicherungsschicht in den Kommunikationsstapel ein. Auf dieser Ebene werden die beiden *Basiseffekte* Rahmenverlust und Verzögerung durch verteilte Emulationswerkzeuge nachgebildet. Alle anderen Netzeigenschaften können auf diese Basiseffekte zurückgeführt werden.

Um Netztechnologien mit gemeinsamen Medien durch verteilte Werkzeuge nachbilden zu können, wird zusätzlich das Konzept des *virtuellen Trägersignals* eingeführt. Hierbei werden die Eigenschaften eines Rundsendemediums nachgebildet, indem kooperative Emulationswerkzeuge Rundsendungen zur Signalisierung eines Trägersignals benutzen. Somit kann jede Werkzeuginstanz lokal ein aktuelles Modell des emulierten gemeinsamen Mediums halten. Auf dieser Basis kann auch das Verhalten von Medienzugriffsprotokollen nachgebildet werden.

Die Arbeit deckt auch die wesentlichen Realisierungsaspekte eines Emulationssytems ab. Mit ausführlichen Messungen wird gezeigt, dass das entwickelte System für die Nachbildung von Netzszenarien sehr gut geeignet ist, selbst wenn die nachzubildenden Parameter sich dynamisch ändern. Die entwickelten Werkzeuge sind in der Lage, Netzeigenschaften in einem weiten Parameterbereich realistisch nachzubilden. Mit diesem System steht nun eine ideale Testumgebung für Leistungsmessungen von verteilten Anwendungen und Netzprotokollen in Abhängigkeit von Netzeigenschaften zur Verfügung.

English Abstract

Network Emulation for the Performance Analysis of Distributed Applications and Network Protocols

1 Introduction

The rapid growth of the available network bandwidth, new network technologies and the general availability of networked devices have lead to a rapid change in network usage [TMW97]. This also results in new types of distributed applications that were developed in the last years, such as peer-to-peer applications [GHN+03], and new protocols, such as transport protocols suitable for wireless environments [BSAK95, MCG+01] or routing protocols for ad-hoc networks [JM96, PR99].

During the design and implementation of these new distributed applications and protocols, it is essential to analyze their performance in various network environments. While mathematical analysis and simulations are commonly used in early design stages, measurements have to verify the theoretical results as soon as implementations become available.

For two reasons, it is problematic to conduct these measurements in real network environments. First, suitable environments may not be *available* to the performance analyst. Secondly, especially for mobile wireless network environments, the *repeatability* of network parameters cannot be guaranteed for several measurements. Therefore, it is hard to *compare* the measurement results.

Thus, there is a strong need for *synthetic* network environments that can be parametrized in order to reproduce the properties of an original or fictitious network. The process of introducing network properties that differ from the actual properties of the hardware in use is called *network emulation*. A *network emulation tool* is software or hardware capable of altering network traffic in a specified way. A testbed with a number of interconnected nodes and suitable emulation tools is called *network emulation testbed.*

Existing approaches to network emulation either consider only small scenarios, or are restricted to simple network properties and do not allow the emulation of shared media networks. While in wired networks, shared media access protocols are becoming less important, the interest in shared media wireless networking is growing.

In this doctoral thesis, a method for network emulation is proposed that is both scalable in terms of scenario size and network traffic, and provides the possibility of emulating shared media networks. The proposed system, the Network Emulation Testbed (NET), combines a centralized scenario control and distributed network emulation tools running on several node PCs. The emulation tools can cooperate to maintain a distributed, yet consistent media model. This facilitates the emulation of shared media networks with distributed tools.

The original thesis is written in German language and consists of seven chapters. In Chapter 1, a general motivation for measurements in an emulated network environment is given and the necessity for a scalable network emulation testbed is identified. In Chapter 2, the method of performance measurement in general is compared to the complementary methods mathematical analysis and simulation. After that, the existing work in network emulation is covered. Chapter 3 proposes several possible architectures for emulation testbeds and compares them regarding efficiency and performance. For one of these candidate architectures, the architecture with centralized control and distributed emulation tools, the functionality of an emulation system and its main components is described in detail in Chapter 4. Chapter 5 covers the most important realization issues of the developed emulation system. Chapter 6 provides measurements proving the applicability of the system. An example for a typical performance measurement of a network protocol is also given. Finally, Chapter 7 concludes the thesis and points out some future research issues. In the following sections of this abstract, a summary of Chapters 2–7 is given.

2 Related Work

In the first part of Chapter 2, the method of performance measurement in general is compared to the complementary methods mathematical analysis and network simulation.

The basic difference between these three methods for performance analysis is that they can be applied to different types of test subjects: Mathematical analysis is commonly used with simple, well-understood protocols that can be described by mathematical formulae. In this case, the mathematical formulae make up a *model* for the protocol [Jai91].

More complex protocols can hardly be described by mathematical formulae. A *simulation model* has to be used instead. The main difference to mathematical analysis is that simulation uses *algorithmic models.* In general, the complexity of simulation models is unlimited. In practice,

the effort of designing and interpreting the simulation model limits the complexity of the model. To judge the behavior of e.g. a protocol in a certain environment, the *whole environment* of the protocol has to be modeled, not only the protocol itself. While some simulation tools provide a large set of existing model components that can easily be combined to a complex simulation environment [BEF+00], it is often problematic to provide a complete simulation model for a complex test subject; consider e.g. an application like a distributed file system. Furthermore, some aspects of the reality are often neglected in simulation models, and therefore may lead to wrong behavior of the simulation model; prominent examples include the timer granularity in operating systems [BP96a]. The only method to really judge the behavior of an *implementation* of a distributed application or a network protocol is therefore the *measurement* of the implementation in a suitable environment.

There are many publications about measurement testbeds that have been designed especially for the analysis of one specific application or network protocol (e.g. [CI95, MBJ99, TCDA00]). However, especially for environments with dynamic network parameters (as in mobile ad-hoc networks, MANETs), it is very hard, if not impossible to conduct several subsequent measurements with comparable network parameters. Therefore, comparative measurements are hardly possible in real-world testbeds. Another problem with testbeds is their limited availability: Few projects have the resources for a complete measurement testbed just for one measurement setup. Therefore, there is a strong demand for measurement testbeds that can *emulate* a large number of network conditions in a reproducible way.

In the second part of this chapter, a comprehensive overview of the related work in network emulation is given.

Hardware-based emulation testbeds are usually restricted to one network technology (e.g. [KR01, HH02, JS03]). Software-based testbeds combine general-purpose hardware with special software tools to emulate a large variety of network scenarios, and are therefore much more flexible. This thesis focuses on software-based emulation.

Since a network scenario often consists of several network links, there are two basic architectures to build the emulated network: Either the whole scenario is emulated by one single, central emulation entity, or several instances of an emulation tool are connected together to form a comprehensive scenario, each of them responsible for emulating its own part of the network.

Centralized emulation approaches often reuse existing network *simulation* tools for emulation. These tools can work with complex network models, including both exclusive links and shared media [Fal99, FTO02]. However, in order to interact with real applications and protocols, the simulation cores have to run in real-time. This requirement becomes a problem when moving to realistic scenario sizes and bandwidths. While this can be addressed to some extent by par-

allelizing the central simulation instance [VYW+02, MYV03], the basic problem of having a performance bottleneck persists.

Since centralized emulation approaches are limited in terms of scenario size and bandwidth, it is common to distribute the emulation efforts among several instances. A coherent idea is to partition the emulation scenario according to the emulated topology, leading to one emulation tool per emulated link in the extreme case. Most emulation tools change network properties by intercepting, delaying, or altering data in the protocol stack. Early emulation attempts aim at the emulation of a single network link only [IP94, ADLY95, ACO97, Riz97, CS03]. They differ in the parameters they can affect, but have in common that the emulation parameters stay constant during the experiments. Recent approaches include dynamic parameter changes, triggered by the replay of previously gathered measurement data [NSNK97].

Two connected machines equipped with network emulation tools can help analyzing some aspects of e.g. a transport protocol. However, to perform measurements within more complex network topologies, e.g. to analyze routing protocols, more machines are necessary. With the growing number of participating nodes, both the setup of the machines and the coordination of the emulation tools becomes a challenge, especially if the emulated scenario includes dynamic parameter changes during the experiment. Emulation testbeds face these problems, as they ease the setup and operation of emulation scenarios consisting of large numbers of nodes and connections. Netbed [WLS+02] at the University of Utah consists of 168 PC nodes connected by a number of switches, working together as large programmable patchpanel to create almost any virtual connection topology. Special link properties can be introduced between the nodes. The special properties of MANETs are not yet considered. Other testbeds consider the emulation of MANET properties [ZL02, LS02], but do not include the effects of frame collisions in their emulation models.

The conclusion from the related work is twofold: First, network emulation testbeds are the ideal environment to analyze the impact of certain network scenarios on the performance of distributed applications and protocols. Secondly, existing solutions to network emulation either consider only small scenarios, or are restricted to simple network properties and do not allow the emulation of shared media networks. Therefore, a scalable solution to network emulation is needed that overcomes these limitations.

3 Architecture

In Chapter 3, it is first discussed at which layer of the protocol stack software-based emulation should operate. Based on the basic insight that the software under test has to operate *on top* of the emulated network, it is concluded that the emulation method should operate as low as

possible in the protocol stack. For a software-based solution, this translates to emulation *below the network layer.* This way, it is possible to include network layer protocol implementations as software under test. As a result, it is proposed to insert an additional *emulation layer* between the network layer and the data link layer [HR02].

Based on the protocol abstraction of the emulation layer, the two *basic effects* that can be emulated at this layer are derived: frame loss and frame delay. It is sketched how these basic effects can be derived from other network properties.

In the remaining part of this chapter, several candidate architectures for emulation systems are proposed and compared. The main classes of architectures are centralized control with centralized emulation tools, centralized control with distributed tools, and distributed control with distributed tools. In addition to that, there are two alternative approaches to integrate emulation tools into the network stack: First, it is possible to intercept network traffic at the emulation layer with an *emulation proxy* and direct this traffic to an emulation tool, which is responsible for further processing. Secondly, it is also possible to integrate the emulation tool functionality directly into the network stack.

The proposed architectures differ in complexity, scalability, and the parameter range they can emulate. At the end of this chapter, all candidate architectures are summarized and compared in a table.

4 Functionality

In Chapter 4, one of the candidate architectures described in Chapter 3 is selected for realization, namely the architecture with centralized control and distributed emulation tools, with the emulation tools integrated directly into the network stack. The functionality of each of the components is described in detail.

The central emulation control is responsible for the overall coordination of an experiment, and especially for the parameter settings of the distributed emulation tools. Based on a central scenario description, which can include dynamic parameters [HLR02], the central emulation control derives the parameters to be emulated at each point of time during an experiment.

These current network parameters are stored in the *global network model*, which includes the parameters bit error rate, bandwidth, and propagation delay for each pair of nodes. The subset of parameters that are relevant for an emulation tool on one individual node is called *local network model*. At the beginning and during the runtime of an experiment, the central emulation control sends the current local network models for each node to the emulation tools on the respective nodes.

During an experiment, there are many instances of local emulation tools running at the same time, at least one instance on every emulation node. Each instance is responsible for the emulation of one network connection of one node. In this chapter of the thesis, it is described in detail how an emulation tool derives the basic effects from the parameters in its local network model, and how the emulation of the basic effects is performed [HR02].

The emulation of shared media networks with distributed emulation tools requires an additional concept: the *virtual carrier* [HMR03]. This concept facilitates the emulation of any medium access protocol that is based on carrier sensing (CSMA). The basic idea is to hold an up-to-date model of the emulated media at every participating station, and to keep the respective media models consistent.

This can be achieved if the emulation tools on all stations listen to all transmissions on the media, whether they are destined to the respective station or not. On the reception of a frame, the emulation tool can calculate the time this frame would occupy the emulated media (according to the emulated bandwidth) and update its local model accordingly. Frames that are destined to the local machine are forwarded to upper layers, all other frames can be discarded by the emulation tool. Frame transmission requests by the local station are handled according to the medium access algorithm, taking the emulated media status into account. If an emulation tool receives an additional transmission while its emulated media is in the *busy* status, it can react accordingly and emulate the effects of a collision.

5 Realization

In Chapter 5, important realization aspects of a concrete emulation system (the NET-System at the University of Stuttgart) are described. It is especially pointed out that this thesis does *not* contain a complete and detailed reference for all implementation and usage questions; for that reason, further reference is given, especially the technical reference manual of the NET-System [HM04].

The realization chapter first gives a brief overview of the hardware of the NET-System and its capabilities, especially how the efficiency of the emulation process can be improved by the usage of VLANs (Virtual LANs) to emulate network devices and connection topologies.

Then, an important detail of the central emulation control is explained, namely an example for an automated process to derive the current network parameters from a dynamic scenario description. The complete derivation process is shown for the network parameters of a MANET scenario, starting with the movement patterns of the MANET nodes and ending with the parameters in the global network model.

For the distributed emulation tools, the main focus of the realization chapter is on the integration to the protocol stack implementation. The tools that were implemented as part of this thesis run as kernel modules for Linux, and offer the same service abstraction as a *virtual network device*.

On the sending side, the registration as a network device is sufficient to integrate the emulation tool with the packet flow: If an emulation tool is registered with a valid layer 3 address, the layer 3 implementation automatically forwards outgoing packets not to a real network device, but to the emulation tool instead.

On the receiving side, the integration is more complex. Two alternative ways of integration to the Linux network stack are explained. In the first alternative, the emulation tool registers itself as a new type of network protocol. If the emulation tool instances at the *sending side* tag all outgoing packets with this new protocol type, the layer 3 protocol multiplexer at the *receiving side* will forward these incoming packets to the respective emulation tool instance. The emulation tool can now delay or drop the packets, according to the configured network parameters. The second alternative of intercepting traffic on the receiving side is based on a patch in a central function of the Linux protocol stack implementation, and facilitates the interception of packets that have not been tagged at the sending side before.

Because of limitations of the execution environment, namely the characteristics of the operating system and the testbed hardware, the specified functionality of an emulation tool cannot always be realized exactly. In these cases, an emulation tool may introduce network properties that differ from the specified properties. These unwanted properties are called *emulation artifacts*. The types of emulation artifacts that could occur, their expected impact and possible methods to reduce them are covered at the end of the realization chapter.

6 Experiments

In Chapter 6, it is shown by measurements that the NET-System is suitable for network emulation, and that the developed tools can emulate network parameters in a wide parameter range and with sufficient accuracy.

First, the minimal delay between two instances of an emulation tool in the NET-System is measured. This delay value is the lower limit of the delay that can be emulated in the NET-System, and also plays an important role in the virtual carrier emulation. The measured delay always stays below $160\,\mu$s, which is two orders of magnitude lower than the delays that are introduced by higher protocol layers, and therefore negligible in most cases. Virtual carrier emulation is also possible with this minimal delay value.

Secondly, the performance of VLAN-configurations on the central switch of the NET-System is measured. It is concluded from the VLAN configuration performance that VLAN topology

emulation can only be used for static, not for dynamic topologies. Dynamic connection topologies have to be emulated by the emulation tools themselves.

The evaluation of the central emulation control performance shows that it is sufficient to control a highly dynamic scenario with 64 nodes, which is the maximum scenario size in the NET-System.

The subsequent evaluation of the emulation tool reveals the parameter range that this tool can emulate in the NET-System. Any loss ratio can be emulated; delay can be emulated with a mean error of $< 2.5\,\%$. Bandwidth limitation scales up to a maximum bandwidth of $b_{\max} \approx 381\,\mathrm{Mbit/s}$ per link, which is the hardware limitation of the PC nodes in the NET-System.

Finally, the setup and the results of a typical measurement in a MANET scenario are provided to serve as an example for the type of measurements that are possible in the NET-System. To measure the performance of a location-based routing protocol implementation, a scenario with 50 mobile nodes is set up. Since the protocol implementation needs access to a location device to work properly, an additional component is needed: the virtual GPS[1]-device [HMTR04]. One instance of the virtual GPS-device is running on each emulation node. During an experiment, the central emulation control supplies the GPS-devices with their respective current location information, according to the virtual movements of the nodes. This way, the unmodified implementation of the location-based protocol can be analyzed in the NET-System. The measurement results are in good agreement with the performance predictions of a comparable simulation model.

7 Conclusion

Chapter 7 concludes this doctoral thesis by summarizing the main results and pointing to future research issues.

Network emulation is essential for comparative performance measurements of distributed applications and protocols. For larger scenarios and higher bandwidths, emulation efforts have to be distributed. Existing approaches to distributed network emulation focus on the emulation of single network links. The distributed emulation of shared media networks has not been addressed before.

In this thesis, different candidate architectures for network emulation systems have been proposed and compared. For one of the architectures, namely the combination of a centralized emulation control with distributed, cooperative emulation tools that are integrated into the protocol stack, the functionality of a network emulation system has been presented in detail. An impor-

[1] Global Positioning System

tant detail in the functionality of the distributed emulation tools is the virtual carrier concept, which facilitates the emulation of any media access protocol that is based on carrier sensing.

The thesis also covers the most important realization issues of an emulation system. Among other things, the integration of the emulation tools into the network stack implementation of Linux is explained in detail. The proposed system has been completely realized, and is now available at the University of Stuttgart (NET-System). Extensive measurements have shown the applicability of the developed system. The NET-System constitutes an ideal environment for comparative performance measurements of distributed applications and protocols.

Future research issues have been identified regarding the emulation tools, resource emulation and scalability. While there are tools available for the emulation of the medium access protocols defined in IEEE 802.3 (Ethernet) [HMR03] and IEEE 802.11b (Wireless LAN) [Yan04], further protocols based on carrier sensing could also be covered by tools that also rely on the virtual carrier concept.

Despite from the emulated GPS-device introduced in Chapter 6, the emulation of resources other than network devices has not been covered by this thesis. Useful emulated resources in an emulation system could include emulated primary and secondary storage devices, which should be configurable with respect to speed and capacity, as well as emulated processing power.

Finally, the scalability of the NET-System is still limited: The number of testbed nodes is the upper limit for the scenario size that can be emulated. There are approaches to node virtualization that can overcome this limitation. This is subject of further research efforts [MHR05].

Inhaltsverzeichnis

1

Einleitung

1.1 Motivation

Der Fortschritt in der digitalen Kommunikationstechnik hat in den letzten Jahren zu einer bemerkenswerten Dynamik in der Verwendung von Rechnernetzen[1] geführt. Steigende Übertragungsraten, sinkende Zugangskosten und die zunehmende Verbreitung von Funktechnologie sind die Motoren für die Weiter- und Neuentwicklung von verteilten Anwendungen und Netzprotokollen.

Statistiken über die Aufteilung des Internet-Verkehrs nach Anwendungen verdeutlichen diese Dynamik. Während um 1990 noch Dateiübertragungen über FTP, E-Mails und Diskussionsgruppen den Hauptteil des Verkehrs ausmachten [CDJM91], waren 1997 HTTP-basierte Anwendungen (das "World Wide Web") bereits für zwei Drittel der übertragenen Datenmenge verantwortlich [TMW97]. Eine Messung aus 2002 zeigt, dass der Netzverkehr aus dezentralen Dateitauschdiensten ("Peer-to-Peer") inzwischen HTTP bei weitem überholt hat [GHN+03].

Nicht nur Anwendungen, sondern auch die darunterliegenden Netzprotokolle werden weiterentwickelt. Beispielsweise können klassische Transportprotokolle nur ineffizient in drahtlosen Netzen mit hohen Fehlerraten betrieben werden, was zu zahlreichen Verbesserungsvorschlägen geführt hat [BB95, BSAK95, MCG+01]. Ein weiteres Beispiel für junge Protokollentwicklungen sind Vermittlungsprotokolle für die hochdynamischen Topologien von Ad-Hoc-Netzen, die anderen Anforderungen genügen müssen als Vermittlungsprotokolle für statische Netze [PB94, JM96, PR99].

[1] In vielen Dokumenten in deutscher Sprache wird für den englischen Begriff *(Computer) Network* der Anglizismus *Netzwerk* verwendet. Ich versuche dies so weit wie möglich zu vermeiden und verwende die korrekte Übersetzung *Netz* bzw. *Rechnernetz*. Einige Komposita des Wortes *Netz* sind jedoch schon durch andere Bedeutungen belegt (z.B. *Netzkabel, Netzgerät*), sodass ich in diesen Sonderfällen zur Vermeidung von Doppeldeutigkeiten auf die gebräuchlicheren Formen ausweiche (*Netzwerkkabel, Netzwerkgerät*).

Wie jede neu entwickelte oder überarbeitete Software müssen auch verteilte Anwendungen und Netzprotokolle sowohl funktionalen als auch nichtfunktionalen Anforderungen genügen. Funktionale Anforderungen definieren die Eignung für eine bestimmte Funktion („das geforderte Ergebnis wird erzielt“), während nichtfunktionale Anforderungen weitere Qualitäten beschreiben („das Ergebnis wird effizient erreicht“, „die Software ist gut bedienbar“, etc.). Unter den nichtfunktionalen Anforderungen nehmen die *Leistungseigenschaften*, für die eine Metrik existiert, einen besonderen Platz ein, denn sie sind mess- und vergleichbar und daher ein gutes „Verkaufsargument“, sowohl im wissenschaftlichen, als auch im wirtschaftlichen Umfeld. Es besteht also ein großes Interesse an systematischen und anerkannten Methoden zur Leistungsanalyse von verteilten Anwendungen und Netzprotokollen.

Die messbare Leistung von Software in einem Gesamtsystem hängt von verschiedenartigen Faktoren ab, neben den Eigenschaften der Software selbst auch von den Ressourcen der ausführenden Systeme (Rechenleistung, Speicherkapazität) und der Vernetzung der Systeme (Vernetzungstopologie, Übertragungsrate, etc.). Die Leistung von verteilten Anwendungen und Netzprotokollen hängt in besonderem Maße von den Eigenschaften der verwendeten Netzinfrastruktur ab. Daher beschränkt sich diese Arbeit auf Methoden zur Leistungsanalyse von Software in Abhängigkeit von Netzeigenschaften. Da wir[2] andere Systemressourcen, die auch Einfluss auf die Gesamtleistung haben können, nicht berücksichtigen, sprechen wir von „vergleichender“ Leistungsanalyse. Mit vergleichender Leistungsanalyse sind im Allgemeinen keine absoluten, sondern nur relative Leistungsaussagen möglich. Eine typische relative Leistungsaussage wäre beispielsweise: „Im gegebenen Netzszenario erreicht Transportprotokoll A eine um 5% höhere Übertragungsrate als Transportprotokoll B.“

Es gibt drei grundsätzliche Methoden zur vergleichenden Leistungsanalyse von verteilten Anwendungen und Protokollen: Mathematische Analyse, Simulation und Messung.

Mathematische Analyse [Jai91] wird vor allem in frühen Phasen des Entwicklungsprozesses angewendet, in denen noch keine zu messende Implementierung existiert. Grundlage der mathematischen Analyse ist ein mathematisches Modell des gesamten zu untersuchenden Systems. Das Modell repräsentiert sowohl die zu analysierende Anwendung und deren Ausführungsumgebung, als auch die Kommunikationsinfrastruktur. Damit ein solches Modell für eine mathematische Darstellung beherrschbar ist, kommt der Modellierungsschritt nicht ohne starke Vereinfachungen aus. Während in klassischen Modellierungsansätzen die Übertragungsrate von Leitungen und die Kapazität von Vermittlungspuffern berücksichtigt werden, ist der vollständige Algorithmus eines gängigen Transportprotokolls schon zu komplex, um mit vertretbarem Aufwand komplett analytisch beschreibbar zu sein. Die Methode der mathematischen Modellierung und Analyse ist gut geeignet für prinzipielle Aussagen zur Leistung von Algo-

[2] In dieser Arbeit wird das Personalpronomen *wir* verwendet, wenn auf Eigenleistungen Bezug genommen wird. Dies ist als *pluralis modestiae* zu verstehen, da an der Entwicklung der Gesamtlösung, die in dieser Dissertation beschrieben wird, nicht nur der Autor selbst, sondern u.a. ein gutes Dutzend Studenten beteiligt waren.

rithmen in einer vollständig verstandenen Umgebung, nicht jedoch für die Analyse von komplexeren Systemen und der Beurteilung von tatsächlichen existierenden Implementierungen.

Modelle, die zu komplex sind, um sie mit vertretbarem Aufwand in einem mathematischen Kalkül darzustellen, können mit Hilfe von Simulationswerkzeugen interpretiert werden. Für die exakte Simulation von Kommunikationsprozessen bieten sich ereignisdiskrete Verfahren an [BEF+00, ZBG98, Ril03]: Alle Ereignisse innerhalb des betrachteten Systems (z.B. das Senden eines Datenpakets) werden chronologisch verarbeitet und können weitere Ereignisse in der Zukunft bewirken (z.B. den Empfang eines Datenpakets). Die Verarbeitungslogik der Ereignisse ist das Verhaltensmodell des Systems und kann prinzipiell beliebig komplex programmiert werden. So stehen im Simulator ns-2 [BEF+00] sogar mehrere Versionen des Transportprotokolls TCP als Modell bereit. Trotz der prinzipiell unbeschränkten Möglichkeiten der Methode Simulation arbeitet jedes konkrete Simulationsmodell mit gewissen Vereinfachungen. Systemnahe Details wie die Betriebsmittelverwaltung eines Betriebssystems, Zeitgebergranularitäten etc. werden in existierenden Werkzeugen bisher nicht berücksichtigt, obwohl sie nachweislich Einfluss auf die Leistung von Netzprotokollen haben [BP96b]. Eine weitere Einschränkung ergibt sich bei der Simulation des Verhaltens von Anwendungen: Ab einer gewissen Komplexität können Anwendungen nicht mehr mit vertretbarem Aufwand in ein Simulationsmodell integriert werden.

Eine verbindliche Aussage über die Leistung einer konkreten Implementierung einer verteilten Anwendung oder eines Netzprotokolls kann man nur durch Messung dieser Implementierung treffen. Diese These wird von Veröffentlichungen unterstützt, die sich kritisch mit der Methode Simulation auseinandersetzen [PF97, FP01], bzw. Simulations- und Messergebnisse vergleichen [HZ03, HMTR04]. Um für den angestrebten Einsatzzweck aussagekräftig zu sein, muss eine Messung in der entsprechenden Zielumgebung stattfinden. Oft kommt jedoch schon aus wirtschaftlichen Gründen die Bereitstellung einer solchen Umgebung zu Analysezwecken nicht in Frage. Soll die Umgebung zudem neuartige Technik beinhalten, kann auch bei ausreichendem Budget die Verfügbarkeit in ausreichenden Stückzahlen problematisch sein. Besonders bei Szenarien mit mobilen Teilnehmern stellt sich zusätzlich die Frage der Wiederholbarkeit eines Experiments. So ist es praktisch unmöglich, ein Experiment mit mobilen, spontan vernetzten Rechnern mehrmals mit exakt denselben Bedingungen zu wiederholen, um beispielsweise die Effizienz verschiedener Vermittlungsprotokolle für spontanvernetzte Umgebungen zu vergleichen [MBJ99].

Die Kernprobleme der Methode Messung in der Realumgebung, nämlich Verfügbarkeit und Wiederholbarkeit, können durch die Bereitstellung einer künstlichen, der Realität nachgebildeten Testumgebung umgangen werden. Wenn es gelingt, die relevanten Netzeigenschaften der gewünschten tatsächlichen oder fiktiven Umgebung in der erforderlichen Genauigkeit und wiederholbar nachzubilden, sind vergleichende Leistungsmessungen im Labor möglich. Den

Prozess der Nachbildung von Netzeigenschaften nennen wir *Emulation*. Werkzeuge, die Netzeigenschaften nachbilden können, nennen wir *Emulationswerkzeuge*.

Frühe Arbeiten zur Emulation von Netzeigenschaften befassen sich mit Veränderungen der Kommunikation zwischen zwei direkt verbundenen Rechnern. Schon einfache Emulationswerkzeuge [IP94, ADLY95, ACO97, Riz97] ermöglichen vergleichende Messungen, beispielsweise zum Leistungsverhalten von Anwendungen bei Veränderung der Parameter Übertragungsrate, Fehlerrate und Verzögerung. Die Kombination von mehreren Instanzen eines solchen Werkzeugs zu einer Testumgebung ermöglicht auch Szenarien mit mehreren emulierten Verbindungen [WLS+02]. Zentralisierte Emulationswerkzeuge, die nicht nur eine einzelne Verbindung, sondern den gesamten Netzverkehr in einem Szenario berücksichtigen, können bis zu einem gewissen Grad auch Interaktionen von mehreren Sendern nachbilden [DBCF95, KGM+01, FTO02]. Durch ihre Architektur sind diese Ansätze jedoch nur für kleinere Szenarien geeignet, weil der gesamte Netzverkehr eines Szenarios von einem einzigen Werkzeug verarbeitet werden muss, das damit einen Flaschenhals darstellt. Es gibt zwar erste verteilte Ansätze, die mehrere Instanzen eines Emulationswerkzeugs von einer zentralen Steuerungsinstanz aus koordinieren [ZL02, LS02], jedoch werden hier nur sehr grundlegende Verbindungsparameter wie die Konnektivität berücksichtigt. Weitergehende Netzeigenschaften, die auch die Interaktion von verschiedenen Sendern mit einbeziehen, konnten mit einem verteilten, skalierbaren Ansatz bisher nicht nachgebildet werden.

Für die vergleichende Leistungsanalyse von verteilten Anwendungen und Netzprotokollen ist eine Testumgebung notwendig, welche die Eigenschaften von bestehenden oder fiktiven Rechnernetzen realistisch emulieren kann. Bestehende Lösungen sind entweder wegen ihrer zentralisierten Architektur nicht skalierbar, oder können wesentliche leistungsrelevante Netzeigenschaften nicht nachbilden. Aus diesem Grund wird in dieser Arbeit eine Lösung vorgestellt, die durch die Kombination einer hochgradig verteilten Architektur mit neuartigen, kooperativen Emulationswerkzeugen eine sowohl realistische, als auch skalierbare Nachbildung von Rechnernetzen ermöglicht.

1.2 Struktur der Arbeit

Im folgenden Kapitel 2 wird die Methode der Emulation von Netzeigenschaften in das Spektrum der Methoden zur Leistungsanalyse eingeordnet. Dabei wird insbesondere klar, für welche Anwendungsfälle die Methode Emulation den verwandten Methoden überlegen ist. Im zweiten Teil des Kapitels werden die existierenden Arbeiten zur Emulation von Netzeigenschaften systematisiert und bewertet.

In Kapitel 3 wird zunächst diskutiert, in welcher konzeptionellen Ebene ein Verfahren zur Emulation von Netzeigenschaften eingreifen sollte. Aus dem ermittelten Eingriffspunkt ergeben sich

die Netzparameter, die an dieser Stelle nachgebildet werden können. Schließlich werden alternative Lösungsarchitekturen zur Nachbildung dieser Parameter vorgestellt und verglichen.

In Kapitel 4 werden die Funktionsweise und das Zusammenspiel der Komponenten eines Emulationssystems mit zentraler Steuerung und verteilten Emulationswerkzeugen im Einzelnen erläutert. Ein wichtiger Teilaspekt ist hierbei die Idee des kooperativen verteilten Emulationswerkzeugs, das die Nachbildung der Eigenschaften eines Netzes mit gemeinsamem Medium erlaubt.

Wichtige Aspekte der Realisierung des für diese Arbeit entwickelten Emulationssystems werden in Kapitel 5 beschrieben. Dabei wird beispielsweise die Integration der Emulationswerkzeuge in den Betriebssystemkern von Linux erläutert. Für detaillierte Informationen zur Implementierung und Bedienung wird jedoch explizit auf die technische Dokumentation des Systems verwiesen [HM04].

Kapitel 6 zeigt durch systematische Messungen die Leistungsfähigkeit und damit auch das Anwendungsspektrum des entwickelten Emulationssystems. Außerdem wird anhand eines typischen Beispiels gezeigt, wie mit einem emulierten Netzszenario Leistungseigenschaften eines Netzprotokolls gemessen werden können. Abschließend werden die Messergebnisse den Ergebnissen aus einer vergleichbaren Simulation gegenübergestellt.

Kapitel 7 beschließt diese Abhandlung mit einer Zusammenfassung des wissenschaftlichen Beitrags und der wichtigsten Ergebnisse. Ein Ausblick auf zukünftige Forschungsthemen rundet die Arbeit ab.

2

Stand der Wissenschaft

In diesem Kapitel werden zunächst die existierenden Methoden für die Leistungsanalyse von verteilten Anwendungen und Netzprotokollen charakterisiert. Aus der Diskussion der Methoden wird die Motivation für die Emulation von Netzeigenschaften klar. Danach werden die existierenden Arbeiten vorgestellt und klassifiziert. In der abschließenden Bewertung wird der Beitrag der vorliegenden Arbeit motiviert und von den existierenden Arbeiten abgegrenzt.

2.1 Einordnung der Arbeit

In Kapitel 1 wurden bereits die drei grundlegenden Methoden der Leistungsanalyse genannt: Mathematische Analyse, Simulation und Messung. Im Folgenden werden die Idee und die charakteristischen Merkmale jeder dieser Methoden kurz anhand von wichtigen Vertretern erklärt. Basierend auf diesem Überblick wird die Methode Emulation motiviert und in das Spektrum der besprochenen Methoden eingeordnet.

Gemeinsames Ziel der in diesem Abschnitt vorgestellten Methoden ist die vergleichende Leistungsanalyse von verteilten Anwendungen und Netzprotokollen (zusammengefasst im Folgenden „Testsubjekte“ genannt). „Vergleichend“ kann hier zweierlei heißen: Entweder wird die Leistung mehrerer Varianten eines Testsubjekts verglichen, oder die Leistung eines Testsubjekts wird in Abhängigkeit von Umgebungsparametern des Testsubjekts gemessen. In dieser Arbeit beschränken wir uns auf Parameter, die direkt oder indirekt die Eigenschaften des verwendeten Rechnernetzes beeinflussen. Ein direkter Parameter ist beispielsweise die Bitfehlerrate eines Funkkanals. Ein typischer indirekter Parameter ist die Mobilität der Benutzer in einem mobilen Ad-Hoc-Netz: Die Mobilität wirkt indirekt auf die Eigenschaften des Ad-Hoc-Netzes, indem beispielsweise die Änderungen der Konnektivität der Teilnehmer bei höherer Mobilität häufiger

auftreten. Testsubjekt könnte in diesem Beispiel ein Vermittlungsprotokoll für Ad-Hoc-Netze sein.

Die jeweiligen Analysemethoden unterscheiden sich vor allem in der Art der verwendeten Modelle. Das notwendige theoretische Hindergrundwissen zur Modellbildung liefert die allgemeine Modelltheorie nach Stachowiak [Sta73]: Jede Modellierung ist eine Abbildung von Attributen eines (realen oder fiktiven) Originals in die eines Modells. Bei dieser Abbildung wird stets nur ein Teil der Attribute des Originals berücksichtigt („Verkürzungsmerkmal“ der Abbildung). Die übrigen Attribute sind nicht im Modell repräsentiert („präterierte Attribute“). Durch das Wesen des Modells können weitere Attribute hinzukommen, die keine Entsprechung im Original haben („abundante Attribute“). Zweck der Modellbildung ist es, durch die Arbeit mit dem Modell zu Aussagen über Attribute zu gelangen, die auch für die zugehörigen Attribute des Urbilds gelten. Diese Übertragung funktioniert nur, wenn die für die Aussagen relevanten Attribute korrekt modelliert sind. Die Auswahl der zu modellierenden Attribute ist also für die Nutzbarkeit des Modells essentiell.

Bei allen modellbasierten Analysemethoden stellt sich zunächst die Frage nach der Granularität des verwendeten Modells, d.h. die Wahl der kleinsten modellierten Einheit des Systems. Die Wahl der geeigneten Granularität muss den naturgemäß höheren Aufwand einer feineren Modellierung gegen den zu erwartenden Erkenntnisgewinn abwägen. Soll wie in unserem Fall Kommunikation in einem Rechnernetz modelliert werden, hat es sicherlich keinen Sinn, auf molekularer Ebene Elektronenbewegungen zu modellieren. Für gewisse Fragestellungen, beispielsweise zur Bewertung von Interferenzen in einem Mobilfunknetz, kann es erforderlich sein, einzelne Bits und sogar Bitcodierungsverfahren in das Modell aufzunehmen. Da in dieser Arbeit verteilte Anwendungen und Netzprotokolle in traditionell paketvermittelnden Rechnernetzen als Testsubjekte betrachtet werden, erscheint es nicht sinnvoll, bitgenaue Modelle zu verwenden. Wir werden deshalb im Folgenden Modelle mit der Granularität einzelner Datenpakete (bzw. Rahmen) betrachten, falls nicht explizit eine andere Granularität angegeben ist. Fast alle existierenden Arbeiten für unseren Problembereich arbeiten mit Modellen dieser Granularität. Eine weitergehende Diskussion zur Wahl der richtigen Modellierungsgranularität für Modelle von Kommunikationsnetzen ist in [HBE+01] zu finden.

2.1.1 Mathematische Analyse

Die Grundlagen der mathematischen Modellierung für die Leistungsanalyse wurden beispielsweise von Raj Jain beschrieben [Jai91]. Die Idee dabei ist, das Verhalten des Testsubjekts und aller anderen beteiligten Komponenten mathematisch darzustellen. Dabei kommen beispielsweise Verfahren aus der Wahrscheinlichkeitstheorie (z.B. zur Modellierung der Wahrscheinlichkeit der Ankunft eines Datenpakets) und der Warteschlangen (z.B. zur Modellierung von Paketpuffern) zum Einsatz.

Der große Vorteil mathematischer Modelle ist deren einfache Auswertung, wenn das Modell einmal existiert. Steht eine mathematisch geschlossene Darstellung für den Zusammenhang der interessierenden Parameter eines Systems zur Verfügung, können relativ leicht Trends erkannt und Optima bestimmt werden. Werden mehrere Testsubjekte in gleicher Weise modelliert, kann sogar die Überlegenheit eines Testsubjekts mathematisch bewiesen werden. Die Herausforderung bei der mathematischen Analyse ist es, die relevanten Attribute eines Testsubjekts und dessen Umgebung zu identifizieren und korrekt zu modellieren. Um mit vertretbarem Aufwand mathematisch darstellbar zu sein, sind die Modelle stets stark vereinfacht, d.h. sie abstrahieren sehr stark von der Realität.

Demers et al. zeigen in [DKS89] anhand eines mathematischen Modells die Überlegenheit des „Fair-Queueing“-Algorithmus gegenüber normalen FIFO-Warteschlangen[1] für den Fall, dass einzelne Datenquellen überdurchschnittlich viel Netzlast erzeugen. Obwohl das Modell nur einen sehr kleinen Aspekt der Realität beschreibt (die Warteschlangen eines Routers) und dabei von scheinbar wesentlichen Attributen abstrahiert (beispielsweise wird das Vermittlungsprotokoll übergangen), genügt das Modell für die gewünschte Aussage.

Schon bei ein wenig komplexeren Modellierungsaufgaben werden die Grenzen der mathematischen Modellierung deutlich. Fuks et al. [FLV01] stellen für ein einfaches paketvermittelndes System ein mathematisches Modell für den Zusammenhang der Größe der Vermittlungstabellen mit der Verzögerung der Pakete auf. Obwohl die Autoren sich auf ein sehr einfaches Vermittlungsprotokoll beschränken, wird ihr Modell nur handhabbar, weil sie von unendlichen Warteschlangen ausgehen. Die Unendlichkeit der Warteschlangen ist ein Beispiel für ein abundantes Attribut des Modells, dessen Einfluss auf die Ergebnisse zumindest genauerer Beachtung bedarf.

Fazit: mathematische Analyse

Leistungsanalyse mit mathematischen Modellen kann für einfache Algorithmen zu klaren Aussagen führen, die in dieser Evidenz von empirischen Verfahren nicht zu erbringen sind. Zwar ist der Modellierungsschritt durch die Auswahl der betrachteten Attribute stets angreifbar, der mathematisch korrekte Schluss auf der Grundlage des Modells jedoch nicht.

Komplexere Algorithmen oder sogar das Verhalten von Anwendungen realistisch mit rein mathematischen Modellen zu beschreiben, ist allerdings – zumindest mit den bisher bekannten mathematischen Methoden – nicht möglich (vgl. auch [AB82]). Da die Leistungsanalyse von verteilten Anwendungen aber explizit zum Problemraum dieser Arbeit gehört, wird die Methode der mathematischen Analyse hier nicht weiter betrachtet.

[1] FIFO steht für *First In First Out* und bezeichnet die Eigenschaft „reihenfolgeerhaltend“.

2.1.2 Simulation

Wie bei der mathematischen Analyse wird auch bei der Methode der Netzsimulation das gesamte betrachtete System modelliert, d.h. neben den Testsubjekten selbst auch alle Komponenten der Umgebung, die mit den Testsubjekten interagieren. Das charakteristische Merkmal der Simulation gegenüber der mathematischen Analyse ist, dass *algorithmische* Modelle verwendet werden. Das Verhalten der Testsubjekte und der Umgebung wird in einer geeigneten Sprache modelliert. Simulationswerkzeuge unterstützen beim Erstellen und Interpretieren („Ausführen“) des Modells, sowie bei der Auswertung der Ergebnisse. Um den Aufwand für die Modellierung und die Interpretation des Modells zu minimieren, wird als Modellierungssprache oft eine existierende Programmiersprache verwendet.

Alle relevanten Werkzeuge für die Netzsimulation in der von uns betrachteten Modellgranularität basieren auf ereignisdiskreter Verarbeitung, d.h. sie verarbeiten in chronologischer Reihenfolge Ereignisse, die in einer zentralen Warteschlange verwaltet werden. Das jeweils nächste anstehende Ereignis der Warteschlange (z.B. das Senden eines Datenpakets) wird durch die im Modell definierte Verarbeitungslogik ausgewertet und kann weitere Ereignisse in der Zukunft bewirken (z.B. den Empfang eines Datenpakets), die wiederum an der chronologisch richtigen Stelle in die Warteschlange einsortiert werden.

Für die Einsatzfähigkeit eines Simulationswerkzeugs ist die Verfügbarkeit einer umfangreichen Bibliothek, die Modelle für Standardkomponenten enthält, essentiell. Solche Komponenten können als Teilmodelle in eigene Simulationsmodelle unverändert oder parametrisiert übernommen werden. Wichtige Komponenten sind beispielsweise einfache Netzverbindungen definierter Übertragungsrate, Modelle von häufig verwendeten Netzprotokollen (z.B. TCP und IP) oder Lastgeneratoren.

Simulationswerkzeuge

Es existieren zahlreiche Werkzeuge für die Netzsimulation. Im Folgenden werden kurz die wichtigsten Vertreter der generell einsetzbaren Werkzeuge mit ihren charakteristischen Eigenschaften vorgestellt.

Das populärste Werkzeug für die Netzsimulation ist „ns-2“, ein ereignisbasierter, paketorientierter Simulator, dessen Kern an der University of Southern California in Berkeley entstanden ist [BEF+00]. Die Modelle für ns-2 werden in den Programmiersprachen OTcl[2] (Topologie und Szenariodefinition) und C++ (Protokollverhalten) definiert. Da ns-2 ein offenes System ist, das von zahlreichen Forschergruppen auf der ganzen Welt verwendet wird, stehen inzwischen sehr viele Komponenten für alle wichtigen Netzprotokolle als Teilmodelle zur Verfügung.

[2] OTcl ist die objektorientierte Version der Skriptsprache Tcl *(Tool Command Language)*.

Um die Effizienz beim Erstellen und Auswerten eines Modells zu erhöhen, sind im Umfeld von ns-2 weitere Werkzeuge entstanden. Erstellungswerkzeuge können Teile eines Modells automatisch erzeugen, z.B. eine typische statische Netztopologie [MLMB01] oder einen für mobile Benutzer eines Rechnernetzes typischen Bewegungsablauf [SHB+03]. Mit einem Visualisierungswerkzeug kann beispielsweise eine graphische Aufbereitung der Paketflüsse eines Szenarios dargestellt werden [EHH+00].

An der University of California in Los Angeles wurde mit „GloMoSim“ ein ereignisbasiertes Simulationswerkzeug speziell für drahtlose Netze entwickelt [ZBG98]. Im Gegensatz zu ns-2 ist GloMoSim darauf ausgelegt, dass mehrere parallel arbeitende Prozesse sich die Bearbeitung der Ereignisse teilen. Bei entsprechend partitionierbaren Simulationsszenarien kann so die Bearbeitung eines Szenarios auf mehrere Rechner verteilt und damit beschleunigt werden. Das Verhaltensmodell der Protokolle und Netzkomponenten wird komplett in der Programmiersprache C erstellt. Im Vergleich mit ns-2 sind bisher deutlich weniger Modellkomponenten verfügbar.

Der „Georgia Tech Network Simulator“ ist ebenfalls ein ereignisbasiertes, parallelisierbares Simulationswerkzeug [Ril03]. Das gesamte Werkzeug und alle vorhandenen Modellkomponenten sind in der Programmiersprache C++ geschrieben, was dank der durchgängig objektorientierten Architektur die Benutzung und Erweiterung vereinfachen soll. Sowohl klassische Netze mit statischer Topologie, als auch drahtlose Ad-Hoc-Netze werden unterstützt.

Auch an der Universität Stuttgart ist ein paralleles, ereignisbasiertes Simulationswerkzeug für komplexe Netzszenarien entwickelt worden [Nec98]. Vorteil dieses Werkzeugs ist vor allem die generelle, durchgänging objektorientierte Architektur. Nicht nur das Netzmodell, sondern auch der parallele Simulator ist objektorientiert angelegt. Mehrere alternative Synchronisierungskonzepte für die Parallelisierung der Simulation stehen zur Verfügung. So kann für jedes Simulationsmodell das geeignete Synchronisierungsverfahren ausgewählt werden.

Fazit: Simulation

Netzsimulation hat sich als Standardmethode für die Leistungsanalyse von Netzprotokollen etabliert. Die meisten Veröffentlichungen in diesem Bereich verwenden zur Zeit dasselbe Simulationswerkzeug (ns-2), wodurch die Ergebnisse zumindest vergleichbar sind. Prinzipiell stellt sich auch bei Simulationsmodellen die Frage nach der korrekten Modellierung, d.h. vor jeder Schlussfolgerung aus Simulationsergebnissen muss die Frage gestellt werden, ob das verwendete Modell alle für die Schlussfolgerung relevanten Attribute des Originals korrekt abbildet. Zur grundsätzlichen Diskussion um die Grenzen der Modelle, die in der Netzsimulation verwendet werden, haben besonders Autoren aus dem Umfeld des Projekts ns-2 beigetragen [PF97, FP01, Flo01].

Ein beschränkender Faktor für die Methode Simulation ist der Aufwand für die Modellierung. So ist es zwar prinzipiell möglich, das Verhalten von verteilten Anwendungen in einem Simulationsmodell nachzubilden. Für die meisten existierenden Anwendungen – man denke beispielsweise an ein verteiltes Dateisystem – würde dies jedoch einen beträchtlichen Modellierungsaufwand bedeuten und wäre damit wirtschaftlich nicht vertretbar. Ein weiterer beschränkender Faktor ist der Betriebsmittelbedarf bei der Auswertung eines Simulationsmodells. Vor allem die Rechenzeit, die für die Simulation umfangreicher Szenarien benötigt wird, beschränkt im praktischen Einsatz die Szenariengröße und -komplexität.

