MASTERARBEIT

Simulation mobiler Ad-Hoc Netze

Ausgeführt am Institut für
Breitbandkommunikation
der Technische Universität Wien

unter der Anleitung von
O.Univ.Prof. Dipl.-Ing. Dr.techn. Harmen R. van As

durch
Lukas Wallentin
Bahnlände 43-45, 1100 Wien

Wien, im Dezember 2008

Abstract

Due to the ongoing progress in the development of wireless communication systems and the advances in size and capacity of computer systems, mobile devices are going to be part of our daily life. From the statistical point of view every person in Austria possesses at the moment at least one mobile phone. Also other mobile devices like handheld computers or laptops are common today.

Such devices can be used to build so called ad-hoc networks. In these networks every free moving node can act as router. One cost effective way to solve open issues concerning ad-hoc networks is by simulation which is the topic of these theses.

Besides the theoretical background the theses present a simulation framework for mobile ad-hoc networks. This framework is build upon the network simulator of the Institute of Broadband Communications. It is characterized by its expandability as well as its support of large networks. Furthermore, the theses discuss the possible usage of the framework in education.

Zusammenfassung

Durch die laufenden Fortschritte in der Entwicklung drahtloser Kommunikationssysteme sowie der immer kleineren und leistungsstärkeren Computersysteme nimmt die Verbreitung mobiler Geräte fortwährend zu. Statistisch gesehen besitzt zum aktuellen Zeitpunkt jeder Österreicher zumindest ein Handy. Auch weitere mobile Geräte wie etwa Pocket PC oder Laptops sind keine Seltenheit mehr.

Solche Geräte können verwendet werden um Ad-Hoc Netze zu bilden. In einem solchen Netz agiert jeder der frei beweglichen Knoten als Router. Um Fragestellungen über solche Netze zu beantworten, bietet es sich aus praktikablen Gründen an, solche Netze zu Simulieren. Diese Arbeit beschäftigt sich mit der Simulation dieser Netze.

Neben der Behandlung des theoretischen Hintergrundes wird ein Framework zur Simulation von mobilen Ad-Hoc Netzen vorgestellt. Dieses Framework baut auf dem Simulator des Instituts für Breitbandkommunikation auf. Es zeichnet sich neben der einfachen Erweiterbarkeit vor allem durch die Unterstützung großer Netze aus. Weiters wird die Verwendbarkeit des Frameworks als Unterrichtswerkzeug behandelt.

Danksagung

Mein besonderer Dank gilt Herrn o.Univ. Prof. Dr.-Ing. Harmen R. van As, der es mir ermöglicht hat, die hier vorliegende Diplomarbeit am Institut für Breitbandkommunikation durchzuführen.

Bedanken möchte ich mich auch bei meinen Kollegen am Institut für Breitbandkommunikation, die immer ein offenes Ohr für meine Anliegen und Probleme hatten.

Weiters möchte ich mich auch bei Petra Schaner für die Verbesserungsvorschläge und Korrekturen an meiner schriftlichen Arbeit bedanken.

Nicht zuletzt möchte ich mir noch bei meiner Familie und Freunden bedanken, die mich während der Studienzeit unterstützten.

Wien, November 2008 Lukas Wallentin

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den letzten Jahren wurde die Verwendung von drahtlosen Kommunikationssystemen immer populärer. Egal ob Laptop, Handy oder Handheld Computer, sie alle besitzen heutzutage meist bereits mehrere kabellose Kommunikationssysteme. Systemen wie beispielsweise Wireless LAN können genutzt werden um Geräte miteinander zu verbinden, ohne dass eine entsprechende Infrastruktur nötigt ist. Sie bilden sogenannte Ad-Hoc Netzwerke. Zurzeit sehen die meisten Protokolle nur die Kommunikation zwischen Teilnehmern vor, die im direkten Funkkontakt zueinander stehen. Durch die Verwendung entsprechender Routingprotokolle ist es allerdings möglich Netze zu schaffen, in denen ein Teilnehmer mit allen anderen Teilnehmern des Netzes kommunizieren kann, solange er zumindest mit einem Teilnehmer in direkten Kontakt treten kann. Das wird dadurch ermöglicht, indem die Knoten in einem solchen Netz als Router agieren und die Pakete weiterleiten.

Um das Verhalten solcher Netze zu untersuchen oder interessante Fragestellungen zum Beispiel im Bezug auf das Routing zu beantworten, bietet es sich an solche Netze zu simulieren. Gerade im Bezug auf große mobile Ad-Hoc Netze mit mehreren tausenden Teilnehmern wäre es relativ teuer entsprechende Testnetze aufzubauen. Ein weiter Vorteil der Simulation ist, dass alle Einflussgrößen bekannt sind, sodass man etwa recht gut verschiedene Routing-Algorithmen unter gleichen Bedingungen testen kann.