2.1.3 Messung

Unter den Methoden zur Leistungsanalyse nimmt die Messung einen besonderen Platz ein. Charakteristisches Merkmal der Messung ist es, dass nicht ein Modell analysiert wird, sondern das Original. Das eigentliche Testsubjekt ist Gegenstand der Messung, in unserem Fall also eine Implementierung einer verteilten Anwendung oder eines Netzprotokolls. Eine Motivation für Messungen an Implementierungen ist die beschränkte Aussagekraft von Simulationsmodellen. Beispielsweise haben Brakmo und Peterson 1996 gezeigt, dass sich gängige Simulationsmodelle für TCP nicht exakt wie reale TCP-Implementierungen in UNIX-Betriebssystemen verhalten [BP96a]. Die von den Autoren identifizierte Ursache war in diesem Fall, dass aus technischen Gründen, unter anderem durch eine beschränkte Zeitgebergranularität, die spezifizierten Algorithmen nicht exakt in der Implementierung umgesetzt werden konnten. Generell kann nur eine Messung mit Sicherheit Aussagen über ein Testsubjekt machen; jede Form der Modellierung des Testsubjekts ist prinzipiell angreifbar.

Für die Messung eines Testsubjekts muss eine geeignete Messumgebung zur Verfügung stehen. Um die Eigenschaften einer Software-Implementierung messen zu können, ist eine Ausführungsumgebung notwendig (Rechner, Betriebssystem etc.). Da wir ausschließlich Testsubjekte betrachten, die über ein Rechnernetz kommunizieren, hat es keinen Sinn, eine einzelne Instanz eines Testsubjekts isoliert zu betrachten. Vielmehr muss jede Messung mindestens zwei Kommunikationspartner berücksichtigen. Darüberhinaus muss eine Kommunikationsinfrastruktur zur Verfügung stehen, die der „Zielumgebung“ des Testsubjekts entspricht.

Messumgebungen zur Leistungsmessung von verteilten Anwendungen und Netzprotokollen werden häufig speziell nur für eine ganz bestimmte Messung eingerichtet. Es existiert eine Vielzahl von Veröffentlichungen, die spezielle Messumgebungen für jeweils ein Testsubjekt beschreiben. Da in diesen Arbeiten der Schwerpunkt auf den Messergebnissen und nicht auf der Messumgebung und der Messmethodik liegt, sind sie an dieser Stelle nicht von zentralem Interesse. Um einen Eindruck von der Methodik und den Problemen und Einschränkungen solcher Testumgebungen zu bekommen, werden in den folgenden Abschnitten exemplarisch eini-

ge typische Messaufbauten beschrieben, die als Kommunikationsinfrastruktur drahtlose lokale Netze (Wireless LAN, WLAN) betrachten.

Beispiel: WLAN-Messumgebungen

Caceres und Iftode beschreiben 1995 einen der ersten systematischen Versuche, den Einfluss von WLAN-typischen Netzcharakteristika auf Kommunikationsprotokolle zu messen [CI95]. In ihrem Versuchsaufbau bewegt sich ein mobiler Rechner mit WLAN-Karte durch den Abdeckungsbereich von zwei verschiedenen WLAN-Zugangspunkten (Access Points, APs). Die APs realisieren eine Brücke zu einem traditionellen LAN. An diesem LAN ist ein zweiter Rechner angeschlossen. Der mobile Rechner baut in dem Versuch eine TCP-Verbindung zu dem stationären Rechner auf. Während der Datenübertragung wird der mobile Rechner zwischen den Abdeckungsbereichen der APs hin- und hergetragen. Wenn die WLAN-Karte ihre Verbindung zwischen den APs wechselt (Handover), ist für kurze Zeit keine Kommunikation zwischen mobilem und stationärem Rechner möglich. Ziel des Versuchs war die Untersuchung des Einflusses dieser Unterbrechung auf die Leistung der TCP-Verbindung.

Rechnernetze aus spontanvernetzten mobilen Teilnehmern, so genannte MANETs (Mobile Ad Hoc Networks), sind in den letzten Jahren für die Protokollforschung und damit auch für die Leistungsanalyse besonders interessant geworden. Maltz et al. beschreiben 1999 ihre Messungen einer eigenen Implementierung des MANET-Vermittlungsprotokolls DSR (Dynamic Source Routing) [MBJ99]. Wiederum wird der Durchsatz von TCP gemessen; allerdings wird die Leistung von TCP in dieser Messung als indirekte Leistungsmetrik für das darunterliegende Vermittlungsprotokoll verwendet: Nach der Annahme der Autoren ist der Durchsatz von TCP ein Gradmesser für die Leistung des Vermittlungsprotokolls. Insbesondere soll die Geschwindigkeit der Routenanpassungen bei sich schnell bewegenden Teilnehmern beurteilt werden. Dazu wird ein Messaufbau mit sechs Fahrzeugen verwendet. In jedem Fahrzeug befindet sich ein tragbarer Rechner mit WLAN-Ausstattung. Während eines Messlaufs werden die sechs Fahrzeuge von Studenten über den Universitätscampus gefahren, um die möglichen Vernetzungstopologien ständig zu ändern. Zusätzlich sind zwei stationäre Teilnehmer vorhanden. Zu den methodischen Erfahrungen der Versuche gehört, dass allein der Versuchsaufbau großen logistischen und materiellen Aufwand verursacht. Bei den Bewegungsparametern mussten große Einschränkungen gemacht werden, allein aufgrund der Tatsache, dass nur wenige befahrbare Straßen in dem Gelände zur Verfügung standen. Allerdings betonen die Autoren, dass die Messungen trotz des Aufwands lohnende Ergebnisse gebracht haben. Eine der Schlussfolgerungen ist, dass es sich lohnen würde, eine künstliche Umgebung für Messungen solcher Art zu entwickeln.

Toh et al. beschreiben im Jahre 2000 eine ähnliche Messung [TCDA00]: Für Leistungsmessungen eines Vermittlungsprotokolls (ABR, „Associativity-Based Routing“) tragen fünf Studenten

Rechner mit WLAN-Ausstattung über den Campus. Während der Messungen werden die fünf Rechner allerdings nicht bewegt. Die Autoren sind unter anderem an der Zeit interessiert, die für das initiale Finden einer Route benötigt wird. Wie Maltz et al. betonen auch Toh et al., dass Messungen extrem zeitaufwändig sind. Trotzdem entscheiden sich die Autoren ausdrücklich gegen eine Analyse durch Simulation. Sie wollen Leistungsaspekte der ABR-Implementierung in der realen Umgebung messen, weil sie davon ausgehen, dass die gängigen Simulationsmodelle einige leistungsrelevante Attribute nicht realistisch nachbilden.

Lundgren et al. beschreiben 2002 ebenfalls einen Testaufbau für die Leistungsmessung von MANET-Vermittlungsprotokollen [LLN+02]. Die Messumgebung besteht aus 37 tragbaren Rechnern mit WLAN-Ausstattung. Die Autoren betonen vor allem, dass vergleichende Leistungsmessungen nur bei reproduzierbaren Umgebungsbedingungen möglich sind. Sie kritisieren, dass bei gängigen Messaufbauten die Bewegung der Teilnehmer nicht genau definiert sei. Daher führen sie eine Metrik für die Mobilität ein, und definieren einen exakten Bewegungsablauf für jeden der 37 Teilnehmer ihrer Messumgebung. Jeder mobile Rechner gibt während eines Experiments Bewegungsanweisungen auf dem Bildschirm aus. Mit Hilfe von 37 freiwilligen Helfern werden die Rechner dann entsprechend dieser Choreographie bewegt. Natürlich ist durch die Art der Ausführung die Genauigkeit des Bewegungsablaufs beschränkt.

Fazit: Messung

Messumgebungen werden oft speziell für ein bestimmtes Szenario eingerichtet. Am Beispiel von MANET-Messumgebungen wurde klar, dass die Vorbereitung und Durchführung von Messungen einen erheblichen materiellen, zeitlichen und oft auch personellen Aufwand darstellt. Ein besonders bei Messungen mit mobilen Teilnehmern auftretendes und bisher nicht zufriedenstellend gelöstes Problem ist die Wiederholbarkeit von Netzeigenschaften. Mit den vorgestellten Messaufbauten für MANETs sind keine zuverlässig wiederholbaren und damit wirklich vergleichbaren Messläufe möglich.

2.1.4 Diskussion

Zweifellos hat jede der drei dargestellten Methoden für die Leistungsanalyse von verteilten Anwendungen und Netzprotokollen ihre Anwendungsgebiete. Die mathematische Analyse kann in frühen Stadien der Protokollentwicklung grundsätzliche Zusammenhänge von wichtigen Parametern aufzeigen. Der Anwendungsbereich der mathematischen Analyse ist klar limitiert durch die begrenzte beherrschbare Komplexität der Modelle.

Simulationsmodelle können auch das Verhalten von komplexeren Algorithmen repräsentieren; Standardbibliotheken für Simulationswerkzeuge bieten die Chance, schnell komplexe Modelle

aus vielen vorhandenen Teilmodellen zusammenzustellen. Mit einer Serie von Simulationsläufen können die Auswirkungen eines Parameters auf die betrachteten Leistungseigenschaften isoliert betrachtet werden. Die Aussagekraft von Simulationen hängt stark von der Qualität der verwendeten Modelle ab. Darüberhinaus ist die Simulation ebenfalls durch die Komplexität der zu modellierenden Testsubjekte beschränkt.

Sollen explizit die Eigenschaften einer vorhandenen Implementierung einer Anwendung oder eines Protokolls analysiert werden, muss eine Messung durchgeführt werden. Aussagekräftige Messungen erfordern eine geeignete Messumgebung, mit deren Bereitstellung zwei Probleme verbunden sind: die Verfügbarkeit der Messumgebung und die Wiederholbarkeit der Netzeigenschaften. Eine Messumgebung mit speziellen Anforderungen an die Netztechnologie in der erforderlichen Größe zu beschaffen und zu betreiben, ist mit erheblichem logistischem und materiellem Aufwand verbunden. Darüber hinaus ist eine Messumgebung für die vergleichende Leistungsanalyse nur dann einsetzbar, wenn sie wiederholbare Bedingungen für mehrere Messläufe garantieren kann. Gerade im exemplarisch dargestellten Anwendungsbereich der drahtlosen mobilen Netzumgebungen ist die Wiederholbarkeit der Netzeigenschaften ein Problem, das in traditionellen Messumgebungen nicht zufriedenstellend gelöst werden kann.

Messung in emulierter Umgebung

Eine flexible Messumgebung mit nachgebildeten („emulierten“) Netzeigenschaften begegnet sowohl dem Problem der Verfügbarkeit, als auch der Anforderung nach Wiederholbarkeit der Netzeigenschaften. Eine Umgebung, die mit denselben, nur einmal beschafften Geräten eine große Vielfalt an möglichen Netzszenarien nachbilden kann, stellt eine eher zu rechtfertigende Investition dar, als eine spezielle Messumgebung für eine bestimmte Art von Messungen. Sind die Netzeigenschaften nicht von Umgebungseinflüssen abhängig, sondern für jeden Messlauf exakt steuer- und kontrollierbar, werden mehrere, vergleichbare Messläufe möglich.

Die Emulation von Netzeigenschaften stellt also keine grundsätzlich neue Methode zur Leistungsanalyse von verteilten Anwendungen und Netzprotokollen dar, sondern erweitert die existierende Methode der Messung um die Möglichkeit der Messung in einer nachgebildeten Umgebung.

2.2 Verwandte Arbeiten

In diesem Abschnitt werden die existierenden Arbeiten zur Emulation von Netzeigenschaften klassifiziert, sowie ihre Möglichkeiten und Beschränkungen diskutiert. Schließlich wird der Beitrag der vorliegenden Arbeit motiviert.

Alle hier vorgestellten Ansätze beeinflussen Netzeigenschaften in definierter Weise, jedoch auf unterschiedliche Art. Zunächst besprechen wir Lösungen, die auf modifizierten Netzwerkgeräten basieren. Danach wenden wir uns Lösungen zu, die ausschließlich Software-Werkzeuge benutzen. Dabei stellen wir zunächst Werkzeuge für die Emulation von genau einer Netzverbindung vor, und gehen abschließend auf die Emulation von Netzszenarien ein, die aus mehr als einer Netzverbindung bestehen.

2.2.1 Hardware-basierte Emulation

Die Idee von Hardware-basierter Emulation ist es, den nötigen Aufwand und das Wiederholbarkeitsproblem von Messungen zu verringern, indem man mit spezieller Hardware in definierter Weise Einfluss auf gewisse Netzeigenschaften nimmt. Da hierbei Geräte eingesetzt werden, die in der physischen Schicht (Bitübertragungsschicht) arbeiten, sind sie jeweils speziell auf eine Netztechnologie zugeschnitten. In diesem Abschnitt werden beispielhaft einige Arbeiten vorgestellt, die sich mit Hardware-basierter Emulation von WLAN-Szenarien befassen.

Kaba und Raichle beschreiben einen Versuchsaufbau, mit dessen Hilfe die geometrischen Ausmaße einer WLAN-Testumgebung von Campusgröße auf Laborgröße verkleinert werden können [KR01]. Dazu werden die in den WLAN-Geräten integrierten Antennen teilweise abgeschirmt und damit die Reichweite von einigen hundert Metern auf wenige Zentimeter verringert. Schon mit dieser einfachen Modifikation kann so der Messaufwand für einige Standardszenarien deutlich reduziert werden. In einem weiteren Schritt schlagen die Autoren vor, auf die Funkausbreitung komplett zu verzichten, und verbinden die WLAN-Geräte direkt über Antennenkabel. Mit zwischengeschalteten Kombinations- und Dämpfungsgliedern kann die Signalausbreitung definiert geändert werden. So werden die Eigenschaften der Wellenausbreitung, die direkten Einfluss auf die Netzeigenschaften haben, steuer- und damit wiederholbar.

Hernandez und Helal beschreiben einen ähnlichen Ansatz, bei dem ebenfalls die Antennensignale durch steuerbare Dämpfungsglieder beeinflusst werden [HH02]. Die Arbeit befasst sich außerdem mit der Frage, wie realistische Sollwerte für die Dämpfungsglieder ermittelt werden können.

Im Versuchsaufbau von Judd und Steenkiste werden ebenfalls direkt die Antennensignale von WLAN-Geräten beeinflusst [JS03]. Allerdings werden die Signale jedes Geräts zunächst digitalisiert, in digitaler Form von spezieller Hardware verarbeitet und wieder in ein analoges Antennensignal gewandelt. Durch diesen Versuchsaufbau ist es möglich, an zentraler Stelle Dämpfung und Interferenzen für ein ganzes Szenario einzustellen. Abhängig von der Leistungsfähigkeit der zentralen Komponente können die Netzeigenschaften von WLAN-Szenarien bis zu einer gewissen Größe wiederholbar emuliert werden. Die Autoren schätzen, dass ein aktueller digitaler Signalprozessor Signale von bis zu 50 Antennen verarbeiten könnte.

Fazit: Hardware-basierte Emulation

Wie wir am Beispiel von WLAN-Testumgebungen gesehen haben, kann Hardware-basierte Emulation den Aufwand für traditionelle Messungen deutlich verringern. Wenn es zusätzlich die Möglichkeit gibt, relevante Netzeigenschaften nach einer Szenariovorgabe zu steuern, sind wiederholbare Messläufe möglich. Damit ist die Grundlage für vergleichende Leistungsanalyse geschaffen. Der Nachteil Hardware-basierter Emulationsumgebungen ist jedoch, dass sie speziell auf eine bestimmte Netztechnologie zugeschnitten und auf besondere Hardware angewiesen sind. Die Möglichkeiten einer solchen Umgebung sind somit stark eingeschränkt.

2.2.2 Software-basierte Emulation einer Netzverbindung

Auch Software-basierte Emulation hat zum Ziel, Netzeigenschaften definiert zu beeinflussen. Da der Eingriff jedoch nicht in der Bitübertragungsschicht, sondern in höheren Schichten des Kommunikationsstapels erfolgt, ist keine spezielle Hardware notwendig. Die verwendeten „Emulationswerkzeuge“ sind vielmehr reine Software-Werkzeuge. Wir betrachten zunächst Werkzeuge, die sich auf die Emulation der Eigenschaften einer einzigen Netzverbindung beschränken.

Zhang und Bhargava schlagen 1993 vor, den Netzverkehr zwischen zwei Rechnern in einem LAN abzufangen und über einen entfernten Rechner zu leiten [ZB93]. Die dadurch entstehende Verzögerung wird für die Leistungsanalyse von verteilten Transaktionsprotokollen genutzt. In einer Erweiterung wird zunächst die Verzögerung zu einem entfernten Rechner gemessen. In einem zweiten Schritt wird diese Verzögerung dann von einem Emulationswerkzeug in einem lokalen Testaufbau künstlich eingefügt. Das Werkzeug arbeitet in diesem Fall auf einem dritten Rechner, der zwischen die beiden kommunizierenden Rechner eingefügt wird.

Ingham und Parrington stellen 1994 mit „Delayline“ ein Emulationswerkzeug vor, das nicht nur die Verzögerung, sondern auch mögliche Rahmenverluste einer WAN-Verbindung nachbilden kann [IP94]. Es wird auf demselben Rechner installiert, der auch die Anwendung ausführt, die analysiert werden soll. Das Werkzeug stellt sich der Anwendung als Transportschicht-Implementierung dar und kann über eine eigene Socket-Schnittstelle angesprochen werden. Anwendungen, die eine emulierte Verbindung zur Kommunikation verwenden sollen, müssen statt der vom Betriebssystem bereitgestellten Sockets die Socket-Implementierung des Emulationswerkzeugs verwenden. Die zu analysierenden Anwendungen müssen also für die Messung angepasst werden.

Ahn et al. schlagen 1995 vor, direkt in den Kommunikationsstapel des Betriebssystems einzugreifen [ADLY95]. Für die Leistungsanalyse von TCP Vegas wird ein Emulationswerkzeug in das Betriebssystem FreeBSD integriert. Das Werkzeug kann ausgehende Rahmen zwischen der Vermittlungs- und der Sicherungsschicht abfangen und verändern. Neben einer zusätzlichen

Verzögerung und einer Rahmenverlustrate kann eine Begrenzung der Übertragungsrate eingeführt werden. Durch die Integration mit den Protokollimplementierungen des Betriebssystems ist das Emulationswerkzeug transparent für die logisch darüberliegenden Protokollschichten. Ein ähnlich arbeitendes Werkzeug für Linux-Betriebssysteme wird später von Carson und Santay vorgestellt [CS03].

Für die Emulation von WAN-Verbindungen mit einem Windows-Betriebssystem existiert das kommerzielle Emulationswerkzeug „The Cloud“ [Shu05]. Neben einer konstanten Verzögerung und einer begrenzten Übertragungsrate fügt „The Cloud“ zusätzlich für jedes Paket eine individuelle, zufällige Verzögerungskomponente hinzu, die durch statistische Funktionen definiert werden kann (Gleich- und Normalverteilung). Dieses Verhalten soll die wechselnden Füllstände von Vermittlungswarteschlangen nachbilden.

Allman et al. beschreiben 1997 den „Ohio Network Emulator“, ein Emulationswerkzeug, das verschiedene Arten von Verzögerung nachbilden kann [ACO97]. Das Werkzeug läuft auf einem dedizierten Rechner mit zwei Netzschnittstellen, der als Vermittler zwischen zwei Subnetzen fungiert. Das Werkzeug ist logisch am Übergangspunkt dieser Subnetze positioniert. Sämtlicher Netzverkehr, der über den Vermittler geleitet wird, ist den zusätzlichen Verzögerungen unterworfen; es ist also nicht nötig, die zu analysierenden Anwendungen oder Protokolle zu ändern. Allerdings wird stets ein zusätzlicher Rechner für das Emulationswerkzeug benötigt.

Luigi Rizzo stellt 1997 mit „Dummynet“ erstmals ein Werkzeug vor, das im Kommunikationsstapel des Betriebssystems arbeitet und dabei sowohl in Sende- als auch in Empfangsrichtung Verkehr verarbeiten kann [Riz97]. Dummynet ist eine Erweiterung des Betriebssystems FreeBSD, und fängt den Netzverkehr zwischen der Transportschicht und der Vermittlungsschicht ab. Verzögerung, Übertragungsrate, Warteschlangengröße und Paketverlustrate können an dieser Stelle eingestellt werden.

Noble et al. kombinieren in ihrer Arbeit aus 1997 die Messung von dynamischen Netzeigenschaften eines drahtlosen Netzes mit der Nachbildung der Eigenschaften durch ein Emulationswerkzeug [NSNK97]. Während einer Aufzeichnungsphase werden Verzögerung und Rahmenverlustrate zwischen einem mobilen Teilnehmer und mehreren WLAN-Basisstationen gemessen. Später werden die aufgezeichneten Netzeigenschaften im Labor durch ein Emulationswerkzeug nachgebildet, das im Kommunikationsstapel des Betriebssystems NetBSD zwischen der Vermittlungs- und der Sicherungsschicht eingreift. Neu ist hier vor allem die Idee, dass die Netzeigenschaften, die ein Emulationswerkzeug nachbildet, während eines Messlaufs ständig geändert werden.

Kojo et al. stellen 2001 ein Emulationswerkzeug vor, das speziell die Eigenschaften einer drahtlosen Mobilfunk-Datenverbindung (z.B. GPRS) nachbildet [KGM+01]. Wie bei [ZB93] und [ACO97] wird das Emulationswerkzeug auf einem dedizierten Rechner ausgeführt, der zwischen zwei Kommunikationspartner geschaltet wird. Der Beitrag der Arbeit ist vor allem die

Modellierung von besonderen Eigenschaften einer Mobilfunk-Verbindung wie den plötzlich auftretenden Rahmenverlusten bei einem Zellenwechsel, stark asymmetrischen Übertragungsraten etc.

Fazit: Software-basierte Emulation einer Netzverbindung

Gängige Emulationswerkzeuge können die direkte Kommunikation zwischen zwei Rechnern beeinflussen, indem sie Verzögerungen einführen, die Übertragungsrate begrenzen und Rahmen- bzw. Paketverluste verursachen. Damit sind die leistungsrelevanten Eigenschaften einer Punkt-zu-Punkt-Verbindung abgedeckt. Es gibt verschiedene Methoden, um den Netzverkehr abzufangen; es hat sich jedoch gezeigt, dass der direkte Eingriff im Kommunikationsstapel des Betriebssystems vorteilhaft ist, weil dadurch das Werkzeug für logisch darüberliegende Protokollimplementierungen und Anwendungen transparent ist.

2.2.3 Software-basierte Emulation von Netzszenarien

Die bisher vorgestellten Emulationswerkzeuge können die Netzeigenschaften einer Verbindung zwischen zwei Kommunikationspartnern nachbilden. Für die Analyse vieler Testsubjekte sind jedoch Szenarien mit mehr als zwei Kommunikationspartnern notwendig; etwa zur Beurteilung der Leistungsfähigkeit eines Vermittlungsprotokolls oder einer hierarchisch organisierten verteilten Anwendung. In den folgenden beiden Abschnitten werden Software-Lösungen für die Nachbildung von Netzszenarien mit mehr als zwei Kommunikationspartnern vorgestellt. Die existierenden Arbeiten lassen sich in zentralisierte und verteilte Verfahren klassifizieren.

Zentralisierte Verfahren

Zentralisierte Verfahren für die Emulation von Netzszenarien haben gemeinsam, dass der gesamte Netzverkehr aller Kommunikationspartner in einem Szenario über eine einzige Instanz eines Emulationswerkzeugs geleitet wird.

Davies et al. stellen 1995 erstmalig ein zentralisiertes Emulationswerkzeug vor [DBCF95]. Anwendungen, die ausschließlich UDP-Pakete zur Kommunikation benutzen, senden ihre Pakete nicht direkt zum eigentlichen Empfänger, sondern zunächst zum Emulationswerkzeug. Dieses entscheidet, ob das Paket nicht, verzögert oder sofort dem Empfänger zugestellt wird. Bei dieser Entscheidung kann prinzipiell auch eine Rolle spielen, ob in unmittelbarer zeitlicher Nähe ein anderes Paket durch das Werkzeug verarbeitet wurde. So könnten Effekte wie kollidierende Rahmen nachgebildet werden. Allerdings ist die Modellierung von Kollisionen problematisch, da das Werkzeug keine echte Gleichzeitigkeit erkennen kann: Aus der Sicht des Werkzeugs kommen die UDP-Pakete der verschiedenen Sender immer hintereinander an, auch wenn sie

gleichzeitig abgesendet wurden, denn die Netztechnologie des Emulationssystems serialisiert die Rahmen zwangsläufig. Außerdem kann der exakte Empfangszeitpunkt eines Rahmens von ein Werkzeug, das auf Anwendungsschicht arbeitet, nicht zuverlässig ermittelt werden.

Raman und Katz benutzen ebenfalls ein zentralisiertes Emulationswerkzeug auf UDP-Basis [RK02]. Das Werkzeug bildet den Verlust und die Verzögerung von Paketen nach. Um realistische Netzparameter zu erhalten, führen die Autoren Referenzmessungen über tatsächliche WAN-Verbindungen durch.

In Abschnitt 2.1.2 wurde bereits ns-2 als wichtiges Simulationswerkzeug vorgestellt [BEF+00]. Kevin Fall schlägt 1999 vor, eine erweiterte Version von ns-2 („nse“) als zentralisiertes Werkzeug für die Emulation von Netzeigenschaften zu verwenden [Fal99]. Dazu muss die ereignisdiskrete Verarbeitung des Simulators mit der Realzeit synchronisiert werden, es wird also eine Abbildung von Simulationszeit auf Realzeit definiert. Während eines Experiments werden alle Ereignisse im Simulationsmodell abgearbeitet, die in der Vergangenheit liegen. Wenn das nächste Ereignis in der Zukunft liegt, wird der Simulationsprozess solange unterbrochen, bis die Realzeit das nächste Ereignis „eingeholt“ hat. Außerdem kann das Simulationsmodell über eine Schnittstelle mit der realen Welt verbunden werden: An definierten Übergangspunkten des Modells können reale Pakete, die am Simulationsrechner ankommen, in Modellrepräsentationen von Paketen umgewandelt und so Teil des Simulationsmodells werden. In der anderen Kommunikationsrichtung werden an denselben Übergangspunkten Pakete aus dem Simulationsmodell in tatsächliche Pakete umgewandelt und über eine reale Netzwerkkarte weitergeschickt. Damit dieser Übergang verlustfrei funktioniert, ist auch das Paketmodell von nse gegenüber dem von ns-2 erweitert, beispielsweise um eine Repräsentation von Nutzdaten.

Durch die umfangreichen Bibliotheken, die für ns-2 zur Verfügung stehen, wird es mit der Erweiterung nse auch zu einem mächtigen Emulationswerkzeug. Beispielsweise benutzen Ke et al. das ns-2-Modell eines Ad-Hoc-Netzes, um über die Emulationsschnittstelle mehrere reale Benutzer eines Coda-Dateisystems anzuschließen [KMJ00]. Der Ansatz, nse als zentralisiertes Emulationswerkzeug für Netzszenarien zu verwenden, unterliegt zwei grundsätzlichen Einschränkungen: Einerseits muss die zentrale Simulationskomponente das verwendete Modell schnell genug verarbeiten können, damit sie mit der Realzeit synchron bleiben kann. Dies limitiert die Komplexität des Modells im Hinblick auf die Anzahl der Kommunikationspartner und den Netzverkehr. Andererseits werden „Artefakte“, also unerwünschte Netzeigenschaften eingeführt, welche den Realismus der Modelle beeinflussen: Durch die Tatsache, dass jede Sendung zwischen zwei Kommunikationspartnern über eine dritte Instanz geleitet wird, entsteht eine zusätzliche, unerwünschte Verzögerung. Weitere, nichtdeterministische Verzögerungen können durch das Rechnernetz entstehen, über das der zentrale Simulationsrechner angeschlossen ist (beispielsweise bei der Serialisierung gleichzeitiger Sendungen). Weil die tatsächlichen Sendezeitpunkte nicht exakt bekannt sind, ist es nicht zuverlässig möglich, Effekte wie Kollisionen von überlappenden Sendungen im Modell zu berücksichtigen.

Mahrenholz und Ivanov verringern die von nse unerwünscht eingeführten Verzögerungen u.a. dadurch, dass alle Testsubjekte und das zentrale Emulationswerkzeug in unterschiedlichen Ausführungsumgebungen desselben Rechners ausgeführt werden [MI04, MI05]. Dadurch wird allerdings die mögliche Szenariogröße noch weiter beschränkt.

Lin et al. stellen 2002 einen mit nse verwandten Ansatz vor [LMP02]. Das zentrale Emulationswerkzeug arbeitet allerdings im Betriebssystemkern und basiert auf einem einfachen Modell, das als Netzparameter nur Übertragungsrate und Paketverlust, aber keine Kollisionen berücksichtigt. Die Autoren betonen, dass ihr Modell wesentlich weniger aufwändig und damit performanter ist als das bei nse verwendete. Dadurch kann die ungewollt eingeführte Verzögerung gegenüber nse verkleinert werden. Die Geschwindigkeit des Werkzeugs wird auch dadurch erreicht, dass die gesamte Paketverarbeitung direkt im Betriebssystemkern erfolgt.

Auch Flynn et al. benutzen ein zentrales Emulationswerkzeug für die Emulation von Netzszenarien. Das Werkzeug ist in der Sprache Java realisiert [FTO02]. Durch eine gröbere Abstraktion bei der Modellierung der Protokolle erreicht es laut den Autoren eine bessere Leistung als nse.

Vahdat et al. beschreiten einen anderen Weg, um die Leistung einer zentralen Emulationsinstanz zu erhöhen: Sie verteilen das Emulationswerkzeug auf mehrere Rechner [VYW+02, MYV03]. Logisch betrachtet sind noch immer mehrere Endgeräte an eine einzige Instanz angeschlossen, welche die Netzeigenschaften des Szenarios nachbildet. Diese Instanz besteht jedoch aus bis zu zehn Rechnern, die jeweils nur für einen Teil des Szenarios zuständig sind. Damit die Vermittlung der Pakete sowohl innerhalb einer Partition, als auch zwischen mehreren Partitionen eines Szenarios funktioniert, können bei diesem Ansatz keine Originalimplementierungen von Vermittlungsprotokollen verwendet werden. Vielmehr betonen die Autoren, dass die Eigenschaften mehrerer Vermittlungsprotokolle speziell für die Verwendung mit dem Emulationswerkzeug nachprogrammiert wurden. Dadurch ist die Analyse von existierenden Vermittlungsprotokollen mit diesem Ansatz nicht möglich.

Fazit: zentralisierte Emulation von Netzszenarien

Eine zentralisierte Emulation von Netzszenarien hat den Vorteil, dass die Steuerung eines kompletten Szenarios an nur einer Stelle erfolgt. Auch komplexere Szenarien können mit überschaubarem Aufwand eingerichtet werden. Ein gemeinsamer Nachteil aller zentralisierter Verfahren ist es allerdings, dass die zentrale Instanz den möglichen Netzverkehr in einem Szenario beschränkt. Außerdem können Effekte, die einen sehr genauen Zeitbegriff erfordern, etwa das zeitliche Überlappen zweier Rahmen, von einer zentralisierten Instanz auf Anwendungsebene nur unzuverlässig oder gar nicht festgestellt werden.

Verteilte Verfahren

In Abschnitt 2.2.2 wurden Emulationswerkzeuge vorgestellt, welche die Eigenschaften einzelner Netzverbindungen nachbilden können. Um Szenarien mit mehreren Netzverbindungen nachzubilden, liegt es daher nahe, in einer Testumgebung mehrere Instanzen solcher Werkzeuge auf mehreren Rechnern zusammenzuschalten.

Das „Utah Network Testbed“ besteht aus 168 Rechnern, die entweder als Ausführungsplattform für Testsubjekte oder als zwischengeschaltete „Emulationsknoten“[3] verwendet werden können [WLS+02]. Auf einem Emulationsknoten wird Dummynet [Riz97] oder nse [Fal99] als Emulationswerkzeug eingesetzt. Das gewünschte Szenario wird von einer zentralen Instanz automatisch eingerichtet. Auch während eines Experiments können die Netzeigenschaften des Szenarios noch geändert werden, um beispielsweise das Ausfallen einer Verbindung nachzubilden. Da die Vernetzung der 168 Knoten in der Testumgebung nicht homogen ist, stellt es ein nichttriviales Problem dar, eine optimale Abbildung von den logischen Knoten eines spezifizierten Szenarios auf die physischen Knoten der Testumgebung zu berechnen. Ein wichtiger Beitrag der Autoren ist eine effiziente näherungsweise Lösung dieses im allgemeinen Fall NP-harten Problems [RAL03].

Eine Erweiterung des Utah Network Testbeds bezieht drahtlos angebundene Knoten in die Testumgebung mit ein [WLG02]. Allerdings werden die Netzeigenschaften der Funkstrecken nicht emuliert. Stattdessen können einige reale Funkstrecken, die an der University of Utah für diesen Zweck eingerichtet wurden, in ein Emulationsszenario eingebunden werden. In diesem Fall wird anhand der in einer Szenariobeschreibung spezifizierten gewünschten Eigenschaften einer Funkstrecke aus den tatsächlich vorhandenen Funkstrecken diejenige ausgesucht und eingebunden, die der Spezifikation am nächsten kommt.

„MobiEmu“ ist eine Testumgebung, die speziell für die Emulation von dynamischen Topologien in MANETs konzipiert wurde [ZL02]. Die Testsubjekte laufen auf mehreren Knotenrechnern, die von einer zentralen Instanz auf einem Steuerungsrechner koordiniert werden. Die Steuerungsinstanz benutzt ein Mobilitätsszenario, um die Erreichbarkeit der Knoten untereinander zu berechnen. Dabei wird von einer fixen Funkreichweite ausgegangen, bis zu der Kommunikation möglich ist. *Vor* dem Messlauf wird jedem Knotenrechner eine Liste mit Zeitstempeln und Änderungen in seiner Erreichbarkeit übergeben. *Während* des Messlaufs sendet die Steuerungsinstanz dann nur noch Zeitstempel an die Knotenrechner, um die Änderungen zeitlich zu koordinieren. Statt eines dedizierten Emulationswerkzeugs wird auf den einzelnen Rechnern die ohnehin vorhandene Linux-Betriebssystemfunktion „IPTables“ benutzt, die es erlaubt, Netzverkehr von bestimmten Sicherungsschicht-Adressen zuzulassen oder zu sperren.

[3] In Anlehnung an die Darstellung einer Netzverbindungstopologie als Graph aus Knoten und Kanten wird oft der Begriff „Knoten“ als Synonym für „Rechner als Teil einer Netztopologie“ verwendet.

Damit bleibt die Erreichbarkeit die einzige Netzeigenschaft, die nachgebildet werden kann. Verzögerung, Übertragungsrate oder Verlust werden nicht betrachtet.

Liu und Song stellen eine ähnlich strukturierte Testumgebung vor, bei der ebenfalls ein zentraler Steuerungsrechner die Erreichbarkeit der Knoten in einem MANET-Szenario ermittelt [LS02]. Der Steuerungsrechner ist über Ethernet an mehrere Knotenrechner angeschlossen, welche die Knoten des MANETs repräsentieren. Während eines Messlaufs wird die Erreichbarkeitsinformation ständig über Ethernet-Rundsendungen an alle Rechner übermittelt. Auf jedem Knotenrechner arbeitet ein Emulationswerkzeug, das neben der dynamischen Erreichbarkeit auch eine fest eingestellte Verzögerung und eine begrenzte Übertragungsrate nachbilden kann. Weitergehende Effekte wie Kollisionen werden nicht betrachtet.

„EMPOWER“ ist ein verteilter Emulationsansatz für Netzszenarien, der ohne zentrale Steuerung auskommt [ZN03]. Auch hier werden mehrere Rechner mit einem Emulationswerkzeug ausgestattet, das wechselnde Verbindungstopologien nachbilden kann. Die Parameter für jeden Rechner werden jedoch nicht während des Experiments von einer zentralen Instanz gesteuert, sondern im Voraus berechnet und als Ablaufplan auf die Rechner verteilt. Während des eigentlichen Experiments werden die geplanten Parameteränderungen durch das Emulationswerkzeug auf jedem Rechner anhand des Ablaufplans selbstständig durchgeführt. Damit das Szenario hierbei konsistent bleibt, sind synchronisierte Uhren notwendig. Als Parameter werden dynamische Erreichbarkeit, Verzögerung, Übertragungsrate und Rahmenverlust nachgebildet. Kollisionen werden nicht berücksichtigt. Das verwendete Werkzeug greift zur Nachbildung der dynamischen Erreichbarkeit in die Vermittlungsschicht ein, sodass die Vermittlungsschicht nicht als Testsubjekt in Frage kommt, sondern nur höhere Schichten.

Fazit: verteilte Emulation von Netzszenarien

Durch den koordinierten parallelen Einsatz mehrerer Instanzen eines Emulationswerkzeugs können auch größere Netzszenarien nachgebildet werden. Die Leistungsfähigkeit der vorgestellten Ansätze ist vor allem von den Möglichkeiten der eingesetzten Werkzeuge abhängig. Existierende Testumgebungen können grundlegende Netzeigenschaften wie Topologieänderungen, Verzögerungen, begrenzte Übertragungsrate und Rahmenverlust nachbilden; Interaktionseffekte mehrerer Rahmensendungen (Kollisionen, konkurrierender Zugriff) wurden bisher nicht betrachtet.

2.3 Diskussion

Die Vielzahl der in den letzten Jahren veröffentlichten Arbeiten im Bereich der Emulation von Netzeigenschaften lässt darauf schließen, dass neben den etablierten Methoden Simulation

und Messung in der Realumgebung die Messung in einer emulierten Netzumgebung auf immer größeres Interesse stößt. Im Bereich der Emulationswerkzeuge haben sich eindeutig die Software-basierten Werkzeuge als Grundlagentechnologie durchgesetzt, weil sie mit wesentlich weniger Aufwand einsetzbar sind als Hardware-basierte Werkzeuge. Auf diese Werkzeuge aufbauend haben sich zwei alternative Möglichkeiten entwickelt, um ganze Szenarien nachzubilden, die sich in der Anordnung der verwendeten Werkzeuge unterscheiden. Zentralisierte Verfahren leiten den gesamten Netzverkehr eines Szenarios über eine zentrale Instanz eines Emulationswerkzeugs, während verteilte Verfahren mehrere Instanzen verwenden, die jeweils nur für einen Teil des Szenarios zuständig sind. Messungen mit zentralisierten Werkzeugen sind administrativ einfacher durchzuführen, jedoch in der möglichen Szenariogröße beschränkt. Verteilte Architekturen ermöglichen größere Szenarien, erfordern aber einen wesentlich größeren Aufwand für die Koordination der Werkzeuge. Zentrale wie verteilte Verfahren sind in ihrer Mächtigkeit durch die Möglichkeiten der verwendeten Emulationswerkzeuge beschränkt.

Für die verteilte Emulation von Szenarien mit mehreren, unabhängigen Netzverbindungen existieren schon einige Ansätze. Je nach verwendetem Werkzeug können gewisse Eigenschaften solcher Verbindungen nachgebildet werden. Zu Beginn unserer Forschungsarbeiten im Jahre 2001 war die verteilte Emulation von *dynamischen* Netzszenarien noch völlig unbekannt; zu den zeitlich überlappenden ersten Veröffentlichungen zu dynamischen Netzszenarien gehören unsere eigenen Arbeiten ([ZL02, LS02, HLR02, HR02]). Bis heute bietet kein anderes existierendes Verfahren die realistische Nachbildung aller leistungsrelevanter Netzparameter in einem Szenario. Beispielsweise blieben Seiteneffekte beim Senden durch die Reduzierung der Übertragungsrate (Rückstau von Rahmen) von den existierenden Werkzeugen unberücksichtigt, obwohl dieses Verhalten Einfluss auf die Leistung von Protokollen und Anwendungen hat. Außerdem können mit keinem der bisher existierenden Verfahren die Effekte von Netztechnologien mit gemeinsamen Medien realistisch emuliert werden.

Ziel dieser Arbeit ist es daher, ein vielseitig einsetzbares Emulationsverfahren für Rechnernetze zu entwickeln, das sowohl skalierbar ist, als auch wichtige Netztechnologien realistisch nachbilden kann. Dabei sollen explizit Technologien mit gemeinsamen Medien und dynamischen Netzeigenschaften eingeschlossen sein. Dazu wird zunächst untersucht, mit welcher Architektur die Anforderungen an Skalierbarkeit, Realismus und Einsetzbarkeit erfüllt werden können, und welche Netzeigenschaften und Seiteneffekte nachgebildet werden müssen. Danach wird ein Verfahren entwickelt, mit dem es möglich ist, die geforderten Netzeigenschaften und Seiteneffekte realistisch nachzubilden.

3

Architektur

In den ersten beiden Kapiteln wurde die Motivation für ein vielseitig einsetzbares und skalierbares Emulationsverfahren für Rechnernetze deutlich. In diesem Kapitel wird zunächst argumentiert, auf welcher konzeptionellen Ebene ein Emulationsverfahren in die Rechnerkommunikation eingreifen sollte. Danach werden die Netzparameter identifiziert, die nachgebildet werden müssen. Auf dieser Grundlage werden verschiedene Verteilungsalternativen für eine Lösung vorgestellt und bewertet.

3.1 Einordnung in das Schichtenmodell

Um Netzverkehr zu beeinflussen, muss jedes Emulationsverfahren in den Protokollstapel eingreifen bzw. diesen modifizieren. In Abschnitt 2.2.2 wurde bereits deutlich, dass dieser Eingriff auf verschiedenen konzeptionellen Ebenen erfolgen kann. In diesem Abschnitt wird erklärt, welche Ebene für unsere Aufgabenstellung am besten geeignet ist.

Es sind verschiedene Referenzmodelle für Protokollstapel in Gebrauch. In Anlehnung an das Standardlehrbuch von Andrew S. Tanenbaum [Tan96] verwenden wir im Folgenden ein vereinfachtes Fünf-Schichten-Modell.[1] Abb. 3.1 zeigt, wie die fünf Schichten üblicherweise umgesetzt werden: Die Bitübertragungsschicht und ein Teil der Sicherungsschicht (z.B. die Medienzugriffssteuerung, MAC) sind in Hardware realisiert. In einem typischen LAN entspricht dies den Funktionen der Netzwerkkarte, der Verkabelung und ggf. weiterer Infrastrukturkomponenten. Optional wird ein zusätzlicher Teil der Sicherungsschicht in Software implementiert

[1] Das Fünf-Schichten-Modell ergibt sich aus dem OSI-Referenzmodell, wenn man die beiden selten explizit realisierten Schichten fünf (Sitzungsschicht) und sechs (Darstellungsschicht) vernachlässigt. Es stellt damit einen Kompromiss aus dem siebenschichtigen OSI-Referenzmodell und dem vierschichtigen TCP/IP-Referenzmodell dar.

(z.B. „Logical Link Control“, LLC). Die darüberliegenden Protokollschichten, Vermittlungsschicht und Transportschicht, sind als Teil des Betriebssystems implementiert. Schließlich folgen die Anwendungen, welche die Funktionen der Transportschicht über eine Betriebssystem-Schnittstelle (meist „Sockets“) benutzen.

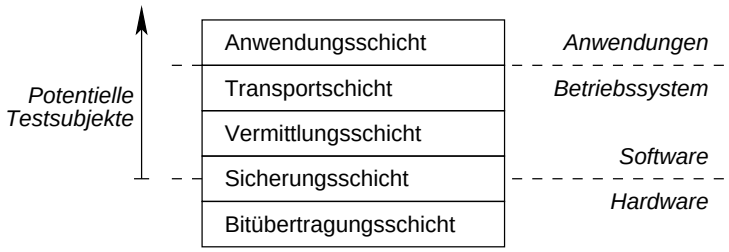

Abbildung 3.1: Vereinfachtes Schichtenmodell

Im Rahmen dieser Arbeit sind sowohl verteilte Anwendungen, als auch Software-Implementierungen von Netzprotokollen als Testsubjekte zu betrachten. Um ein Testsubjekt geänderten Netzparametern auszusetzen, muss ein Emulationsverfahren konzeptionell *unter* dem Testsubjekt eingreifen. Sollen also Testsubjekte bis hinunter zur Vermittlungsschicht (bzw. dem in Software realisierten Teil der Sicherungsschicht) unterstützt werden, muss ein in Software realisiertes Emulationsverfahren die tiefstmögliche Software-Schicht im Protokollstapel bilden, und somit direkt auf der Netz-Hardware aufsetzen (Abb. 3.2). Dies bedeutet die Einführung einer zusätzlichen *Emulationsschicht* innerhalb der Sicherungsschicht-Implementierung, welche dieselbe Dienstabstraktion wie die darunterliegende (Hardware-)Schicht bietet. In den meisten praxisrelevanten Fällen ist die von der Netz-Hardware angebotene Dienstabstraktion ein unzuverlässiger, verbindungsloser Dienst. So wird auch die von uns eingeführte Emulationsschicht diese Dienstabstraktion verwenden [HR02].

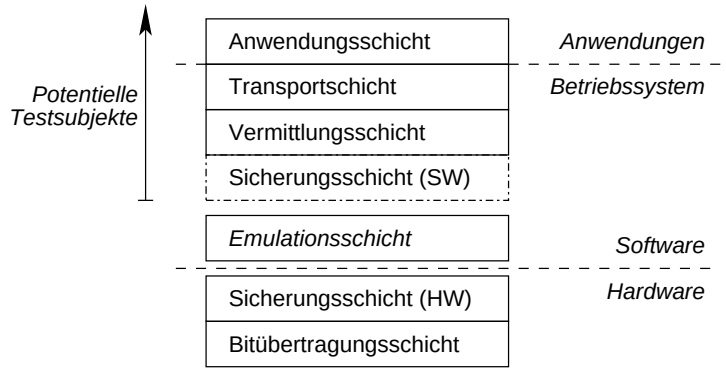

Abbildung 3.2: Einordnung der Emulationsschicht im Protokollstapel

3.2 Nachzubildende Netzeigenschaften

Aufgabe der Emulationsschicht ist es, die Eigenschaften einer fiktiven Netz-Hardware nachzubilden. Dabei bedient sie sich der Dienste der tatsächlich vorhandenen Netz-Hardware, deren Eigenschaften somit für die Schichten oberhalb der Emulationsschicht maskiert werden. Die Emulationsschicht bietet der darüberliegenden Schicht einen unzuverlässigen, verbindungslosen Dienst zur Übermittlung von Rahmen an direkt benachbarte (d.h. über die Sicherungsschicht erreichbare) Kommunikationspartner an.

Die prinzipiell möglichen Eigenschaften dieses Dienstes ergeben sich direkt aus der Definition der Dienstabstraktion, die von der Emulationsschicht realisiert wird: Ein zur Übermittlung übergebener Rahmen wird entweder fehlerfrei oder überhaupt nicht übertragen, denn fehlerhaft übertragene Rahmen werden beim Empfang erkannt und nicht ausgeliefert. Außerdem nimmt die Übertragung eines Rahmens eine gewisse Zeit in Anspruch. Damit sind die beiden *Basiseffekte* klar, welche die Emulationsschicht nachbilden kann: Rahmenverlust und Verzögerung. Wie wir im Folgenden sehen werden, lassen sich alle weiteren Eigenschaften der nachzubildenden Netz-Hardware auf diese beiden Basiseffekte zurückführen.

Da die Emulationsschicht auf existierender Netz-Hardware aufsetzt, die selbst gewisse Eigenschaften hat, kann sie nicht völlig beliebige Netzeigenschaften nachbilden. Vielmehr kann sie nur *zusätzliche Rahmenverluste* und *zusätzliche Verzögerungen* einfügen. Um hier einen möglichst großen Parameterbereich abdecken zu können, ist es also notwendig, ein reales Emulationssystem mit einer sowohl *zuverlässigen* als auch *möglichst schnellen* Netztechnologie auszustatten. Diese Forderungen werden in Kapitel 4 konkretisiert. Eine mögliche Realisierung wird in Abschnitt 5.1 vorgestellt.

3.2.1 Basiseffekt: Rahmenverlust

Der häufigste Grund für einen Rahmenverlust ist eine physikalische Störung des Übertragungsmediums. Ein Rahmen mit mindestens einem fehlerhaft übertragenen Bit wird von der Sicherungsschicht des Empfängers mit großer Wahrscheinlichkeit anhand der Prüfsumme erkannt und verworfen. Während Störungen in Glasfaser- oder Kupferleitungen vernachlässigbar selten auftreten (die Wahrscheinlichkeit für einen Bitfehler in einer Glasfaserleitung liegt in der Größenordnung $p_b \approx 10^{-12}$), sind Übertragungsfehler bei Funkübertragungen sehr viel wahrscheinlicher. Neben aktiven Störeinflüssen ist vor allem die Entfernung von Sender und Empfänger, sowie die baustoffliche Beschaffenheit der Umgebung entscheidend [Lan99]. In drahtlosen, mobilen Szenarien ist der Rahmenverlust also ein sehr dynamischer Parameter.

Bei gegebener Bitfehlerrate p_b und einem Rahmen der Länge l bit berechnet sich unter der vereinfachenden Annahme, dass Bitfehler unkorreliert auftreten, die Rahmenverlustwahrscheinlichkeit[2] zu $p_v = (1-(1-p_b)^l)$.

Neben einem Bitfehler kann bei Verwendung eines gemeinsamen Mediums auch ein Medienzugriffsprotokoll einen Rahmenverlust verursachen, wenn beispielsweise durch ein dauerhaft überlastetes Medium keine Übertragung möglich ist. Dieser Effekt kann nur durch eine genaue Nachbildung des Medienzugriffsprotokolls emuliert werden (siehe Abschnitt 4.4).

$p_v = 0$ entspricht einem fehlerfreien Kanal; $p_v = 1$ bedeutet, dass im Moment keine Datenübertragung möglich ist. Die Extremwerte der Rahmenverlustwahrscheinlichkeit können also auch als Parameter für die Konnektivität von zwei Kommunikationspartnern dienen.

3.2.2 Basiseffekt: Verzögerung

Die Gesamtverzögerung eines Rahmens zwischen der Übergabe an die Netz-Hardware des Senders und der Auslieferung durch die Netz-Hardware des Empfängers setzt sich aus drei unabhängigen Verzögerungseffekten zusammen: Medienwartezeit, Serialisierungsverzögerung[3] und Ausbreitungsverzögerung [HLR02].

- *Medienwartezeit t_m:* Die Zeit zwischen dem Entgegennehmen eines Rahmens durch die Netz-Hardware und dem Start des tatsächlichen Sendens des Rahmens nennen wir Medienwartezeit. Während der Medienwartezeit wird der zu übertragende Rahmen von der Netz-Hardware zwischengespeichert. Der Zeitpunkt des Sendevorgangs wird vom Medienzugriffsprotokoll bestimmt. Die Medienwartezeit für einen konkreten Rahmen kann nur dann ermittelt werden, wenn das gegenwärtige Belegungszustand des Mediums bekannt ist und eine Nachbildung des Medienzugriffsprotokolls existiert (siehe Abschnitt 4.4).

- *Serialisierungsverzögerung t_s:* Jeder reale Kommunikationskanal ist in der Übertragungsrate b begrenzt. Die Zeit, die zur Übertragung eines Rahmens der Länge l benötigt wird, ist $t_s = l/b$. Bei kleinen Übertragungsraten, wie z.B. Übertragungen über eine Telefonleitung (max. 64 kbit/s), ist die Serialisierungsverzögerung die dominierende Verzögerungskomponente.

[2] Wir sprechen hier von *Rahmenverlustwahrscheinlichkeit* und nicht von *Rahmenverlustrate*, weil p_v nur für einen einzelnen Rahmen gilt, und damit nicht eine länger andauernde „Rate“ darstellt.

[3] Hier wird absichtlich der im Deutschen unübliche Begriff *Serialisierungsverzögerung* eingeführt, obwohl bereits ein anderer deutscher Begriff für den Quotienten l/b existiert, nämlich die *Übertragungsdauer*. Dies hat zwei Gründe: Erstens wird in manchen deutschsprachigen Dokumenten, u.a. in Vorlesungsskripten, auch die *Ausbreitungsverzögerung* zur Übertragungsdauer gerechnet. Dies kann in unserem Kontext zu Begriffsverwirrungen führen. Zweitens sind die Begriffe *serialization delay* bzw. *serialization time* in englischsprachigen Veröffentlichungen durchgängig in Gebrauch, und die direkte Übersetzung wird auch im Deutschen intuitiv richtig verstanden.

Beispiel zur Größenordnung: Ein Rahmen der typischen Länge 1500 Byte erfährt auf einer ISDN-Leitung die Serialisierungsverzögerung $t_s = 1500 \cdot 8\,\text{bit}/(64000\,\text{bit/s}) \approx 190\,\text{ms}$.

- *Ausbreitungsverzögerung t_a:* Die Ausbreitungsverzögerung ist die Zeit, die das Signal benötigt, um die Strecke d zwischen Sender und Empfänger zurückzulegen. Bei Funkkommunikation ist d die Wegstrecke des Funksignals; normalerweise wird dabei von der Luftlinie zwischen Sender und Empfänger ausgegangen. Bei Kommunikation über Kabelmedien entspricht d der Kabellänge, bzw. der Länge des Kabelsegments zwischen Sender und Empfänger. Je nach Medium variiert die Ausbreitungsgeschwindigkeit c des Signals von ca. $0{,}66 \cdot c_0$ (Kupferkabel, Lichtwellenleiter) bis zu $c_0 \approx 3 \cdot 10^8\,\text{m/s}$ (Ausbreitung einer Funkwelle im Vakuum). Die Ausbreitungsverzögerung ergibt sich zu $t_a = d/c$. Bei großen Entfernungen, wie z.B. Satellitenkommunikation, ist die Ausbreitungsverzögerung die dominierende Verzögerungskomponente.
  Beispiel zur Größenordnung: Ein Interkontinentalkabel zwischen Europa und den USA verursacht eine Ausbreitungsverzögerung von $t_a = 7000\,\text{km}/(0{,}66 \cdot c_0) \approx 35\,\text{ms}$.

Anschaulich kann man die drei Verzögerungskomponenten so beschreiben: Die Medienwartezeit ist die Zeit, *bis* das erste Bit des Rahmens gesendet werden kann, die Serialisierungsverzögerung ist die Zeit, *während* der Rahmen gesendet wird, und die Ausbreitungsverzögerung ist die Zeit, die ein Bit des Rahmens *auf dem Medium* verbringt.

In Netzen niedriger Übertragungsrate dominiert normalerweise die Serialisierungsverzögerung, bei hoher Übertragungsrate und großen Entfernungen die Ausbreitungsverzögerung. In lokalen Netzen mit gemeinsam benutztem Medium spielt bei großer Last die Medienwartezeit eine Rolle; ansonsten dominiert die Verarbeitungszeit der höheren, über der Emulationsschicht liegenden Schichten auf den Endgeräten.

Außer den drei genannten Verzögerungskomponenten können weitere Effekte die Übertragung eines Rahmens verzögern, beispielsweise die Verwendung von aktiven Infrastrukturkomponenten. Diese Verzögerungen bewegen sich im Mikrosekundenbereich und werden daher normalerweise von anderen Effekten verdeckt. Wenn sie trotzdem modelliert werden sollen, können sie der Ausbreitungsverzögerung zugeschlagen werden.

Effekte auf höheren Schichten

An dieser Stelle ist zu bemerken, dass auf höheren Protokollschichten Effekte vorkommen können, die ähnliche Auswirkungen wie die beschriebenen Basiseffekte auf der Sicherungsschicht haben. Prominentestes Beispiel sind die Effekte der Warteschlangen auf der Vermittlungsschicht: Je nach Last führen diese Warteschlangen signifikante Verzögerungen ein und

verursachen bei einem Überlauf Paketverluste. Diese Effekte werden hier nicht etwa vernachlässigt, sondern liegen außerhalb des Modellierungsbereichs der Emulationsschicht und brauchen deshalb nicht nachgebildet zu werden. Da wir in unserer Architektur die Vermittlungsschicht als potentielles Testsubjekt und damit über der Emulationsschicht sehen, entstehen die damit verbundenen Effekte automatisch durch das Verhalten tatsächlicher Protokollimplementierungen über nachgebildeter Netz-Hardware.

3.2.3 Höherwertige Parameter

Wären die Basiseffekte Rahmenverlust und Verzögerung rein statischer Natur, könnte man für die Emulation einer Netzverbindung diese Parameter einfach errechnen bzw. messen, um sie dann durch die Emulationsschicht nachbilden zu lassen. In vielen Fällen haben die Basiseffekte jedoch eine dynamische Komponente, die sich nicht einfach algorithmisch erfassen lässt. Vielmehr sind die Basiseffekte von weiteren Parametern abhängig, die wir *höherwertige Parameter* nennen.

Beispielsweise hat der Belegungszustand eines von mehreren Teilnehmern gemeinsam genutzten Mediums Einfluss auf die Medienwartezeit; die Bitfehlerraten und damit die Rahmenverlustwahrscheinlichkeiten in einem drahtlosen Medium ändern sich je nach Position der Kommunikationspartner. Auch mehrstufige Abhängigkeiten sind möglich. Fast jeder denkbare Parameter könnte von einem weiteren höherwertigen Parameter abhängig sein. Beispielsweise ist die momentane Position eines mobilen Kommunikationsteilnehmers von seiner Geschwindigkeit abhängig, die momentane Geschwindigkeit wiederum von seiner eingeschlagenen Route und seinen Fortbewegungsmöglichkeiten, die Route ist abhängig von seinem gegenwärtigen Ziel usw. Es ist offenbar nicht möglich, eine geschlossene Menge mit allen denkbaren höherwertigen Parametern für ein Szenario anzugeben. Vielmehr müssen für ein *konkretes* Szenario diejenigen Parameter ausgewählt werden, auf deren Basis die Beschreibung eines Szenarios sinnvoll ist. Außerdem muss es möglich sein, von diesen Parametern auf die nachzubildenden Basiseffekte zu schließen.

Zwei der dynamischen Parameter, die zur Berechnung der Basiseffekte benötigt werden, können *nicht* sinnvoll in einer Szenariodefinition beschrieben werden: die Rahmenlänge und der Medienzustand. Sie müssen vielmehr im laufenden Betrieb ermittelt werden (zum Zustand eines emulierten Mediums siehe Abschnitt 4.4). Alle anderen Parameter können entweder direkt spezifiziert (vorgegeben) oder aus anderen bekannten Parametern berechnet werden. Die Menge der direkt spezifizierten Parameter definiert das nachzubildende *Netzszenario* aus Nutzersicht.

Die Berechnung der Basiseffekte aus den wichtigsten höherwertigen Parametern (Übertragungsrate, Bitfehlerrate, Rahmenlänge etc.) wird in Kapitel 4 erklärt. Abschnitt 5.3 beschreibt

darüberhinaus die konkrete Realisierung eines Verfahrens, um aus der Position zweier Kommunikationspartner in einem Funknetz auf eine Bitfehlerrate zu schließen.

3.3 Verteilungsalternativen

In Abschnitt 3.1 wurde bereits erläutert, in welcher *konzeptionellen* Schicht des Protokollstapels ein Emulationsverfahren eingreifen sollte. Auch die auf dieser Schicht nachzubildenden Netzparameter sind bekannt. Ein Werkzeug, das die Nachbildung der gewünschten Netzparameter leisten kann, wird *Emulationswerkzeug* genannt. Soll nicht nur eine einzelne Netzverbindung, sondern ein gesamtes Netzszenario nachgebildet werden, kann es sinnvoll sein, nicht nur ein Werkzeug, sondern mehrere, kooperierende Werkzeuge zu verwenden. In diesem Abschnitt werden verschiedene Verteilungsalternativen für Emulationswerkzeuge vorgestellt und bewertet. Die Varianten werden mit den Kürzeln ① bis ③ bezeichnet.

Annahmen zur Testumgebung

Alle vorgestellten Architekturen gehen von der Annahme aus, dass für die Nachbildung des Szenarios eine Testumgebung mit mehreren physischen Knoten zur Verfügung steht, die über ein zuverlässiges, schnelles Rechnernetz verbunden sind. Zudem soll die Anzahl der physischen Knoten ausreichend sein, damit jeder im Testszenario vorkommende Knoten jeweils durch mindestens einen physischen Knoten der Testumgebung dargestellt werden kann. Die Annahme einer eins-zu-eins Zuordnung zwischen Knoten im Szenario und physischen Knoten vereinfacht die Architektur und die Implementierung. Möglichkeiten, die sich durch die Aufhebung dieser Einschränkung ergeben, werden im Ausblick skizziert (Abschnitt 7.2).

Weiterhin nehmen wir an, dass jeder physische Knoten einer Testumgebung mindestens so viele Netzschnittstellen besitzt wie der Knoten im Szenario, den er nachbildet. Dadurch wird die Skizzierung der verschiedenen Architekturen erleichtert. In Abschnitt 5.2 wird diese Einschränkung durch die Einführung von „virtuellen Netzwerkgeräten“ komplett aufgehoben.

Annahmen zum Netzmodell

Wir gehen davon aus, dass zu jedem Zeitpunkt eines Experiments die nachzubildenden Eigenschaften aller Verbindungen an zentraler Stelle bekannt sind und somit ein aktuelles globales Netzmodell existiert. Als Grundlage dieses Netzmodells wird eine Szenariobeschreibung herangezogen, die vom Benutzer eines Emulationssystems vorgegeben wird. Eine solche Szenariobeschreibung kann sowohl statische, als auch dynamische Parameter enthalten. Im letzteren Fall enthält die Szenariobeschreibung *Änderungsvorschriften* für die dynamischen Parameter.

Eine zentrale Instanz wertet diese Änderungsvorschriften aus und hält das globale Netzmodell auf dem aktuellen Stand.

3.3.1 Zentralisiertes Emulationswerkzeug

In einer *zentralisierten* Architektur (Variante ①) wird nur ein einziges, zentrales Emulationswerkzeug verwendet, das auf einem explizit dafür reservierten Knoten *E* ausgeführt wird (siehe Abb. 3.3). Dieses Werkzeug bildet auf der Grundlage des Netzmodells die Eigenschaften aller Netzverbindungen eines Szenarios nach. Die im Testszenario definierten Knoten *(A, B, C)* werden auf jeweils einen physischen Knoten abgebildet. Zusätzlich zu den Testsubjekten – in der Abbildung ist das Testsubjekt ein Vermittlungsprotokoll – werden auf den Knoten eines Testszenarios üblicherweise *Lastgeneratoren* ausgeführt, die als Quellen und Senken für den Netzverkehr fungieren. Der gesamte Netzverkehr zwischen den Knoten *(A, B, C)* muss zur Emulation über den Knoten *E* geleitet werden.

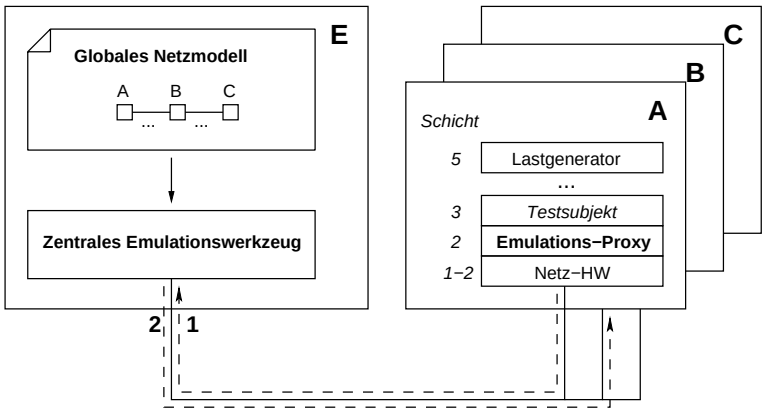

Abbildung 3.3: Zentralisierte Emulation (①)

Um dies zu erreichen, wird auf jedem Szenarioknoten ein zusätzliches Werkzeug auf der logischen Ebene der Emulationsschicht eingeführt, das wir *Emulations-Proxy*[4] nennen. Die Emulations-Proxys sind dafür zuständig, dass Netzverkehr innerhalb des Szenarios nicht direkt, sondern indirekt über das zentrale Emulationswerkzeug zugestellt wird. Dabei müssen sie auch dafür sorgen, dass diese Umleitung für die höheren Protokollschichten transparent erfolgt. Ein Rahmen, der beispielsweise von *A* nach *B* geschickt werden soll, wird vom Emulations-Proxy auf *A* abgefangen und zunächst zum zentralen Emulationswerkzeug auf *E* umgeleitet. Damit trotzdem die komplette Adressinformation des Rahmens erhalten bleibt, muss der Emulations-Proxy dazu den ursprünglichen Rahmen in einen eigenen Rahmen „einpacken“. Das Emulationswerkzeug entscheidet, ob und wann der Rahmen zugestellt werden soll (es bildet also die

[4] Eine deutsche Entsprechung für den Begriff Emulations-Proxy wäre etwa *Emulations-Stellvertreterobjekt*.

Basiseffekte nach), und schickt ihn zum Auslieferungszeitpunkt ggf. selbst weiter an *B*. Der Emulations-Proxy auf dem empfangenden Knoten *B* muss dann die Artefakte der Umleitung wieder entfernen und den Rahmen an die nächsthöhere Protokollschicht ausliefern.

Da bei der zentralisierten Emulation einer zentralen Instanz der Zustand aller Medien und der gesamte Netzverkehr eines Szenarios bekannt ist, können prinzipiell auch Technologien nachgebildet werden, bei denen sich mehrere Teilnehmer ein gemeinsames Medium teilen. Allerdings ergeben sich dadurch besondere Anforderungen an das Emulationsnetz, über das die Emulations-Proxys mit dem zentralen Emulationswerkzeug kommunizieren. Nur wenn die durch dieses Netz eingeführten Verzögerungen sehr klein sind, kann die zentrale Instanz die realzeitliche Zuordnung von Nachrichten verschiedener Sender auf dasselbe nachgebildete Medium in der erforderlichen Genauigkeit vornehmen, um Effekte wie Kollisionen nachzubilden. Diese zeitlichen Anforderungen werden in Abschnitt 4.4 genauer betrachtet.

3.3.2 Verteilte Emulationswerkzeuge, zentrale Steuerung

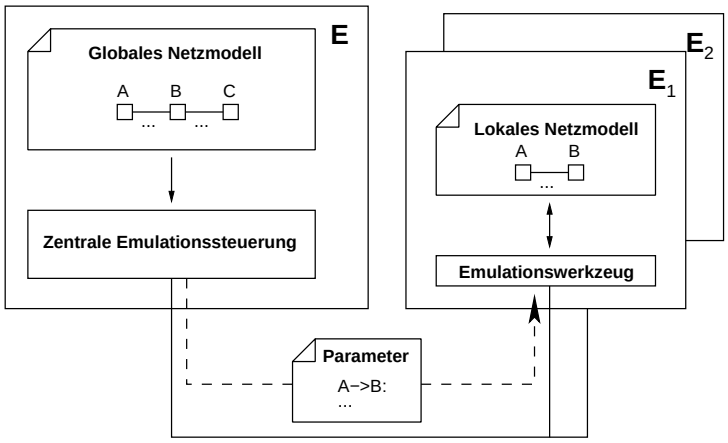

Abbildung 3.4: Zentrale Steuerung verteilter Emulationswerkzeuge (②)

Da die Eigenschaften der einzelnen Netzverbindungen eines Szenarios oft voneinander unabhängig sind, ist es sinnvoll, mehrere Instanzen eines Emulationswerkzeugs für die Nachbildung eines Szenarios zu verwenden. Weil das Netzmodell nach unserer Annahme an einer zentralen Stelle vorliegt, ist es für dynamische Szenarien notwendig, während eines Experiments Änderungen im Modell an die verteilten Werkzeuge zu kommunizieren. Diese Aufgabe übernimmt die *zentrale Emulationssteuerung* (Abb. 3.4).

Für die Platzierung verteilter Emulationswerkzeuge gibt es zwei verschiedene Möglichkeiten: Entweder sie werden jeweils auf einem explizit reservierten Emulationsknoten ausgeführt (Variante ②*a*), oder sie werden direkt auf den Knoten betrieben, die auch die Testsubjekte ausführen (Variante ②*b*).

Explizite Emulationsknoten (②*a*)

Verteilte Emulationswerkzeuge auf expliziten Emulationsknoten arbeiten funktional gleich wie im zentralisierten Fall, mit dem Unterschied, dass je ein Werkzeug für die Nachbildung einer Netzverbindung im Szenario zuständig ist. Die Last der Verarbeitung des Verkehrs im Netzszenario ist also auf mehrere Werkzeuge verteilt. Wiederum werden Emulations-Proxys verwendet, um den Netzverkehr zwischen den Testsubjektinstanzen über die Emulationswerkzeuge zu leiten. Auch Technologien mit gemeinsam benutztem Medium können prinzipiell nachgebildet werden; es gelten die gleichen Einschränkungen wie im zentralisierten Fall.

Abb. 3.5 zeigt die Realisierung mit einem expliziten Emulationsknoten für die Nachbildung einer Verbindung zwischen *A* und *B*.

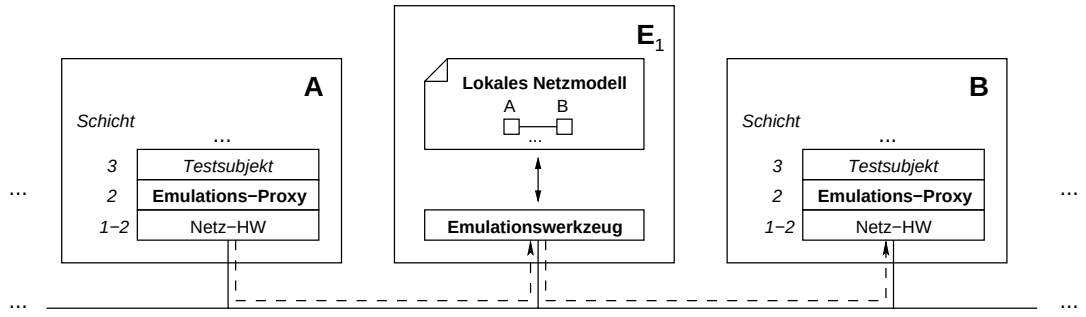

Abbildung 3.5: Verteilte Emulation mit expliziten Emulationsknoten (②*a*)

Testsubjekt und Emulation auf denselben Knoten (②*b*)

Unter der Annahme, dass der Betrieb eines Emulationswerkzeugs, das nur für die Nachbildung eines kleinen Szenarioteils zuständig ist, nur einen Bruchteil der Ressourcen eines Rechners beansprucht, kann man die Architektur deutlich vereinfachen: Werden statt der Emulations-Proxys direkt Instanzen von Emulationswerkzeugen auf den Testsubjekt-Knoten betrieben, kann auf die zusätzlichen Emulationsknoten verzichtet werden (Abb. 3.6). Die Umleitung des Netzverkehrs durch die Emulations-Proxys entfällt. Sendungen zwischen zwei nach Szenariodefinition direkt verbundenen Knoten können auch im Emulationssystem direkt zugestellt werden; somit werden zusätzliche, systembedingte Verzögerungen vermieden.

Der Verzicht auf explizite Emulationsknoten hat Auswirkungen auf die Aufgabenzuordnung der Emulationswerkzeuge: Während in Variante ②*a* jede nachzubildende *Netzverbindung* eindeutig einem Emulationswerkzeug zugeordnet ist, hat in Variante ②*b* jeder *Knoten* eines Netzszenarios mindestens ein zugeordnetes Emulationswerkzeug. Der Netzverkehr zwischen zwei benachbarten Knoten durchläuft also die Emulationsschicht auf beiden Knoten; sowohl beim Senden als auch beim Empfangen eines Rahmens kann ein Emulationswerkzeug eingreifen.

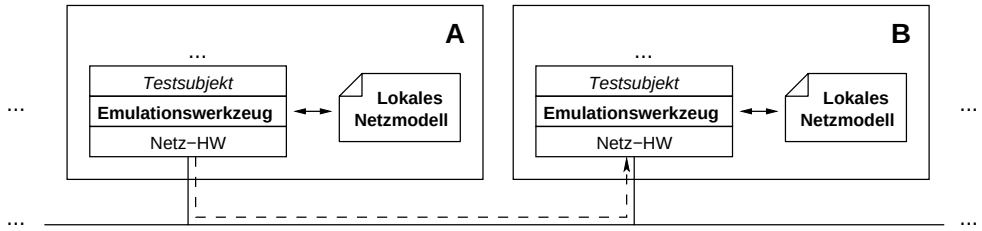

Abbildung 3.6: Verteilte Emulation ohne explizite Emulationsknoten (②*b*)

Die knotenbasierte Zuordnung von Emulationswerkzeugen verkompliziert allerdings die Nachbildung von Netzen mit gemeinsamem Medium: An der Nachbildung eines Netzes mit *n* Teilnehmern ist nicht mehr nur ein einziges, sondern sind *n* Emulationswerkzeuge beteiligt. Da die von den Emulationswerkzeugen nachzubildenden Basiseffekte unter anderem vom gegenwärtigen Zustand des emulierten Mediums abhängen können, dieser aber von allen Teilnehmern beeinflusst werden kann, müssen die Emulationswerkzeuge kooperieren, um ein gemeinsames Modell des emulierten Mediums zu halten (siehe Abschnitt 4.4).

3.3.3 Verteilte Emulationswerkzeuge, lokale Steuerung

Bei dynamischen Szenarien mit verteilten Emulationswerkzeugen findet während des Experiments Kommunikation zwischen der zentralen Emulationssteuerung und den Emulationswerkzeugen statt. Bei großen und hochdynamischen Szenarien kann die zentrale Emulationssteuerung mit der Bereitstellung der Parameter in Echtzeit überfordert sein. Außerdem ist möglicherweise administrative Kommunikation während des Experiments unerwünscht, wenn die Architektur der Testumgebung es zulässt, dass administrativer Netzverkehr und Netzverkehr innerhalb des Szenarios sich gegenseitig beeinflussen können.

Administrative Kommunikation während eines Experiments kann ganz vermieden werden, indem alle für eine Werkzeuginstanz notwendigen Parametersätze mit zugehörigen Zeitstempeln *vor* dem Beginn des Experiments auf dem jeweiligen Knoten abgelegt werden. Während des Experiments wird dann jedes Emulationswerkzeug von einer *lokalen Steuerungsinstanz* mit den zur jeweiligen Zeit gültigen Parametern versorgt (Abb. 3.7). Damit die Parameteränderungen der einzelnen Steuerungsinstanzen zu einem konsistenten Gesamtszenario führen, müssen sie zeitlich genau koordiniert sein. Dies setzt synchronisierte Uhren auf den Knoten voraus.

Das Konzept der lokalen Steuerung kann wiederum sowohl mit expliziten Emulationsknoten (Variante ③*a*) als auch mit Emulationswerkzeugen und Testsubjekt auf denselben Knoten (Variante ③*b*) verwendet werden.

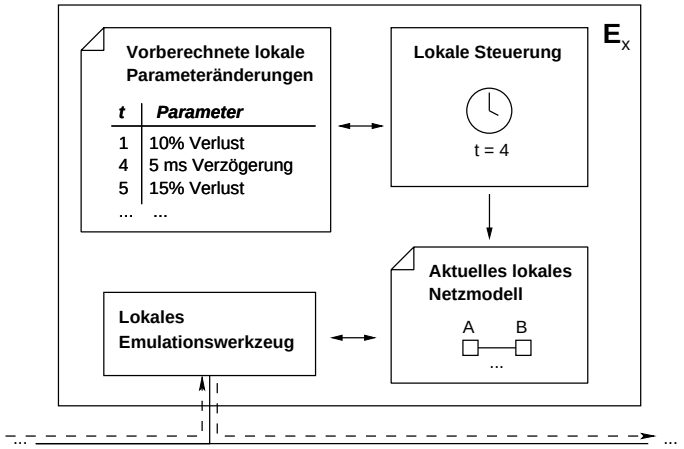

Abbildung 3.7: Lokale Steuerung der Emulationsparameter (③a, ③b)

3.3.4 Bewertung

Jede der vorgestellten Verteilungsalternativen ermöglicht die Nachbildung der Basiseffekte Rahmenverlust und Verzögerung für mehrere Verbindungen in einem Netzszenario. Dennoch sind sie nicht für alle möglichen Szenarien gleich gut geeignet. Im Folgenden wird zunächst diskutiert, welche Einschränkungen die verschiedenen Verteilungsalternativen auf die Nachbildung der Basiseffekte haben. Danach werden die Unterschiede im Bezug auf die erreichbare Szenariengröße erörtert.

Rahmenverlust

Wir gehen davon aus, dass ein Emulationswerkzeug jede Rahmenverlustwahrscheinlichkeit $0 \leq p_v \leq 1$ zuverlässig nachbilden kann; dabei spielt es keine Rolle, ob das Werkzeug verteilt oder zentral realisiert ist. Zu den nachgebildeten Rahmenverlusten addieren sich die Rahmenverluste, die über das Rechnernetz der Testumgebung eingeführt werden. Da wir jedoch davon ausgehen, dass dieses Rechnernetz zuverlässig ist, Rahmenverluste also vernachlässigbar selten auftreten, ergeben sich hier keine Unterschiede zwischen den verschiedenen Alternativen.

Verzögerung

Zusätzlich zu der von einem Emulationswerkzeug absichtlich eingeführten Verzögerung werden noch weitere, unvermeidliche Verzögerungen eingeführt: die Verarbeitungszeit eines Rahmens im Emulations-Proxy bzw. im Emulationswerkzeug (t_v), sowie die Verzögerung, die eine Nachricht beim Versenden über das Rechnernetz der Testumgebung erfährt (t_n). Je nach Art und

Anzahl der beteiligten Komponenten in einer Architektur addieren sich diese Verzögerungskomponenten ggf. mehrfach auf. Die Summe dieser Komponenten ist die Verzögerung $t_{\min}$, die ein Rahmen in einem Emulationssystem *mindestens* erfährt. Ist der Wert von $t_{\min}$ bekannt und soll eine Rahmenverzögerung $t \geq t_{\min}$ nachgebildet werden, so kann ein Emulationswerkzeug vor der Nachbildung $t_{\min}$ vom Sollwert abziehen. Es ist jedoch nicht möglich, Verzögerungen $t < t_{\min}$ nachzubilden. Es ist also wünschenswert, $t_{\min}$ so klein wie möglich zu halten.

Konkrete Werte für $t_{\min}$ können nur durch Messung in einem existierenden Emulationssystem ermittelt werden; Messwerte aus dem für diese Arbeit entwickelten System sind in Abschnitt 6.1 zu finden. In Tab. 3.1 ist die prinzipielle Zusammensetzung von $t_{\min}$ für alle Architekturalternativen angegeben.

Szenariengröße

Die mit einem Emulationssystem nachbildbare Szenariengröße ist vor allem durch die Testumgebung begrenzt – durch die Anzahl der zur Verfügung stehenden Knoten, sowie durch die Leistungsfähigkeit der Knoten und des Rechnernetzes. Die vorgestellten Architekturen unterscheiden sich in der Effizienz der Ausnutzung dieser Ressourcen. Zur Nachbildung eines gegebenen Szenarios benötigen die Alternativen also unterschiedliche Ressourcen, bzw. bei gegebenen Ressourcen sind unterschiedliche Szenariengrößen möglich. Als abkürzende Schreibweise wollen wir hier die Anzahl der Knoten in einem Szenario mit n_k, die Anzahl der Netzverbindungen mit n_v bezeichnen, wobei die Verbindung mehrerer Knoten über ein gemeinsames Netz (z.B. LAN) nur als eine Verbindung zählen soll. Die Anzahl der für die Nachbildung benötigten physischen Knoten der Testumgebung sei n_p.

Bei Verwendung eines zentralisierten Emulationswerkzeugs wird jedem Knoten im Szenario ein Knoten in der Testumgebung zugeordnet. Zusätzlich wird ein zentraler Emulationsknoten E benötigt. Die Anzahl der erforderlichen physischen Knoten ist also $n_p = n_k + 1$. Zusätzlich ist die Szenariogröße durch den insgesamt im Szenario entstehenden Netzverkehr beschränkt, da dieser von E bearbeitet werden muss.

Auch Architekturen mit verteilten Emulationswerkzeugen benötigen einen physischen Knoten pro Knoten im Szenario sowie einen zentralen Steuerungsrechner. Ohne explizite Emulationsknoten bleibt die Anzahl der erforderlichen physischen Knoten bei $n_p = n_k + 1$. Bei Verwendung von expliziten Emulationsknoten wird zusätzlich ein weiterer Emulationsknoten E_x für jede Netzverbindung im Szenario verwendet; die Anzahl der erforderlichen Knoten erhöht sich also auf $n_p = n_k + n_v + 1$. Der gesamte Netzverkehr in einem Szenario ist im Gegensatz zur zentralen Architektur nicht mehr beschränkt. Da während eines Experiments die Parameter aller Emulationswerkzeuge von der zentralen Emulationssteuerung aktualisiert werden, kann bei hochdynamischen Szenarien und sehr vielen Emulationsknoten die zentrale Steuerung trotzdem zum Flaschenhals werden.

Die Kombination von verteilten Emulationswerkzeugen und lokalen Steuerungsinstanzen für die Emulationsparameter vermeidet, dass die zentrale Steuerungsinstanz zum Flaschenhals wird. Die Anzahl der benötigten Knoten bleibt gleich.

Fazit

Var.	*Steuerung*	*Emulationswerkzeug*	n_p	$t_{\min}$	*Flaschenhals*
①	zentral	zentral	n_k+1	$3t_v+2t_n$	zentrales E.-Werkzeug
②a	zentral	pro Netzverbindung	n_k+n_v+1	$3t_v+2t_n$	zentrale E.-Steuerung
②b	zentral	pro Szenario-Knoten	n_k+1	$2t_v+t_n$	zentrale E.-Steuerung
③a	verteilt	pro Netzverbindung	n_k+n_v+1	$3t_v+2t_n$	–
③b	verteilt	pro Szenario-Knoten	n_k+1	$2t_v+t_n$	–

steigender Realisierungsaufwand ↓

Tabelle 3.1: Vergleich der Architekturalternativen

Tab. 3.1 fasst die Eigenschaften der Architekturalternativen zusammen. Zentralisierte Emulationssysteme sind einfach zu realisieren, jedoch nur für kleine Szenarien geeignet, da das zentrale Emulationswerkzeug einen Flaschenhals darstellt. Bei den Varianten mit verteilten Emulationswerkzeugen ist die Zuordnung von Emulationswerkzeugen zu Szenario-Knoten vorteilhaft, weil sie sowohl die Anzahl der benötigten Knoten n_p, als auch die Untergrenze der Verzögerung $t_{\min}$ minimiert. Für hochdynamische Szenarien mit sehr vielen Knoten ist zusätzlich die Einführung einer verteilten Steuerungsinstanz sinnvoll.

3.4 Zusammenfassung

In diesem Kapitel wurde zunächst argumentiert, warum ein Emulationverfahren möglichst tief im Protokollstapel eingreifen sollte. Die Einführung einer Emulationsschicht als unterste in Software realisierte Schicht wurde als idealer Eingriffspunkt identifiziert. Funktional ist die Emulationsschicht dabei ein Teil der Sicherungsschicht. Auf der Grundlage der Dienstabstraktion, die die Emulationsschicht anbietet, wurden die beiden Basiseffekte ersichtlich, mit denen die Emulationsschicht Netzeigenschaften nachbilden kann: Rahmenverlust und Verzögerung. Es wurde skizziert, wie sich diese Basiseffekte aus höherwertigen Netzparametern ergeben können.

Für die Realisierung eines Emulationssystems, das ein Netzszenario aus mehreren Verbindungen nachbilden kann, wurden mehrere Verteilungsalternativen vorgeschlagen und in Hinblick auf ihre Eignung in Bezug auf die Szenariogröße und die nachzubildenden Parameter bewertet.

Im folgenden Kapitel wird die Funktionsweise eines Emulationssystems für eine der Architekturalternativen detailliert vorgestellt.

4

Funktionsweise eines verteilten Emulationssystems

In diesem Kapitel wird die Funktionsweise eines Emulationssystems am konkreten Beispiel einer Architektur mit zentraler Steuerung und verteilten, kooperativen Emulationswerkzeugen beschrieben. Zunächst wird erklärt, warum genau diese Architekturvariante ausgewählt wurde. Danach werden die Funktion und das Zusammenspiel der Emulationssteuerung, der Emulationswerkzeuge und der Netzinfrastruktur des Emulationssystems detailliert erläutert. Dabei wird auch das Konzept des virtuellen Trägersignals eingeführt, das kooperative verteilte Emulationswerkzeuge für die Nachbildung von Netzen mit gemeinsamem Medium ermöglicht.

4.1 Architekturüberblick

Von den im vorigen Kapitel vorgestellten Architekturalternativen werden wir uns im Folgenden mit der Variante ②*b* genauer befassen (zentrale Steuerung, verteilte Emulationswerkzeuge, ein Emulationswerkzeug pro Szenario-Knoten). Diese Architekturvariante wurde ausgewählt, weil sie sich durch eine kleine minimale Verzögerung t_{min} und eine effiziente Knotennutzung auszeichnet. Ein zentrales Emulationswerkzeug wurde vermieden, um die Summe des möglichen Netzverkehrs in einem Szenario nicht zu beschränken.[1] Die Einführung einer zusätzlichen lokalen Steuerung (Variante ③*b*) hätte einen erheblich größeren Realisierungsaufwand bedeutet. Hierauf wurde verzichtet, da für die in unserer Realisierung zu erwartenden Szenariogrößen

[1] Auch bei Verwendung von verteilten Emulationswerkzeugen ist die emulierbare Übertragungsrate einer *einzelnen* Verbindung durch die Leistungsfähigkeit der Testumgebung beschränkt (siehe Abschnitt 4.3.3, sowie Messungen in Abschnitt 6.4).

von deutlich unter 100 Szenarioknoten eine zentrale Steuerung keinen Flaschenhals darstellt. Messungen, die dies belegen, finden sich in Abschnitt 6.3.

Die ausgewählte Architektur besteht aus drei funktionalen Komponenten (Abb. 4.1): Aus der zentralen Emulationssteuerung, den auf jedem Knoten vorhandenen verteilten Emulationswerkzeugen und dem Rechnernetz, das die Knoten untereinander und mit der zentralen Steuerung verbindet.

- Die zentrale Emulationssteuerung koordiniert das gesamte Experiment auf der Grundlage einer Szenariobeschreibung. Sie verwaltet das globale Netzmodell und schickt während eines Messlaufs die jeweils benötigten Teile des globalen Modells an die verteilten Emulationswerkzeuge.

- Die verteilten Emulationswerkzeuge, die auf jedem Emulationsknoten ausgeführt werden, haben eine lokal beschränkte Sicht auf das nachzubildende Netz. Aufgrund ihres lokalen Netzmodells bilden sie Netzeigenschaften nach.

- Die Netzinfrastruktur des Emulationssystems ermöglicht die Kommunikation aller Komponenten untereinander. Außerdem kann die Infrastruktur einen Teil der Aufgaben der Emulationswerkzeuge übernehmen und diese damit entlasten.

In den folgenden Abschnitten wird die Funktionalität der einzelnen Komponenten im Detail erklärt.

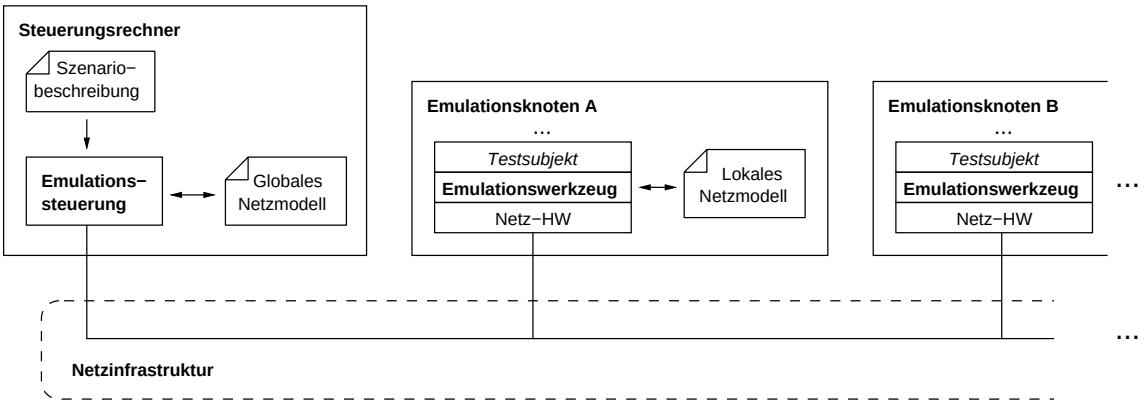

Abbildung 4.1: Funktionale Komponenten des Emulationssystems

4.2 Emulationssteuerung

Die Emulationssteuerung ist für die Gesamtkoordination eines Experiments zuständig. Alle für die Einrichtung und den Ablauf eines Experiments notwendigen Informationen sind in einer *Szenariobeschreibung* vorgegeben. Die Szenariobeschreibung enthält allgemeine, statische Konfigurationsinformationen und potentiell dynamische Informationen über die Eigenschaften der nachzubildenden Netze. Auf der Grundlage der Konfigurationsinformationen (Anzahl und Typ der Knoten, Testsubjekte, Hilfsprogramme etc.) wird das Experiment vorbereitet. Anhand der Informationen über die Netzeigenschaften und deren Änderungsvorschriften erstellt die Emulationssteuerung ein *globales Netzmodell*, das während des Experiments ständig aktualisiert wird. Mit den jeweils aktuellen Netzparametern aus dem globalen Netzmodell werden die verteilten Emulationswerkzeuge zu Beginn und während des Experiments konfiguriert. Nach dem Ende des Messlaufs koordiniert die Emulationssteuerung das kontrollierte Herunterfahren des Experiments.

Im Folgenden wird zunächst das globale Netzmodell als wichtigste Datenstruktur der Emulationssteuerung vorgestellt. Danach wird die Koordinationsfunktion der Emulationssteuerung in der Reihenfolge der drei Phasen Vorbereitung, Messung und Nachbereitung erklärt.

4.2.1 Globales Netzmodell

In der Szenariobeschreibung sind die nachzubildenden Parameter und deren Änderungsvorschriften für das gesamte Szenario beschrieben. Da nur die Basiseffekte Rahmenverlust und Verzögerung direkt nachgebildet werden können, in der Szenariobeschreibung aber auch höherwertige Parameter enthalten sein können, müssen daraus die nachzubildenden Basiseffekte zunächst berechnet werden (vgl. Abschnitt 3.2.3).

Um die Funktionalitäten der Komponenten eines Emulationssystems klar zu trennen, erscheint es sinnvoll, die gesamte *Berechnung* der nachzubildenden Basiseffekte in der Emulationssteuerung durchzuführen, während die verteilten Emulationswerkzeuge für die *Nachbildung* der Basiseffekte zuständig sind. Die berechneten Basiseffekte wären dann im zentralen Netzmodell vorzuhalten und bei Bedarf den verteilten Emulationswerkzeugen zu übermitteln. Die Schnittstelle zwischen den beiden Komponenten wäre also durch die Sollwerte für die nachzubildenden Basiseffekte bestimmt.

Diese strikte Aufgabenteilung ist allerdings in der von uns gewählten Architektur nicht möglich. Für einige Berechnungsschritte werden Informationen über den momentanen Netzverkehr benötigt, die der zentralen Instanz nicht zur Verfügung stehen. So wird beispielsweise zur Berechnung der Rahmenverlustwahrscheinlichkeit aus der Bitfehlerrate die Rahmenlänge benötigt (vgl. Abschnitt 3.2.1), die nur dem verarbeitenden Emulationswerkzeug bekannt ist. Diesen und

andere Berechnungsschritte kann also nur das Emulationswerkzeug direkt für jeden zu bearbeitenden Rahmen durchführen.

Da die Berechnung der Basiseffekte nicht komplett zentral erfolgen kann, könnte man vorschlagen, sie vollständig auf die verteilten Werzeuge zu verlagern. Dies ist jedoch aus zwei Gründen nicht ratsam: Erstens sollten die Emulationsknoten, die in unserer Architektur ja auch die Testsubjekte ausführen, nicht unnötig belastet werden; ansonsten bestünde die Gefahr, dass die Rechenkapazität der einzelnen Emulationsknoten zum Flaschenhals wird. Zweitens können die Berechnungsschritte je nach Art der höherwertigen Parameter, die als Ausgangspunkt zur Verfügung stehen, die Interaktion mit externen Werkzeugen oder Datenbanken erfordern (siehe Abschnitt 5.3). Es ist zweckmäßig, die Schnittstelle zu diesen externen Komponenten an einem Punkt zu bündeln, nämlich bei der Emulationssteuerung.

Es ist also sinnvoll, einen Teil der Parameterberechnung zentral, einen weiteren lokal in den Emulationswerkzeugen durchzuführen. Alle Berechnungsschritte, die ohne Informationen über den aktuellen Netzverkehr ausgeführt werden können, obliegen der zentralen Steuerung. Die Schnittstelle zwischen Emulationssteuerung und Emulationswerkzeugen ist somit durch die Menge der nachzubildenden Netzparameter bestimmt, die ohne lokales Wissen nicht weiterverarbeitet werden können. Diese Parameter ergeben dann zusammengenommen das globale Netzmodell. Ausgehend von den Basiseffekten wollen wir im Folgenden diese Parametermenge bestimmen und deren Repräsentation im Netzmodell erläutern.

Rahmenverlust

Innerhalb der nachzubildenden Schichten kann der Effekt Rahmenverlust durch sporadische Bitfehler (Kanalqualität), fehlende Konnektivität (Verbindungstopologie) oder als Seiteneffekt des Medienzugriffs (Kollisionen etc.) entstehen. Rahmenverluste, die durch den Medienzugriff entstehen, müssen ohnehin durch einen emulierten Medienzugriff nachgebildet werden, und werden somit nicht direkt aus der Szenariobeschreibung abgeleitet (siehe Abschnitt 4.4). Rahmenverluste aufgrund von Bitfehlern hängen von der Länge des Rahmens ab, die nur lokal bekannt ist. Die Rahmenverlustwahrscheinlichkeit p_v berechnet sich aus der Bitfehlerrate und der Rahmenlänge ($p_v = (1-(1-p_b)^l)$, vgl. Abschnitt 3.2.1). Die Bitfehlerrate p_b ist nicht von weiteren lokalen Größen abhängig. p_b ist somit ein Parameter, der im Netzmodell repräsentiert sein muss. Rahmenverluste aufgrund eingeschränkter Konnektivität können durch einen Extremwert der Bitfehlerrate modelliert werden ($p_b = 1$) und benötigen daher keine gesonderte Betrachtung.

In einem Netzszenario kann jedem Knotenpaar eine individuelle Bitfehlerrate zugeordnet sein. Bitfehlerraten müssen zudem nicht symmetrisch sein. Die Zuordnung von Knotenpaaren zu Bitfehlerraten wird im globalen Netzmodell daher zweckmäßig als $n_k \times n_k$ Matrix repräsentiert.

Verzögerung

Die nachzubildende Verzögerung setzt sich aus den Komponenten Serialisierungsverzögerung, Ausbreitungsverzögerung und Medienwartezeit zusammen (vgl. Abschnitt 3.2.2).

Die Serialisierungsverzögerung t_s berechnet sich aus der Rahmenlänge und der Übertragungsrate ($t_s = l/b$) und kann daher ebenfalls nur lokal bestimmt werden. Die (Brutto-) Übertragungsrate b ist dagegen in der Szenariobeschreibung enthalten, bzw. kann daraus ermittelt werden. Statt t_s ist also b in das globale Netzmodell aufzunehmen. Übertragungsraten können für jedes Knotenpaar unterschiedlich und asymmetrisch sein. Obwohl Übertragungsraten oft statische Parameter sind, gibt es auch Szenarien mit dynamischen Übertragungsraten, etwa bei automatischer Anpassung der Übertragungsraten an die Kanalqualität bei WLAN. Die Übertragungsraten werden entsprechend den Bitfehlerraten in einer $n_k \times n_k$ Matrix im globalen Netzmodell repräsentiert.