Simulationen können aber nicht nur dafür verwendet werden neue Fragestellungen zu beantworten, sondern sie haben auch einen didaktischen Wert. Mit Hilfe von Simulationen können zum Bespiel das Hidden-Station-Problem oder die Funktionsweise des MAC Protokolls anschaulich darge-

stellt werden.

Im Rahmen dieser Diplomarbeit wurde ein Framework zur Simulation mobiler Ad-Hoc Netze basierend auf dem IEEE802.11 Standard erstellt. Neben der Möglichkeit, dieses Framework auf einfache Weise zu erweitern, wurde auch auf die Unterstützung von großen Netzen mit mehreren hundert Knoten geachtet. Das Framework ist dabei eine Erweiterung des IBKSims, des Simulators des Instituts für Breitbandkommunikation.

Zum Aufbau der Diplomarbeit:

In Kapitel 2 wird noch genauer beschrieben, was ein mobiles Ad-Hoc Netz eigentlich ist und was alles zur Simulation eines solchen Netzes benötigt wird. Es schließt mit einer genauen Anforderungsliste an das Framework.

Kapitel 3 behandelt allgemein das Thema Simulation. Dabei werden speziell das Gebiet der Computersimulation sowie die verschiedenen Simulationsverfahren genauer beschrieben. Anschließend wird der Simulator des Instituts für Breitbandkommunikation beschrieben. Es werden die Konzepte des Simulators behandelt sowie einige grundlegende Module, auf die das Framework aufbaut.

In Kapitel 4 wird der IEEE802.11 Standard beschrieben, wobei besonders auf die Teile des Standards, die im Framework verwendet werden, eingegangen wird.

Kapitel 5 behandelt verschiedene Konzepte, die zur Simulation von mobilen Ad-Hoc Netzen benötigt werden. Es werden etwa verschiedene Radio- und Bewegungsmodelle vorgestellt, der Realismus der verschiedenen Modelle diskutiert, sowie die Ansätze des Frameworks mit denen anderer Mobility-Frameworks verglichen.

Schließlich wird in Kapitel 6 das eigentliche Framework erörtert. Die einzelnen Module, deren Schnittstellen sowie deren Zusammenhang werden beschrieben.

Kapitel 7 beschreibt noch den didaktischen Nutzen des Frameworks, bevor die Arbeit mit einem Ausblick auf mögliche Weiterentwicklungen sowie einer Zusammenfassung in Kapitel 8 schließt.

Kapitel 2

Problemstellung

Soll ein mobiles Ad-Hoc Netz simuliert werden, stellt sich anfangs die Frage, welche Funktionalität ein Simulator beziehungsweise ein dazu gehöriges Framework besitzen muss. Dafür muss allerdings die Frage geklärt werden, was genau ein mobiles Ad-Hoc Netz ist und welche Eigenheiten es besitzt. Anschließend muss geklärt werden, was genau simuliert werden soll. Nachdem diese beiden Fragen im Weiteren beantwortet werden, werden die Anforderungen an das Framework formuliert, wobei auch darauf eingegangen wird, welche Funktionen nicht unbedingt unterstützt werden müssen.

2.1 Mobile Ad-Hoc Netze

Ein mobiles Ad-Hoc Netz besteht aus Plattformen, auch Knoten genannt, die sich frei bewegen können. Diese besitzen mindestens ein Interface zur drahtlosen Kommunikation, einen Router und eventuell einen oder mehrere Hosts wobei all diese Teile auch in einem Gerät kombiniert sein können [Macker98].

Ein solcher Knoten in einem mobilen Ad-Hoc Netz kann vieles sein. Angefangen von kleinen mobilen Geräten wie Handys, Laptops oder PDAs über Roboter, bis hin zu Autos, Flugzeugen und Schiffen. Wichtig ist allein die Tatsache, dass die Knoten sich bewegen können, eine drahtlose Kommunikationseinrichtung besitzen und als Router fungieren können.

Abbildung 2.1 stellt ein solches Netz dar. In dieser Darstellung sind die Knoten rund eingezeichnet, ihre möglichen Kommunikationsverbindungen sind durch jeweils eine Linie markiert. Wie in der Abbildung zu erkennen ist, besitzt nicht jeder Knoten eine direkte Verbindung zu jedem anderen Knoten des Netzes. Da allerdings jeder Knoten als Router fungieren kann, ist es