Die Ausbreitungsverzögerung t_a ist nicht von lokalen Größen abhängig. Für Szenarien mit statischen Knoten kann sie direkt in der Szenariobeschreibung angegeben werden, bei sich bewegenden Knoten ist sie dynamisch aus den Positionen der Teilnehmer berechenbar. Sie ist daher direkt in das globale Netzmodell aufzunehmen. Jedes Knotenpaar kann eine eigene Ausbreitungsverzögerung besitzen. t_a ist normalerweise symmetrisch, kann aber in Sonderfällen (z.B. Satelliten-DSL mit Telefon-Rückleitung) auch asymmetrisch sein. Auch die Ausbreitungsverzögerung wird im globalen Netzmodell durch eine $n_k \times n_k$ Matrix repräsentiert.

Die Medienwartezeit t_m wird indirekt durch die Nachbildung eines Medienzugriffsprotokolls ermittelt. In diese Nachbildung gehen verschiedene Parameter ein, die entweder ohnehin im Netzmodell repräsentiert sind (p_b, b, t_a) oder lokal vom Emulationswerkzeug ermittelt werden können (l, Medienzustand). Die einzige zusätzliche Information, die zur Emulation eines Medienzugriffsprotokolls benötigt wird, ist die Konfiguration, *welches* Medienzugriffsprotokoll nachgebildet werden soll. Diese Information ist statischer Natur und wird nur zu Beginn des Experiments bei der Konfiguration der Emulationswerkzeuge benötigt; sie ist daher nicht Teil des (dynamischen) Netzmodells.

Zusammenfassung

Das globale dynamische Netzmodell der zentralen Emulationssteuerung enthält nicht direkt die nachzubildenden Basiseffekte, sondern für jedes Knotenpaar eine Menge von Parametern, aus denen die jeweiligen Instanzen des verteilten Emulationswerkzeugs zusammen mit lokal zur Verfügung stehenden Informationen die Basiseffekte bestimmen können. Diese Parameter sind die Bitfehlerrate p_b, die Übertragungsrate b und die Ausbreitungsverzögerung t_a. Das globale Netzmodell ist also eine $n_k \times n_k$ Matrix, welche die Tripel (p_b, b, t_a) enthält (vgl. Tab. 4.1 auf Seite 68).

Auf der Grundlage der in der Szenariobeschreibung enthaltenen Parameter und deren Änderungsvorschriften berechnet die zentrale Emulationssteuerung während eines Experiments die jeweils aktuellen Parameter für das globale Netzmodell. Der konkrete Berechnungsprozess hängt von der Art der Szenariobeschreibung ab und kann sehr komplex sein. Ein Beispiel für eine solche Parameterberechnung wird in Abschnitt 5.3 beschrieben. Wir gehen im Folgenden davon aus, dass der Berechnungsprozess und somit die Parameter im globalen Netzmodell bekannt sind.

4.2.2 Vorbereitungsphase

In der Vorbereitungsphase eines Experiments liest die Emulationssteuerung zunächst die Szenariobeschreibung ein und ermittelt die Anzahl der Knoten im Szenario (n_k). Da jeder Szenarioknoten durch einen physischen Knoten repräsentiert werden soll, werden aus der Menge der aktuell verfügbaren Knoten der Testumgebung n_k Knoten ausgewählt und den Knoten in der Szenariobeschreibung zugeordnet. Verfügt die Testumgebung über genügend viele gleichartige Knoten, so ist diese Zuordnung trivial. Wir wollen hier von einer homogenen Testumgebung mit ausreichender Knotenanzahl ausgehen. In einer heterogenen Testumgebung wären zusätzlich die Eigenschaften der Knoten und deren Netzanbindung bei der Zuordnung zu berücksichtigen [RAL03].

Nach der Knotenzuordnung werden auf allen verwendeten Knoten die benötigten Instanzen der verteilten Emulationswerkzeuge gestartet und mit den Parametern initialisiert, die zu Beginn des Szenarios gelten. Wenn die Netzinfrastruktur des Emulationssystems auch Emulationsaufgaben übernimmt, müssen die dafür notwendigen Konfigurationen ebenfalls zu diesem Zeitpunkt durchgeführt werden. Weiterhin müssen die Testsubjekte und für die Messung zusätzlich benötigte Komponenten, z.B. Datenbanken, Lastgeneratoren und Messwerkzeuge, nun installiert und konfiguriert werden. Damit sich die Testsubjekte und deren Umgebung zu Beginn des Messlaufs in einem definierten, reproduzierbaren Zustand befinden, kann eine Vorlaufphase nötig sein, in der beispielsweise Datenbanken gestartet und initialisiert, Zwischenspeicher vorgeladen und Protokolldateien angelegt werden.

Bis zu diesem Zeitpunkt ist für den Vorbereitungsprozess keine besondere zeitliche Koordination der Knoten notwendig, zumindest wenn die Testsubjekte nicht für ihre Vorlaufphase interagieren müssen. Die Emulationssteuerung kann die notwendigen Anweisungen also sequentiell geben. Aus Effizienzgründen wird die Vorlaufphase zwar in der Praxis parallel, aber nicht notwendigerweise synchron ablaufen. Der Start des eigentlichen Experiments, also der Beginn der Messphase, muss allerdings bei allen Knoten synchron erfolgen. Der synchrone Start der Testsubjekte und Messwerkzeuge auf allen Knoten ist u.a. deshalb notwendig, damit schon die ersten (typischerweise Initialisierungs-) Nachrichten der Testsubjekte sinnvoll bearbeitet werden können.

Die synchrone Ausführung auf mehreren Rechnern eines verteilten Systems erfordert entweder synchronisierte Uhren oder ein externes synchrones Signal. Damit wir keine synchronisierten Uhren voraussetzen müssen, wählen wir eine Lösung mit einem synchronen Signal.

Wenn wir davon ausgehen, dass eine Rundsendung im Rechnernetz des Emulationssystems allen Empfängern gleichzeitig zugestellt wird, die Ausbreitungsverzögerung vom Steuerungsrechner zu allen anderen Knoten in der Testumgebung also identisch ist, können wir Rundsendungen zu Synchronisationszwecken einsetzen. Nach erfolgreichem Abschluss der lokalen Vorbereitungsphase sendet jeder Knoten eine „Bereit“-Nachricht an die zentrale Emulationssteuerung und wartet auf ein Signal zum Beginn der Messung. Hat die Emulationssteuerung die „Bereit“-Nachricht von allen beteiligten Knoten erhalten, sendet sie eine einzige „Start“-Nachricht durch eine Rundsendung an alle Knoten. Durch den verwendeten Ethernet-Vermittlungsknoten wird sichergestellt, dass die Rundsendung (innerhalb einer Toleranz von wenigen Mikrosekunden) an alle Knoten gleichzeitig zugestellt wird. Sofort nach dem Erhalt der „Start“-Nachricht wird auf jedem Knoten mit der Messphase begonnen, das heißt, die Testsubjekte, Lastgeneratoren und Messwerkzeuge werden aktiv (Abb. 4.2).

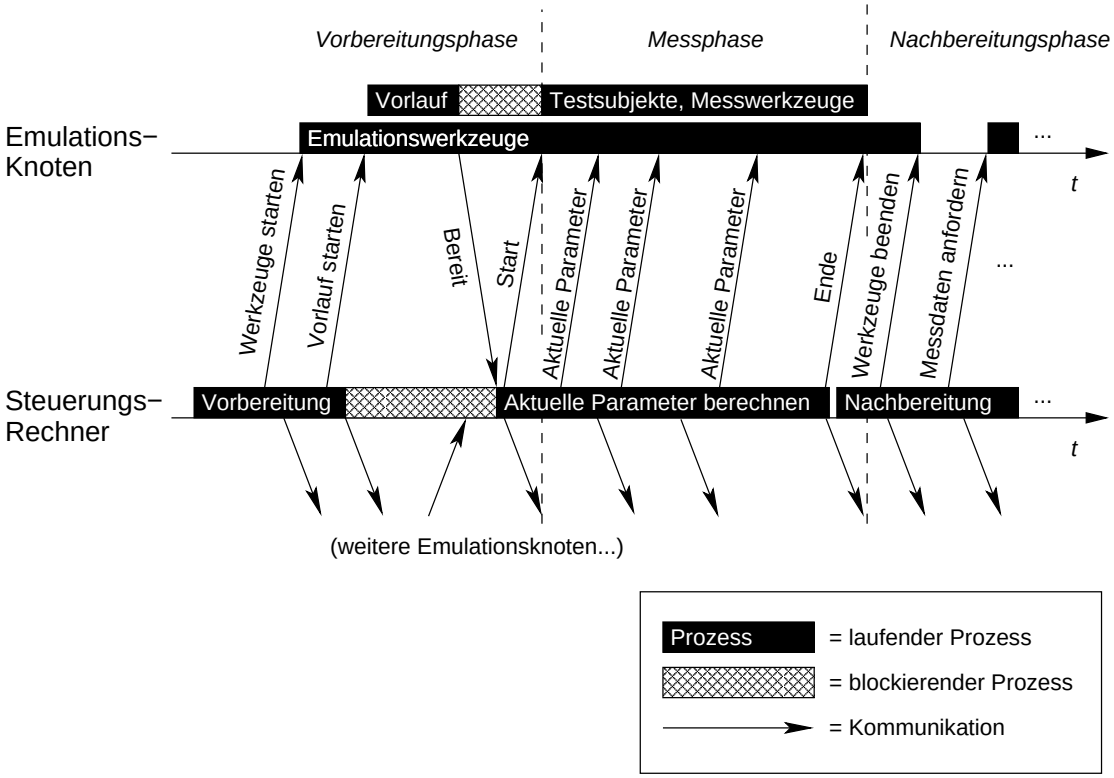

Abbildung 4.2: Ablaufsteuerung eines Experiments

4.2.3 Messphase

Während der Messphase bilden die verteilten Emulationswerkzeuge die konfigurierten Netzparameter autonom nach, ohne dazu Kontakt mit der zentralen Steuerungsinstanz zu benötigen. Wenn sich die nachzubildenden Parameter im globalen Netzmodell jedoch während des Experiments ändern, muss die Konfiguration der Emulationswerkzeuge aktualisiert werden. Dies geschieht über Aktualisierungsnachrichten, die von der Emulationssteuerung bei Bedarf an die verteilten Emulationswerkzeuge verschickt werden. Die Werkzeuge setzen diese Parameteränderungen sofort um.

Zur Häufigkeit von Änderungsnachrichten kann kein allgemeingültiger Richtwert gegeben werden; die optimale Aktualisierungsfrequenz hängt von der Leistung der zentralen Steuerung und vor allem von der Größe und der Dynamik des Szenarios ab. Es ist in jedem Fall sinnvoll, Änderungsnachrichten nicht regelmäßig, sondern nur bei Bedarf zu verschicken, also wenn Parameter sich in relevantem Maße geändert haben. Messungen zu den Kosten von Aktualisierungsnachrichten finden sich in Abschnitt 6.3.

Sind zu einem Zeitpunkt umfangreiche Parameteränderungen nötig, kann wie zu Beginn eines Experiments ein Synchronisierungsproblem entstehen: Um beispielsweise alle Netzverbindungen aller Knoten eines Szenarios zum gleichen Zeitpunkt auszuschalten, muss die Emulationssteuerung an die Werkzeuge auf allen Knoten eine Aktualisierungsnachricht schicken. Werden die Nachrichten nacheinander geschickt, so tritt das emulierte Ereignis nicht gleichzeitig ein. Die exakte Synchronisierung von Parameteränderungen erfordert zusätzlichen Aufwand und kann wiederum entweder durch synchronisierte Uhren oder durch ein externes Synchronisationssignal erreicht werden: Stehen synchronisierte Uhren zur Verfügung, können die Aktualisierungsnachrichten *vor* dem Ereignis verschickt werden und tragen einen Zeitstempel, zu dem sie gültig werden. Ohne synchronisierte Uhren kann eine Rundsendung der Emulationssteuerung den Gültigkeitszeitpunkt zuvor versendeter Aktualisierungsnachrichten signalisieren. Die exakte Synchronisierung von Parameteraktualisierungen hat sich allerdings in der Praxis in den bisher betrachteten Szenarien nicht als zwingend notwendig erwiesen und wird daher in dieser Arbeit nicht weiter berücksichtigt.

4.2.4 Nachbereitungsphase

Ein Experiment läuft normalerweise so lange, wie in der Szenariobeschreibung angegeben. Alternativ zu einer vorgegeben fixen Experimentdauer kann es bei manchen Experimenten sinnvoll sein, eine anwendungsabhängige Entscheidung über das Ende eines Experiments zu treffen, etwa wenn keine Nachrichten mehr unterwegs sind. Eine solche anwendungsabhängige Terminierung muss der Emulationssteuerung von einem zusätzlichen Terminierungsmodul signalisiert werden, das hier nicht weiter betrachtet wird.

Ist die spezifizierte Dauer des Experiments abgelaufen, oder das Experiment wurde aus einem anderen Grund terminiert, beendet die Emulationssteuerung die Messphase. Wenn die Testsubjekte, Messwerkzeuge und weiteren Hilfsprogramme, die auf den Knoten ausgeführt werden, sich nicht selbstständig nach der vorgegebenen Zeit beenden, müssen sie von der Emulationssteuerung beendet werden. Danach werden die Emulationswerkzeuge auf Grundeinstellungen rückgesetzt bzw. ebenfalls beendet, sowie die experimentspezifischen Einstellungen an der Netzinfrastruktur der Testumgebung zurückgenommen.

Nach dem erfolgreichen Abschluss einer Messung schließt sich eine Auswertungsphase an, in der lokale Messergebnisse von den einzelnen Knoten an einer zentralen Stelle gesammelt, in zeitliche Korrelation gebracht und nach den gewünschten Kriterien ausgewertet werden. Die Auswertungsphase ist spezifisch für jedes Experiment und wird daher an dieser Stelle nicht weiter betrachtet.

4.3 Emulationswerkzeug für Netze mit exklusivem Medienzugriff

Auf jedem Knoten der Testumgebung läuft eine Instanz eines Emulationswerkzeugs, das den Netzverkehr in Sende- und Empfangsrichtung beeinflussen kann. In diesem und dem folgenden Abschnitt wird im Einzelnen erklärt, wie ein Emulationswerkzeug die Basiseffekte (p_v, t_s, t_a, t_m) auf der Grundlage der Parameter aus dem Netzmodell (p_b, b, t_a) nachbildet.

Netze mit exklusivem Medienzugriff sind einfacher nachzubilden als solche mit gemeinsamem Medienzugriff. Daher wird in diesem Abschnitt zunächst die Funktionsweise eines Werkzeugs für die Emulations von Netzen mit exklusivem Zugriff erklärt. Im nächsten Abschnitt wird die Funktionalität auf Netze mit gemeinsamem Zugriff erweitert.

4.3.1 Lokales Netzmodell

Im globalen Netzmodell ist während eines Messlaufs für jedes Knotenpaar (i, j) ein aktueller Parametersatz (p_b, b, t_a) vorhanden. Das Emulationswerkzeug auf einem Knoten k bekommt von der zentralen Emulationssteuerung den Teil des Netzmodells übermittelt, der relevante Informationen für diesen Knoten enthält. Für die Matrixdarstellung M des Netzmodells (Tab. 4.1) heißt dies beispielsweise, dass das Emulationswerkzeug auf dem Knoten k die Werte $\{m_{ij} | i = k \vee j = k\}$ erhält; dies entspricht der Zeile und der Spalte der Matrix, die k enthält. Wenn Effekte nur in Sende- bzw. Empfangsrichtung nachgebildet werden, genügt auch nur die Zeile $\{m_{ij} | i = k\}$ bzw. die Spalte $\{m_{ij} | j = k\}$ einer Parametermatrix. Die einem Emulationswerkzeug lokal bekannten Teile des globalen Netzmodells nennen wir *lokales Netzmodell*. In Tab. 4.1 sind die Teile des globalen Modells, die das lokale Netzmodell des Emulationswerkzeugs auf Knoten 2 ergeben, grau hinterlegt.

	1	2	⋯	j	⋯	n
1	$(p_b, b, t_a)_{11}$	$(p_b, b, t_a)_{12}$	⋯	$(p_b, b, t_a)_{1j}$	⋯	$(p_b, b, t_a)_{1n}$
2	$(p_b, b, t_a)_{21}$	$(p_b, b, t_a)_{22}$	⋯	$(p_b, b, t_a)_{2j}$	⋯	$(p_b, b, t_a)_{2n}$
⋯	⋯	⋯	⋯	⋯	⋯	⋯
i	$(p_b, b, t_a)_{i1}$	$(p_b, b, t_a)_{i2}$	⋯	$(p_b, b, t_a)_{ij}$	⋯	$(p_b, b, t_a)_{in}$
⋯	⋯	⋯	⋯	⋯	⋯	⋯
n	$(p_b, b, t_a)_{n1}$	$(p_b, b, t_a)_{n2}$	⋯	$(p_b, b, t_a)_{nj}$	⋯	$(p_b, b, t_a)_{nn}$

Tabelle 4.1: Globales Netzmodell / lokales Netzmodell für Knoten 2

4.3.2 Rahmenverlust

Die Verlustwahrscheinlichkeit eines Rahmens wird vom Emulationswerkzeug aus der individuellen Rahmenlänge und der im Netzmodell vorhandenen Bitfehlerrate berechnet. Abhängig vom Rahmentyp kann die Rahmenverlustentscheidung vom Emulationswerkzeug auf dem sendenden oder dem empfangenden Knoten gefällt werden. Dies wird nachfolgend diskutiert.

Emulation in Senderichtung

Ein im Emulationswerkzeug des Senders verworfener Rahmen hat den Vorteil, dass er weder von der Netzinfrastruktur noch vom Empfänger verarbeitet werden muss. Damit werden die Ressourcen der Testumgebung geschont. Die Rahmenverlustentscheidung sollte also möglichst schon beim Senden getroffen werden. Dies ist genau dann möglich, wenn es sich um einen Rahmen an nur einen Empfänger handelt („Unicast“).

Nach der Berechnung der Rahmenverlustwahrscheinlichkeit p_v für einen konkreten Rahmen kann die Entscheidung über einen Rahmenverlust gefällt werden. Dazu wird zunächst eine Zufallszahl R mit $0 \leq R < 1$ generiert. Der Rahmen wird genau dann verworfen, wenn $R < p_v$. Wir gehen hier explizit davon aus, dass das Rechnernetz des Emulationssystems zuverlässig ist, zusätzliche Rahmenverluste durch dieses Netz also vernachlässigbar selten auftreten.

Emulation in Empfangsrichtung

Für Rahmen, die an mehrere Empfänger gleichzeitig adressiert sind, also Rundsendungen („Broadcast“) und Gruppensendungen („Multicast“), muss die Entscheidung über einen Rahmenverlust für jedes Sender-Empfänger-Paar getrennt getroffen werden. Wollte man diese Rahmenverlustentscheidungen beim Sender treffen, müssten Rund- und Gruppensendungen über mehrere Einzelsendungen realisiert werden. Es ist daher wesentlich effizienter, die Berechnung der Rahmenverlustwahrscheinlichkeit und die Emulation des Rahmenverlusts durch das

Emulationswerkzeug des empfangenden Knoten durchzuführen. Der Prozess der Rahmenverlustentscheidung entspricht dabei dem bei der Emulation in Senderichtung.

4.3.3 Verzögerung

Die nachzubildende Verzögerung setzt sich aus der Medienwartezeit, der Serialisierungsverzögerung und der Ausbreitungsverzögerung zusammen (siehe Abschnitt 3.2.2).

Medienwartezeit

Die Medienwartezeit t_m ist die Zeit zwischen dem Entgegennehmen eines Rahmens durch die Netz-Hardware und dem Beginn des Sendevorgangs. Da wir hier von einem Medium mit exklusivem Zugriff ausgehen, kann immer sofort gesendet werden, es sei denn, es befindet sich noch ein Rahmen *desselben Senders* in der Serialisierungsphase. Während dieser Phase signalisiert ein Netzwerkgerät der darüberliegenden Protokollschicht mit einem *Belegt-Signal*, dass das Medium temporär nicht verfügbar ist. Dieselbe Semantik muss ein Emulationswerkzeug, das ein Netzwerkgerät nachbildet, ebenfalls erfüllen. Wenn höhere Schichten weitere Rahmen über ein Gerät senden wollen, das im Moment belegt ist, dürfen sie den Rahmen dem Gerät nicht übergeben, sondern müssen ihn selbst zwischenspeichern (oder verwerfen). Die dadurch entstehende Wartezeit wird also implizit durch Warteschlangen in höheren Schichten erzeugt und braucht nicht von der Emulationsschicht nachgebildet zu werden.

Serialisierungsverzögerung und Ausbreitungsverzögerung

Die Werte für die Ausbreitungsverzögerung t_a werden von den Emulationswerkzeugen direkt aus dem Netzmodell entnommen; die Serialisierungsverzögerung t_s wird lokal aus der aktuellen Rahmenlänge und der Übertragungsrate aus dem Netzmodell berechnet ($t_s = l/b$). Die Serialisierungsverzögerung eines Rahmens hat zwei Effekte: Während sich ein Rahmen in der Serialisierungsphase befindet, gilt das Netzwerkgerät als belegt (s.o.). Außerdem wird die Auslieferung des Rahmens um t_s verzögert. Da auch die Ausbreitungsverzögerung zu einer zusätzlichen Verzögerung der Auslieferung eines Rahmens führt, können die beiden Effekte gemeinsam nachgebildet werden. Vom Sollwert der Verzögerung muss noch die ohnehin durch das Emulationssystem eingeführte Verzögerung $t_{\min}$ abgezogen werden. Somit berechnet sich die zusätzlich einzuführende Verzögerung zu $t_z = (t_s + t_a) - t_{\min}$. Da wir nur zusätzliche Verzögerungen einführen können, muss gelten: $t_z \geq 0$. Damit ergibt sich eine weitere Bedingung: $t_{\min} \leq (t_s + t_a)$; das heißt, Verzögerungen, die kleiner sind als $t_{\min}$, können nicht nachgebildet werden.

Realisierung durch eine Warteschlange

Die zusätzlich einzuführende Verzögerung t_z eines Rahmens kann sowohl beim Sender als auch beim Empfänger realisiert werden; aus der Sicht der höheren Schichten ist dabei kein Unterschied wahrzunehmen. Da eine Verzögerung beim Senden einfacher zu realisieren ist, gehen wir in der folgenden Beschreibung von einer Verzögerung beim Sender aus.

Ein Rahmen r wird zur Zeit t_0 dem Emulationswerkzeug zum Senden übergeben. Zunächst wird nun t_z für diesen Rahmen berechnet. Somit ergibt sich für diesen Rahmen die Aussendezeit $t' = t_0 + t_z$. Der Rahmen muss also bis zum Zeitpunkt t' im Emulationswerkzeug zwischengespeichert werden. Dazu wird eine Warteschlange benutzt, die auszusendende Rahmen in der Reihenfolge ihrer Aussendezeit enthält (die *Emulationswarteschlange*). Der Rahmen wird mit dem Zeitstempel t' versehen und an die chronologisch richtige Stelle[2] in die Emulationswarteschlange eingefügt. Der Rahmen mit dem kleinsten Zeitstempel befindet sich somit immer am Kopf der Warteschlange. Erreicht die Realzeit den Wert dieses Zeitstempels, wird der dazugehörige Rahmen ausgesendet und aus der Emulationswarteschlange gelöscht.

Die Größe der Emulationswarteschlange ist begrenzt: Zu einem Zeitpunkt befindet sich maximal ein Rahmen in der Serialisierungsphase. Die restliche Warteschlange repräsentiert die Kapazität des nachgebildeten Kanals, also das Produkt aus Übertragungsrate und Verzögerung („Bandwidth-Delay-Product“). Die maximale Warteschlangengröße in Bit berechnet sich also (bei maximaler Rahmenlänge $l_{\max}$ in Bit) zu: $q_{\max} = l_{\max} + b \cdot t_a$.

An dieser Stelle müssen wir annehmen, dass das Netzwerkgerät des Emulationssystems zum berechneten Aussendezeitpunkt bereit ist, einen Rahmen entgegenzunehmen; ansonsten würde eine zusätzliche, unerwünschte Verzögerung bis zur tatsächlichen Rahmensendung entstehen. Wir können genau dann garantieren, dass der berechnete Aussendezeitpunkt eingehalten werden kann, wenn die Übertragungsrate des Emulationssystems mindestens so groß ist wie die emulierte Übertragungsrate einer Verbindung. Die Übertragungsrate des Emulationssystems stellt somit eine obere Schranke für die nachbildbaren Übertragungsraten in einem Netzszenario dar. Das Rechnernetz eines Emulationssystems sollte also neben einem möglichst kleinen $t_{\min}$ auch eine möglichst hohe Übertragungsrate aufweisen, um einen großen nachbildbaren Parameterraum zu ermöglichen. Messungen zur tatsächlich erreichbaren Übertragungsrate in einem Emulationssystem finden sich in Abschnitt 6.4.

[2] Aufeinanderfolgende Rahmen an denselben Empfänger tragen ansteigende Zeitstempel. Die „chronologisch richtige Stelle“ ist in diesem Fall stets das Ende der Warteschlange. Werden jedoch abwechselnd Rahmen an verschiedene Empfänger mit stark unterschiedlicher Ausbreitungsverzögerung geschickt, kann es auch erforderlich sein, einen Rahmen *zwischen* zwei existierenden Rahmen in der Warteschlange einzufügen. Die Emulationswarteschlange ist also meistens, aber nicht immer reihenfolgeerhaltend.

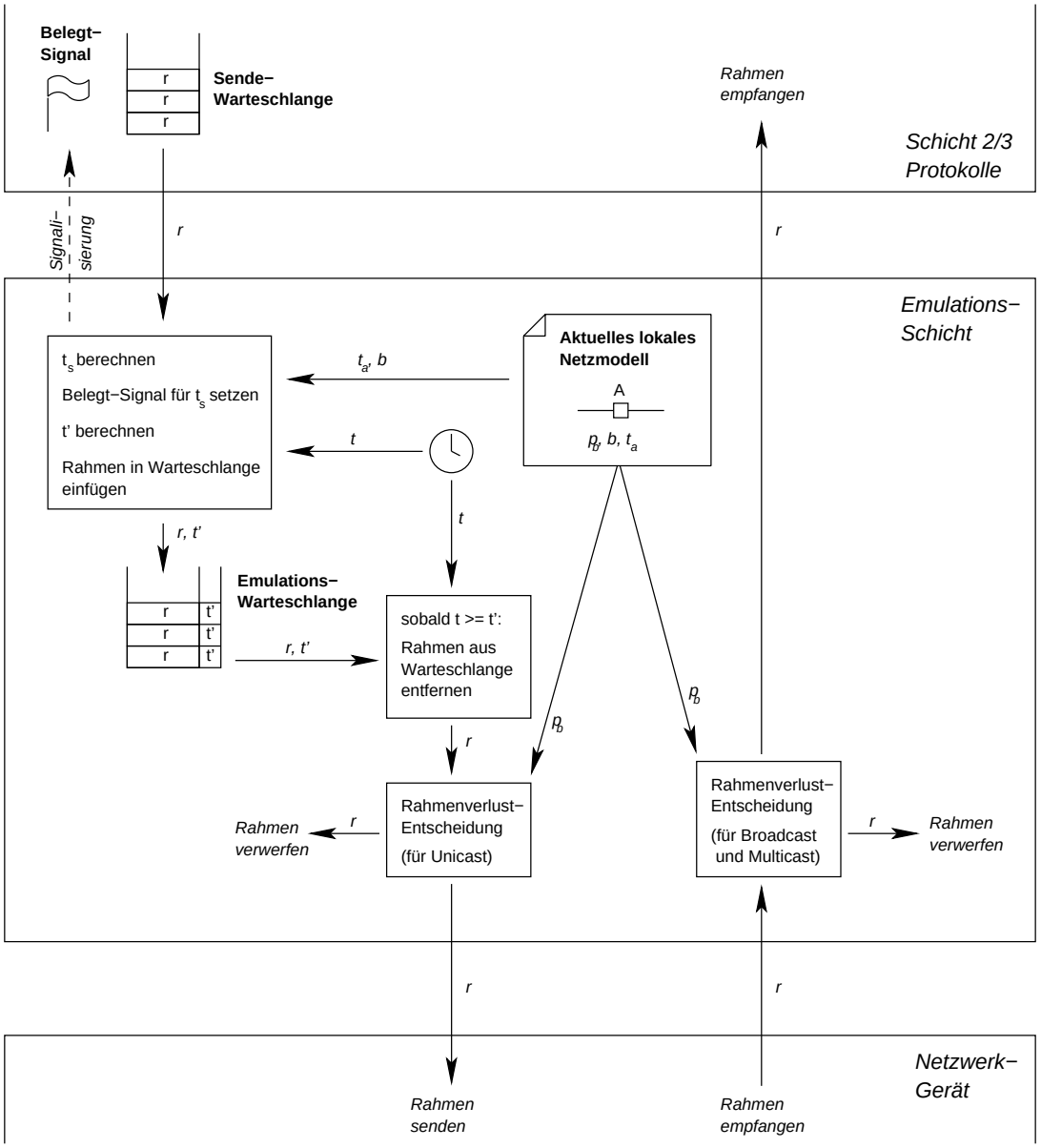

Abbildung 4.3: Funktionsweise eines verteilten Emulationswerkzeugs für exklusiven Medienzugriff

4.3.4 Zusammenfassung

Abb. 4.3 zeigt die Funktionsweise eines verteilten Emulationswerkzeugs bei exklusivem Medienzugriff in der Zusammenfassung. Grundlage der Emulation sind die Parameter im lokalen Netzmodell, das von der zentralen Emulationssteuerung ständig aktualisiert wird. Das lokale Netzmodell enthält die Übertragungsrate b, die Ausbreitungsverzögerung t_a und die Bitfehlerrate p_b. In der Abbildung sind diese Parameter zur Vereinfachung der Darstellung nur jeweils einmal vorhanden; tatsächlich kann für jedes Sender-Empfänger-Paar ein eigener Parametersatz existieren.

Wird einem Emulationswerkzeug ein Rahmen r zum Senden übergeben, werden zunächst die Verzögerungskomponenten berechnet. Dabei wird auch das „Belegt-Signal“ gesetzt, das dafür sorgt, dass während der Serialisierungsphase eines Rahmens dem Emulationswerkzeug kein weiterer Rahmen übergeben wird. Der Rahmen wird mit seiner berechneten Aussendezeit t' versehen und in die Emulationswarteschlange eingestellt. Zum Zeitpunkt t' wird er wieder aus der Warteschlange entnommen. Nun wird für Unicast-Rahmen die Rahmenverlustwahrscheinlichkeit berechnet und die Rahmenverlustentscheidung getroffen. Hier ist die Reihenfolge zu beachten: Unabhängig davon, ob ein Rahmen verloren gehen wird oder nicht, verbraucht er Übertragungskapazitäten des Senders; das heißt, er blockiert das Netzwerkgerät während des Sendevorgangs. Die Emulation der Serialisierungsverzögerung muss also *vor* der Emulation des Rahmenverlusts geschehen.

Auf Empfängerseite kommt der Rahmen am Emulationswerkzeug schon zum berechneten Auslieferungszeitpunkt an. Nur für Broadcast- und Multicastrahmen muss noch der Rahmenverlust emuliert werden. Wenn der Rahmen nicht verworfen wird, wird er umgehend an die höhere Schicht ausgeliefert.

4.4 Emulationswerkzeug für Netze mit gemeinsamem Medienzugriff

Auch für die Emulation einer Netztechnologie mit einem von mehreren Teilnehmern gemeinsam genutzten Medium sind die beiden Basiseffekte Rahmenverlust und Verzögerung nachzubilden. Trotzdem reicht die Funktionalität des in Abschnitt 4.3 vorgestellten Emulationswerkzeugs hierfür nicht aus, denn die Basiseffekte hängen nicht nur von den Parametern im Netzmodell und der Rahmenlänge ab, sondern auch vom Verhalten des Medienzugriffsprotokolls, das wiederum vom aktuellen Medienzustand beeinflusst wird.

Um trotzdem das Leistungsverhalten dieser Netztechnologien realistisch nachbilden zu können, ist es nötig, das Verhalten des jeweiligen Medienzugriffsprotokolls zu emulieren. Es bietet

sich also an, diese normalerweise in Hardware realisierte Protokollschicht als Teil eines speziellen Emulationswerkzeugs in Software nachzubilden, das spezifisch für diese Netztechnologie geeignet ist.

Es gibt sehr viele verschiedene Medienzugriffsprotokolle, die auf unterschiedlichen Grundkonzepten beruhen; in der Praxis sind neben zentralisierten (Koordinator-basierten) und kooperativen (z.B. Token-basierten) Verfahren vor allem Protokolle mit wahlfreiem Zugriff, die auf Trägererkennung („Carrier Sense Multiple Access“, CSMA) basieren, wichtig (siehe z.B. [Tan96]). Durch die sehr unterschiedlichen Konzepte ist es nicht möglich, das Verhalten von sämtlichen Protokollen mit einem einzigen Werkzeug nachzubilden. Um die prinzipielle Umsetzbarkeit der Emulation eines Medienzugriffsprotokolls zu zeigen, werden wir im Folgenden das Konzept des „virtuellen Trägersignals“ einführen, das als Basistechnologie für die Emulation von allen Protokollen geeignet ist, die mit Trägererkennung arbeiten. Auf der Grundlage dieses Konzepts können dann konkrete Medienzugriffsprotokolle der CSMA-Familie implementiert werden. Hierzu gehören neben der „Ethernet“-Protokollfamilie (IEEE 802.3) auch die WLAN-Protokolle (IEEE 802.11), die immer häufiger Verwendung finden. Für Einzelheiten zu den Implementierungen der Medienzugriffsprotokolle von Ethernet und WLAN wird auf eine Veröffentlichung [HMR03] und zwei Diplomarbeiten verwiesen [Mai02, Yan04].

4.4.1 Verzögerung und virtuelles Trägersignal

In der Realität wird der Belegungszustand des Netzmediums durch die Netz-Hardware ermittelt (Trägererkennung) und dem Medienzugriffsprotokoll bei Bedarf zur Verfügung gestellt. In der Emulation benötigt dazu jede Werkzeuginstanz ein lokales Medienmodell, das den aktuellen Zustand des emulierten Mediums aus der Sicht des jeweiligen Knotens repräsentiert. Der Zustand eines Mediums wird durch zwei Ereignisse verändert: Beginnt ein Knoten mit einer Rahmenübertragung, so wechselt der Medienzustand aller anderen beteiligten Knoten auf „belegt“, sobald das Signal sie erreicht, also nach der Ausbreitungsverzögerung t_a. Ist die Übertragung des Rahmens nach der Serialisierungsverzögerung t_s beendet, wechselt der Medienzustand wieder auf „frei“.

Die Idee des „virtuellen Trägersignals“[3] ist, dass jede Rahmensendung nicht nur an den oder die eigentlichen Empfänger, sondern über einen Rundsenderahmen an alle erreichbaren Knoten gleichzeitig geschickt wird. Ein solcher Rahmen dient nun nicht nur der Nutzdatenübertragung, sondern zusätzlich als Signalisierung eines virtuellen Trägersignals. Eigenschaften eines Rundsendemediums werden also auf der Grundlage von Rundsenderahmen nachgebildet, ohne

[3] Ein ähnliche Idee wird im IEEE 802.11-Standard für die Signalisierung von Kanalbelegungen zur Lösung des „Hidden Terminal Problem“ verwendet und dort „Network Allocation Vector“ genannt. Die Idee der Verwendung eines virtuellen Trägersignals für die Emulation der Eigenschaften eines gemeinsamen Mediums ist jedoch völlig neu.

annehmen zu müssen, dass die tatsächlich eingesetzte Netztechnologie des Emulationssystems Rundsendeeigenschaften hat.

Abb. 4.4 zeigt die Funktionsweise des virtuellen Trägersignals in einem Raum-Zeit-Diagramm: Die drei Knoten *A*, *B* und *C* teilen sich laut Szenariodefinition ein gemeinsames Übertragungsmedium. Das Emulationswerkzeug auf Knoten *B* bekommt einen Rahmen *r* zur Übertragung an Knoten *A* übergeben. Die Nachbildung des Medienzugriffsprotokolls auf *B* entscheidet, wann *r* gesendet werden darf. Im Beispiel ist diese Aussendezeit t'. Das Emulationswerkzeug schickt nun *r* nicht nur an *A*, sondern per Rundsendung an alle erreichbaren Knoten, und zwar sofort, ohne zuvor die Serialisierungsverzögerung t_s abzuwarten.

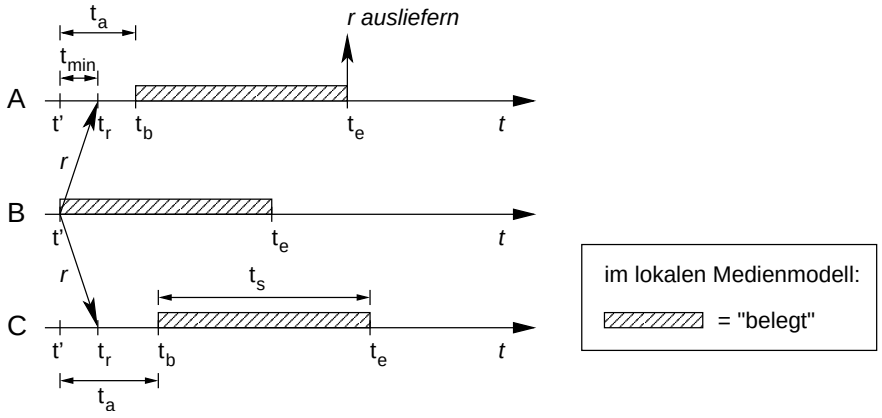

Abbildung 4.4: Nachbildung eines Trägersignals

Jedes Emulationswerkzeug muss nun seinen eigenen Medienzustand genau für den Zeitraum auf „belegt“ schalten, in dem in der Realität ein Trägersignal wahrnehmbar wäre. Dieser Zeitraum beginnt, wenn das Trägersignal des Rahmens bei dem jeweiligen Knoten ankommt, also nach der individuellen Ausbreitungsverzögerung t_a. Das Trägersignal endet, wenn der Rahmen komplett angekommen ist, und dauert damit für die Zeit der Serialisierungsverzögerung t_s an. Es gilt folglich für den Zeitpunkt des Beginns des Trägersignals $t_b = t' + t_a$, für das Ende $t_e = t_b + t_s$. Für den Sender gilt $t_b = t'$, da keine Ausbreitungsverzögerung zu berücksichtigen ist. Der tatsächliche Empfänger des Rahmens, im Beispiel Knoten *A*, muss dann den Rahmen zum Zeitpunkt t_e ausliefern. Dies gilt sinngemäß auch für Rahmen mit mehreren Empfängern. Diejenigen Knoten, die nicht Empfänger sind, benutzen den Rahmen nur für die Signalisierung des Trägersignals und können ihn anschließend verwerfen.

Zur Berechnung von t_b wird der Sendezeitpunkt t' benötigt. Alle Knoten außer dem Sender selbst kennen jedoch nicht t', sondern nur den Empfangszeitpunkt t_r. Die Differenz von t_r und t' ist $t_{\min}$, also die Zeit, die das Rechnernetz des Emulationssystems zur Übertragung des Rahmens benötigt. Wenn $t_{\min}$ bekannt ist, kann t_b folglich aus dem Empfangszeitpunkt t_r bestimmt werden: $t_b = t_r - t_{\min} + t_a$.

Hieraus folgt aber auch, dass $t_b < t_r \Leftrightarrow t_{\min} > t_a$. Das bedeutet, dass das virtuelle Trägersignal beginnen müsste, *bevor* der dazugehörige Rahmen und damit die Signalisierung des Trägersignals vom Emulationswerkzeug empfangen wurde, wenn $t_{\min} > t_a$. Da der Medienzustand frühestens mit dem Empfang des zugehörigen Signals aktualisiert werden kann, also zur Zeit t_r, ist das Medienmodell kurzzeitig falsch: Es wechselt zu spät von „frei“ auf „belegt“ (Abb. 4.5). Dieser Zusammenhang bekräftigt erneut die Forderung nach einem möglichst kleinen $t_{\min}$ (siehe Abschnitt 3.3.4 und Messungen in Abschnitt 6.1).[4]

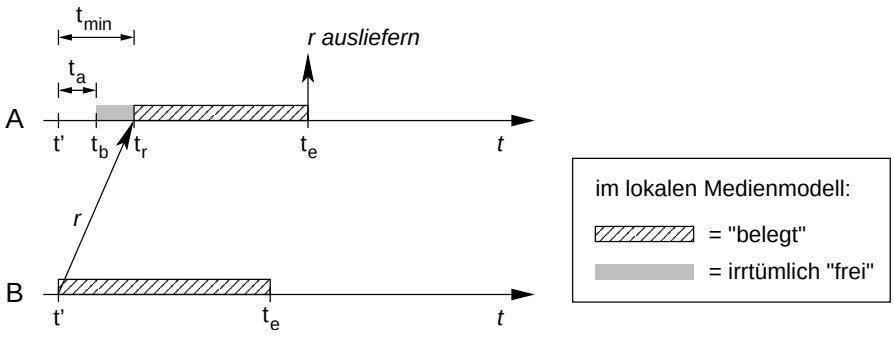

Abbildung 4.5: Vorübergehend falsches Medienmodell bei $t_{\min} > t_a$

Das hier beschriebene Vorgehen zur Emulation eines Trägersignals beinhaltet implizit die Emulation der Serialisierungsverzögerung und der Ausbreitungsverzögerung beim Empfänger. Durch das Bereitstellen des aktuellen lokalen Medienzustandes auf jedem Knoten ist außerdem die Grundlage für die Nachbildung von Medienzugriffsprotokollen gegeben, die auf Trägererkennung basieren. Ein nachgebildetes Medienzugriffsprotokoll kann somit auch die Medienwartezeit emulieren, indem es zu sendende Rahmen zurückhält.

4.4.2 Rahmenverlust

Außer der Verzögerung müssen auch bei der Emulation eines gemeinsamen Mediums Rahmenverluste nachgebildet werden, die sich aus sporadischen Bitfehlern ergeben oder die Konnektivität der Knoten definieren. Diese Rahmenverluste entstehen in der Realität auf dem Übertragungsmedium, müssen also auch in der Emulation logisch *unterhalb* des nachgebildeten Medienzugriffs behandelt werden. Da das Konzept des virtuellen Trägersignals ausschließlich Rundsenderahmen verwendet, kann die Entscheidung über den Rahmenverlust immer erst beim Empfänger getroffen werden. Ansonsten ist der Prozess der Rahmenverlustentscheidung mit dem für die Emulation eines exklusiven Mediums beschriebenen Prozess identisch (siehe Abschnitt 4.3.2).

[4] $t_{\min} \approx 100\,\mu s \gg t_a$, das entspricht je nach emulierter Technologie etwa $\frac{1}{10} \cdot t_s$. Das Medienmodell ist also tatsächlich kurzzeitig inkonsistent, allerdings nur zu max. 10% der Zeit.

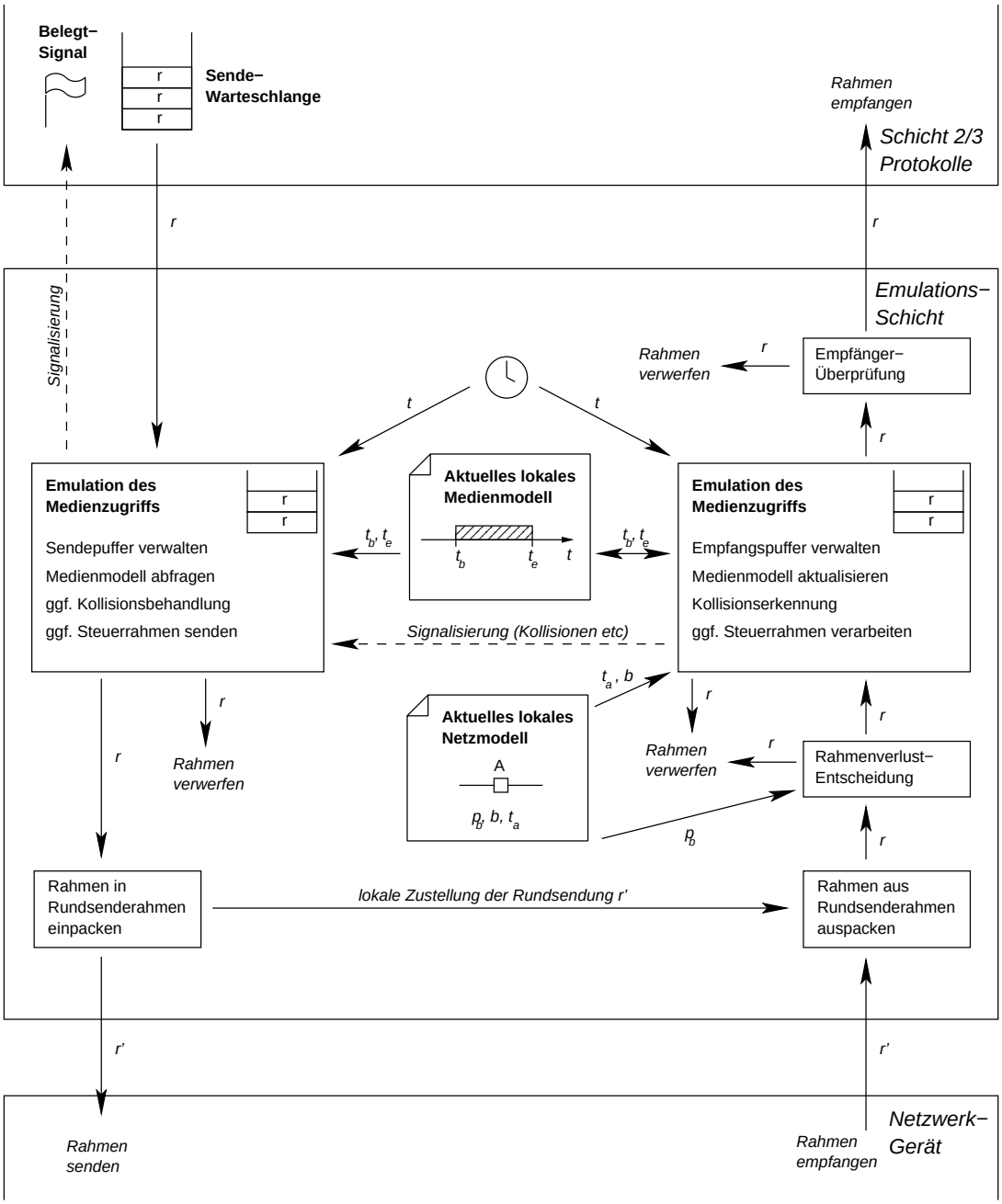

Abbildung 4.6: Funktionsweise eines verteilten Emulationswerkzeugs für Medien mit gemeinsamem Zugriff

4.4.3 Zusammenfassung

Abb. 4.6 zeigt die prinzipielle Funktionsweise eines verteilten Emulationswerkzeugs für Medien mit gemeinsamem Zugriff in der Zusammenfassung, ohne auf die Details eines bestimmten Medienzugriffsprotokolls einzugehen. Entsprechend dem Emulationswerkzeug für exklusiven Medienzugriff bildet auch hier ein ständig von der zentralen Emulationssteuerung aktualisiertes lokales Netzmodell die Grundlage der nachgebildeten Parameter. Im lokalen Netzmodell sind die aktuellen Werte für Ausbreitungsverzögerung, Übertragungsrate und Bitfehlerwahrscheinlichkeit abrufbar. Zusätzlich wird ein aktuelles lokales Medienmodell verwaltet, in dem der Belegungszustand des Mediums repräsentiert ist.

Wir betrachten zunächst die Senderichtung. Die Komponente, die das Medienzugriffsprotokoll nachbildet, kann einen eigenen Sendepuffer für Rahmen haben. Dies modelliert die Tatsache, dass auch in der Realität Netz-Hardware einen Rahmen schon entgegennimmt, bevor sicher ist, ob und wann er gesendet werden kann. Erst wenn dieser Sendepuffer voll ist, wird den höheren Protokollschichten „belegt“ signalisiert. Befinden sich ein oder mehrere Rahmen im Sendepuffer, muss die Rahmensendung vorbereitet werden, wie es das Medienzugriffsprotokoll vorsieht. Eventuell ist dazu zunächst der Austausch von Steuerrahmen mit dem Empfänger nötig (z.B. in IEEE 802.11 mit RTS / CTS: „Ready To Send“ / „Clear To Send“). In jedem Fall muss gewartet werden, solange das lokale Medienmodell im Zustand „belegt“ ist. Die Wartezeit in diesem Sendepuffer entspricht der Medienwartezeit t_m. Das Medienzugriffsprotokoll kann auch entscheiden, dass ein Rahmen an dieser Stelle verworfen wird (beispielsweise wenn eine Steuerrahmen-Interaktion fehlgeschlagen ist). Sobald ein Rahmen laut Medienzugriffsprotokoll gesendet werden darf, wird er in einen Rundsenderahmen eingepackt und verschickt. Da wir das Medienmodell im Empfangspfad aktualisieren werden, wird der Rahmen zusätzlich auch an den eigenen Empfangspfad weitergeleitet.[5]

In Empfangsrichtung wird zunächst der ursprüngliche Rahmen aus dem Rundsenderahmen ausgepackt. Vor jeder anderen Entscheidung wird überprüft, ob der Rahmen diesen Empfänger überhaupt hätte erreichen können, es wird also die Rahmenverlustentscheidung aufgrund der Bitfehlerwahrscheinlichkeit aus dem Netzmodell getroffen. Wird der Rahmen nicht verworfen, muss er zunächst im Empfangspuffer gespeichert werden. Jetzt wird der Sendezeitpunkt des Rahmens abgeschätzt ($t' = t_r - t_{\min}$). Dann kann Beginn ($t_b = t' + t_a$) und Ende ($t_e = t_b + l/b$) seines Trägersignals berechnet werden. Anhand dieser Werte wird das aktuelle Medienmodell überprüft und aktualisiert. Überlappt das errechnete Trägersignal mit einem schon existierenden Trägersignal im Medienmodell, so entspricht dies einer Kollision zweier Rahmen. Beide

[5] Die tatsächliche Implementierung ist an dieser Stelle effizienter: Selbstverständlich muss bei einer Rahmensendung nicht zusätzlich der gesamte Empfangspfad durchlaufen werden, um das lokale Medienmodell zu aktualisieren, Kollisionen zu erkennen etc. Da sich die Rechenschritte zur Überprüfung und Aktualisierung des Medienmodells für eigene und fremde Rahmen jedoch gleichen, wird hier eine einfache, wenn auch nicht die effizienteste Variante der Implementierung vorgestellt.

kollidierenden Rahmen werden verworfen, sowohl der neu angekommene, als auch der zum schon existierenden Trägersignal gehörende, der sich ebenfalls noch im Empfangspuffer befinden muss. Wenn das Medienzugriffsprotokoll mit Kollisionserkennung arbeitet, muss die Kollision auch der Sendeseite der Protokollimplementierung signalisiert werden. Erkannte Kollisionen können beispielsweise die erneute Sendung eines Rahmens bewirken. Wird das Ende eines Trägersignals ohne Störung (Kollision mit einem anderen Rahmen) erreicht, darf der entsprechende Rahmen weiterverarbeitet werden. Handelt es sich um einen an diesen Empfänger adressierten Steuerrahmen, muss er direkt vom Medienzugriffsprotokoll verarbeitet (z.B. beantwortet) werden. Datenrahmen werden schließlich, sofern sie die richtige Empfängeradresse tragen, an die höheren Protokollschichten ausgeliefert.

4.5 Netzinfrastruktur des Emulationssystems

Die Netzinfrastruktur eines Emulationssystems hat in erster Linie die Aufgabe, die Emulationsknoten untereinander und mit der zentralen Steuerung zu verbinden. Wenn die verwendeten Geräte weitergehende Einstellungen (z.B. Filterregeln) zulassen, können sie außerdem einen Teil der Emulationsaufgaben übernehmen. Verteilte Emulationswerkzeuge mit der oben beschriebenen Funktionalität sind zwar in der Lage, die Basiseffekte ohne Hilfe der Netzinfrastruktur autonom nachzubilden. Für einige Szenarien ist es jedoch wesentlich effizienter, wenn die Netzinfrastruktur zusätzlich für die Nachbildung benutzt wird.

Skalierungsproblem durch Rundsendungen

In vielen praktisch relevanten Netzszenarien ist die Konnektivität gering, das heißt, jeder Knoten hat im Vergleich zur Gesamtknotenanzahl nur zu wenigen anderen Knoten eine direkte Verbindung. Beispielsweise liegt in existierenden Netztopologien mit mehreren tausend Knoten die gemessene durchschnittliche Konnektivität bei ca. 2 bis 7 [RTY+00]. Ein globales Netzmodell enthielte bei diesen Szenarien also sehr häufig Bitfehlerraten von 1. Besonders bei Rundsendungen, die eine Rahmenverlustentscheidung beim empfangenden Knoten erfordern, wird ein Skalierungsproblem offensichtlich: Eine Rundsendung, die zunächst an alle Knoten eines Szenarios zugestellt wird, muss von den Emulationswerkzeugen auf fast allen Knoten beim Empfangen verworfen werden, weil keine direkte Konnektivität zum Sender besteht. Bei großen Szenarien mit vielen Rundsendungen bedeutet dies eine hohe unnötige Belastung der Emulationswerkzeuge. Das Problem der Rundsendungen verschärft sich noch, wenn Trägersignale emuliert werden, denn durch das Konzept des virtuellen Trägersignals wird mindestens eine Rundsendung für jede Rahmensendung erzeugt.

Emulation von Rahmenverlust durch die Netzinfrastruktur

Moderne Netzinfrastrukturen arbeiten mit programmierbaren Zentralkomponenten, die es erlauben, die Ausbreitung von Rahmen im Netz nach bestimmten Kriterien zu begrenzen. Die Möglichkeiten unterscheiden sich je nach Technologie und Geräteklasse. Übliche Methoden sind beispielsweise Filterregeln, die auf der Grundlage von Adressinformationen Rahmen verwerfen können, oder die Definition von abgeschlossenen Teilnetzen. Diese Möglichkeiten können zur Emulation des Effekts Rahmenverlust eingesetzt werden. Für alle Verbindungen mit $p_b = 1$ gilt auch (unabhängig von der Länge) $p_v = 1$; diese Rahmen können von der Netzinfrastruktur verworfen werden. Die Funktionalität der Emulationswerkzeuge auf den Knoten wird hiervon nicht beeinflusst. Die Emulation des Rahmenverlusts für Rahmen mit der Verlustwahrscheinlichkeit $p_v < 1$ übernehmen weiterhin die Emulationswerkzeuge, alle Rahmen mit $p_v = 1$ kommen gar nicht erst beim Werkzeug an.

Die praktische Umsetzung dieser Optimierung ist stark von den zur Verfügung stehenden Funktionen der Netz-Hardware abhängig. Eine mögliche Lösung, die das „Virtual LAN“-Konzept eines Ethernet-Vermittlungsknotens zu diesem Zweck ausnutzt, ist in Abschnitt 5.2 beschrieben.

4.6 Zusammenfassung

In diesem Kapitel wurde die Funktionsweise eines verteilten Emulationssystems mit zentraler Steuerung erläutert. Zu den Aufgaben der zentralen Steuerung gehört hauptsächlich die Verwaltung des globalen Netzmodells und die Koordination des Gesamtexperiments inklusive der verteilten Emulationswerkzeuge. Diese Werkzeuge bilden die spezifizierten Netzeigenschaften nach, indem sie in Sende- und Empfangsrichtung in den Kommunikationsstapel eingreifen und Rahmen verzögern bzw. verwerfen. Außerdem werden durch das Setzen des „Belegt-Signals“ erwünschte Seiteneffekte auf höheren Protokollschichten generiert. Netze mit exklusivem Zugriff können von Emulationswerkzeugen autonom nachgebildet werden. Für die Emulation von Netzen mit gemeinsamem Zugriff müssen mehrere Instanzen eines Emulationswerkzeugs kooperieren. Hierfür wurde das Konzept des virtuellen Trägersignals entwickelt, das als Grundlage für die Nachbildung von allen Netztechnologien verwendet werden kann, die mit Trägererkennung arbeiten. Schließlich wurde auf die Möglichkeit verwiesen, zusätzlich die Netzinfrastruktur einer Testumgebung zu Emulationszwecken einzusetzen, um einem Skalierungsproblem bei Szenarien mit vielen Rundsendungen vorzubeugen.

5

Realisierung

Nachdem im vorigen Kapitel die Funktionsweise eines Emulationssystems mit zentraler Steuerung und verteilten, kooperativen Werkzeugen erläutert wurde, zeigt dieses Kapitel die praktische Umsetzbarkeit eines solchen Systems anhand einer konkreten Realisierung. Dieses Kapitel beschreibt jedoch *nicht* die Realisierung eines Emulationssystems in allen technischen Details; dies würde den Rahmen dieser Arbeit bei weitem sprengen. Vielmehr wird im Folgenden ein Überblick über das Gesamtsystem gegeben, sowie einige wichtige Aspekte der Realisierung näher betrachtet. Für weitergehende technische Informationen verweisen wir auf das Handbuch des entwickelten Systems [HM04], in dem auch die Benutzung des Emulationssystems für Leistungsmessungen in verschiedenartigen Szenarien konkret erklärt wird. Weitere Details, vor allem zu Implementierungen von Werkzeugen, können in Veröffentlichungen [HLR02, HR02, HMR03, HMTR04], sowie in den Ausarbeitungen von Diplomarbeiten [Dud02, Mai02, Rep03, Yan04] und Studienarbeiten [Wur02, Wei03, Stö04, Cas05] nachgelesen werden.

Zunächst wird ein Überblick über die Hardware-Plattform gegeben, die uns für die Realisierung eines Emulationssystems zur Verfügung steht. Aufbauend auf den speziellen Möglichkeiten dieser Hardware wird erklärt, wie die Netzinfrastruktur zur Emulation von Netzwerkgeräten und Netztopologien benutzt werden kann. Danach wird für einen wichtigen Anwendungsfall – die Berechnung von Bitfehlerraten aus Knotenbewegungen in einem MANET – der Prozess der automatischen Berechnung von Netzparametern erläutert. Anschließend wird erklärt, wie sich Emulationswerkzeuge als Linux-Kernmodule realisieren lassen. Schwerpunkt ist dabei die Integration in den Kontroll- und Datenfluss des Kommunikationsstapels. Zum Schluss des Kapitels werden die möglichen Emulationsartefakte, also die unbeabsichtigten Effekte von Emulationswerkzeugen, diskutiert und Methoden zu deren Verminderung vorgestellt.

5.1 Das „Network Emulation Testbed“ (NET)

Im Jahre 2002 wurde von der Abteilung „Verteilte Systeme“ des IPVS der Universität Stuttgart ein Clustersystem beschafft, das ausschließlich für den Betrieb als Testumgebung mit emulierten Netzeigenschaften vorgesehen war: Das „Network Emulation Testbed“, abgekürzt NET (Abb. 5.1). NET besteht aus 64 identisch ausgestatteten Standard-PCs[1], die als Emulationsknoten verwendet werden, und einem leistungsfähigen Steuerungsrechner[2], der die zentrale Emulationssteuerung ausführt. Auf allen Rechnern läuft ein Linux-Betriebssystem. Das „Emulationsnetz“, das alle Rechner verbindet, basiert auf einem monolithischen Gigabit-Ethernet-Vermittlungsknoten, der VLANs[3] unterstützt. Dieses Netz ist für die Kommunikation der Testsubjekte innerhalb eines Emulationsszenarios vorgesehen. Ein vollkommen separates „Steuerungsnetz“ (Fast Ethernet Technologie) ermöglicht die Kommunikation des Steuerungsrechners mit allen Emulationsknoten unabhängig vom Emulationsnetz. Das Steuerungsnetz ist ausschließlich für administrativen Netzverkehr bestimmt.

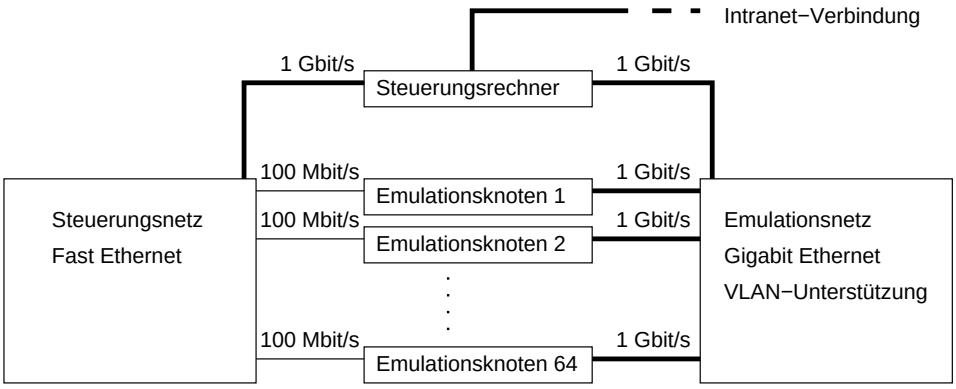

Abbildung 5.1: Hardwarearchitektur des NET

Diese Hardwarearchitektur eignet sich aus mehreren Gründen besonders gut für die Emulation von Rechnernetzen:

- Die große Anzahl von 64 Emulationsknoten ermöglicht die Emulation relativ großer Szenarien. In der von uns gewählten Architektur mit einem Emulationsknoten pro Szenarioknoten sind also Netzszenarien mit 64 Knoten möglich.[4]

[1] Intel Pentium 4 Prozessor mit 2,4 GHz Taktfrequenz, 512 MB Hauptspeicher, 40 GB Festplattenspeicher

[2] zwei Intel Xeon Prozessoren, jeweils 2,4 GHz Taktfrequenz, 2 GB Hauptspeicher, RAID Plattensystem mit 541 GB Brutto-Kapazität

[3] „Virtual LAN“, virtuelles lokales Netz

[4] Viele Szenarien benötigen wesentlich mehr als 64 Knoten. Durch Knotenvirtualisierung kann die Anzahl der für ein Szenario verfügbaren Knoten um mindestens eine Größenordnung erhöht werden. Dies ist jedoch nicht Inhalt dieser Arbeit. Auf die Idee der Knotenvirtualisierung wird im Ausblick (Abschnitt 7.2) verwiesen.

- Die hohe Übertragungsrate im Emulationsnetz[5] ermöglicht hohe Übertragungsraten in Emulationsszenarien (vgl. Abschnitt 4.3.3 und Messungen in Abschnitt 6.4).
- Durch die Verwendung eines Gigabit-Vermittlungsknotens als Grundlage des Emulationsnetzes ist der Wert für die systembedingte Verzögerung ($t_{\min}$) sehr klein (Messungen siehe Abschnitt 6.1). Ein möglichst kleiner Wert von $t_{\min}$ ist wichtig für einen großen emulierbaren Parameterbereich (Abschnitt 4.3.3) und die erzielbare Genauigkeit bei der Emulation eines gemeinsamen Mediums (Abschnitt 4.4.1).
- Die verwendete Netztechnologie Gigabit Ethernet garantiert zwar theoretisch nicht die fehlerfreie Zustellung von Rahmen, ist jedoch im praktischen Betrieb äußerst zuverlässig: Rahmenverluste bzw. -veränderungen treten vernachlässigbar selten auf. Dies wird von unserem Emulationskonzept vorausgesetzt, insbesondere für die Emulation von Rahmenverlusten (Abschnitt 4.3.2).
- Durch die physische Trennung von Emulationsnetz und Steuerungsnetz ist garantiert, dass der Netzverkehr im Emulationsszenario nicht durch administrativen Netzverkehr gestört wird.
- Das Emulationsnetz unterstützt den Betrieb von VLANs. Dies ist aus zwei Gründen vorteilhaft: Einerseits können dadurch vom Emulationsnetz Emulationsaufgaben übernommen werden, andererseits ist knotenseitig die effiziente Einführung von virtuellen Netzwerkgeräten möglich. Virtuelle Netzwerkgeräte ermöglichen die Emulation von Knoten mit mehr Netzwerkgeräten als tatsächlich vorhanden sind (siehe Abschnitt 5.2).
- Durch die Kombination von kostengünstiger Standard-Hardware (PCs, Ethernet) und kostenlos erhältlicher Betriebssoftware (Linux) sind Systeme der NET-Architektur in der Anschaffung im Vergleich mit anderen Clustersystemen dieser Größe sehr preiswert. Somit ist die Voraussetzung dafür gegeben, dass andere Forschergruppen ähnliche Systeme beschaffen bzw. bestehende Systeme um die erforderliche Netzinfrastruktur aufrüsten und auf der Grundlage unserer Forschungsergebnisse eigene Messungen durchführen können.[6]

5.2 Emulation mit VLANs

In diesem Abschnitt wird beschrieben, wie sich VLANs, die durch die Netz-Hardware von NET unterstützt werden, für die Emulation nutzen lassen. Zwar sind sie für die Funktionalität

[5] Gigabit Ethernet war zur Beschaffungszeit von NET (2002) die schnellste verfügbare Standard-Technologie für Rechnernetze.

[6] Derzeit (Ende 2005) werden von uns entwickelte Emulationswerkzeuge außer am IPVS der Universität Stuttgart auch von Forschergruppen an der Philipps-Universität Marburg, der Case Western Reserve University in Cleveland (Ohio, USA) und der University of Utah in Salt Lake City (Utah, USA) in ähnlichen Testumgebungen benutzt.

eines Emulationssystems, wie es in Kapitel 4 beschrieben wurde, nicht zwingend notwendig; allerdings kann durch den Einsatz von VLANs die Belastung der Emulationswerkzeuge und der Bedarf an Netz-Hardware auf den Emulationsknoten minimiert werden.

Im Folgenden umreißen wir dazu zunächst kurz die grundlegende Funktionalität von VLANs. Danach zeigen wir, wie sich durch VLANs Netzressourcen virtualisieren lassen. Schließlich erläutern wir den Einsatz von VLANs für die Emulation von Verbindungstopologien.

5.2.1 Funktion von VLANs

Die VLAN-Technologie wurde durch die IEEE standardisiert (IEEE 802.1Q, [IEE03]). VLANs erlauben die Definition von in sich geschlossenen, privaten Teilnetzen innerhalb eines LANs. Die Netzinfrastruktur sorgt dafür, dass Netzverkehr nur innerhalb eines VLANs verbreitet wird. Davon sind insbesondere auch Gruppen- und Rundsendungen betroffen. Bei der „tagged VLANs“-Variante, die wir im NET-System verwenden, erfolgt die Zuordnung von Teilnehmern zu VLANs durch ein „VLAN tag“ (ein zusätzliches, das VLAN identifizierendes Attribut) im Rahmenkopf eines jeden Rahmens, der über das Netz gesendet wird. Anhand dieser Attribute werden die Rahmen von der Netzinfrastruktur gefiltert und nur an diejenigen Rechner weitergeleitet, die ebenfalls Teilnehmer des betreffenden VLANs sind. Um diese Attribute beim Sender eines Rahmens einzufügen und beim Empfänger wieder zu entfernen, wird eine zusätzliche Schicht im Protokollstapel benötigt: Der „VLAN-Treiber“. Der VLAN-Treiber setzt auf der Netz-Hardware auf und bietet höheren Protokollschichten die gleiche Schnittstelle wie ein Netzwerkgerät an.

5.2.2 Emulation von Netzwerkgeräten

Ein Rechner mit nur einem Netzwerkgerät kann Teil mehrerer VLANs sein. Daher ist es möglich, mehrere Instanzen eines VLAN-Treibers zu benutzen, die auf demselben Netzwerkgerät aufsetzen. Da ein VLAN-Treiber logisch ein eigenes Netzwerkgerät repräsentiert, das nur innerhalb des VLANs kommunizieren kann, ist es möglich, durch die VLAN-Technologie mehrere *virtuelle* Netzwerkgeräte auf einem *physischen* Netzwerkgerät zu emulieren. Die Eigenschaften jedes dieser virtuellen Netzwerkgeräte können wiederum getrennt durch jeweils eine Instanz eines Emulationswerkzeugs beeinflusst werden. Abb. 5.2 zeigt, wie ein Rechner mit drei physischen Netzwerkgeräten durch einen Rechner mit nur einem Netzwerkgerät, aber drei VLANs nachgebildet werden kann.

Durch virtuelle Netzwerkgeräte kann ein Emulationsknoten, der nur *eine* physische Verbindung zum Emulationsnetz hat, einen Knoten mit einer beliebigen Anzahl von Netzverbindungen emulieren. Der maximale Knotengrad im nachzubildenden Netzszenario ist also nicht auf

den Knotengrad des Emulationssystems beschränkt (im Gegensatz zu vielen alternativen Emulationsansätzen, z.B. [ZN03]). Die Tatsache, dass im NET-System jeder Emulationsknoten nur über eine einzige Verbindung zum Emulationsnetz verfügt, stellt daher keine Einschränkung für den maximalen Knotengrad der nachzubildenden Topologien dar.

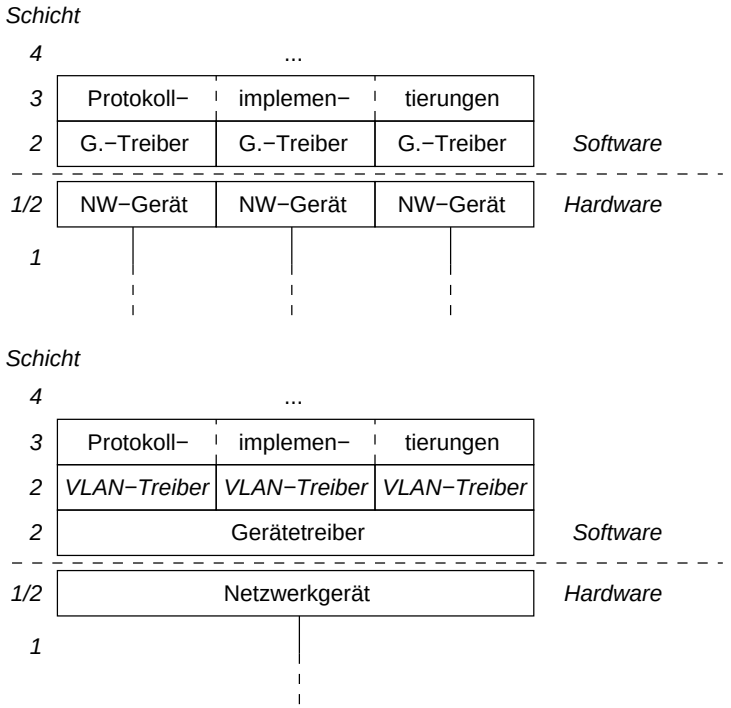

Abbildung 5.2: Mehrere Netzwerkgeräte in einem Knoten (oben) nachgebildet durch virtuelle Netzwerkgeräte (unten)

5.2.3 Emulation einer Verbindungstopologie

Neben der Einführung von virtuellen Netzwerkgeräten ermöglicht das VLAN-Konzept auch die Emulation einer Verbindungstopologie. Ein Rundsenderahmen, der mit einem VLAN-Attribut versehen ist, wird von der Netzinfrastruktur des Emulationsnetzes nur an diejenigen Knoten weitergeleitet, die als Teilnehmer des betreffenden VLANs registriert sind. Ein an einen einzelnen Empfänger adressierter Rahmen wird entsprechend nur dann zugestellt, wenn Sender und Empfänger Teilnehmer desselben VLANs sind. Mit VLANs können also beliebige Verbindungstopologien nachgebildet werden. Abb. 5.3 zeigt ein einfaches Beispiel: Laut Vorgabe können *A* und *B* direkt kommunizieren; *B*, *C* und *D* teilen sich zu dritt ein Kommunikationsmedium. Mit zwei VLAN-Definitionen im Emulationsnetz und den entsprechenden VLAN-Treibern auf den Emulationsknoten kann man diese Verbindungstopologie nachbilden, ohne dazu Emulationswerkzeuge auf den Knoten verwenden zu müssen.

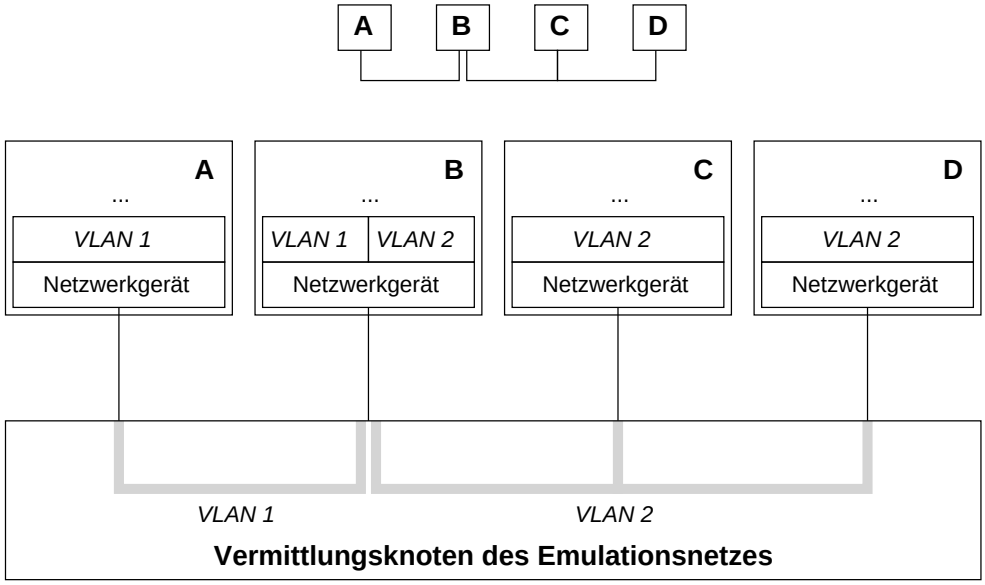

Abbildung 5.3: Verbindungstopologie (oben) und mit VLANs emulierte Topologie (unten)

Der Vermittlungsknoten des Emulationsnetzes nimmt damit den Emulationswerkzeugen einen Teil der Aufgaben ab, indem er den Rahmenverlust für alle Knotenpaare nachbildet, die nicht direkt miteinander kommunizieren können ($p_b = 1$). Für Szenarien mit statischer Verbindungstopologie und geringer Konnektivität bedeutet dies eine deutliche Entlastung der Emulationswerkzeuge. Die Rahmenverlustentscheidung für Rahmen mit einer nichttrivialen Bitfehlerrate ($0 < p_b < 1$) muss jedoch in jedem Fall weiterhin von Emulationswerkzeugen getroffen werden.

Grenzen der Topologieemulation mit VLANs

Die Anzahl der unterschiedlichen VLANs in einem System ist begrenzt. Der IEEE 802.1Q-Standard und die Hardware von NET erlauben die Definition von maximal 4092 verschiedenen VLANs.[7] Für die bisher von uns betrachteten Szenarien mit bis zu 64 Knoten ist diese Einschränkung jedoch irrelevant.

Vor der Verwendung von VLANs muss der Vermittlungsknoten des Emulationsnetzes mit der Information konfiguriert werden, welche VLAN-Attribute an welchen Anschlüssen zulässig sind. Diese Konfiguration benötigt einige Zeit (Messungen siehe Abschnitt 6.2). Während dieser Zeit sind die Anschlüsse des Vermittlungsknotens nicht benutzbar. Es ist also nicht möglich, *während eines Experiments* die VLAN-Konfiguration zu ändern, da sonst die Eigenschaften der

[7] Der Adressraum der VLAN-Attribute ist $2^{12} = 4096$ Adressen groß. Durch den VLAN-Standard sind drei besondere Adressen reserviert. In der Konfiguration des NET-Systems wird eine weitere VLAN-Adresse für die administrative Kommunikation mit dem Vermittlungsknoten benötigt.

emulierten Netzverbindungen im Emulationsnetz in unbeabsichtigter Weise verändert würden. Topologieemulation mit VLANs ist also nur für *statische* Verbindungstopologien geeignet. Verändert sich die Konnektivität von Knoten in einem Szenario während eines Experiments (z.B. aufgrund von sich bewegenden Knoten), muss die Topologieemulation weiterhin von Emulationswerkzeugen realisiert werden.

5.3 Beispiel einer Parameterberechnung mit höherwertigen Parametern

In Kapitel 4 wurde das Prinzip der Parameterberechnung als Funktion der Emulationssteuerung erläutert: Ausgehend von den höherwertigen Parametern, die in einer Szenariobeschreibung spezifiziert sind, müssen die im globalen Netzmodell repräsentierten Parameter (p_b, b, t_a) berechnet werden, bevor sie nachgebildet werden können. Dieser Berechnungsprozess wurde bisher nicht konkret beschrieben, da er spezifisch für die Menge der Parameter in der Szenariobeschreibung ist. Im Folgenden wird eine konkrete Realisierung für einen häufig benötigten Berechnungsprozess beschrieben: die Ermittlung von Bitfehlerraten aus Bewegungsanweisungen in einem MANET-Szenario. Dazu skizzieren wir zunächst das betrachtete Szenario, insbesondere die Szenariobeschreibung, die den Ausgangspunkt des Berechnungsprozesses darstellt. Danach beschreiben wir Schritt für Schritt den Prozess der Parameterberechnung bis zur Bitfehlerrate, wobei zwei alternative Lösungen unterschiedlicher Komplexität vorgestellt werden.

5.3.1 Beispielszenario: MANET

MANETs sind Netze aus Funkverbindungen, die „ad hoc“ zwischen mobilen Teilnehmern gebildet werden, ohne dass dazu zusätzliche Kommunikationsinfrastruktur benutzt wird. Wir gehen in unserem Beispielszenario davon aus, dass die Teilnehmer Personen sind, die sich in der Innenstadt von Stuttgart außerhalb von Gebäuden bewegen. Jeder Teilnehmer ist mit einem mobilen Rechner mit WLAN-Netzwerkgerät ausgerüstet. Durch die Mobilität der Teilnehmer und die begrenzte Reichweite der eingesetzten Funktechnologie ändern sich die nachzubildenden Netzeigenschaften zwischen den potentiellen Kommunikationspartnern ständig.

Szenariobeschreibung

Es wäre möglich, für das MANET-Szenario eine Szenariobeschreibung zu verwenden, welche die im globalen Netzmodell repräsentierten Parameter (p_b, b, t_a) für jedes Knotenpaar zu jedem Zeitpunkt während eines Experiments enthält. Dies ist jedoch so nicht praktikabel. Vielmehr ist es wünschenswert, als höherwertigen Parameter den *Bewegungsablauf* der einzelnen

Knoten vorzugeben. Für die Bewegung von Fußgängern, Autos etc. existieren „Mobilitätsmodelle“, auf deren Grundlage randomisierte, typische Bewegungsabläufe automatisch generiert werden können [SHB+03]. Wir wollen annehmen, dass die Szenariobeschreibung des MANET-Szenarios solche automatisch generierten Bewegungsabläufe für jeden Knoten enthält. Die Bewegungsabläufe stellen den Ausgangspunkt für die Parameterberechnung dar.

Um die Bewegungsabläufe der Knoten darzustellen, verwenden wir ein Beschreibungsformat, das für das Simulationswerkzeug *ns-2* entwickelt wurde [FV02]. Im Wesentlichen enthält eine solche Beschreibung eine chronologisch sortierte Abfolge von zweidimensionalen Bewegungsanweisungen der Art:

> Zur Zeit t: Knoten n beginnt mit einer Bewegung in Richtung der Koordinaten (x,y) mit Geschwindigkeit v.

Aufgrund der Popularität von ns-2 existieren zahlreiche Implementierungen von Mobilitätsmodellen, die Bewegungsabläufe in diesem Format ausgeben können. Durch die Übernahme des Beschreibungsformats von ns-2 werden diese Implementierungen auch für unser Emulationssystem nutzbar.

Zu einer kompletten Szenariobeschreibung gehören weitere Konfigurationsinformationen wie die Anzahl der Knoten, Angaben zu den Testsubjekten etc. (vgl. Abschnitt 4.2). Wir wollen uns hier jedoch auf die Berechnung der Parameter im globalen Netzmodell aus den Knotenbewegungen beschränken und setzen daher alle weiteren Konfigurationsinformationen als bekannt voraus. Alle statischen Parameter, die wir später im Berechnungsprozess benötigen werden, sind in Tab. 5.1 zusammengefasst: Wir verwenden als Szenariofläche die Innenstadt von Stuttgart. Für die Wellenlänge haben wir die mittlere Wellenlänge des WLAN-Frequenzbandes gewählt. Die Werte für die Sende- und die minimal benötigte Empfangsleistung (den Empfangsschwellwert) sind dem Datenblatt einer aktuellen WLAN-Karte[8] entnommen.

Szenariofläche	$2365\,\mathrm{m} \times 1905\,\mathrm{m}$
Antennentyp	Kugelstrahler
Antennenhöhe (h)	$1{,}5\,\mathrm{m}$
Wellenlänge (λ)	$0{,}123\,\mathrm{m} \quad (f = 2{,}442\,\mathrm{GHz})$
Brutto-Übertragungsrate (b)	$11\,\mathrm{Mbit/s}$
Sendeleistung (P_s)	$16\,\mathrm{dBm}$
Empfangsschwellwert (P_{es})	$-83\,\mathrm{dBm}$

Tabelle 5.1: Statische Parameter des Beispielszenarios

[8] „D-Link DWL-610“, erhältlich von: D-Link Deutschland GmbH, Schwalbacher Straße 74, 65760 Eschborn

5.3.2 Berechnungsprozess

Alle Parameter im globalen Netzmodell (p_b, b, t_a) können in einem realen WLAN von der Knotenposition abhängen. Zur Vereinfachung nehmen wir für dieses Beispiel jedoch an, dass die Brutto-Übertragungsrate b konstant ist. Außerdem vernachlässigen wir die Ausbreitungsverzögerung, die in einem WLAN bei 450 m Reichweite maximal $t_a = 450\,\mathrm{m}/c \approx 1{,}5\,\mu\mathrm{s}$ betragen kann. Wir beschränken uns also auf die Betrachtung der Bitfehlerrate p_b als dynamischen Parameter.

Abb. 5.4 zeigt den Berechnungsprozess im Überblick. Ausgehend von den Bewegungsanweisungen aus der Szenariobeschreibung werden die Bewegungen aller Knoten nachgebildet. Für jedes Paar von potentiellen Sendern und Empfängern wird dann die beim Empfänger ankommende Leistung P_e berechnet. Dieser Berechnungsschritt kann optional unter Berücksichtigung eines geographischen Umgebungsmodells durchgeführt werden. Schließlich wird die Bitfehlerrate p_b bestimmt.

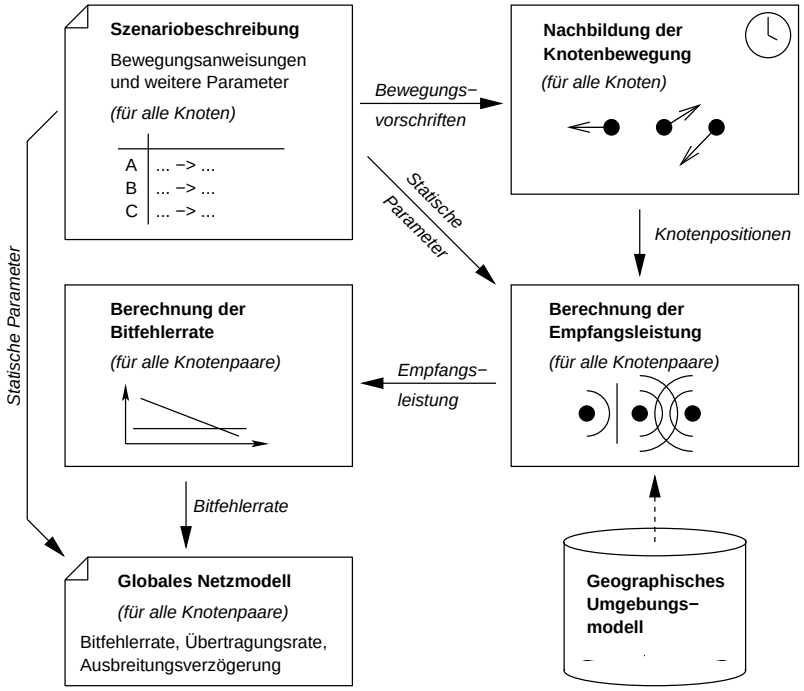

Abbildung 5.4: Vorgehen zur dynamischen Berechnung des globalen Netzmodells für ein MANET-Szenario

Nachbildung der Knotenbewegung

Auf der Grundlage der Bewegungsanweisungen, die im ns-2 Format vorliegen, werden während eines Experiments die Bewegungen aller Knoten in Echtzeit nachgebildet. Da wir die Netzpara-

meter iterativ berechnen, muss die prinzipiell kontinuierliche Bewegung der Knoten an dieser Stelle diskretisiert werden. Wir verwenden als Berechnungsintervall 1 s, d.h., die Positionen aller Knoten im Modell werden sekündlich neu berechnet. Die Ausgabe dieses Prozessschrittes ist eine Tabelle, in der für jeden Knoten dessen aktuelle geographische Position auf der Szenariofläche verzeichnet ist.

Berechnung von P_e ohne Berücksichtigung eines geographischen Umgebungsmodells

In einem MANET ist jeder Knoten potentieller Sender und Empfänger. Daher wird nun jedes *Paar* von Knoten (k_1, k_2) betrachtet. Für eine fiktive Sendung von k_1 an k_2 mit der Leistung P_s wird berechnet, wie groß die Leistung P_e des bei k_2 ankommenden Signals wäre. Hierzu betrachten wir zunächst ein Berechnungsmodell, das von einer ungestörten Wellenausbreitung ausgeht und daher keine geographischen Informationen über die Umgebung benötigt. Dieses Modell wird auch von den meisten MANET-Simulationen benutzt, die mit ns-2 durchgeführt werden [FV02].

Das „Free space / Two-ray ground model“ kombiniert zwei einfache Wellenausbreitungsmodelle: Zunächst wird die Entfernung d zwischen k_1 und k_2 berechnet. Für Entfernungen bis zu einem Schwellwert wird das „Free space model“ von Friis benutzt [Fri46]. Es geht von einer sich kugelförmig ausbreitenden Welle aus und modelliert die abnehmende Leistung proportional zur anwachsenden Kugeloberfläche. Für größere Entfernungen kommt das „Two-ray ground model“ zum Einsatz, das neben dem direkten Strahl zwischen Sender und Empfänger auch einen durch den Boden reflektierten zweiten Strahl in die Berechnung mit einbezieht [Rap01].

Hintergrund der Kombination zweier unterschiedlicher Modelle ist die Tatsache, dass das „Free space model“ für kleinere, das „Two-ray ground model“ für größere Entfernungen realistische Werte liefert. Der Entfernungsschwellwert für das Umschalten zwischen den beiden Modellen wird pragmatisch durch den Schnittpunkt der beiden Einzelfunktionen definiert. Somit erhält man eine stetige, abschnittsweise definierte Funktion für $P_e(d)$:

$$P_e(d) = \begin{cases} P_s \left(\frac{\lambda}{4\pi d}\right)^2 & \text{für} \quad d \leq \frac{4\pi h^2}{\lambda} \\ P_s \left(\frac{h}{d}\right)^4 & \text{für} \quad d > \frac{4\pi h^2}{\lambda} \end{cases}$$

Abb. 5.5 zeigt den Verlauf der Empfangsleistung in Abhängigkeit von der Entfernung für die in Tab. 5.1 angegebenen Parameter. Durch den ebenfalls eingezeichneten Empfangsschwellwert P_{es} kann man ablesen, dass die Kommunikationsreichweite in diesem Fall ca. 450 m beträgt.

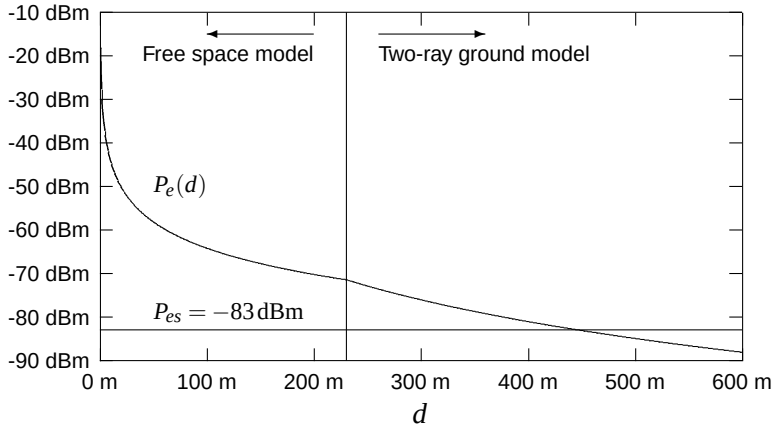

Abbildung 5.5: Verlauf der Empfangsleistung nach „Free space / Two-ray ground model“

Berechnung von P_e mit Berücksichtigung eines geographischen Umgebungsmodells

Das „Free space / Two-ray ground model“ bringt nur dann realistische Ergebnisse, wenn von einem unbebauten Gebiet ausgegangen werden kann. In einem Innenstadtszenario, wie wir es betrachten wollen, beeinflussen Gebäude die Funkwellenausbreitung jedoch erheblich. Wir haben deshalb ein alternatives Verfahren zur Berechnung von P_e betrachtet, das ein geographisches Modell der Umgebung in die Berechnung mit einbezieht.

Landstorfer beschreibt verschiedene Verfahren, die mit Strahlverfolgungsalgorithmen auf der Basis von 2,5-dimensionalen geographischen Daten realistische Vorhersagen für die Funkwellenausbreitung ermöglichen [Lan99]. Die Details dieser Verfahren sind zu komplex, als dass sie hier in Kürze wiedergegeben werden könnten. Auch eine Implementierung als Teil der Emulationssteuerung kommt wegen der Komplexität nicht in Frage. Um dennoch ein solches Verfahren in NET integrieren zu können, bedienen wir uns eines kommerziellen Werkzeugs, das den von Landstorfer beschriebenen „Intelligent Ray Tracing“-Algorithmus implementiert („ProMan“).[9]

Wenn P_e für alle Knotenpaare während eines Experiments berechnet werden soll, bleibt für eine Berechnung nur wenig Zeit: Bei einer Positionsaktualisierung pro Sekunde und 64 Knoten sind, wenn alle Knoten in Bewegung sind, bis zu 4032 Berechnungen pro Sekunde nötig.[10] Die Berechnungen für den „Intelligent Ray Tracing“-Algorithmus sind jedoch bei weitem zu aufwändig, um sie in dieser kurzen Zeit durchführen zu können. Daher haben wir uns entschieden, die Werte für P_e für sämtliche möglichen Positionspaare in einem Szenario vorzuberechnen. Um eine endliche Menge von Positionspaaren zu erhalten, muss die Szenariofläche durch ein

[9] ProMan ist ein Produkt der Firma AWE Communications, Moltkestr. 28, 71116 Gärtringen, http://www.awe-communications.de

[10] Wenn man von *symmetrischen* Werten ausgeht, sind $n_k \cdot (n_k - 1)/2 = 2016$ Paarungen zu betrachten, bei *asymmetrischen* Werten $n_k \cdot (n_k - 1) = 4032$ Paarungen.

Raster diskretisiert werden. Wir haben für ein konkretes Gebiet (Stuttgarter Innenstadt)[11] Werte in einem Raster von $5\,\mathrm{m} \times 5\,\mathrm{m}$ berechnet.[12] Während eines Experiments können die jeweils benötigten Werte für P_e direkt aus den vorberechneten Daten abgerufen werden.

Berechnung der Bitfehlerrate p_b

Aus der Empfangsleistung P_e kann nun ein Wert für die Bitfehlerrate p_b bestimmt werden, der als dynamischer Parameter im globalen Netzmodell abgelegt wird. Im einfachsten Fall werden nur die Extremwerte der Bitfehlerrate berücksichtigt; dazu kann der Empfangsschwellwert P_{es} aus Tab. 5.1 benutzt werden:

$$\begin{aligned} P_e \geq P_{es} \quad &\Rightarrow \quad p_b = 0 \\ P_e < P_{es} \quad &\Rightarrow \quad p_b = 1 \end{aligned}$$

Um zusätzlich Zwischenwerte der Bitfehlerrate zu berücksichtigen, können beispielsweise empirisch gewonnene Werte herangezogen werden. Wir übernehmen hierbei die Vorgehensweise von Pavon und Choi aus [dPPC03], die für eine *Simulation* denselben Berechnungsschritt benötigen. Grundlage der Berechnung sind Messwerte zum Zusammenhang von Signal-/Rauschabstand und Bitfehlerrate, die einem Datenblatt entnommen sind (Tab. 5.2).

P_e/P_r	5 dB	6 dB	7 dB	8 dB	9 dB	10 dB	11 dB
p_b	$1{,}2 \cdot 10^{-2}$	$6 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$7 \cdot 10^{-4}$	$2{,}5 \cdot 10^{-4}$	$8 \cdot 10^{-5}$	$2{,}7 \cdot 10^{-5}$
P_e/P_r	12 dB	13 dB	14 dB	15 dB	16 dB	17 dB	
p_b	$8 \cdot 10^{-6}$	$1{,}9 \cdot 10^{-6}$	$3{,}9 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$3 \cdot 10^{-8}$	$4 \cdot 10^{-9}$	

Tabelle 5.2: Zusammenhang von Signal-/Rauschabstand und Bitfehlerrate (empirische Daten aus [Int00], gemessen in einem WLAN bei 11 Mbit/s)

Zum Berechnen einer Bitfehlerrate muss zunächst ein Wert für den Rauschpegel P_r des Kanals angenommen werden. Für einen konkreten Signal-/Rauschabstand (P_e/P_r) kann dann in der Tabelle ein empirischer Wert für die Bitfehlerrate abgelesen werden. Zwischenwerte werden interpoliert.

[11] Die geographischen Daten wurden freundlicherweise vom Stadtmessungsamt Stuttgart zur Verfügung gestellt. Sie stammen aus der automatisch generierten „Automatischen Liegenschaftskarte“ (ALK).

[12] Bei einer Szenariofläche von $2365\,\mathrm{m} \times 1905\,\mathrm{m}$ ergibt dies 180213 mögliche Positionen und ca. 32 Mrd. Positionspaare. Wir speichern P_e in einer 4-Byte-Fließkommazahl und benötigen dafür insgesamt ca. 121 GB Speicherplatz. Durch die Benutzung der 64 PCs des NET-Systems zum parallelen Berechnen der Wellenausbreitung konnte die gesamte Vorberechnung unseres Szenarios in weniger als drei Tagen durchgeführt werden.

5.3.3 Diskussion

Abb. 5.6 veranschaulicht den Effekt der beiden alternativen Verfahren zur Berechnung der Empfangsleistung. Für dieselbe Senderposition (markiert durch ein weißes Kreuz) ist in der Abbildung jeweils der Bereich grau eingefärbt, in der das Signal des Senders empfangen werden könnte (also der Bereich von $P_e \geq P_{es}$, die Reichweite des Senders). Die schwarzen Flächen stellen Gebäude dar.

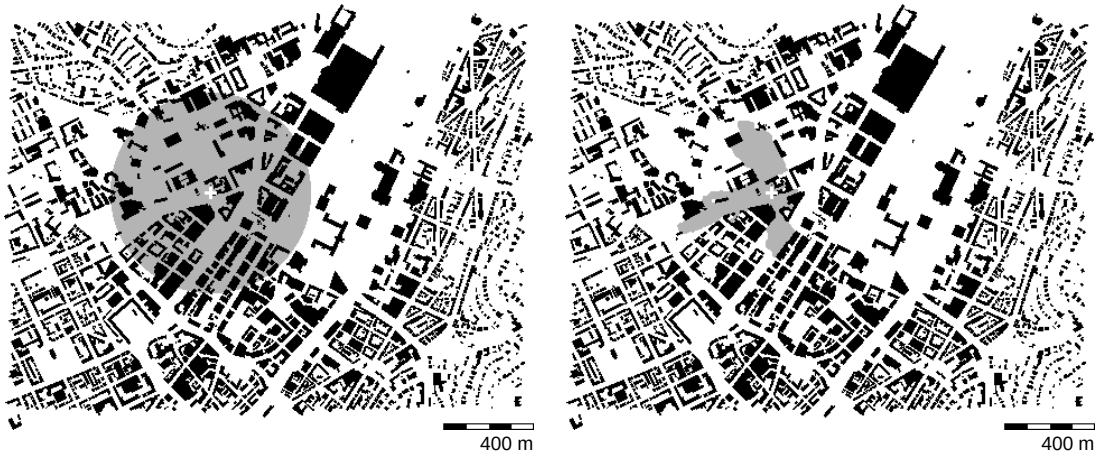

Abbildung 5.6: Sendereichweite eines WLAN-Senders in der Stuttgarter Innenstadt (links ohne, rechts mit Berücksichtigung von Gebäudedaten)

Aus dem Vergleich der Abbildungen schließen wir zwei Dinge. Einerseits sind die Ergebnisse beider Verfahren quantitativ vergleichbar: In beiden Fällen liegt die maximale Reichweite bei ca. 450 m.[13] Andererseits ist augenfällig, dass Gebäude in einer Stadt die Reichweite von WLAN erheblich verringern. Dies hat nachweislich auch einen großen Einfluss auf die Leistung von Vermittlungsprotokollen, wie wir mit vergleichenden Simulationen gezeigt haben [SHR05]. Für eine realistische Nachbildung solcher Szenarien ist die Verwendung eines geographischen Umgebungsmodells für die Parameterberechnung also unverzichtbar.

Beispielmessungen für ein MANET-Szenario, bei denen das hier beschriebene Verfahren zur Parameterberechnung verwendet wurde, finden sich in Kapitel 6.

[13] Im Datenblatt der WLAN-Karte „D-Link DWL-610“ wird eine maximale Reichweite von ca. 400 m angegeben. Wir vermuten, dass im Datenblatt aus rechtlichen Gründen absichtlich eine pessimistische Schätzung veröffentlicht wurde.

5.4 Realisierung eines Emulationswerkzeugs für Linux

In den Abschnitten 4.3 und 4.4 wurde die prinzipielle Funktionsweise eines Emulationswerkzeugs erläutert. In diesem Abschnitt wird erklärt, wie ein solches Emulationswerkzeug in transparenter Weise in einen konkreten Protokollstapel integriert werden kann. Der Protokollstapel ist als Teil des Betriebssystems realisiert. Die hier vorgestellte Lösung arbeitet mit dem Betriebssystem Linux. Eine Integration in andere Betriebssysteme wäre in ähnlicher Weise möglich, wenn die erforderlichen Programmierschnittstellen bzw. der Quellcode des Betriebssystems zur Verfügung stehen.

5.4.1 Integration in den Protokollstapel

In Abschnitt 3.1 wurde motiviert, auf welcher konzeptionellen Schicht ein Emulationswerkzeug eingreifen muss: zwischen den Protokollimplementierungen des Betriebssystems und der Netz-Hardware. Wir betrachten nun diesen Eingriffspunkt in der tatsächlichen Implementierung des Protokollstapels in Linux.

Die unterste in einem Linux-System in Software realisierte Protokollschicht ist die Vermittlungsschicht.[14] Die Implementierung der Vermittlungsschicht interagiert mit Netzwerkgeräten, um Rahmen zu senden und zu empfangen. Die Spezifika unterschiedlicher Netzwerkgeräte werden durch einen *Gerätetreiber* maskiert, der für jedes reale Netzwerkgerät unterschiedlich sein kann. Der Treiber stellt keine zusätzliche Funktionalität im Sinne einer Protokollschicht zur Verfügung,[15] sondern vereinheitlicht und vereinfacht lediglich die Schnittstelle zu den verschiedenen Geräten. Die Schnittstelle zwischen Vermittlungsschicht und Gerätetreiber stellt also den Eingriffspunkt für ein Emulationswerkzeug dar.

Abb. 5.7 zeigt die wesentlichen Funktionen der Schnittstelle zwischen Vermittlungsschicht und Gerätetreiber. Hier sind nur die Funktionen aufgeführt, die für die Kernfunktionalität eines Emulationswerkzeugs wichtig sind. Für Informationen zu den weiteren Elementen der Schnittstelle (Initialisierung, Konfiguration, Gerätestatistik, Fehlerbehandlung etc.) verweisen wir auf die Fachliteratur über die Implementierung des Linux-Protokollstapels [WPR+02].

[14] In Linux ist die Verwendung einer zusätzlichen Software-Sicherungsschicht zwar möglich, aber nicht üblich. Sie wird im Folgenden nicht weiter betrachtet.

[15] Teilweise müssen Gerätetreiber Funktionalität der Sicherungsschicht erbringen, die normalerweise vom Netzwerkgerät erbracht wird. Dies ist allerdings eine Ausnahme und dient wiederum der Vereinheitlichung der Schnittstelle von Netzwerkgeräten aus der Sicht des Betriebssystems.

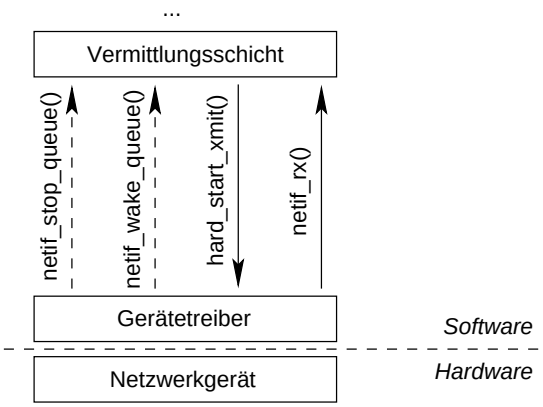

Abbildung 5.7: Schnittstelle zwischen Vermittlungsschicht und Gerätetreiber

Realisierung als Kernmodul

Sowohl die Implementierung der Vermittlungsschicht als auch die Gerätetreiber werden nicht als Benutzerprozesse, sondern im Betriebssystemkern ausgeführt. Ein Emulationswerkzeug, das ständig mit diesen Komponenten kommuniziert, muss aus Effizienzgründen auch als Teil des Betriebssystemkerns realisiert werden. Nur so können Kontextwechsel bei jeder Interaktion vermieden werden.

Die Architektur des Betriebssystemkerns von Linux ist teilweise modular und erlaubt es, Kernmodule dynamisch nachzuladen. Wir haben uns deshalb entschieden, unser Emulationswerkzeug als Kernmodul zu realisieren. Somit ist es ein vollwertiger Teil des Betriebssystemkerns, kann aber trotzdem während der Laufzeit des Betriebssystems modular installiert und wieder entfernt werden.

Es gibt verschiedene Entwurfsvorlagen für Kernmodule, die Schnittstellen vorgeben und somit die Integration eines Moduls in den Kern vereinfachen (z.B. für Dateisysteme, Gerätetreiber etc.). Da das Emulationswerkzeug auf unterster Ebene in den Kommunikationsstapel eingreifen soll, bietet es sich an, es als *virtuellen Gerätetreiber* für Netzwerkgeräte zu realisieren. Damit arbeitet es auf derselben konzeptionellen Schicht wie ein Gerätetreiber für ein reales Netzwerkgerät. Es stellt somit aus Betriebssystemsicht die Repräsentation eines „virtuellen Netzwerkgeräts“ dar. Im Folgenden wird erklärt, wie die Integration eines Emulationswerkzeugs in Sende- und Empfangsrichtung funktioniert.

Integration in Senderichtung

Jeder Gerätetreiber für Netzwerkgeräte stellt die Funktion hard_start_xmit() (für „hardware start transmission“) zur Verfügung, mit der ein Rahmen ausgesendet werden kann. Die Funktion

nimmt eine Referenz auf den zu übertragenden Rahmen als Argument entgegen. Gerätetreiber, die reale Netzwerkgeräte bedienen, sorgen in dieser Funktion für die Übertragung des Rahmens zum Netzwerkgerät. Virtuelle Gerätetreiber können den zur Aussendung übergebenen Rahmen in dieser Funktion verarbeiten (verändern, erweitern, verzögern, verwerfen) und ggf. danach wiederum an einen (virtuellen oder realen) Gerätetreiber zur Weiterverarbeitung übergeben, indem sie dessen Implementierung der hard_start_xmit()-Funktion aufrufen.

Über die Funktionen netif_wake_queue() und netif_stop_queue() teilt ein Gerätetreiber den Warteschlangen der Vermittlungsschicht mit, ob das Netzwerkgerät gerade bereit ist, Rahmen entgegenzunehmen, oder nicht. Ein Emulationswerkzeug kann diese Funktionen benutzen, um die Semantik des „Belegt-Signals“ zu implementieren (siehe Abschnitt 4.3.3). Beispielsweise signalisiert ein Emulationswerkzeug „belegt“, wenn sich gerade ein Rahmen in der Serialisierungsphase befindet, bzw. der Sendepuffer des Emulationswerkzeugs voll belegt ist.

Bei der Initialisierung eines Emulationswerkzeugs als neues virtuelles Netzwerkgerät muss dafür gesorgt werden, dass das Werkzeug in die Verarbeitungskette des Protokollstapels eingefügt wird, d.h. dass bei einer Rahmensendung die Vermittlungsschicht nicht mehr direkt die Funktion hard_start_xmit() eines Gerätetreibers für ein reales Netzwerkgerät, sondern die des vorgeschalteten Emulationswerkzeugs aufruft. Das wird analog zu jedem anderen Gerätetreiber für Netzwerkgeräte dadurch erreicht, dass das Emulationswerkzeug in die Geräteliste des Betriebssystems eingefügt, mit einer gültigen Netzadresse der Vermittlungsschicht versehen und in die Leitwegetabellen der Vermittlungsschicht eingetragen wird.[16] Die Implementierung der Vermittlungsschicht spricht dann automatisch bei ausgehenden Sendungen abhängig von der Zieladresse des zu sendenden Pakets den Gerätetreiber mit der richtigen Netzadresse an (Abb. 5.8).

Durch die nahtlose Integration in den Kommunikationsstapel als zusätzliches virtuelles Netzwerkgerät funktioniert die Einbindung eines Emulationswerkzeugs in Senderichtung für die bestehende Implementierung des Protokollstapels völlig transparent. Dies bedeutet insbesondere, dass die Vermittlungsschicht und alle höheren Schichten als Testsubjekte betrachtet werden können.

Integration in Empfangsrichtung

In Empfangsrichtung ist die Interaktion zwischen Gerätetreiber und Netzprotokollen komplizierter, weil die Rahmenbehandlung teilweise im Unterbrechungskontext ausgeführt wird. Abb. 5.9 zeigt einen Ausschnitt aus der Implementierung des Empfangspfads in Linux: Ein am

[16] Wir gehen hier zur Vereinfachung davon aus, dass als Vermittlungsschicht IP, bzw. ein anderes auf dem IP-Adressschema basierendes Vermittlungsprotokoll (z.B. eine Implementierung von AODV) existiert. Das Betriebssystem Linux unterstützt auch grundsätzlich andere Protokolle auf Vermittlungsschicht (z.B. IPX). Unser Integrationsansatz für die Emulationswerkzeuge arbeitet prinzipiell auch mit diesen Protokollen zusammen.

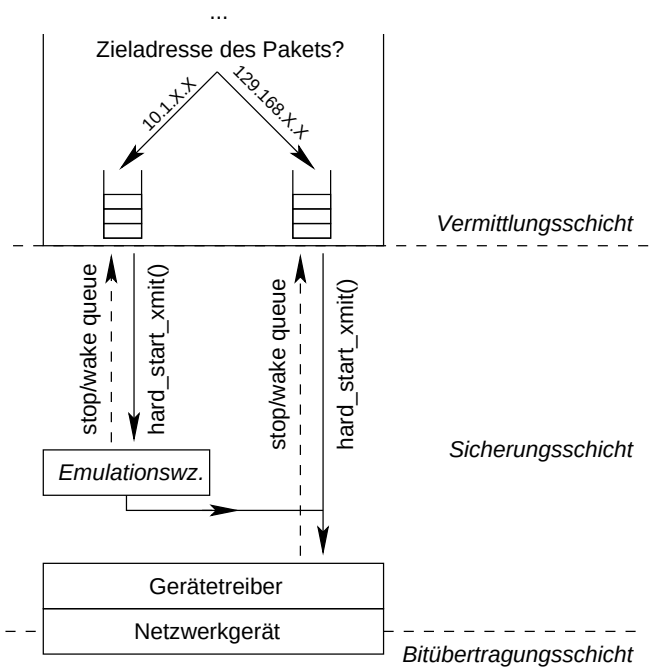

Abbildung 5.8: Integration in den Sendepfad

Netzwerkgerät ankommender Rahmen löst eine Unterbrechung aus. In der Unterbrechungsbehandlung wird der Rahmen vom Gerätetreiber entgegengenommen (1) und von diesem an die Hilfsfunktion netif_rx() übermittelt (2), die den Rahmen für die weitere Verarbeitung in eine Warteschlange der Vermittlungsschicht einstellt (3) und daraufhin die Unterbrechungsbehandlung beendet. Die weitere Verarbeitung geschieht dann nicht mehr im Unterbrechungskontext, sondern im normalen Betriebssystemkontext, sobald das Betriebssystem regelmäßig Rechenzeit zugeteilt bekommt. Auf Vermittlungsschicht beginnt die Verarbeitung mit dem sog. Protokoll-Multiplexer, der möglichst alle[17] seit dem letzten Bearbeitungszyklus eingegangenen Rahmen aus der Warteschlange abarbeitet und abhängig von einem Protokoll-Attribut im Rahmenkopf entscheidet, welches Vermittlungsschicht-Protokoll die weitere Verarbeitung übernehmen soll.

Es gibt zwei verschiedene Methoden, um Rahmen in diesem Prozess abzufangen und einem Emulationswerkzeug zu übergeben. Beide werden in verschiedenen Versionen unseres Emulationswerkzeugs verwendet.

[17] Bei großer Last kann es sein, dass in einem Bearbeitungszyklus nicht alle Rahmen der Warteschlange verarbeitet werden können.

1. Methode: Registrieren eines neuen Vermittlungsprotokolls

Ein Emulationswerkzeug kann sich als Vermittlungsprotokoll mit einem eigenen, bisher noch nicht vorhandenen Protokolltyp registrieren. Wenn das Emulationswerkzeug auf der Sendeseite alle ausgehenden Rahmen mit diesem Protokolltyp versieht, werden eingehende Rahmen automatisch vom Protokoll-Multiplexer nicht direkt an eine tatsächliche Vermittlungsschicht-Implementierung, sondern an das Emulationswerkzeug übergeben (Abb. 5.9, Schritt 4). Nachdem das Emulationswerkzeug den Rahmen verarbeitet hat, versieht es ihn mit dem ursprünglichen Protokolltyp (z.B. IP) und übergibt ihn wieder der Funktion netif_rx() (5), die den Rahmen erneut dem Protokoll-Multiplexer übergibt (6). Entsprechend dem neuen Protokolltyp wird der Rahmen nun zur tatsächlichen Implementierung der Vermittlungsschicht weitergeleitet (7).

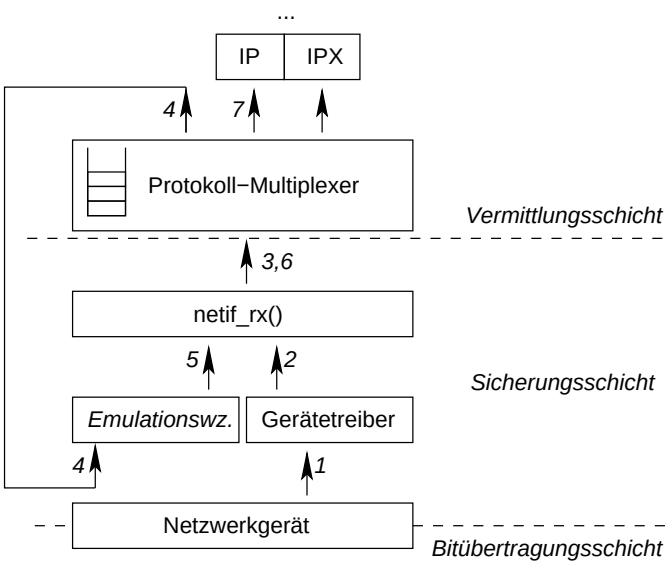

Abbildung 5.9: Integration in den Empfangspfad als zusätzliches Vermittlungsprotokoll

Diese Methode hat den Vorteil, dass die existierende Implementierung des Kommunikationsstapels nicht geändert werden muss. Allerdings kann ein Emulationswerkzeug auf Empfängerseite nur solche Rahmen empfangen, die senderseitig von einem Emulationswerkzeug entsprechend markiert wurden. In der hier beschriebenen Architektur ist diese Einschränkung zwar irrelevant, da alle Emulationsknoten mit Emulationswerkzeugen ausgestattet sind. Allerdings sind auch Architekturen denkbar, in denen Netzverkehr von externen Geräten eingespeist wird, die aus technischen Gründen (z.B. geschlossene Betriebssysteme) nicht mit einem Emulationswerkzeug ausgestattet werden können.

2. Methode: Modifikation von netif_rx()

Alternativ zur Einführung eines neuen Protokolltyps kann die Funktion netif_rx(), über die der gesamte eingehende Netzverkehr läuft, so ergänzt werden, dass sie für jeden empfangenen Rahmen zunächst eine Instanz des Emulationswerkzeugs aufruft.

Abb. 5.10 illustriert den Vorgang im Detail: Jeder von einem Netzwerkgerät empfangene Rahmen (1) wird vom Gerätetreiber an netif_rx() übergeben (2). Die geänderte Funktion ruft für jeden empfangenen Rahmen zunächst eine Instanz des Emulationswerkzeugs auf (3). Das Emulationswerkzeug kann nun entscheiden, ob es den Rahmen abfangen möchte oder nicht. In jedem Fall kehrt der Ausführungsstrang sofort wieder zur aufrufenden Funktion zurück (4). Abhängig vom Rückgabewert des Funktionsaufrufs wird der Rahmen an die Vermittlungsschicht weitergegeben (5) oder nicht.

Einen abgefangenen Rahmen kann das Emulationswerkzeug nun entweder verwerfen (Rahmenverlust) oder zu einem späteren Zeitpunkt durch einen Aufruf der Funktion netif_rx() (6) wieder in den Empfangspfad einspeisen (Verzögerung).

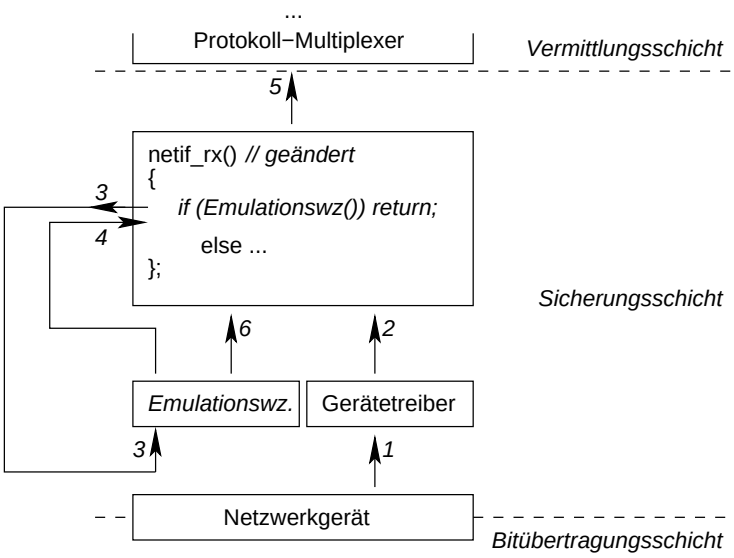

Abbildung 5.10: Integration in den Empfangspfad über die Funktion netif_rx()

Diese Methode des Eingriffs in den Empfangspfad ist flexibler, weil unabhängig vom Sender alle Rahmen abgefangen werden können. Allerdings muss die Funktion netif_rx() geändert werden, die zum Linux-Betriebssystemkern gehört. Es ist also die Anwendung eines „Patch“[18] und

[18]Das englische Wort „patch“ bezeichnet ursprünglich einen Flicken aus Stoff, und wird inzwischen auch in der Bedeutung „ein Stück Programmcode als Korrektur bzw. Ergänzung existierender Software“ verwendet. Eine griffige deutsche Übersetzung ist mir nicht bekannt.

eine erneute Übersetzung des Betriebssystems nötig, um die Integration eines Emulationswerkzeugs auf diese Weise vorzubereiten.

5.4.2 Konfigurationsschnittstelle

Zu Beginn und während eines Emulationslaufs muss ein Emulationswerkzeug mit den Netzparametern konfiguriert werden, die es nachbilden soll. In einem Emulationssystem mit verteilten Werkzeugen, wie wir es betrachten, gilt das sogar für eine große Anzahl von Werkzeugen auf verschiedenen Rechnern, die in effizienter Weise von der zentralen Emulationssteuerung mit Parametern versorgt werden müssen.

Das Emulationswerkzeug wird nicht als Benutzerprozess, sondern als Kernmodul ausgeführt, und kann deshalb nicht mit den üblichen Methoden der Prozesskommunikation angesprochen werden. Da es jedoch über die Schnittstellen eines Gerätetreibers verfügt, ist eine alternative Form der Kommunikation möglich: Um Einstellungen an Geräten vornehmen zu können, existiert in UNIX die „IOCTL“-Schnittstelle.[19] Diese Schnittstelle definiert ein generelles Aufrufformat für Einstellungen an Gerätetreibern. Jeder Treiber kann eine eigene IOCTL-Funktion mit mehreren Unterfunktionen definieren. Auf diese Weise kann ein Emulationswerkzeug die benötigten Funktionen bereitstellen, mit denen Benutzerprozesse Konfigurationseinstellungen vornehmen können. Auf ähnliche Weise ist es möglich, über die IOCTL-Funktion aktuelle Daten des Werkzeugs (z.B. Einstellungen oder Verkehrsstatistiken) abzufragen.

Die IOCTL-Funktion ist nur für lokale Aufrufe geeignet. Um die Emulationswerkzeuge auf den Emulationsknoten von der zentralen Emulationssteuerung aus zu konfigurieren, ist eine Indirektion notwendig: Auf jedem Emulationsknoten wird ein *Konfigurationswerkzeug* ausgeführt, das auf Nachrichten der zentralen Steuerung wartet und ggf. per IOCTL-Funktion an ein lokales Emulationswerkzeug weitergibt (siehe Abb. 5.11). Für die Kommunikation zwischen zentraler Steuerung und lokalen Konfigurationswerkzeugen können die üblichen Methoden zur Interprozesskommunikation verwendet werden. In unserer Implementierung verwenden wir für die Konfigurationsnachrichten einfache UDP-Pakete. Für diese administrative Kommunikation zwischen dem Steuerungsrechner und den Emulationsknoten wird im NET-System das Steuerungsnetz verwendet. Dadurch ist gewährleistet, dass die Konfigurationsnachrichten nicht mit dem Netzverkehr im emulierten Szenario interferieren können.

[19] IOCTL steht für *Input/Output Control*, da es primär eine Schnittstelle für die Konfiguration von Ein-/ Ausgabegeräten ist.

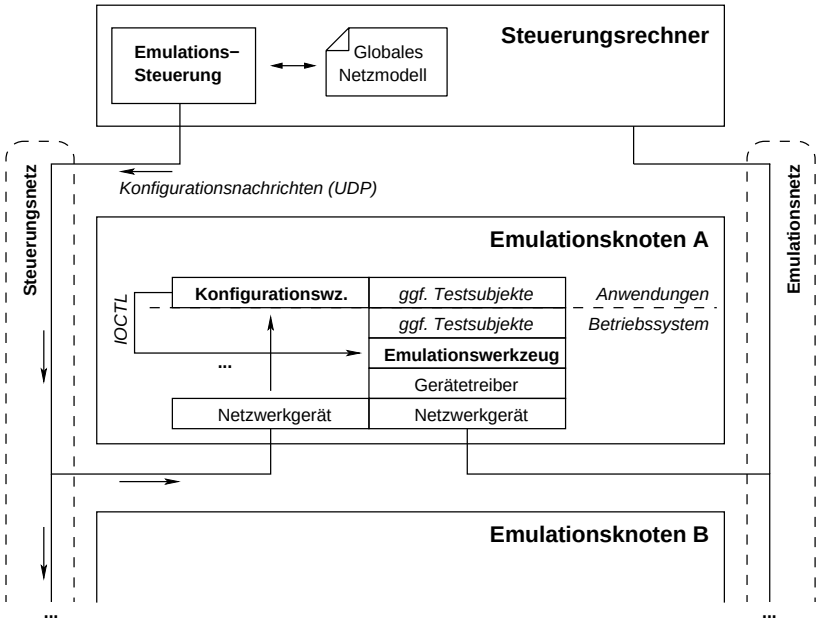

Abbildung 5.11: Konfiguration der lokalen Emulationswerkzeuge durch die zentrale Emulationssteuerung

5.5 Emulationsartefakte

Ein nach den Vorgaben in Kapitel 4 realisiertes Emulationswerkzeug könnte spezifizierte Netzeigenschaften exakt nachbilden, solange sie sich in den theoretisch möglichen Parameterbereichen bewegen (vgl. Abschnitt 4.3.3: $t_{\min}$ ist die Untergrenze der emulierbaren Verzögerung; die Übertragungsrate des Emulationsnetzes ist die Obergrenze der emulierbaren Übertragungsrate).

An einigen Stellen ist es jedoch aus praktischen Gründen nicht möglich, die spezifizierte Funktionalität exakt zu realisieren. Das Verhalten eines realen Emulationswerkzeugs kann also in bestimmten Fällen vom spezifizierten Verhalten abweichen. Dies kann dazu führen, dass der nachgebildete Netzverkehr unerwünschte Eigenschaften hat, die vom Emulationswerkzeug eingeführt wurden. In diesem Fall sprechen wir von „Emulationsartefakten“. Im Folgenden werden kurz mögliche Emulationsartefakte, deren Ursprung und Relevanz, sowie mögliche Methoden zu deren Verringerung besprochen.

5.5.1 Artefakte beim Rahmenverlust

Für die Rahmenverlustentscheidung wird eine Zufallszahl benötigt. Echte Zufallszahlen in einem ansonsten deterministisch organisierten Rechner zu erzeugen ist nur mit zusätzlicher

Hardware möglich, etwa durch Sensorik für thermisches Rauschen. Da wir solche Hardware im Emulationssystem nicht voraussetzen wollen, bleibt nur die Verwendung von „Pseudo-Zufallszahlen“: Als Zufallszahlen werden die aufeinanderfolgenden Zahlen einer Zahlenfolge verwendet. Die Folge muss die Eigenschaft haben, dass jede Zahl des Definitionsbereichs in der Folge gleich häufig vertreten ist. Die Zahlen einer solchen Folge sind also *gleichverteilt*. Aus Effizienzgründen können allerdings nur periodische Folgen verwendet werden, d.h. die „Pseudo-Zufallszahlen“ wiederholen sich nach einer gewissen Anzahl von Zahlen.[20] Wir verwenden eine Folge aus einem Standardlehrbuch [KR88]:

$$Z(n+1) = (Z(n) \cdot 1103515245 + 12345) \bmod 2^{32}$$

Diese Folge hat eine Periodizität von 2^{32} Zahlen. Um dieses deterministische Verhalten abzumildern, wird die Folge bei jeder Initialisierung eines Emulationswerkzeugs mit Hilfe der aktuellen Systemzeit initialisiert. Somit startet die Folge bei jedem Lauf an einer anderen Stelle. Das Problem der Periodizität bleibt allerdings bestehen.

Die Periodizität der Pseudo-Zufallszahlen kann sich wie folgt auswirken: In einem Emulationswerkzeug sei eine Bitfehlerrate p_b eingestellt, die bei Rahmen der Länge l zu einer Rahmenverlustwahrscheinlichkeit von $p_r = 0{,}5$ führt. Werden genau 2^{32} Rahmen der Länge l über dieses Emulationswerkzeug verschickt, so werden ca. 2^{31} „zufällige“ Rahmen verworfen. Werden danach dieselben 2^{32} Rahmen nochmals gesendet, werden wiederum ca. 2^{31} Rahmen verworfen, jedoch *genau dieselben* Rahmen wie zuvor.

Wie das Gedankenexperiment zeigt, erzeugt die Verwendung von Pseudo-Zufallszahlen unter ungünstigen Umständen deterministisches Verhalten, wo nichtdeterministisches Verhalten gewünscht wäre. Durch die große Periodizität von 2^{32} ist es jedoch sehr unwahrscheinlich, dass unerwünschte Wechselwirkungen mit Testsubjekten entstehen.

5.5.2 Artefakte bei der Verzögerung

Ein Emulationswerkzeug muss in Sende- und Empfangsrichtung Rahmen verzögern. Für die Nachbildung von Verzögerungen wird ein *Zeitgeber* benötigt: Ein Rahmen, der erst zum Zeitpunkt t' weiterverarbeitet (ausgesendet bzw. ausgeliefert) werden darf, wird vom Emulationswerkzeug zunächst in einer Warteschlange zwischengespeichert. In der verbleibenden Zeit bis t' muss das Emulationswerkzeug den Prozessor freigeben, damit andere Prozesse (Anwendungsprozesse und andere Teile des Betriebssystems) weiterlaufen können. Zuvor muss das Emulationswerkzeug dafür sorgen, dass es zum Zeitpunkt t' wieder Rechenzeit erhält. Dazu wird t' zusammen mit einer Funktion des Emulationswerkzeugs bei dem vom Betriebssystem verwal-

[20] Es gibt auch nichtperiodische deterministische Folgen, z.B. die Nachkommastellen von π. Solche Folgen lassen sich allerdings nicht sehr effizient und mit konstantem Aufwand berechnen.

teten Zeitgeber registriert. Zum Zeitpunkt t' ruft der Zeitgeber die registrierte Funktion auf, die dann den Rahmen weiterverarbeiten kann.

Der Zeitgeber in Linux ist so realisiert, dass er jedesmal, wenn das Betriebssystem regelmäßig Rechenzeit erhält, die Liste der registrierten Zeitpunkte überprüft und ggf. die dazugehörigen Funktionen aufruft. Dies bedeutet, dass die Genauigkeit des Zeitgebers von der Granularität der Zeitscheiben des Betriebssystems abhängt. Standardmäßig ist in Linux eine Zeitscheibe von 10 ms definiert, d.h. spätestens alle 10 ms wird der Zeitgeber aufgerufen.

Die Granularität der Zeitscheiben bedeutet für das Emulationswerkzeug, dass der Aufruf einer für den Zeitpunkt t' registrierten Rahmenbehandlungsfunktion bis zu 10 ms *nach* t' erfolgen kann. Das Aussenden bzw. Ausliefern eines Rahmens kann also im schlechtesten Fall mit 10 ms zusätzlicher, unerwünschter Verzögerung geschehen.

Auf diese Weise werden stets nur unerwünschte *zusätzliche* Verzögerungen eingeführt. Damit das Emulationswerkzeug trotzdem *im Mittel* korrekte Verzögerungen einführt, können im Ausgleich Rahmen absichtlich *zu früh* verarbeitet werden. Unabhängig davon ändert sich in jedem Fall das zeitliche Muster der Rahmensendungen: Statt eines konstanten Rahmenstroms, wie er beispielsweise als Ergebnis einer reduzierten Übertragungsrate zu erwarten wäre (Abb. 5.12 oben), wird tatsächlich pulsierend alle 10 ms gesendet (Abb. 5.12 unten).

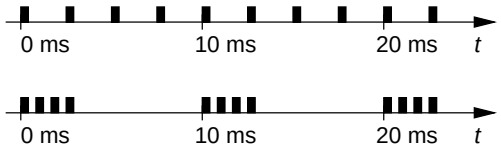

Abbildung 5.12: Verzögerungsartefakte einer Zeitscheibe von 10 ms: erwünschte (oben) und tatsächlich nachgebildete Rahmensendezeiten (unten)

Trotz der zweifellos unerwünschten Verzögerungsartefakte sind Zeitgeber mit 10 ms Granularität für viele Emulationsszenarien ausreichend: Jede gewünschte Übertragungsrate kann nachgebildet werden, und zusätzliche Verzögerungen werden *im Mittel* korrekt, im Einzelfall mit maximal 10 ms Fehler emuliert. An dieser Stelle ist darauf hinzuweisen, dass durch die Implementierung des Protokollstapels im Empfangspfad eines realen Systems ohnehin zusätzliche Verzögerungen eingeführt werden: Durch die zeitliche Entkopplung der Unterbrechungsbehandlung von der Weiterverarbeitung eines eingehenden Rahmens durch die Vermittlungsschicht (und den höheren Schichten) kann zwischen Sicherungsschicht und Vermittlungsschicht eine zusätzliche Verzögerung von bis zu einer Zeitscheibendauer entstehen (vgl. Abb. 5.9). Es hat also wenig Sinn, einen Rahmen auf der Emulationsschicht mikrosekundengenau zuzustellen, wenn derselbe Rahmen beim Empfänger ohnehin nichtdeterministisch mehrere Millisekunden im Eingangspuffer der Vermittlungsschicht verbringen wird.

Auswirkungen auf das virtuelle Trägersignal

Während Zeitgeber mit 10 ms Genauigkeit für die Emulation von Übertragungsrate und Verzögerungen in vielen Fällen ausreichen, benötigt das Konzept des virtuellen Trägersignals deutlich feiner auflösende Zeitgeber.

Eine einfache Beispielrechnung soll dies verdeutlichen: Ein Rahmen der typischen Länge $l = 1500\,\text{Byte}$ belegt in einem WLAN mit $b = 11\,\text{Mbit/s}$ für $t_s = l/b \approx 1\,\text{ms}$ das Medium. Das lokale Medienmodell eines Emulationswerkzeugs muss genau für diesen Zeitraum in den Zustand „belegt“ geschaltet werden. Auf CSMA basierende Medienzugriffsprotokolle, die wir auf der Grundlage dieses Medienmodells nachbilden wollen, definieren für den Zeitpunkt, an dem das Medium wieder frei ist (also nach Ablauf von t_s), eine Aktion, falls ein Sendewunsch besteht. Beispielsweise wird bei „1-persistentem“ Senden direkt nach dem Freiwerden des Mediums ein Sendeversuch gestartet. Daher ist es für die Emulation eines Medienzugriffsprotokolls essentiell, *sofort* nach dem Ende eines emulierten Trägersignals Rechenzeit zu erhalten, um beispielsweise mit einer Rahmensendung beginnen zu können.

Bei Verwendung der normalen Zeitgeber des Betriebssystems mit 10 ms Granularität würde die Implementierung des Medienzugriffsprotokolls mit bis zu 10 ms Verzögerung aufgerufen, was in der Beispielrechnung bis zu 10 Rahmenlängen entspräche. Es ist offensichtlich, dass auf dieser Grundlage das Verhalten eines realen Medienzugriffsprotokolls nicht einmal annähernd realistisch nachgebildet werden kann. Möglichkeiten zur Verminderung solcher Effekte werden nachfolgend behandelt.

5.5.3 Verminderung von Emulationsartefakten

Zur Verminderung von Emulationsartefakten sind feiner auflösende Zeitgeber wünschenswert, für die Emulation eines virtuellen Trägersignals sogar zwingend notwendig. Im Folgenden werden zwei verschiedene Methoden skizziert, mit denen die Genauigkeit der Zeitgeber verbessert werden kann.

Verkürzte Zeitscheiben

Die minimale Zeitscheibe des Betriebssystems hat direkten Einfluss auf die Granularität der Zeitgeber. Man kann die minimale Zeitscheibe eines Betriebssystems verkürzen, um Verzögerungsartefakte zu verringern; Zheng et al. schlagen eine Verkürzung auf 1 ms vor [ZN03]. Die minimale Dauer einer Zeitscheibe ist eine zentrale Betriebssystemkonstante und kann mit einer Neuübersetzung des Betriebssystems verändert werden. Diese Vorgehensweise verringert tatsächlich die zu erwartenden Verzögerungsartefakte von Emulationswerkzeugen.

Allerdings wird dadurch nicht nur das Verhalten des Emulationswerkzeugs, sondern das des gesamten Systems verändert. Insbesondere von TCP-Implementierungen ist bekannt, dass ihr Leistungsverhalten signifikant vom Verhalten der Zeitgeber abhängt [BP96a]. Somit werden die genaueren Zeitgeber durch potentiell viele neue Emulationsartefakte an anderer Stelle erkauft. Aus diesem Grund werden kürzere Zeitscheiben als Lösungsansatz hier nicht weiter betrachtet.

APIC-Zeitgeber

Neuere[21] Prozessoren der Intel-Familie bieten eine zusätzliche, genauere Zeitgeberschaltung an, die sog. „APIC“-Zeitgeber.[22] Diese Zeitgeber basieren auf dem Bustakt des Prozessors und erreichen damit eine Auslösegenauigkeit im Mikrosekunden-Bereich. Da sie komplett unabhängig zu den normalen Zeitgebern funktionieren, beeinflussen sie nicht deren Auslösegenauigkeit.

Das NET-System verfügt über Prozessoren mit APIC-Zeitgebern. Daher können wir diese Funktionalität für unsere Emulationswerkzeuge benutzen. Durch eine Betriebssystemerweiterung, die Vincent Oberlé an der Universität Karlsruhe entwickelt hat, werden die APIC-Zeitgeber unter Linux verfügbar.[23] Neuere Versionen unserer Emulationswerkzeuge verwenden APIC-Zeitgeber für die Emulation des virtuellen Trägersignals. Für weitere Informationen zur Integration von APIC-Zeitgebern in ein Emulationswerkzeug wird auf eine Diplomarbeit verwiesen [Yan04].

5.6 Zusammenfassung

In diesem Kapitel wurden wichtige Aspekte der Realisierung eines verteilten Emulationssystems mit zentraler Steuerung erläutert. Insbesondere wurde die Integration der Emulationswerkzeuge in den Kommunikationsstapel von Linux, sowie die Interaktion der zentralen Steuerung mit den verteilten Werkzeugen abgedeckt. Zahlreiche weitere Details zur Realisierung (z.B. zur Synopsis der Emulationswerkzeuge und -steuerung, Definition der verwendeten Dateiformate, Funktion und Bedienung weiterer Werkzeuge zur Koordination und Vereinfachung der Bedienung) wurden im Rahmen dieser Arbeit nicht dargestellt; zu diesem Zweck wurde auf die Dokumentation des NET-Systems und weitere Veröffentlichungen verwiesen.

[21] Prozessoren ab der P6-Familie (Pentium Pro)
[22] APIC steht für *Advanced Programmable Interrupt Controller*
[23] Die „APIC-Patches“ von Vincent Oberlé sind frei verfügbar unter http://www.oberle.org

6

Experimente

Im vorangegangenen Kapitel wurde das NET-System als konkrete Realisierung eines Emulationssystems vorgestellt. In diesem Kapitel wird anhand von Messungen überprüft, ob und bis zu welchem Grad die Hardware und die Software-Werkzeuge des NET-Systems in der Lage sind, tatsächlich realistisch Netzeigenschaften nachzubilden.

Zunächst wird dazu die systembedingte Verzögerung des NET-Systems gemessen, die wesentlichen Einfluss auf die nachbildbaren Parameter hat. Danach messen wir die Zeit, die im NET-System für die Konfiguration von VLANs benötigt wird. Daraus werden die Einsatzmöglichkeiten von VLANs für die Emulation von Netzverbindungen klar. Es folgt die systematische Evaluation der Leistung der zentralen Emulationssteuerung und der verteilten Emulationswerkzeuge. Schließlich wird anhand eines konkreten Beispiels gezeigt, wie die Emulation eines Netzszenarios für die Evaluation der Implementierung eines Netzprotokolls verwendet werden kann.

6.1 Messung der systembedingten Verzögerung

Als t_{min} bezeichnen wir die Verzögerung, die ein Rahmen systembedingt erfährt, wenn er zwischen zwei Instanzen eines verteilten Emulationswerkzeugs in einer Testumgebung übertragen wird, und die Emulationswerkzeuge keine *zusätzliche* Rahmenverzögerung einführen. t_{min} stellt damit die *minimale emulierbare Verzögerung* dar (vgl. Abschnitt 4.3.3). Darüberhinaus spielt t_{min} bei der Emulation eines Trägersignals eine wichtige Rolle; wenn t_{min} größer ist als die Ausbreitungsverzögerung t_a des emulierten Mediums, ist das lokale Medienmodell des Empfängers eines Rahmens für die Zeit $t_{\text{min}} - t_a$ in einem falschen Zustand (vgl. Abschnitt 4.4.1). Ein möglichst kleiner Wert für t_{min} ist also wünschenswert. Der Wert von t_{min} ist von der Architektur einer Testumgebung, sowie von der Leistungsfähigkeit der Hardware und der Effizienz

der beteiligten Software (Gerätetreiber etc.) abhängig. Wir beschreiben in diesem Abschnitt die Messung von $t_{\min}$ für das NET-System.

6.1.1 Messaufbau

Zur Messung von $t_{\min}$ verwenden wir folgenden Messaufbau (Abb. 6.1): Wir übertragen Rahmen zwischen zwei Emulationsknoten E_1, E_2 über das Emulationsnetz des NET-Systems und messen die Übertragungszeit auf Emulationsschicht, wobei die Emulationswerkzeuge so eingestellt sind, dass sie keine zusätzliche Verzögerung einfügen.

Die Messrahmen werden von einem *Lastgenerator* auf E_1 erzeugt und direkt auf der Sicherungsschicht eingespeist, d.h. dem Emulationswerkzeug auf E_1 übergeben.[1] Die Emulationswerkzeuge sind für diese Messung speziell instrumentiert: Das Emulationswerkzeug auf E_1 markiert jeden ausgehenden Rahmen mit einem Zeitstempel (T_1). Der Rahmen wird dann über den Vermittlungsknoten des Emulationsnetzes an den empfangenden Knoten E_2 weitergeleitet. Das Emulationswerkzeug auf E_2 versieht den Rahmen mit einem zweiten Zeitstempel T_2.

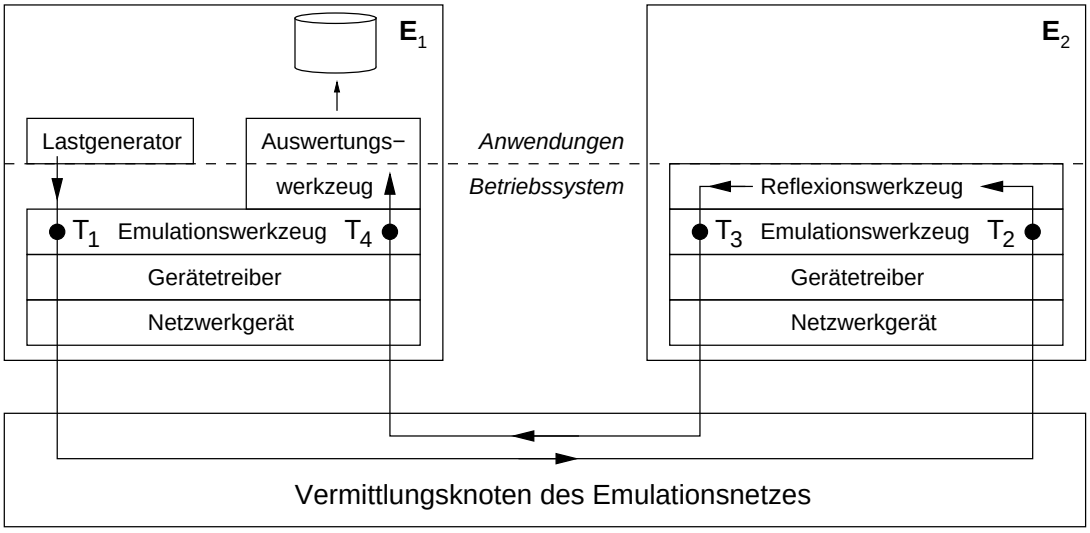

Abbildung 6.1: Messaufbau zur Messung von $t_{\min}$

Die Differenz $T_2 - T_1$ entspräche einem Messwert für $t_{\min}$. Da die lokalen Uhren auf den Emulationsknoten jedoch nicht in der erforderlichen Genauigkeit synchronisiert sind, können wir Differenzen nur zwischen Zeitstempeln *derselben* Uhren vergleichen. Deshalb wird der Rahmen nun von einem speziellen *Reflexionswerkzeug* auf E_2, das aus Effizienzgründen ebenfalls im Betriebssystemkontext läuft, „reflektiert“. Das Reflexionswerkzeug adressiert den Rahmen

[1] Durch die Schnittstelle „packet sockets“ können privilegierte Benutzerprozesse unter Umgehung der höheren Protokollschichten direkt Rahmen auf Sicherungsschicht absetzen.

an E_1 und übergibt ihn wieder an das Emulationswerkzeug auf E_2. Das Emulationswerkzeug versieht den Rahmen mit einem weiteren Zeitstempel T_3 und schickt ihn zurück an E_1. Das Emulationswerkzeug auf E_1 schließlich vergibt den Zeitstempel T_4 und übergibt den Rahmen an ein *Auswertungswerkzeug*, das den Rahmen vom Betriebssystem- in den Anwendungskontext transportiert und zur späteren Auswertung speichert. Die Integration des Reflexions- und des Auswertungswerkzeugs in den Kommunikationsstapel des Betriebssystems erfolgt analog zur Integration von Emulationswerkzeugen durch die Registrierung eines zusätzlichen Vermittlungsschicht-Protokolls (vgl. Abb. 5.9 auf Seite 98).

Als Zeitstempel verwenden wir den jeweils aktuellen Wert des sog. „TSC-Registers“.[2] Dieses Register wird mit jedem Prozessortakt inkrementiert. Bei den mit $f_{\mathrm{CPU}} = 2{,}4\,\mathrm{GHz}$ getakteten Prozessoren des NET-Systems entspricht dies einer zeitlichen Auflösung von $0{,}417\,\mathrm{ns}$.

Wir können nun die Umlaufzeit des Rahmens aus der Sicht des Emulationswerkzeugs auf E_1 berechnen ($T_4 - T_1$). Durch den symmetrischen Aufbau der Messung können wir annehmen, dass die Verzögerung in beiden Kommunikationsrichtungen den gleichen Wert hat. Wenn wir von der Umlaufzeit die Zeit subtrahieren, die der Rahmen im Reflexionswerkzeug verbracht hat ($T_3 - T_2$), erhalten wir einen Messwert für $2 \cdot t_{\mathrm{min}}$. Mit der Annahme, dass beide beteiligten Prozessoren mit derselben Taktfrequenz f_{CPU} betrieben werden, können wir t_{min} wie folgt berechnen:

$$t_{\mathrm{min}} = \frac{(T_4 - T_1) - (T_3 - T_2)}{2 \cdot f_{\mathrm{CPU}}}$$

6.1.2 Messungen

Es ist u.a. wegen der Serialisierungsverzögerung anzunehmen, dass die Rahmenlänge die Übertragungszeit eines Rahmens beeinflusst. Wir messen daher t_{min} für verschiedene Rahmenlängen von 64 bis 1500 Byte.[3] Um außerdem den Einfluss von Last auf die Verzögerung zu untersuchen, variieren wir die vom Lastgenerator erzeugte Übertragungsrate der Testrahmen von $10\,\mathrm{kbit/s}$ bis $500\,\mathrm{Mbit/s}$.

Abb. 6.2 zeigt das arithmetische Mittel von jeweils mindestens 300 Messungen von t_{min} in Abhängigkeit von Rahmengröße und Last. Im Diagramm ist auf der x-Achse nicht die vom Lastgenerator vorgegebene, sondern die tatsächlich erreichte Netto-Übertragungsrate eingetragen. Während des gesamten Experiments gingen keine Rahmen verloren.

Für sehr große Übertragungsraten und kleine Rahmenlängen wurde die Vorgabe des Lastgenerators nicht erreicht, da hier der Anteil der Rahmenköpfe an den Gesamtdaten zu groß ist. Für

[2] TSC steht für *Time Stamp Counter*. Das TSC-Register ist in Intel-Prozessoren ab der Pentium-Generation verfügbar.

[3] Die Angaben zu den Rahmenlängen beziehen sich jeweils die Nutzdaten, die der Sicherungsschicht übergeben werden.

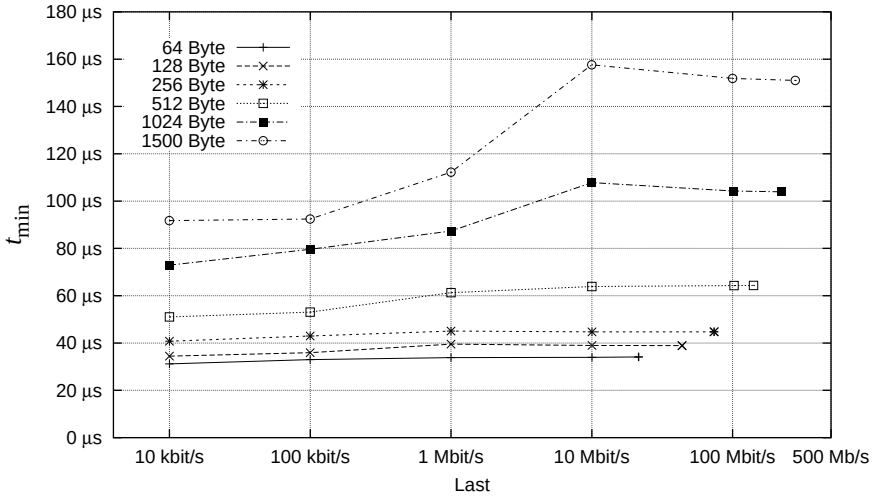

Abbildung 6.2: Messergebnisse für $t_{\min}$ in Abhängigkeit von Rahmengröße und Last

größere Rahmenlängen wird $t_{\min}$ größer, weil Rahmen mit mehr Nutzdaten mehr Zeit bei der Verarbeitung benötigen. Tendenziell nimmt $t_{\min}$ auch dann zu, wenn mehr Last angelegt wird. Dies ist dadurch zu erklären, dass sich mit steigender Last Warteschlangen auf den Netzwerkgeräten der Emulationsknoten und im Vermittlungsknoten bilden, die eine zusätzliche Verzögerung einführen. Es ist allerdings überraschend, dass das Maximum von $t_{\min}$ für große Rahmenlängen schon bei einer Last von 10 Mbit/s erreicht wird, und $t_{\min}$ bei noch größerer Last wieder leicht abnimmt. Eine mögliche Erklärung für dieses Verhalten ist, dass die Verarbeitungslogik des Vermittlungsknotens intern eine Taktung bzw. fixe Puffergrößen besitzt, die sich je nach Rahmenrate und -größe günstig oder ungünstig darstellen. Ferner kann man annehmen, dass die Verarbeitungsstrategie im Vermittlungsknoten für hohe Übertragungsraten optimiert ist (und nicht für eine vergleichsweise niedrige Übertragungsrate von 10 Mbit/s).

Größere Übertragungsraten konnten aus technischen Gründen (Flaschenhals: Bussystem und Taktfrequenz der Emulationsknoten) in unserem Emulationssystem nicht erreicht werden. Bei entsprechend besser ausgestatteten Emulationsknoten wären noch größere Übertragungsraten möglich; hier müssten weitere Messungen zeigen, ob sich prinzipiell andere Ergebnisse für $t_{\min}$ ergeben. Es ist jedoch zu erwarten, dass durch Emulationsknoten mit schnellerer Netzanbindung ein noch kleiner (und damit besserer) Wert für $t_{\min}$ möglich wäre. Weitere Messergebnisse können in der Ausarbeitung einer Studienarbeit nachgelesen werden [Cas05].

6.1.3 Fazit

Die gemessenen Werte für $t_{\min}$ setzen sich aus der Serialisierungsverzögerung, der Ausbreitungsverzögerung und der Verarbeitungszeit der Rahmen auf den Emulationsknoten und dem

Vermittlungsknoten zusammen. Die Serialisierungsverzögerung und die Ausbreitungsverzögerung können auch rechnerisch bestimmt werden: Bei einer Übertragungsrate des Emulationsnetzes von 1 Gbit/s und einem durch das Ethernet-Protokoll verursachten Mehraufwand von 30 Byte[4] pro Rahmen berechnet sich die minimale Serialisierungsverzögerung[5] in unserem Messszenario zu $t_s(94\,\mathrm{B}) \approx 0{,}75\,\mu\mathrm{s}$, die maximale zu $t_s(1530\,\mathrm{B}) \approx 12\,\mu\mathrm{s}$. Die Ausbreitungsverzögerung fällt demgegenüber überhaupt nicht ins Gewicht (bei 10 m Kabellänge ist $t_a \approx 57\,\mathrm{ns}$).

Im Vergleich mit den gemittelten Messwerten für $t_{\min}$, die zwischen $31\,\mu$s und $158\,\mu$s liegen, wird klar, dass die Verarbeitungszeit auf den Emulationsknoten und dem Vermittlungsknoten den größten Anteil an $t_{\min}$ haben muss. Die anderen Verzögerungskomponenten haben eine untergeordnete Bedeutung. An der Verarbeitungszeit auf den Emulationsknoten hat das Emulationswerkzeug selbst einen sehr kleinen Anteil ($< 3\,\mu$s, siehe [HR02]). Wollte man zur Optimierung der Eigenschaften der Testumgebung $t_{\min}$ weiter verkleinern, sollte also zunächst in eine schnellere Rahmenverarbeitung (d.h. schnelleres Bussystem, höhere Taktrate) investiert werden.

Obwohl die Werte für $t_{\min}$ je nach Rahmenlänge und Last variieren, bleiben die gemittelten Werte stets unter $160\,\mu$s. Die Einschränkung, dass kleinere Verzögerungen als $t_{\min}$ nicht nachgebildet werden können, fällt verglichen mit den nichtdeterministisch eingeführten Verzögerungen auf Vermittlungsschicht im Empfangspfad, die bis zu 10 ms betragen können, nicht ins Gewicht (vgl. Abschnitt 5.5.2). Für die Emulation von Verzögerungen, wie sie von der Vermittlungsschicht und den höheren Schichten wahrgenommen werden können, sind die gemessenen Werte von $t_{\min}$ also akzeptabel.

Für die Emulation von Trägersignalen, die *unterhalb* der Vermittlungsschicht abläuft, gelten allerdings strengere Anforderungen. Die von uns ermittelten Werte von $t_{\min}$ sind *größer* als die zu erwartende Ausbreitungsverzögerung im nachgebildeten Netz (in einem WLAN bei einer maximalen Reichweite von 450 m gilt $t_a \leq 1{,}5\,\mu$s); dies bedeutet, dass die lokalen Medienmodelle kurzfristig inkonsistent werden (vgl. Abb. 4.5 auf Seite 75). Da ein Trägersignal der typischen Dauer 1 ms[6] jedoch eine Größenordnung länger andauert als $t_{\min}$, ist die Emulation von Trägersignalen trotz kurzfristiger Inkonsistenten noch möglich.

Wird eine Abschätzung von $t_{\min}$ für die Berechnung von Verzögerungszeiten in einem Emulationswerkzeug benötigt, so ist es sinnvoll, einen Mittelwert zu verwenden, z.B. $t_{\min} \approx 100\,\mu$s.

---

[4] 7 Byte Präambel, 1 Byte „Frame Start Delimiter“, 14 Byte Ethernet-Rahmenkopf, 4 Byte VLAN-Kopf und 4 Byte Prüfsumme ergeben zusammen 30 Byte Mehraufwand.

[5] Die minimale Rahmenlänge beträgt bei allen Ethernet-Varianten 64 Byte. Obwohl in Gigabit Ethernet ein Rahmen stets für mindestens 512 Byte das Medium belegt („Carrier Extension“), kann ein Rahmen beim Empfänger schon ausgeliefert werden, wenn die „Carrier Extension“ noch übertragen wird. Wir zählen sie also nicht zur Serialisierungsverzögerung hinzu.

[6] entspricht dem Trägersignal eines Rahmens mit $l = 1500$ Byte in einem WLAN mit $b = 11$ Mbit/s

6.2 Messung der Konfigurationszeit von VLANs

VLANs im Emulationsnetz können für die Emulation von Netztopologien verwendet werden (vgl. Abschnitt 5.2). Damit können VLANs teilweise die Funktion von Emulationswerkzeugen übernehmen und diese entlasten. Dies kann bei großen Szenarien mit vielen Rundsendungen einem Skalierungsproblem vorbeugen (vgl. Abschnitt 4.5).

Vor der Verwendung müssen VLANs allerdings auf dem Vermittlungsknoten des Emulationsnetzes konfiguriert werden. Dieser Konfigurationsprozess benötigt eine gewisse Zeit. Während dieser Zeit sind die Anschlüsse des Vermittlungsknotens nicht benutzbar. Die Einsetzbarkeit von VLANs für Emulationszwecke ist daher stark von der Länge dieser Konfigurationszeit abhängig. Wir beschreiben in diesem Abschnitt die Messung der VLAN-Konfigurationszeit für den Vermittlungsknoten des Emulationsnetzes im NET-System.

6.2.1 Messaufbau

Der Gigabit-Vermittlungsknoten[7] des NET-Systems kann über verschiedene Schnittstellen angesprochen werden: über einen der 64 Gigabit-Anschlüsse oder über eine zusätzliche serielle Schnittstelle. Da die serielle Schnittstelle mit einer vergleichsweise kleinen Übertragungsrate arbeitet, wählen wir zur Verminderung der Serialisierungsverzögerung einen der wesentlich schnelleren Gigabit-Anschlüsse zur Konfiguration des Vermittlungsknotens. Der Vermittlungsknoten wird über eine Gigabit-Leitung direkt mit dem zentralen Steuerungsrechner verbunden, der für die VLAN-Konfiguration verantwortlich ist (vgl. Abb. 5.1 auf Seite 82). Die Kommunikation mit dem Vermittlungsknoten erfolgt auf der Basis von SNMP.[8]

Zur Messung der VLAN-Konfigurationszeit werden durch ein Messwerkzeug, das auf dem Steuerungsrechner läuft, verschiedene VLAN-Konfigurationen auf dem Vermittlungsknoten vorgenommen. Eine Konfiguration wird dabei durch mehrere sequentielle, synchrone SNMP-Aufrufe realisiert. Das Messwerkzeug misst die Zeit vom Absetzen des ersten SNMP-Aufrufs bis zur Rückkehr des letzten SNMP-Aufrufs einer Konfiguration.

6.2.2 Messungen

Es ist anzunehmen, dass die Konfigurationszeit sowohl von der Anzahl der VLANs, als auch von der Anzahl der Teilnehmer eines VLANs abhängt. Wir messen zunächst die Konfigurationszeit in Abhängigkeit von der Anzahl der VLANs bei jeweils zwei Teilnehmern pro VLAN.

[7] Im NET-System wird ein „Foundry FastIron II“ der Firma „Foundry Networks“ als Gigabit-Vermittlungsknoten verwendet.

[8] *Simple Network Management Protocol*, Standardprotokoll für die Verwaltung von Netzkomponenten

Dies entspricht einer Verbindungstopologie, die ausschließlich Punkt-zu-Punkt-Verbindungen enthält.

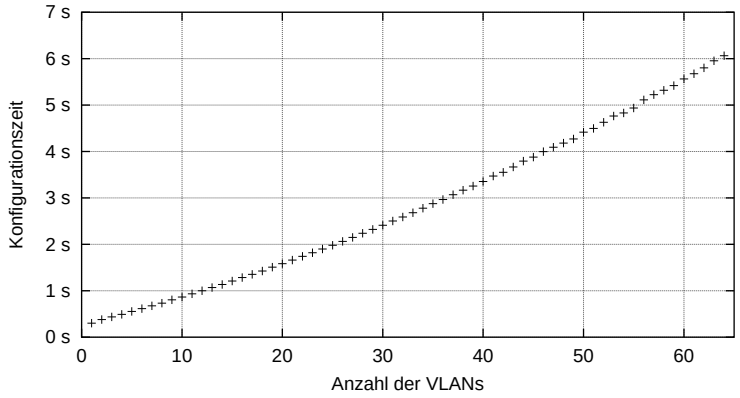

Abbildung 6.3: Konfigurationszeit für 1–64 VLANs mit jeweils zwei Teilnehmern

Abb. 6.3 zeigt die Konfigurationszeit für 1–64 VLANs, wobei jedes VLAN genau zwei Teilnehmer hat. Jeder Punkt in der Abbildung entspricht dem arithmetischen Mittel aus fünf Messläufen. Wie erwartet steigt die für die Konfiguration benötigte Zeit mit der Anzahl der VLANs gleichmäßig an. Schon bei 12 VLANs erreicht die für die Konfiguration benötigte Zeit eine Sekunde.

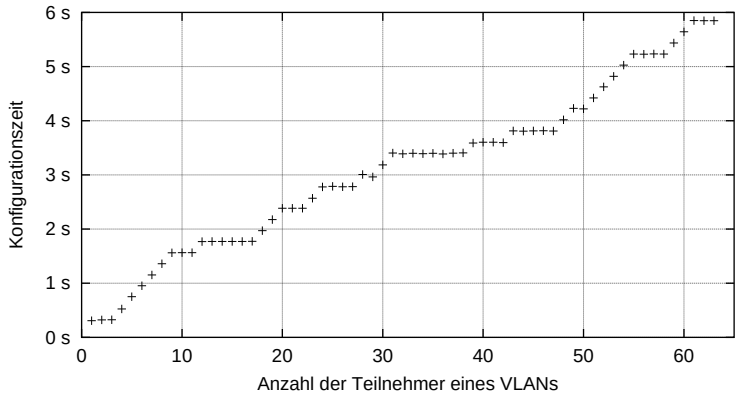

Abbildung 6.4: Konfigurationszeit für ein VLAN mit 1–63 Teilnehmern

In einer zweiten Messung beschränken wir uns auf *ein* VLAN und variieren die Anzahl der Teilnehmer des VLANs von 1 bis 63.[9] Dies entspricht der Verbindungstopologie eines LANs mit 1–63 Teilnehmern. Abb. 6.4 zeigt die Ergebnisse dieser Messung, wobei wiederum jeder Punkt

[9] Da im Messaufbau einer der insgesamt 64 Gigabit-Anschlüsse als Schnittstelle zum Steuerungsrechner verwendet wird, können nur die 63 verbleibenden Anschlüsse für VLANs genutzt werden. Die maximale Teilnehmerzahl in einem VLAN ist somit in dieser Messung 63.

in der Abbildung dem arithmetischen Mittel aus fünf Messläufen entspricht. Die Zunahme der Konfigurationszeit verläuft hier nicht so gleichmäßig wie in Abb. 6.3, was an den internen Datenstrukturen des Vermittlungsknotens liegen muss. Trotzdem liegen die Konfigurationszeiten in derselben Größenordnung von mehreren Sekunden.

6.2.3 Fazit

Im NET-System können Netzszenarien mit bis zu 64 Knoten nachgebildet werden. Um 64 Knoten mit Punkt-zu-Punkt-Verbindungen zu verbinden, sind mindestens 63 Netzverbindungen notwendig. Solche Topologien können zwar mit VLAN-Technologie nachgebildet werden, die Messungen zeigen jedoch, dass die VLAN-Konfiguration des Emulationsnetzes für Topologien dieser Größenordnung mehrere Sekunden dauert. Da die Anschlüsse des Vermittlungsknotens während der Konfigurationszeit nicht benutzbar sind, ist die Emulation von *dynamischen* Netztopologien mit VLAN-Technologie zumindest mit dem von uns verwendeten Vermittlungsknoten nicht möglich.

In Szenarien mit *statischer* Verbindungstopologie geschieht die Konfiguration der Netzinfrastruktur ausschließlich in der *Vorbereitungsphase* (vgl. Abschnitt 4.2.2). Eine Konfigurationszeit von mehreren Sekunden ist hier akzeptabel, denn während dieser Zeit wird das Emulationsnetz noch nicht für Messungen verwendet. Die Topologieemulation mit VLANs ist also ausschließlich für statische Verbindungstopologien geeignet. Dynamische Verbindungstopologien müssen durch Emulationswerkzeuge auf den einzelnen Knoten nachgebildet werden.

6.3 Evaluation der zentralen Emulationssteuerung

Die zentrale Emulationssteuerung steuert den Gesamtablauf eines Experiments durch die Koordination der Vorbereitungsphase, der Messphase und der Nachbereitungsphase (vgl. Abschnitt 4.2). Ändern sich die nachzubildenden Netzparameter während einer Messung, ist die Emulationssteuerung während der Messphase außerdem dafür zuständig, die jeweils aktuellen Netzparameter zu berechnen und an die verteilten Emulationswerkzeuge zu senden. In der von uns gewählten Architektur kann die zentrale Emulationssteuerung während der Messphase zum Flaschenhals werden, falls sie mit der Berechnung und Verteilung der Netzparameter für ein Szenario überfordert ist (vgl. Tab. 3.1 auf Seite 58). Wir messen deshalb die Leistung der Emulationssteuerung im NET-System[10] während der Messphase eines typischen dynamischen Szenarios, um beurteilen zu können, ob sie einen Flaschenhals des Emulationssystems darstellt.

[10] Für die Leistungsmessungen wurde eine aktualisierte Version des Steuerungsrechners verwendet (zwei Intel Xeon Prozessoren, jeweils 3,2 GHz Taktfrequenz, 8 GB Hauptspeicher).

6.3.1 Messaufbau

Die Leistung der Emulationssteuerung kann nur für ein konkretes Szenario gemessen werden, denn der Prozess der Parameterberechnung kann je nach Szenario und dem verwendeten Berechnungsverfahren sehr unterschiedlich sein. Wir beschränken uns im Folgenden auf ein typisches MANET-Szenario und den in Abschnitt 5.3 beschriebenen Berechnungsprozess.

Szenario

Wir verwenden ein Szenario, das in dieser oder ähnlicher Weise schon in zahlreichen Veröffentlichungen benutzt wurde (z.B. [HMTR04]): Mehrere mobile Teilnehmer mit WLAN-Geräten bewegen sich auf den Straßen in der Innenstadt von Stuttgart und bilden damit ein MANET von wechselnder Konnektivität. Wir betrachten eine Variante mit Fußgängern und eine mit Fahrzeugen als Teilnehmern, wobei sich die Varianten lediglich in der Bewegungscharakteristik unterscheiden. Die randomisierten Bewegungsvorschriften der Teilnehmer wurden nach dem „Graph Walk“-Mobilitätsmodell erstellt [SHB+03]: Zu Beginn jedes Messlaufs sind die Teilnehmer zufällig auf den Straßen verteilt. Jeder Teilnehmer sucht sich zufällig ein Ziel in der Stadt aus, das er auf dem kürzesten Weg mit einer zufälligen Geschwindigkeit ansteuert. Er verweilt eventuell eine kurze Zeit und sucht sich dann ein neues Ziel. Alle Teilnehmer bewegen sich dabei stets nur auf den Straßen. Tab. 6.1 fasst die Parameter des Mobilitätsmodells für das Beispielszenario zusammen.

Mobilitätsmodell	„Graph Walk“	
Szenariofläche	$2365\,\mathrm{m} \times 1905\,\mathrm{m}$, Stuttgarter Innenstadt	
Anzahl der Teilnehmer	2–64	
Dauer	300 s	
	Fußgänger	*Fahrzeuge*
Geschwindigkeit v	$1\,\mathrm{m/s} \leq v \leq 2\,\mathrm{m/s}$	$5\,\mathrm{m/s} \leq v \leq 20\,\mathrm{m/s}$
Verweildauer t	0 s	$0\,\mathrm{s} \leq t \leq 20\,\mathrm{s}$

Tabelle 6.1: Parameter des Mobilitätsmodells für das MANET-Beispielszenario

Berechnungsprozess

Neben dem Szenario ist die Komplexität des verwendeten Berechnungsprozesses für die Netzparameter im globalen Netzmodell ausschlaggebend für die Leistung der Emulationssteuerung. Wir verwenden hier den Berechnungsprozess, der in Abschnitt 5.3 ausführlich vorgestellt wurde (siehe Abb. 5.4 auf Seite 89): Ausgehend von den Bewegungsvorschriften des jeweiligen Szenarios werden periodisch die aktuellen Positionen aller Teilnehmer ermittelt. Für jedes

mögliche Kommunikationspaar wird dann die Empfangsleistung und daraus die aktuelle Bitfehlerrate ermittelt. Für die Berechnung der Empfangsleistung verwenden wir alternativ die beiden Verfahren, die in Abschnitt 5.3 eingeführt wurden: das kombinierte „Free space / Two-ray ground"-Modell (abgekürzt *TRG*, ohne geographisches Umgebungsmodell), sowie den „Intelligent Ray Tracing"-Algorithmus (abgekürzt *IRT*, mit geographischem Umgebungsmodell), wobei wir die von uns im Voraus berechneten Wellenausbreitungsdaten für die Stuttgarter Innenstadt verwenden. Die Werte aus Tab. 5.1 auf Seite 88 bilden dabei die physikalische Grundlage für die Berechnung der Wellenausbreitung.

Berechnungszyklus

Während der Messphase berechnet die Emulationssteuerung die aktuellen Netzparameter in einem periodischen Zyklus. Wir gehen hier davon aus, dass die Periode des Berechnungszyklus $t_{\text{Zyklus}} = 1\,\text{s}$ ist.[11] In jedem Zyklus werden zunächst die aktuellen Parameter des globalen Netzmodells berechnet. Wenn sich Parameter gegenüber dem letzten Berechnungszyklus geändert haben, werden diese über Konfigurationsnachrichten an die zuständigen Emulationswerkzeuge auf den jeweiligen Emulationsknoten geschickt (vgl. Abb. 5.11 auf Seite 101). Die Zeit für das Berechnen der Parameter und das Verschicken der Konfigurationsnachrichten nennen wir t_{BV}. Am Ende eines Berechnungszyklus wartet die Emulationssteuerung für $t_{\text{Zyklus}} - t_{\text{BV}}$, bevor sie mit dem nächsten Zyklus beginnt.

Damit die Emulationswerkzeuge möglichst zeitnah die Konfigurationsnachrichten mit den jeweils aktuellen Netzparametern erhalten, ist ein möglichst kleiner Wert für t_{BV} wünschenswert. Insbesondere muss $t_{\text{BV}} \leq t_{\text{Zyklus}}$ gelten, damit die Emulationssteuerung rechtzeitig mit dem nächsten Bearbeitungszyklus beginnen kann.

6.3.2 Messungen

Mit einer für diese Messung speziell instrumentierten Version der Emulationssteuerung messen wir nun t_{BV}, wobei wir die Anzahl der beteiligten Teilnehmer im Szenario von 2 bis 64 steigern. Der Wert für t_{BV} wird jeweils über die Dauer eines Szenarios (300 s) gemittelt.

Abb. 6.5 zeigt die Messergebnisse für beide Szenariovarianten bei Verwendung des TRG-Modells. Jeder Messpunkt entspricht dem durchschnittlichen Wert von t_{BV} aus fünf Läufen mit *unterschiedlichen* Bewegungsvorschriften für jeden Lauf, wobei auch die maximalen Abweichungen vom Mittelwert eingezeichnet sind.[12] Wie erwartet steigt t_{BV} bei steigender Teil-

[11] Der Wert für t_{Zyklus} könnte an dieser Stelle natürlich auch größer oder kleiner gewählt werden. 1 s hat sich für die von uns betrachteten Szenarien als sinnvoller Mittelwert erwiesen.

[12] Durch die randomisierten Bewegungsabläufe, insbesondere durch die unterschiedlichen Geschwindigkeiten der Teilnehmer, können sich relativ große Unterschiede für t_{BV} ergeben.

nehmerzahl an, da die Größe des globalen Netzmodells und damit die Anzahl der zu berechnenden Netzparameter *quadratisch* mit der Teilnehmerzahl steigt. Die Tatsache, dass t_{BV} für den betrachteten Parameterbereich dennoch augenscheinlich nur etwa *linear* mit der Anzahl der Teilnehmer zunimmt, weist darauf hin, dass hier die zahlreichen linearen Komponenten des Berechnungsalgorithmus (Bewegungssimulation der Teilnehmer etc.) den Gesamtaufwand dominieren. Man kann ebenfalls ablesen, dass der Aufwand der zentralen Emulationssteuerung von der Mobilität der Teilnehmer abhängt: Fahrzeuge, die sich im Vergleich zu Fußgängern schnell bewegen, erzeugen erheblich mehr Berechnungsaufwand. Trotzdem bleibt t_{BV} bei Verwendung des TRG-Modells auch bei Szenarien mit 64 Fahrzeugen im Bereich von wenigen Millisekunden.

Abb. 6.6 zeigt die Messwerte für t_{BV} bei Verwendung des IRT-Modells. Zur besseren Vergleichbarkeit mit dem vorigen Experiment verwenden wir *dieselben* Bewegungsabläufe. Wie zu erwarten sind die Berechnungen auf Basis des IRT-Modells deutlich aufwändiger, da für jedes Paar von Teilnehmern ein Wert für die Empfangsleistung aus den vorberechneten Wellenausbreitungsdaten gelesen werden muss. Trotz effizienter Implementierung des Zugriffs auf die vorberechneten Daten[13] liegen die Messwerte für t_{BV} ca. um den Faktor 150 über denen des TRG-Modells. Für große Szenarien mit hoher Mobilität der Teilnehmer wird sogar $t_{\mathrm{BV}} > t_{\mathrm{Zyklus}}$, die Zykluszeit von 1 s kann also nicht eingehalten werden.

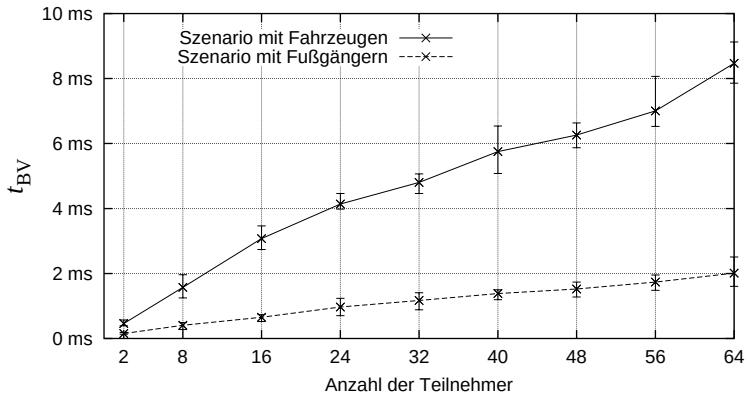

Abbildung 6.5: t_{BV} für 2–64 Teilnehmer (TRG-Modell)

[13] Die vorberechneten Daten der Szenariofläche (ca. 121 GB) sind auf einem RAID-System gespeichert, das direkt an den zentralen Steuerungsrechner angeschlossen ist. In der Implementierung der Emulationssteuerung wird auf der Anwendungsebene ein lokaler Zwischenspeicher verwendet, der für die hier verwendeten Szenarien genügend groß ist, um alle einmal angeforderten Daten eines Szenarios bei späterer erneuter Verwendung nicht nochmals vom Plattenspeicher lesen zu müssen.

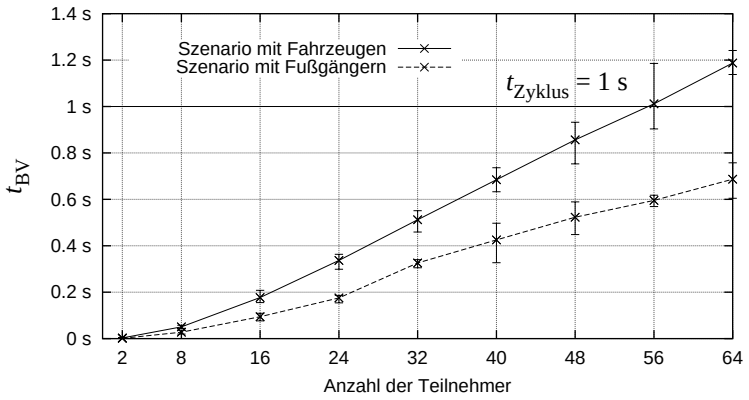

Abbildung 6.6: t_{BV} für 2–64 Teilnehmer (IRT-Modell)

Optimierung durch Vorladen der Daten

Obwohl der Hauptspeicher des Steuerungsrechners deutlich zu klein ist, um die *gesamten* Wellenausbreitungsdaten der Szenariofläche vorzuhalten, passen für Szenarien der hier betrachteten Dauer die benötigten Daten für einen *einzigen Szenariolauf* durchaus in den Hauptspeicher: Wenn wir von symmetrischen Ausbreitungsdaten ausgehen, benötigen wir pro Berechnungszyklus bei $n_k = 64$ Knoten maximal $n_k \cdot (n_k - 1)/2 = 2016$ Werte, für einen ganzen Szenariolauf (300 s) also maximal 604 800 Werte. Bei 4 Byte pro Wert ergibt sich ein maximaler Speicherbedarf von $\approx 2{,}3$ MB (zzgl. der Indexstrukturen). Um den Wert für t_{BV} während der Messphase eines Experiments zu minimieren, ist es deshalb möglich, bereits in der *Vorbereitungsphase* die benötigten Daten für ein konkretes Szenario vorzuladen, indem der Berechnungsprozess für das Szenario einmal komplett durchlaufen wird. In einem zweiten Durchlauf des Szenarios kann nun die eigentliche Messphase beginnen; die benötigten Wellenausbreitungsdaten sind dann schon im Hauptspeicher des Steuerungsrechners vorhanden. Abb. 6.7 zeigt für jeweils ein Szenario mit Fahrzeugen und Fußgängern die Messwerte für t_{BV} ohne und mit Vorladen der Wellenausbreitungsdaten. t_{BV} bleibt bei vorgeladenen Daten auch für große Szenarien deutlich unter t_{Zyklus}.

6.3.3 Fazit

Für ein typisches Szenario mit bis zu 64 mobilen Teilnehmern haben wir die Leistung der zentralen Emulationssteuerung gemessen, wobei wir von einem periodischen Berechnungszyklus mit $t_{Zyklus} = 1$ s ausgegangen sind. Bei Verwendung eines einfachen Berechnungsverfahrens benötigt die Emulationssteuerung auch bei großen Szenarien nur wenige Millisekunden, um die Parameter im globalen Netzmodell zu berechnen und an die verteilten Emulationswerkzeuge zu verschicken. Wird ein komplexes Verfahren zur Parameterberechnung verwendet, das Zu-

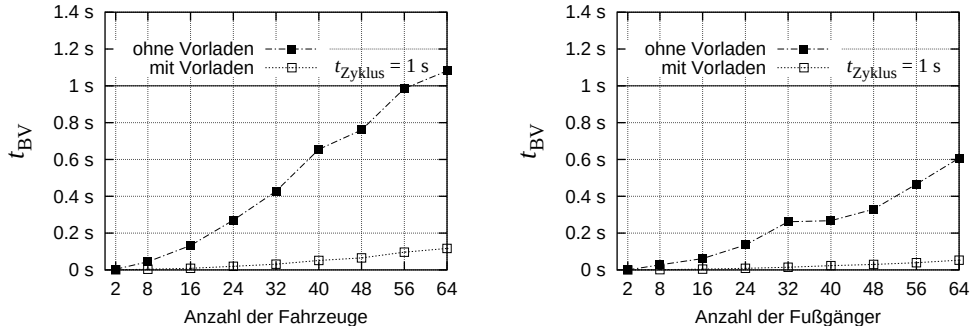

Abbildung 6.7: t_{BV} ohne und mit Vorladen der Wellenausbreitungsdaten (IRT-Modell)

griffe auf Sekundärspeicher erfordert, kann die Berechnungszeit bei großen Szenarien t_{Zyklus} überschreiten, die Emulationssteuerung wäre also mit der Berechnung und dem Verschicken der Netzparameter in Echtzeit überfordert. Durch Vorladen der benötigten Daten in der Vorbereitungsphase eines Experiments kann jedoch auch in solchen Fällen die Leistung der Emulationssteuerung in der Messphase in ausreichendem Maße erhöht werden. Für ein typsches Szenario mit bis zu 64 Knoten wurde also gezeigt, dass die zentrale Emulationssteuerung die Parameter im globalen Netzmodell bei einem Berechnungszyklus von $t_{Zyklus} = 1\,s$ rechtzeitig berechnen und verschicken kann.

Prinzipiell ist die Größe eines Szenarios in unserer Architektur durch die Anzahl der Knoten der Emulationsumgebung beschränkt, im NET-System sind dies 64 Knoten. Daher wurden in den Messungen dieses Abschnitts maximal 64 Emulationsknoten betrachtet. Selbstverständlich unterstützt unsere Architektur auch Emulationsumgebungen mit mehr als 64 Knoten. Die Messergebnisse lassen vermuten, dass auch Szenarien mit wenigen 100 Emulationsknoten problemlos über eine zentrale Instanz gesteuert werden können. Insbesondere bei der Einführung von virtuellen Emulationsknoten werden jedoch Szenarien mit 1000 Knoten realistisch (vgl. Ausblick, Abschnitt 7.2). Für solche Szenarien muss die Architektur der Emulationssteuerung neu überdacht werden; eventuell muss die Emulationssteuerung für solche sehr großen Szenarien verteilt (z.B. hierarchisch) organisiert werden.

6.4 Evaluation der verteilten Emulationswerkzeuge

Emulationswerkzeuge erbringen die Kernfunktionalität eines Emulationssystems, nämlich die Emulation von Netzeigenschaften. Von der korrekten und exakten Funktion der Emulationswerkzeuge hängt die Einsetzbarkeit eines Emulationssystems ab. In diesem Abschnitt wird anhand von mehreren Messungen dargestellt, mit welcher Genauigkeit und in welchem Wertebereich die für das NET-System implementierten Emulationswerkzeuge spezifizierte Netzeigenschaften nachbilden können.

6.4.1 Messaufbau

Wir wollen die Genauigkeit der Emulation aller Netzparameter messen, die am Emulationswerkzeug einstellbar sind. Dies sind die Parameter aus dem globalen Netzmodell: die Bitfehlerrate, die Ausbreitungsverzögerung und die Übertragungsrate (vgl. Abschnitt 4.2.1). Der Messaufbau ist für alle drei Messungen ähnlich: Im NET-System wird durch ein VLAN zwischen den Emulationsknoten E_1 und E_2 eine exklusive Verbindung eingerichtet. Auf beiden Emulationsknoten läuft eine Instanz des Emulationswerkzeugs. Die Verbindung zwischen E_1 und E_2 kann durch die Emulationswerkzeuge mit beliebigen Netzeigenschaften konfiguriert werden. Diese Eigenschaften werden dann mit Hilfe von Messwerkzeugen überprüft.

Da in diesem Fall die Emulationswerkzeuge selbst die Testsubjekte sind, vermeiden wir eine spezielle Instrumentierung der Werkzeuge zu Testzwecken, um die Messung nicht zu beeinflussen. Stattdessen kommen die Programme ping[14] und netio[15] zum Einsatz, die hier gleichzeitig als Lastgeneratoren und Messwerkzeuge fungieren.

6.4.2 Messungen

Wir messen die Genauigkeit der Emulation der Bitfehlerrate p_b, der Ausbreitungsverzögerung t_a und der Übertragungsrate b.

Bitfehlerrate

Da wir ohne Eingriff in das Emulationswerkzeug messen und sich die Messwerkzeuge deshalb logisch *oberhalb* der Emulationsschicht befinden, ist ein direktes Messen der Bitfehlerrate nicht möglich: Das Emulationswerkzeug berechnet aus der Länge eines Rahmens und der Bitfehlerrate die Rahmenverlustwahrscheinlichkeit und bildet dann *Rahmenverluste* nach. Nur diese Rahmenverluste können wir messen. Wir verwenden Testrahmen fixer Länge[16] und können dadurch für jede gewünschte Rahmenverlustwahrscheinlichkeit die zugehörige Bitfehlerrate errechnen und am Emulationswerkzeug auf E_1 einstellen. Für einen Messpunkt werden mit ping 1000 Pakete von E_1 nach E_2 geschickt. Aus der Anzahl der bei E_2 angekommenen Pakete wird p_r errechnet.

[14] ping benutzt „ICMP“-Pakete (Internet Control Message Protocol), um damit Rundsendezeiten zu messen und Paketverluste zu zählen. ping ist Bestandteil jedes UNIX-Betriebssystems.

[15] netio ist ein Standardwerkzeug zum Messen von Übertragungsraten mit TCP. Es wurde von Kai Uwe Rommel geschrieben und ist frei erhältlich von http://freshmeat.net/projects/netio/

[16] Die hier vorgestellten Messungen wurden mit Rahmen der Länge 64 Byte (Nutzlast auf Sicherungsschicht) durchgeführt. Vergleichsmessungen mit anderen Rahmenlängen haben gezeigt, dass die Rahmenlänge erwartungsgemäß keinen Einfluss auf die Emulation des Rahmenverlusts hat.

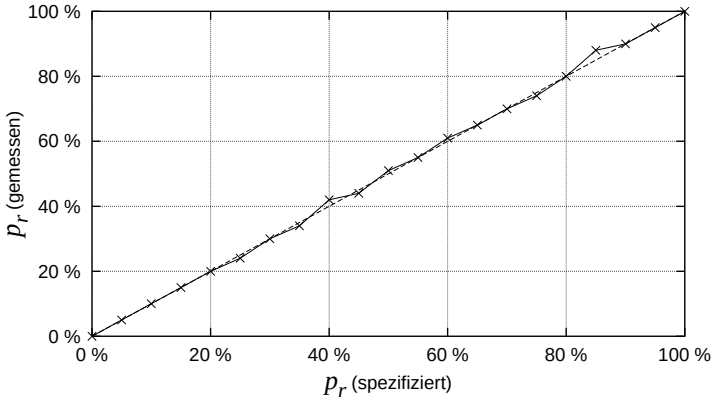

Abbildung 6.8: Genauigkeit der Emulation von p_r

Abb. 6.8 zeigt die Ergebnisse der Messung von p_r. Da Bitfehler spezifiziert, aber Rahmenverluste gemessen werden, sind in der Abbildung zur besseren Vergleichbarkeit von Soll- und Istwerten die spezifizierten Werte ebenfalls in Rahmenverlustwahrscheinlichkeiten umgerechnet dargestellt. Die Abbildung zeigt nur kleine Abweichungen zwischen spezifizierten und gemessenen Verlustwahrscheinlichkeiten. Da es sich hier um ein Zufallsexperiment handelt, sind Abweichungen in dieser Größenordnung normal.

Ausbreitungsverzögerung

Die Ausbreitungsverzögerung kann ohne Instrumentierung der Emulationswerkzeuge und exakt synchronisierte Uhren auf den Emulationsknoten ebenfalls nicht direkt gemessen werden. Wir messen stattdessen mit ping die *Rundsendezeit* eines Pakets, das von E_1 an E_2 und wieder zurück an E_1 gesendet wird. Zunächst messen wir die Rundsendezeit von Paketen *ohne* zusätzliche Verzögerung durch das Emulationswerkzeug. Danach stellen wir am Emulationswerkzeug auf E_1 eine *zusätzliche*[17] Verzögerung t_a für ausgehende Rahmen ein. Die Übertragungsrate wird dabei auf „unendlich“ gesetzt, um keine zusätzliche Serialisierungsverzögerung zu erhalten. Nun messen wir die Rundsendezeit erneut. Die Differenz der Rundsendezeiten ist ein Messwert für die vom Emulationswerkzeug eingeführte Verzögerung t_a.

Abb. 6.9 zeigt die Ergebnisse der Messung von t_a. Jeder Messpunkt entspricht dem arithmetischen Mittel von 100 Messungen. Für große Verzögerungen von mehreren hundert Millisekunden (linkes Diagramm) sind die relativen Abweichungen der gemittelten Messwerte von den Sollwerten so gering, dass sie in der Abbildung nicht erkennbar sind ($< 0{,}1\,\%$ Fehler bei $t_a > 100\,\text{ms}$). Die Ergebnisse für Verzögerungen unter 100 ms sind daher im rechten Diagramm

[17] Wenn keine *zusätzliche*, sondern eine *absolute* Verzögerung eingeführt werden soll, muss vom Sollwert der Verzögerung zunächst $t_{\min}$ subtrahiert werden. Auf diese Kalibrierung im Emulationswerkzeug wurde in dieser Messung verzichtet, da sie bereits in der Subtraktion der Rundsendezeit enthalten ist.

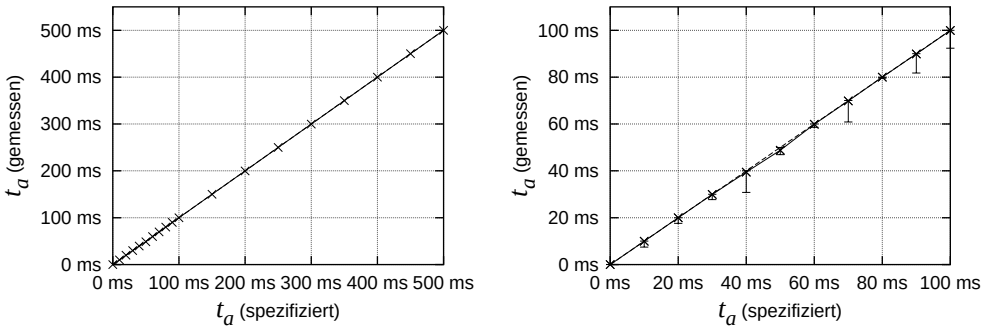

Abbildung 6.9: Genauigkeit der Emulation von t_a für große (links) und kleine Verzögerungen (rechts)

nochmals vergrößert und mit Fehlerbalken dargestellt. Während die mittleren Werte um maximal 2,5% abweichen, sind einzelne Ausreißer mit bis zu 10 ms Abweichung zu erkennen. Dies entspricht genau der minimalen Zeitscheibe des Betriebssystems und ist daher durch eine ungünstige Rechenzeitzuteilung zu erklären (vgl. Abschnitt 5.5.2). Aus diesem Grund wurden *kleinere* Verzögerungen als 10 ms (mit Ausnahme der Referenzmessung *ohne* zusätzliche Verzögerung) in dieser Messung nicht betrachtet.

Übertragungsrate

Zur Messung der Übertragungsrate verwenden wir netio. Das Werkzeug baut zunächst eine TCP-Verbindung von E_1 nach E_2 auf und misst dann die maximale Übertragungsrate der TCP-Verbindung.

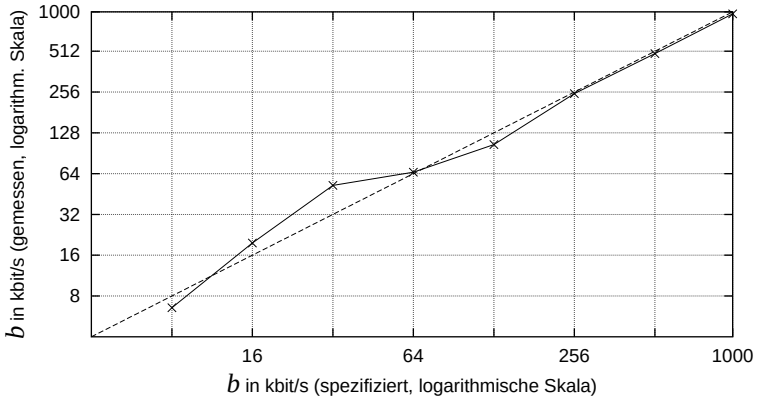

Abbildung 6.10: Genauigkeit der Emulation von b für kleine Übertragungsraten

Da die im NET-System sinnvoll nachbildbaren Übertragungsraten einen sehr großen Wertebereich umfassen, sind die Messergebnisse in drei Diagrammen dargestellt. Die relativ großen

Abweichungen bei sehr kleinen Übertragungsraten (Abb. 6.10) führen wir auf Quantisierungseffekte bei der Messung zurück.[18] Bei größeren Übertragungsraten (Abb. 6.11, linkes Diagramm) verschwindet dieser Effekt; die gemessenen Werte stimmen mit den spezifizierten gut überein. Bei sehr großen Übertragungsraten (Abb. 6.11, rechtes Diagramm) wird die Leistungsfähigkeit des NET-Systems erreicht. Die verwendeten Netzwerkgeräte sollten zwar nominell $b_{\max} = 1\,\text{Gbit/s}$ erreichen, durch die beschränkte Leistungsfähigkeit des Bussystems in den Emulationsknoten ist die Grenze allerdings schon bei $b_{\max} \approx 381\,\text{Mbit/s}$ erreicht. Größere Übertragungsraten lassen sich im NET-System mit der derzeitigen Hardware also nicht nachbilden.

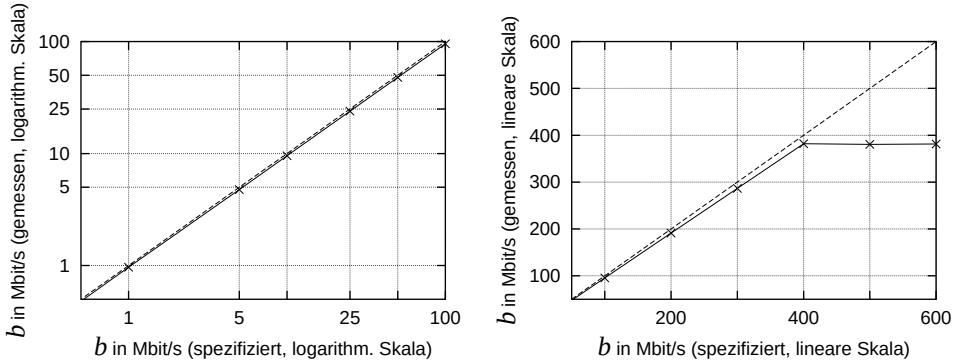

Abbildung 6.11: Genauigkeit der Emulation von b für mittlere (links) und große Übertragungsraten (rechts)

An dieser Stelle ist anzumerken, dass der im NET-System verwendete Vermittlungsknoten mit einer internen Übertragungsrate von $64\,\text{Gbit/s}$ arbeitet und damit niemals zum Flaschenhals des Systems werden kann, was die Übertragungsrate angeht. Die Obergrenze von $381\,\text{Mbit/s}$ gilt also für *jeden einzelnen* Emulationsknoten, nicht für das gesamte System. Weitere Messungen hierzu sind in [Cas05] zu finden.

6.4.3 Fazit

Die Evaluation der verteilten Emulationswerkzeuge im NET-System zeigt, dass die spezifizierten Netzparameter im technisch möglichen Parameterbereich zuverlässig nachgebildet werden können. Es gibt zwei Einschränkungen: Bei der Nachbildung von Rahmenverzögerungen kann in Einzelfällen ein Fehler von bis zu 10 ms entstehen, im Mittel sind die nachgebildeten Verzö-

[18] netio misst die übertragene Datenmenge in einer fixen Zeitspanne. Unabhängig von der Übertragungsrate sind die von TCP erzeugten Segmente in unserer Messung immer gleich groß. Während einer Messphase wird immer eine *ganzzahlige* Anzahl von Segmenten übertragen. Dadurch sind die Messergebnisse quantisiert, was sich bei kleinen Übertragungsraten besonders bemerkbar macht.

gerungen jedoch korrekt. Für die Nachbildung von Übertragungsraten gilt im NET-System mit der derzeitigen Harwdare die Obergrenze $b_{\max} \approx 381\,\mathrm{Mbit/s}$.

Aus dem Nachweis, dass einzelne Netzverbindungen mit unserem Emulationsansatz realistisch nachgebildet werden, folgt auch, dass ein emuliertes Netzszenario, das aus emulierten Verbindungen besteht, als Ganzes realistische Eigenschaften hat. Im folgenden Abschnitt wird ein ausführliches Beispiel für die Emulation eines Netzszenarios gegeben.

6.5 Beispiel einer Messung im NET-System

In diesem Abschnitt wird ein typisches Beispiel für eine Messung im NET-System beschrieben, um einen Eindruck von der Art der Experimente zu geben, die mit emulierten Netzszenarien im NET-System möglich sind. Zunächst wird das Testsubjekt, ein Vermittlungsprotokoll für MANETs, kurz vorgestellt. Danach wird das *virtuelle GPS*[19]*-Gerät* skizziert, eine Erweiterung des NET-Systems, die für die Ausführung dieses Testsubjekts notwendig ist. Schließlich werden die Messergebnisse dargestellt und den Ergebnissen einer vergleichbaren *Simulation* gegenübergestellt.

6.5.1 Testsubjekt

Als Testsubjekt für diese Messung verwenden wir die Implementierung eines Protokolls zur ortsbasierten Nachrichtenvermittlung („Greedy Location-Based Routing“, abgekürzt GLBR) in einem MANET. Die Implementierung des Protokolls wurde im Rahmen des CarTalk2000-Projekts erstellt [THRC03] und dient als Grundlage für die Kommunikation zwischen Fahrzeugen.

Es gibt zahlreiche Vermittlungsprotokolle für MANETs. Viele Vermittlungsprotokolle kommen ohne Ortsinformation aus. Durch die Verwendung von Ortsinformation kann jedoch die Effizienz eines MANET-Vermittlungsprotokolls erheblich gesteigert werden [MWH01]. Das GLBR-Protokoll geht davon aus, dass jeder Teilnehmer seine absolute Position kennt (beispielsweise durch ein Positionierungssystem). Außerdem kennt jeder Teilnehmer die Positionen seiner Nachbarn, wobei Nachbarn alle Teilnehmer sind, die sich in direkter Kommunikationsreichweite befinden. Dies wird dadurch erreicht, dass jeder Teilnehmer periodisch in einer Rundsendung seine eigene Position bekannt gibt.

Bevor eine Nachricht gesendet wird, trägt der Absender neben der Adresse zur Identifikation des Empfängers auch dessen letzte bekannte Position in absoluten Koordinaten in den Nachrichtenkopf ein. Zum Senden oder Weiterleiten einer Nachricht wird aus der Menge der Nach-

[19] *Global Positioning System*, weltweites satellitengestütztes Positionierungssystem

barn derjenige Teilnehmer ausgesucht, der den kleinsten geographischen Abstand zum Empfänger hat. An diesen Teilnehmer wird die Nachricht weitergeleitet. Das GLBR-Protokoll versucht also, in jedem Weiterleitungsschritt dem Empfänger um den größtmöglichen Entfernungsbetrag näher zu kommen (daher der Namensteil *greedy,* gierig).

Um eine Nachricht zu senden, wird die Position des Empfängers benötigt. Diese Position muss ggf. zunächst von einem *Lokationsdienst* erfragt werden. Für das GLBR-Protokoll wurde ein einfacher Lokationsdienst implementiert: Eine Positionsanfrage wird an *alle* Nachbarn geschickt und von diesen ebenso weitergeleitet („Flooding“). Die Antwort auf eine Positionsanfrage kann dann mit dem GLBR-Protokoll an den Anfragenden zurückgeschickt werden.[20]

Eine Nachrichtensendung mit dem GLBR-Protokoll führt, wie bei anderen MANET-Vermittlungsprotokollen auch, abhängig von den Positionen der Teilnehmer und deren Kommunikationsreichweite nicht immer zum Ziel. Der Anteil der erfolgreich zugestellten an den insgesamt gesendeten Nachrichten (die „Zustellrate“) wird als wichtigste Leistungsmetrik für das Protokoll verwendet. Eine weitere wichtige Metrik ist die durchschnittliche Pfadlänge, also die Anzahl der Weiterleitungen einer Nachricht: Kleine Pfadlängen deuten darauf hin, dass viele Nachrichten direkt zugestellt werden konnten (Pfadlänge = 1), das Vermittlungsprotokoll also keine Rolle spielt.

Emulation eines GPS-Geräts

Die Implementierung des GLBR-Protokolls geht davon aus, dass die aktuelle Position des Teilnehmers von einem lokal verfügbaren GPS-Gerät bereitgestellt wird. Die Emulationsknoten im NET-System sind nicht mit GPS-Geräten ausgestattet; selbst wenn dies der Fall wäre, würden die GPS-Geräte die Position des Emulationssystems ausgeben, womit das Verhalten des GLBR-Protokolls verfälscht würde. Um das Testsubjekt unverändert und unter realistischen Bedingungen ausführen zu können, muss also auf jedem Emulationsknoten ein Gerät bereitgestellt werden, das ein GPS-Gerät nachbildet und die jeweils aktuelle Position des Teilnehmers ausgibt, den dieser Emulationsknoten repräsentiert.

Zu diesem Zweck haben wir eine Software-Komponente entwickelt, die das Verhalten und die Schnittstellen eines realen GPS-Geräts naturgetreu nachbilden kann, also ein *virtuelles GPS-Gerät*. Auf jedem Emulationsknoten läuft eine Instanz dieses virtuellen GPS-Geräts. Die zentrale Emulationssteuerung wurde so ergänzt, dass sie während der Messphase eines Szenarios

[20] Die Verwendung von „Flooding“ im Lokationsdienst ist sehr ineffizient und stellt für einzelne Sendungen den Sinn des GLBR-Protokolls in Frage: Soll nur eine einzige Nachricht übertragen werden, lohnt sich die Anfrage einer Position erst gar nicht – es wäre effizienter, gleich die eigentliche Nachricht per „Flooding“ zu übertragen. Wir gehen in unserem Szenario allerdings davon aus, dass *zahlreiche* aufeinanderfolgende Sendungen an *denselben* Empfänger gesendet werden sollen. In diesem Fall fällt der Mehraufwand durch das anfängliche Erfragen der Position durch „Flooding“ weniger ins Gewicht.

außer den regelmäßigen Konfigurationsnachrichten für die Emulationswerkzeuge zusätzlich aktuelle Ortsinformationen an alle virtuellen GPS-Geräte verschickt. So kann jedes virtuelle GPS-Gerät während eines Messlaufs die Position ausgeben, die der zugeordnete Teilnehmer entsprechend der Szenariodefinition zu diesem Zeitpunkt hat. Weitere Informationen zum Konzept und der Implementerung des virtuellen GPS-Geräts sind in einer Studienarbeit [Wei03] und einer Veröffentlichung [HMTR04] zu finden.

6.5.2 Simulationsmodell

Um schon *vor* der Verfügbarkeit der Implementierung des GLBR-Protokolls Aussagen über dessen Leistung treffen zu können, wurde im Rahmen des CarTalk2000-Projekts zunächst ein *präskriptives Modell* des Protokolls für das Simulationswerkzeug ns-2 erstellt. Wie jedes Modell weist auch dieses Simulationsmodell Vereinfachungen gegenüber der tatsächlichen Implementierung auf. In diesem Fall ist u.a. auf eine realistische Modellierung des Lokationsdienstes verzichtet worden: Im Simulationsmodell kennt jeder Teilnehmer zu jedem Zeitpunkt die Position aller Teilnehmer, hat also einen perfekten Lokationsdienst zur Verfügung. Für das GLBR-Protokoll bedeutet dies, dass vor einer Sendung an einen bestimmten Empfänger keine Positionsanfragen versendet werden müssen. Im Simulationsmodell wurde also hauptsächlich die *Weiterleitungsfunktion* des GLBR-Protokolls nachgebildet. Weitere Vereinfachungen gegenüber der Realität, die sich in ns-2-Modellen grundsätzlich nicht vermeiden lassen, betreffen Zeitgebergranularitäten, den Einfluss des Betriebssystems (Rechenzeitzuteilung) etc. (vgl. [BP96b, HZ03]).

Im Folgenden werden für dasselbe Szenario die Messergebnisse des Testsubjekts im NET-System den Simulationsergebnissen des Simulationsmodells in ns-2 gegenübergestellt.

6.5.3 Messungen und Simulationen

Wir gehen von folgendem Szenario aus: In der Innenstadt von Stuttgart bewegen sich 50 Fahrzeuge, die mit WLAN-Geräten ausgestattet sind. Die Bewegungsvorschriften wurden entsprechend den Daten aus Tab. 6.1 auf Seite 115 erstellt. Wir benutzen das TRG-Modell (ohne geographische Information)[21] zur Berechnung der Netzparameter und wählen bei gleichbleibendem Empfangsschwellwert die Sendeleistung so, dass sich eine Reichweite zwischen 50 m und 400 m ergibt.

Um Last zu erzeugen, wählt jedes Fahrzeug zu Beginn des Szenarios zufällig einen Kommunikationspartner aus und sendet während der gesamten Szenariodauer jede Sekunde eine Nach-

[21] Wie in Abschnitt 5.3 gezeigt, ist die Verwendung von geographischer Information für Szenarien in einer Stadt empfehlenswert. Wir verwenden trotzdem das TRG-Modell ohne geographische Information, um die Messergebnisse mit Simulationsergebnissen von ns-2 vergleichen zu können.

richt mit 64 Byte Nutzlast an dieses Fahrzeug. Wir messen sowohl die Zustellrate als auch die durchschnittliche Pfadlänge der Nachrichten. Abb. 6.12 zeigt die Ergebnisse der Messung zusammen mit den Ergebnissen aus der Simulation mit ns-2.

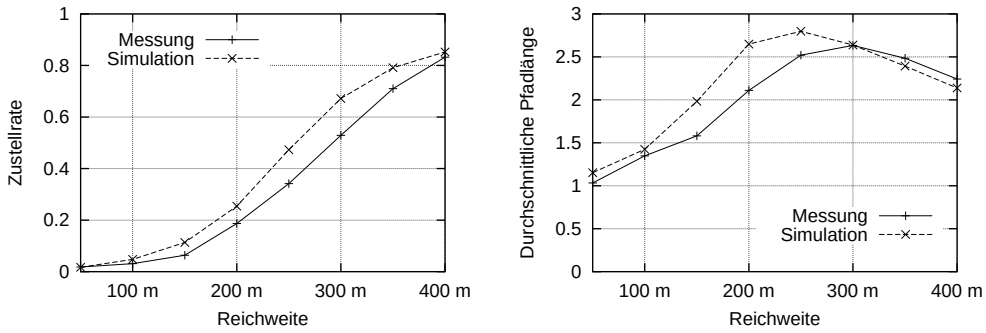

Abbildung 6.12: Zustellrate und durchschnittliche Pfadlänge des GLBR-Protokolls nach Simulation mit ns-2 und Messung im NET-System

6.5.4 Fazit

Die Ergebnisse aus Messung und Simulation dürfen nicht leichtfertig verglichen werden, da die untersuchten Testsubjekte nicht identisch sind: Die Messung im NET-System misst die Leistung der *Implementierung* des GLBR-Protokolls in einem realen System mit emulierten Netzeigenschaften, während die Simulation die Leistung eines *Modells* des GLBR-Protokolls in einer simulierten Umgebung misst. Unser wichtigstes Fazit aus den Messungen in diesem Abschnitt ist, dass es mit unserem Emulationssystem erstmals möglich ist, die *Implementierung* eines solchen Protokolls im Labor unter wiederholbaren Bedingungen zu analysieren. Mit keiner anderen Analysemethode war dies bisher möglich, insbesondere auch nicht mit einem simulativen Ansatz.

Unter Beachtung der methodischen Besonderheit dieses Vergleichs lassen sich trotzdem einige Schlüsse aus dem Vergleich der Messergebnisse ziehen: Die Implementierung des GLBR-Protokolls zeigt qualitativ und quantitativ ein ähnliches Verhalten, wie es vom Simulationsmodell vorhergesagt wurde. Der Vergleich der Zustellrate zeigt, dass die Implementierung stets etwas unter der Leistung des Simulationsmodells liegt, was durch die idealisierenden Annahmen im Simulationsmodell erklärt werden kann. Wenn man davon ausgeht, dass die Funktion des Simulationsmodells korrekt ist, ist das Leistungsverhalten der Implementierung ein Anzeichen für deren korrekte Funktion, wenn auch natürlich kein Beweis.

Außerdem ist die weitgehende Übereinstimmung der Ergebnisse von Simulation und Messung auch ein Indiz dafür, dass das NET-System in der Lage ist, ein typisches, für Simulationen ver-

wendetes Netzszenario so nachzubilden, dass damit die Untersuchung einer Implementierung eines Protokolls möglich ist.

An dieser Stelle ist ausdrücklich darauf hinzuweisen, dass aus dem Vergleich der Ergebnisse aus Simulation und Messung keinesfalls ein *Qualitätsunterschied* der beiden *Methoden* Simulation und Messung in emulierter Umgebung abgeleitet werden kann. Vielmehr sind Simulation und Messung stets als komplementäre Methoden zur Untersuchung verwandter, aber *unterschiedlicher* Testsubjekte zu verstehen, und daher gleichberechtigte Analysemethoden für verschiedene Anwendungsfelder bzw. unterschiedliche Arten von Testsubjekten.

Während der Vorbereitung und Durchführung der hier vorgestellten Messungen haben sich auch wichtige *methodische* Erkenntnisse ergeben: Es ist möglich, dass ein Testsubjekt für seine Ausführung auf spezielle Hardware-Komponenten angewiesen ist, die nicht im NET-System vorhanden sind, wie in diesem Fall das GPS-Gerät. Für die korrekte Ausführung solcher Testsubjekte im NET-System ist die Bereitstellung von *emulierten Geräten* notwendig. Die Emulation von Geräten, die nicht die Funktion von Netzwerkgeräten haben, gehört jedoch nicht zum Kern dieser Arbeit und wird im Ausblick (Abschnitt 7.2) nochmals aufgegriffen.

Als weitere methodische Erkenntnis aus den Messungen des GLBR-Protokolls ist anzumerken, dass durch die Ausführung des Testsubjekts im NET-System zunächst mehrere *funktionale Fehler* in der Implementierung entdeckt wurden, die in vorigen Tests (auf wenigen einzelnen Rechnern) nicht aufgefallen waren. Erst durch die Ausführung in einer realistischen Netzumgebung konnten die Fehler analyisert und behoben werden. Das NET-System, das eigentlich nur für die *Leistungsanalyse* von verteilten Anwendungen und Netzprotokollen gedacht war, hat sich also auch als wertvolles Werkzeug für den *Funktionstest* solcher Anwendungen herausgestellt.

6.6 Zusammenfassung

In diesem Kapitel wurden zunächst wichtige Eigenschaften des NET-Systems gemessen. Aus den Ergebnissen wurde klar, dass das NET-System als Grundlage für die Emulation von Netzeigenschaften sehr gut geeignet ist. Die Evaluation der zentralen Emulationssteuerung hat gezeigt, dass deren Leistung ausreicht, um Messläufe von Netzszenarien bis zur Maximalgröße des NET-Systems zu steuern. Die Evaluation der verteilten Emulationswerkzeuge hat ergeben, dass die spezifizierten Netzeigenschaften innerhalb der technischen Grenzen zuverlässig nachgebildet werden können. Diese Grenzen wurden identifiziert.

Aus dem Nachweis, dass einzelne Netzverbindungen mit unserem Emulationsansatz realistisch nachgebildet werden, folgt auch, dass ein emuliertes Netzszenario, das aus emulierten Verbindungen besteht, als Ganzes realistische Eigenschaften hat. Anhand eines durchgängigen Beispiels, der Evaluation eines Vermittlungsprotokolls in einer emulierten MANET-Umgebung,

wurde schließlich gezeigt, dass es mit dem von uns entwickelten System erstmals möglich ist, im Labor Implementierungen von Netzprotokollen in wiederholbaren Messläufen zu analyisieren.

7

Resümee

7.1 Zusammenfassung

Für die Leistungsanalyse von verteilten Anwendungen und Netzprotokollen in verschiedenen Netzszenarien ist es notwendig, die Eigenschaften von Rechnernetzen zuverlässig in einer Testumgebung emulieren zu können. In der vorliegenden Arbeit wurde ein Verfahren zur Emulation von Rechnernetzen entwickelt, das sowohl skalierbar ist, als auch Netze mit gemeinsamen Medien einschließt. Im Folgenden sind die wesentlichen Ergebnisse der Arbeit kurz zusammengefasst.

Zunächst wurde in dieser Arbeit die Methode der Messung in einer emulierten Netzumgebung motiviert und von den verwandten Analysemethoden – der Simulation und der Messung in einer Realumgebung – abgegrenzt. Immer dann, wenn nicht die Leistung eines Simulationsmodells, sondern die einer *Implementierung* in einem speziellen Netzszenario beurteilt werden soll, gibt es keine Alternative zu einer Messung. Messungen in einer Realumgebung sind nur dann sinnvoll, wenn die gewünschte Netzumgebung verfügbar ist und nur statische Eigenschaften hat. In allen anderen Fällen ist die Messung in einer Umgebung mit emulierten Netzeigenschaften die Methode der Wahl, weil nur so die Verfügbarkeit eines beliebigen Netzszenarios, sowie die Wiederholbarkeit der Netzeigenschaften und damit die Vergleichbarkeit der Ergebnisse verschiedener Messläufe garantiert werden kann.

Die meisten existierenden Lösungen für die Emulation von Netzeigenschaften betrachten entweder nur die Emulation einer einzelnen Netzverbindung, oder sind durch eine komplett zentralisierte Lösungsarchitektur nur für sehr kleine Szenarien geeignet. Die wenigen Ansätze, die explizit auch größere Szenarien einbeziehen, können nur mehrere *unabhängige* Netzverbindungen nachbilden, und schließen damit alle Netztechnologien mit gemeinsamen Medien aus.

Als mögliche Lösungsvorschläge wurden in dieser Arbeit verschiedene *Architekturvarianten* für eine Testumgebung mit emulierten Netzeigenschaften vorgestellt und bewertet. Für die vielversprechende Architekturvariante mit zentraler Steuerung und verteilten Emulationswerkzeugen wurde dann detailliert die Funktionalität eines Emulationssystems mit seinen wesentlichen Komponenten beschrieben.

Das in dieser Arbeit entwickelte Emulationsverfahren greift auf der logischen Ebene der Sicherungsschicht in den Kommunikationsstapel ein. Auf dieser Ebene werden die beiden *Basiseffekte* Rahmenverlust und Verzögerung durch verteilte Emulationswerkzeuge nachgebildet. Alle anderen Netzeigenschaften können auf diese Basiseffekte zurückgeführt werden.

Um Netztechnologien mit gemeinsamem Medium durch verteilte Werkzeuge nachbilden zu können, wurde zusätzlich das Konzept des *virtuellen Trägersignals* eingeführt. Hierbei werden die Eigenschaften eines Rundsendemediums nachgebildet, indem kooperative Emulationswerkzeuge Rundsendungen zur Signalisierung eines Trägersignals benutzen. Durch diese Signalisierungen kann jede Werkzeuginstanz lokal ein aktuelles Modell des emulierten gemeinsamen Mediums halten. Auf der Basis dieses Medienmodells kann dann das Verhalten von Medienzugriffsprotokollen nachgebildet werden.

Die vorgestellte Lösung wurde in vollem Umfang realisiert und existiert nun in Form des *NET-Systems*. Die wichtigsten Realisierungsaspekte wurden in dieser Arbeit besprochen; auf weiterführende Dokumentation wurde an den entsprechenden Stellen verwiesen. Mit ausführlichen Messungen wurde gezeigt, dass das entwickelte System für die Emulation von Netzszenarien sehr gut geeignet ist. Die entwickelten Werkzeuge sind in der Lage, Netzeigenschaften in einem weiten Parameterbereich realistisch nachzubilden. Mit dem NET-System steht nun eine ideale Testumgebung für vergleichende Leistungsmessungen von verteilten Anwendungen und Netzprotokollen zur Verfügung.

7.2 Ausblick

Die in der vorliegenden Arbeit vorgestellte Architektur eines Emulationssystems ist als umfassendes Rahmenwerk zu verstehen. Das NET-System ist eine mögliche Realisierung dieses Rahmenwerks, die an mehreren Stellen erweiterbar ist.

Parameterberechnung

Der Prozess der automatischen Parameterberechnung wurde für den konkreten Fall der Berechnung von Bitfehlerraten aus Bewegungsvorschriften in einem MANET-Szenario mit WLAN-Kommunikation realisiert. Die Dämpfung und Reflexion der Funkwellen durch Gebäude wurde

dabei berücksichtigt. Viele weitere Effekte, die ebenfalls Auswirkungen auf die Bitfehlerrate haben können, wurden jedoch vernachlässigt: z.B. die Orientierung der Antennen, die Abschattung durch die Teilnehmer selbst oder durch andere mobile Objekte, aktive Störquellen, sowie der Einfluss des Wetters auf die Dämpfung. Genauere Berechnungsmodelle können hier die Qualität der Parameterberechnung weiter erhöhen, bzw. weitere dynamische Parameter mit einbeziehen.

Emulationswerkzeuge

Die entwickelten Emulationswerkzeuge können die spezifizierten Netzeigenschaften im technisch möglichen Parameterbereich sehr gut nachbilden. Für die Nachbildung von Medienzugriffsprotokollen auf der Basis des virtuellen Trägersignals existieren bereits Werkzeuge für IEEE 802.3 (Ethernet) [HMR03] und IEEE 802.11b (WLAN) [Yan04]. Die Werkzeuge sind allerdings auf die Nachbildung von Netzen mit diesen Medienzugriffsprotokollen beschränkt. Für die Nachbildung von anderen Medienzugriffsprotokollen müssen weitere Werkzeuge nach diesem Vorbild entwickelt werden.

Emulation von zusätzlichen Ressourcen

Manche Testsubjekte benötigen außer den Netzwerkgeräten noch weitere Ressourcen, die in einer Testumgebung entweder überhaupt nicht verfügbar sind, oder andere als die gewünschten Eigenschaften haben. Für die Messungen in Abschnitt 6.5 wurde beispielsweise ein GPS-Gerät auf den Emulationsknoten benötigt. Die Emulation von solchen zusätzlichen Ressourcen wurde in dieser Arbeit nur am Rande besprochen. Neben dem emulierten GPS-Gerät [HMTR04] wurde die prototypische Realisierung einer emulierten Batterie [Stö04] erstellt, um Testsubjekte untersuchen zu können, deren Verhalten vom Batteriestatus abhängt („*energy-aware*“). Weitere Ressourcen, von denen es sinnvoll wäre, sie in einer emulierten Version in der Testumgebung bereitzustellen, wären beispielsweise ein Prozessor mit spezifizierbarer Rechenleistung oder Primär- und Sekundärspeicher mit spezifizierbarer Größe und Zugriffsgeschwindigkeit.

Szenariogröße

In der von uns gewählten Architektur ist die maximale Anzahl der Knoten in einem emulierten Netzszenario durch die Anzahl der Knoten der Testumgebung beschränkt; in der derzeitigen Ausbaustufe des NET-Systems ist also die Emulation von Szenarien mit bis zu 64 Knoten möglich. Durch den Ausbau der Hardware des NET-Systems wären schon mit der jetzigen Architektur größere Szenarien möglich. Die Einführung von *virtuellen Knoten* ermöglicht allerdings die Nachbildung von wesentlich größeren Szenarien, ohne neue Hardware zu benötigen. Abhängig

von der Leistungsfähigkeit der Emulationsknoten und dem Ressourcenbedarf der Testsubjekte können auf einem physischen Knoten mehrere virtuelle Knoten eingerichtet werden. Jeder virtuelle Knoten kann dabei einen Knoten in einem Netzszenario repräsentieren. Es muss jedoch gewährleistet sein, dass die virtuellen Knoten untereinander mit den im Netzszenario definierten Eigenschaften kommunizieren können und sich dabei nicht gegenseitig in unerwünschter Weise beeinflussen. Je nach Effizienz der Knotenvirtualisierung kann die maximale Größe der Netzszenarien durch einen solchen Ansatz um eine bis zwei Größenordnungen erhöht werden. Methoden für die Emulation von mehreren virtuellen Knoten auf *einem* physischen Knoten sind bereits bekannt [MI04, ESHF04], die effiziente und transparente Verbindung *mehrerer* solcher Knoten zu einem sehr großen Gesamtszenario ist Thema einer weiterführenden Arbeit [MHR05].

A

Abkürzungsverzeichnis

ABR	Associativity-Based Routing (Vermittlungsprotokoll für MANETs)
AODV	Ad-hoc On-demand Distance Vector routing (Vermittlungsprotokoll für MANETs)
AP	Access Point (Zugangspunkt einer WLAN-Infrastruktur)
APIC	Advanced Programmable Interrupt Controller (genauer Zeitgeber von Intel-Prozessoren)
BSD	Berkeley Software Distribution (UNIX-Implementierung der University of California, Berkeley. „FreeBSD“ und „NetBSD“ sind Varianten hiervon)
CSMA	Carrier Sense Multiple Access (Mehrfachzugriff durch Trägererkennung; Grundlage für Medienzugriffsprotokolle)
CTS	Clear To Send (Signalisierung, z.B. als Rahmen in IEEE 802.11)
DSR	Dynamic Source Routing (Vermittlungsprotokoll für MANETs)
FIFO	First In First Out (reihenfolgeerhaltend)
FTP	File Transfer Protocol (Anwendungsprotokoll für Dateiübertragungen)
GLBR	Greedy Location-Based Routing (Protokoll zur ortsbasierten Nachrichtenvermittlung)
GPRS	General Packet Radio Service (Paketorientierter Datendienst im Mobilfunknetz)
GPS	Global Positioning System (Globales satellitengestütztes Positionierungssystem)
HTTP	Hyper Text Transfer Protocol (Anwendungsprotokoll des „World Wide Web“)
HW	Hardware
ICMP	Internet Control Message Protocol (Protokoll für Signalisierungen in IP)
IEEE	Institute of Electrical and Electronics Engineers (weltweiter Berufsverband von Ingenieuren aus den Bereichen Elektrotechnik und Informatik)

IOCTL Input/Output Control (Betriebssystemschnittstelle für die Gerätekonfiguration und -steuerung)

IPX Internetworked Packet Exchange (Vermittlungsprotokoll des OSI-Referenzstapels)

IP Internet Protocol (Vermittlungsprotokoll im Internet)

IPVS Institut für Parallele und Verteilte Systeme

IRT Intelligent Ray Tracing (Modell zur Berechnung von Funkwellenausbreitung)

LAN Local Area Network (lokal begrenztes Rechnernetz)

LLC Logical Link Control (oberster Teil der Sicherungsschicht)

MAC Medium Access Control (Medienzugriffssteuerung)

MANET Mobile Ad Hoc Network (Mobiles Ad-Hoc-Netz)

NET Network Emulation Testbed (Testumgebung mit nachgebildeten Netzeigenschaften am IPVS)

OSI Open Systems Interconnect (Projekt für die Standardisierung von Netzprotokollen)

OTcl Object Tcl (objektorientierte Version von Tcl)

PC Personal Computer (eigentlich „persönlich zugeordneter Rechner“, hier gebraucht im Sinne von „preiswerter Standardrechner“)

RAID Redundant Array of Inexpensive / Independent Disks (System zur Organisation mehrerer Festplatten zu einem leistungsfähigen bzw. sicheren logischen Laufwerk)

RTS Ready To Send (Signalisierung, z.B. als Rahmen in IEEE 802.11)

SNMP Simple Network Management Protocol (Standardprotokoll für die Verwaltung von Netzinfrastrukturgeräten)

SW Software

Tcl Tool Command Language (einfache Skriptsprache)

TCP Transmission Control Protocol (verbindungsorientiertes Transportprotokoll)

TRG Free space / Two-ray ground (Modell zur Berechnung von Funkwellenausbreitung)

TSC Time Stamp Counter (Prozessorregister, das bei jedem Prozessortakt erhöht wird)

UDP User Datagram Protocol (verbindungsloses Transportprotokoll)

VLAN Virtual LAN (virtuelles lokales Rechnernetz)

WAN Wide Area Network (Weitverkehrsnetz)

WLAN Wireless Local Area Network (drahtloses lokal begrenztes Rechnernetz)

B

Formelzeichen

b	Übertragungsrate (von engl. *bandwidth*)
$b_{\max}$	maximal erreichbare Übertragungsrate
c	Ausbreitungsgeschwindigkeit
c_0	Lichtgeschwindigkeit ($\approx 3 \cdot 10^8\,\mathrm{m/s}$)
d	Länge des Übertragungsmediums zwischen Sender und Empfänger
f	Frequenz
f_{CPU}	Taktfrequenz eines Prozessors
h	Höhe einer Antenne über dem Boden
l	Rahmenlänge (in Bit)
$l_{\max}$	maximale Rahmenlänge (in Bit)
λ	Wellenlänge
n_k	Anzahl der Knoten in einem Szenario
n_v	Anzahl der Netzverbindungen in einem Szenario
p_b	Bitfehlerrate
p_v	Rahmenverlustwahrscheinlichkeit
P_e	empfangene Leistung
P_r	Kanalrauschen
P_s	Sendeleistung
P_{es}	Empfangsschwellwert
$q_{\max}$	maximale Größe der Emulationswarteschlange (in Bit)
r	Rahmen (als Symbol in Abbildungen)
R	Zufallszahl ($0 \leq R < 1$)
t	(Real-)Zeit
t'	Sendezeitpunkt eines Rahmens
t_0	Übergabezeitpunkt eines Rahmens an ein Emulationswerkzeug

t_a	Ausbreitungsverzögerung
t_b	Zeitpunkt des Beginns eines Trägersignals
t_{BV}	Zeit für das Berechnen und Verschicken der Netzparameter in der Emulationssteuerung
t_e	Zeitpunkt des Endes eines Trägersignals
t_m	Medienwartezeit
t_{min}	minimale durch die Testumgebung eingeführte Rahmenverzögerung
t_n	Rahmenverzögerung durch das Rechnernetz der Testumgebung
t_r	Empfangszeitpunkt eines Rahmens (von engl. *receive*)
t_s	Serialisierungsverzögerung
t_v	Verarbeitungszeit eines Emulationswerkzeugs bzw. Proxys
t_{Zyklus}	Zeit für einen Berechnungszyklus der Emulationssteuerung
$T_{1..4}$	Zeitstempel
v	Geschwindigkeit (eines mobilen Teilnehmers in einem Netzszenario)

Abbildungsverzeichnis

Literaturverzeichnis

[AB82] I. F. Akyildiz and G. Bolch. Möglichkeiten und Grenzen der analytischen Modellbildung von Warteschlangensystemen. *Informatik Fachberichte*, Band 56:66–78, 1982. Springer Verlag Hamburg.

[ACO97] Mark Allman, Adam Caldwell, and Shawn Ostermann. ONE: The Ohio Network Emulator. Technical Report TR-19972, Ohio University, School of Electrical Engineering and Computer Science, 18 August 1997.

[ADLY95] Jong Suk Ahn, Peter B. Danzig, Zhen Liu, and Limin Yan. Evaluation of TCP Vegas: Emulation and Experiment. *ACM SIGCOMM Computer Communication Review*, 25(4):185–195, October 1995.

[BB95] A. Bakre and B. R. Badrinath. I-TCP: indirect TCP for mobile hosts. In *in Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS 1995)*, pages 136–143, Vancouver, Canada, May 1995.

[BEF+00] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in Network Simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[BP96a] Lawrence S. Brakmo and Larry L. Peterson. Experiences with Network Simulation. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 80–90, Philadelphia, Pennsylvania, USA, May 1996. ACM Press.

[BP96b] Lawrence S. Brakmo and Larry L. Peterson. Experiences with Network Simulation. *ACM SIGMETRICS Performance Evaluation Review*, 24(1):80–90, May 1996.

[BSAK95] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking (MobiCom 1995)*, Berkeley, CA, USA, November 1995. ACM Press.

[Cas05] Mirko Casper. Verzögerungszeiten von Vermittlungsstellen in einem Netzwerkemulationssystem. Studienarbeit Nr. 1969, Universität Stuttgart, IPVS, Abteilung Verteilte Systeme, March 2005.

[CDJM91] R. Caceres, P. Danzig, S. Jamin, and D. Mitzel. Characteristics of Wide-Area TCP/IP Conversations. In *Proceedings of the Conference on Communications Architecture and Protocols (ACM SIGCOMM 91)*, pages 101–112, Zürich, Switzerland, September 1991. ACM.

[CI95] Ramon Caceres and Liviu Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE Journal of Selected Areas in Communications*, 13(5):850–857, 1995.

[CS03] Mark Carson and Darrin Santay. NIST Net — A Linux-based Network Emulation Tool. *ACM SIGCOMM Computer Communications Review (CCR)*, 33(3):111–126, July 2003.

[DBCF95] Nigel Davies, Gordon S. Blair, Keith Cheverst, and Adrian Friday. A Network Emulator To Support The Development Of Adaptive Applications. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location Independent Computing*, pages 47–55, Ann Arbor, Michigan, U.S.A., April 1995.

[DKS89] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of the ACM Symposium on Communications Architectures and Protocols (ACM SIGCOMM 89)*, pages 3–12, Austin, Texas, September 1989. ACM.

[dPPC03] Javier del Prado Pavon and Sunghyun Choi. Link adaptation strategy for IEEE 802.11 WLAN via received signal strength measurement. In *Proceedings of the IEEE International Conference on Communications (ICC'03)*, pages 1108–1113, May 2003.

[Dud02] Dominique Dudkowski. Emulationskonzepte für Mobilfunk- und Ad-Hoc-Netze. Diplomarbeit Nr. 2004, Universität Stuttgart, Abteilung Verteilte Systeme, 2002.

[EHH+00] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network visualization with Nam, the VINT network animator. *IEEE Computer*, 33(11):63–68, November 2000.

[ESHF04] Michael Engel, Matthew Smith, Sven Hanemann, and Bernd Freisleben. Wireless Ad-Hoc Network Emulation Using Microkernel-Based Virtual Linux Systems. In *Proceedings of the 5th EUROSIM Congress on Modeling and Simulation*, pages 198–203, Marne la Vallee, France, 2004.

[Fal99] Kevin Fall. Network Emulation in the Vint/NS Simulator. In *Proceedings of the Fourth IEEE Symposium on Computers and Communications (ISCC'99)*, pages 244–250, Red Sea, Egypt, July 6–8 1999.

[Flo01] Sally Floyd. Simulation is Crucial. *Sidebar, IEEE Spectrum*, January 2001.

[FLV01] Henryk Fuks, Anna T. Lawniczak, and Stanislav Volkov. Packet delay in models of data networks. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 11(3):233–250, July 2001.

[FP01] Sally Floyd and Vern Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking (TON)*, 9(4):392–403, August 2001.

[Fri46] H. T. Friis. A note on a simple transmission formula. *Proceedings of the IRE*, 34:254–256, 1946.

[FTO02] Juan Flynn, Hitesh Tewari, and Donal O'Mahony. A Real Time Emulation System for Ad hoc Networks. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, pages 115–120, San Antonio, Texas, January 27–31 2002.

[FV02] Kevin Fall and Kannan Varadhan. *The ns Manual*. UC Berkeley, LBL, USC/ISI, and Xerox PARC, 2002.

[GHN+03] Alexandre Gerber, Joseph Houle, Han Nguyen, Matthew Roughan, and Subhabrata Sen. P2P, The Gorilla in the Cable. In *Proceedings of the National Cable and Telecommunications Association (NCTA) 2003 National Show*, Chicago, IL, USA, June 2003.

[HBE+01] John Heidemann, Nirupama Bulusu, Jeremy Elson, Chalermek Intanagonwiwat, Kun chan Lan, Ya Xu, Wei Ye, Deborah Estrin, and Ramesh Govindan. Effects of Detail in Wireless Network Simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–11, Phoenix, Arizona, USA, January 2001.

[HH02] Edwin Hernandez and Abdelsalam (Sumi) Helal. RAMON: Rapid-Mobility Network Emulator. In *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN'02)*, pages 809–817, Tampa, Florida, November 6–8 2002.

[HLR02] Daniel Herrscher, Alexander Leonhardi, and Kurt Rothermel. Modeling Computer Networks for Emulation. In *Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '02)*, pages 1725–1731, Las Vegas, June 2002.

[HM04] Daniel Herrscher and Steffen Maier. *NET: The Network Emulation Testbed Manual*. IPVS, 2004.

[HMR03] Daniel Herrscher, Steffen Maier, and Kurt Rothermel. Distributed Emulation of Shared Media Networks. In *Proceedings of the 2003 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2003)*, pages 226–233, Montréal, Quebec, Canada, July 20–24 2003.

[HMTR04] Daniel Herrscher, Steffen Maier, Jing Tian, and Kurt Rothermel. A Novel Approach to Evaluating Implementations of Location-Based Software. In *Proceedings of the 2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2004)*, pages 484–490, San Jose, CA, USA, July 25–29 2004.

[HR02] Daniel Herrscher and Kurt Rothermel. A Dynamic Network Scenario Emulation Tool. In *Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, pages 262–267, Miami, October 2002.

[HZ03] E. Hasenleithner and T. Ziegler. Comparison of Simulation and Measurement using State of the Art Web Traffic Models. In *Proceedings of the Eigth IEEE Symposium on Computers and Communications (ISCC'03)*, Antalya, Turkey, July 2003.

[IEE03] IEEE. *802.1Q: Virtual Bridged Local Area Networks*, 2003.

[Int00] Intersil Corporation. HFA3861B Direct Sequence Spread Spectrum Baseband Processor Datasheet. Available from `http://www.intersil.com/`, 2000.

[IP94] David B. Ingham and Graham D. Parrington. Delayline: A Wide-Area Network Emulation Tool. *Computing Systems*, 7(3):313–332, 1994.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., New York, NY, 1991.

[JM96] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, volume 353, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[JS03] Glenn Judd and Peter Steenkiste. Repeatable and Realistic Wireless Experimentation through Physical Emulation. In *Proceedings of the 2nd Workshop on Hot Topics in Networks (HotNets-II)*, Cambridge, MA, USA, November 20–21 2003.

[KGM+01] Markku Kojo, Andrei Gurtov, Jukka Manner, Pasi Sarolahti, Timo Alanko, and Kimmo Raatikainen. Seawind: a Wireless Network Emulator. In *Proceedings of*

the 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB 2001), Aachen, Germany, September 11–14 2001.

[KMJ00] Qifa Ke, David A. Maltz, and David B. Johnson. Emulation of Multi-Hop Wireless Ad Hoc Networks. In *Proceedings of the Seventh International Workshop on Mobile Multimedia Communications (MoMuC 2000)*, Tokyo, Japan, October 23–26 2000. IEEE Communications Society.

[KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988.

[KR01] James T. Kaba and Douglas R. Raichle. Testbed on a Desktop: Strategies and Techniques to Support Multi-hop MANET Routing Protocol Development. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHOC 2001)*, pages 164–172, Long Beach, CA, USA, October 4–5 2001.

[Lan99] F. M. Landstorfer. Wave propagation models for the planning of mobile communication networks. In *Proceedings of the 29th European Microwave Conference*, pages 1–6, Munich, Germany, October 1999.

[LLN+02] Henrik Lundgren, David Lundberg, Johan Nielsen, Erik Nordström, and Christian Tschudin. A Large-scale Testbed for Reproducible Ad hoc Protocol Evaluations. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2002)*, volume 1, pages 412–418, Orlando, Florida, USA, March 2002. IEEE.

[LMP02] Tao Lin, Scott F. Midkiff, and Jahng S. Park. A Dynamic Topology Switch for the Emulation of Wireless Mobile Ad Hoc Networks. In *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN'02)*, pages 791–798, Tampa, Florida, USA, November 6–8 2002.

[LS02] Weiguo Liu and Hantao Song. Research and Implementation of Mobile Ad Hoc Network Emulation System. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02)*, pages 749–754, Vienna, Austria, July 2002.

[Mai02] Steffen Maier. Emulationskonzepte für Netze mit gemeinsamem Medium. Diplomarbeit Nr. 2000, Universität Stuttgart, Abteilung Verteilte Systeme, April – Oktober 2002.

[MBJ99] David A. Maltz, Josh Broch, and David B. Johnson. Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed. Technical Report CMU-CS-99-116, Carnegie Mellon University, School of Computer Science, Pittsburgh, Pennsylvania, March 1999.

[MCG+01] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom 2001)*, Rome, Italy, July 2001. ACM Press.

[MHR05] Steffen Maier, Daniel Herrscher, and Kurt Rothermel. On Node Virtualization for Scalable Network Emulation. In *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005)*, pages 917–928, Philadelphia, PA, USA, July 24–28 2005.

[MI04] Daniel Mahrenholz and Svilen Ivanov. Real-time network emulation with ns-2. In *Proceedings of the 8-th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT'04)*, pages 29–36, Budapest, Hungary, October 21–23 2004.

[MI05] Daniel Mahrenholz and Svilen Ivanov. Adjusting the ns-2 Emulation Mode to a Live Network. In *Proceedings of Kommunikation in Verteilten Systemen 2005 (KiVS 2005)*, Kaiserslautern, Germany, February 28 – March 3 2005.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 01)*, Cincinnati, Ohio, August 2001.

[MWH01] Martin Mauve, Jörg Widmer, and Hannes Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network*, 15(6):30–39, November 2001.

[MYV03] Priya Mahadevan, Ken Yocum, and Amin Vahdat. Emulating Large-Scale Wireless Networks using ModelNet. *ACM SIGMOBILE Mobile Computing and Communications Review, Poster Session*, 7(1):62–64, January 2003.

[Nec98] Thomas Necker. *Entwicklung eines objektorientierten Werkzeugs für verschiedene Verfahren der parallelen ereignisgesteuerten Simulation*. 69. Bericht über verkehrstheoretische Arbeiten, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1998.

[NSNK97] Brian D. Noble, M. Satyanarayanan, Giao T. Nguyen, and Randy H. Katz. Trace-Based Mobile Network Emulation. In *Proceedings of the ACM Conference on*

Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '97), pages 51–61, Cannes, France, September 14–18 1997.

[PB94] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM Computer Communications Review*, 24(4):234–244, October 1994.

[PF97] Vern Paxson and Sally Floyd. Why We Don't Know How To Simulate The Internet. In *Proceedings of the 1997 Winter Simulation Conference*, pages 1037–1044, Atlanta, Georgia, U.S.A., December 1997.

[PR99] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications (WMCSA'99)*, pages 90–100, New Orleans, Louisiana, February 25–26 1999. IEEE Computer Society.

[RAL03] Robert Ricci, Chris Alfeld, and Jay Lepreau. A Solver for the Network Testbed Mapping Problem. *ACM SIGCOMM Computer Communications Review*, 33(2):65–81, April 2003.

[Rap01] Theodore Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 2001.

[Rep03] Rainer Repp. Vergleich der Verfahren Simulation und Emulation für die Evaluation von Protokollen. Diplomarbeit Nr. 2119, Universität Stuttgart, Abteilung Verteilte Systeme, 2003.

[Ril03] George Riley. The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network research*, pages 5–12, New York, NY, USA, November 2003. ACM Press.

[Riz97] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, January 1997.

[RK02] Bhaskaran Raman and Randy H. Katz. Emulation-based Evaluation of an Architecture for Wide-Area Service Composition. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002)*, July 2002.

[RTY+00] Pavlin Radoslavov, Hongsuda Tangmunarunkit, Haobo Yu, Ramesh Govindan, Scott Shenker, and Deborah Estrin. On characterizing network topologies and analyzing their impact on protocol design. Technical Report USC-CS-TR-00-731, University of Southern California, Department of Computer Science, March 2000.

[SHB+03] Illya Stepanov, Jörg Hähner, Christian Becker, Jing Tian, and Kurt Rothermel. A Meta-Model and Framework for User Mobility in Mobile Networks. In *Proceedings of the 11th International Conference on Networking (ICON 03)*, pages 231–238, Sydney, Australia, September 2003.

[SHR05] Illya Stepanov, Daniel Herrscher, and Kurt Rothermel. On the impact of radio propagation models on manet simulation results. In *Proceedings of the Seventh IFIP International Conference on Mobile and Wireless Communication Networks (MWCN 2005)*, Marrakech, Morocco, September 19–21 2005.

[Shu05] Shunra Software. The Cloud, 2005. http://www.shunra.com.

[Stö04] Andreas Störzbach. Emulation mobiler Geräte: Integration eines Batteriemodells. Studienarbeit Nr. 1917, Universität Stuttgart, Abteilung Verteilte Systeme, 2004.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer, Wien, Austria, 1973.

[Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 3rd edition, 1996.

[TCDA00] C.-K. Toh, Richard Chen, Minar Delwar, and Donald Allen. Experimenting with an Ad Hoc wireless network on campus: insights and experiences. *ACM SIGMETRICS Performance Evaluation Review*, 28(3):21–29, 2000.

[THRC03] Jing Tian, Lu Han, Kurt Rothermel, and Christian Cseh. Spatially Aware Packet Routing for Mobile Ad Hoc Inter-Vehicle Radio Networks. In *Proceedings of the 6th International Conference on Intelligent Transportation Systems (ITSC)*, Shanghai, China, October 12–15 2003.

[TMW97] K. Thompson, G. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6), November 1997.

[VYW+02] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, December 9–11 2002.

[Wei03] Benedict Weisshaar. Emulation eines Positionierungsgerätes. Studienarbeit Nr. 1904, Universität Stuttgart, Abteilung Verteilte Systeme, 2003.

[WLG02] Brian White, Jay Lepreau, and Shashi Guruprasad. Lowering the Barrier to Wireless and Mobile Experimentation. In *Proceedings of the First Workshop on Hot Topics in Networks (Hotnets-I)*, Princeton, New Jersey, USA, October 28–29 2002.

[WLS+02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of*

the 5th Symposium on Operating Systems Design and Implementation (OSDI '02), pages 255–270, Boston, MA, December 2002. USENIX Association.

[WPR+02] Klaus Wehrle, Frank Pählke, Hartmut Ritter, Daniel Müller, and Marc Bechler. *Linux Netzwerkarchitektur: Design und Implementierung von Netzwerkprotokollen im Linux-Kern*. Addison-Wesley, 2002.

[Wur02] Ulrich Wurst. Realistische Lasteinspeisung in Emulationsszenarien. Studienarbeit Nr. 1856, Universität Stuttgart, Abteilung Verteilte Systeme, 2002.

[Yan04] Zhenxiang Yang. Entwicklung eines Verfahrens zur Emulation der Medienzugriffssteuerung in Wireless LAN. Diplomarbeit Nr. 2167, Universität Stuttgart, Abteilung Verteilte Systeme, 2004.

[ZB93] Yongguang Zhang and Bharat Bhargava. A Facility For Experimenting Distributed Software in the Internet. In *Proceedings of the IEEE Workshop in Advances in Parallel and Distributed Systems*, pages 40–45, Princeton, New Jersey, U.S.A., October 1993.

[ZBG98] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulations (PADS '98)*, pages 154–161, Banff, Canada, May 1998.

[ZL02] Yongguang Zhang and Wei Li. An Integrated Environment for Testing Mobile Ad-Hoc Networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '02)*, pages 104–111, EPFL Lausanne, Switzerland, June 2002.

[ZN03] Pei Zheng and Lionel M. Ni. EMPOWER: A Network Emulator for Wireless and Wired Networks. In *Proceedings of the Conference on Computer Communications (INFOCOM 2003)*, volume 3, pages 1933–1942, San Francisco, March 30 – April 3 2003. IEEE.