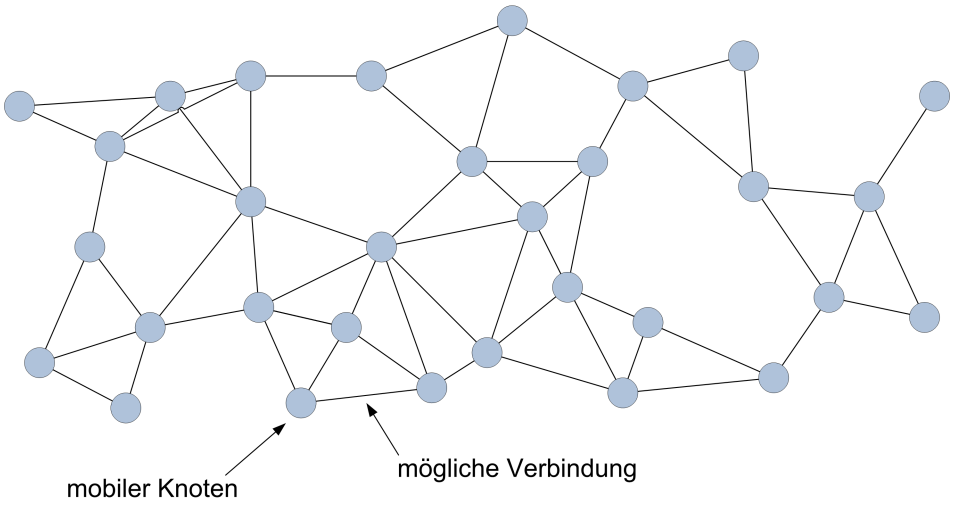

Abbildung 2.1: Darstellung eines mobilen Ad-Hoc Netzes

möglich, dass die Knoten indirekt miteinander kommunizieren. Wichtig ist dabei, dass in der Abbildung nur eine Momentaufnahme von einem solchen Netz dargestellt ist. Da die Knoten mobil sind, ändern sich die möglichen Verbindungen laufend.

Aufgrund der sich ständig ändernden Topologie stellt das Routing in einem solchen Netz eine Herausforderung dar. In einem solchen System müssen Routingprotokolle nicht nur schnell auf Änderungen in der Topologie reagieren, sondern auch auf andere Faktoren wie Übertragungsengpässe oder auch den Energieverbrauch Rücksicht nehmen.

Für die verschiedensten Szenarien wurden Routingprotokolle entwickelt. Abbildung 2.2 zeigt eine Übersicht über die wichtigsten Kategorien von Routingprotokollen nach [Hong02].

In Systemen mit flachem Routing sind alle Knoten gleichberechtigt. Zu den wichtigsten Vertretern dieser Kategorie zählen unter anderem "Dynamic source routing" [Johnson03], "Ad hoc on-demand distance routing" [Perkins03], "Optimized Link State Routing" [Clausen03] sowie "Topology dissemination based on reversed forwarding" [Ogier04]. Diese Protokolle werden auch teilweise in der einen oder anderen Form in den IEEE802.11s Standard eingehen [Perkins01]. Da man ein mobiles Ad-Hoc Netz als Spezialfall eines vermaschten Netzes sehen kann, ist es nicht verwunderlich, dass diese Protokolle teilweise auch in fix verbunden vermaschten Netzen zum

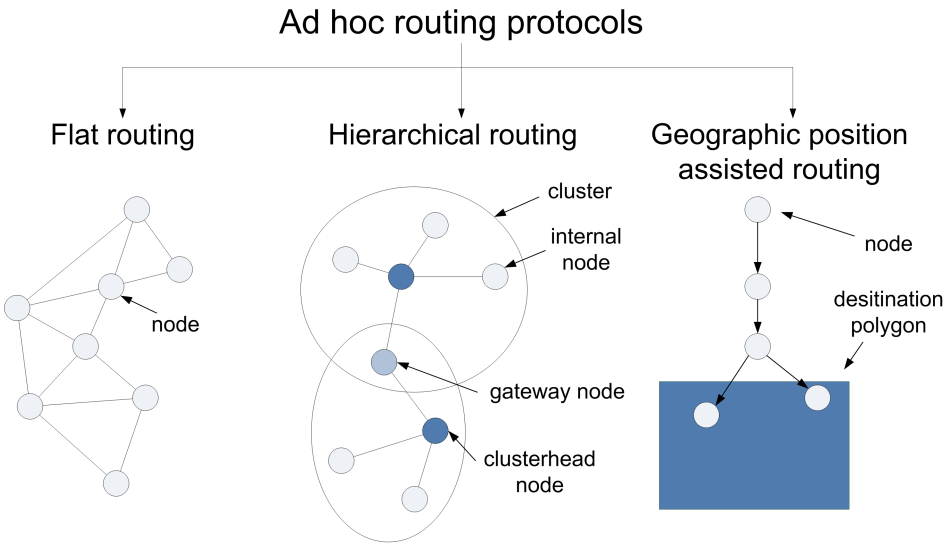

Abbildung 2.2: Übersicht über die verschiedenen Routing-Protokoll-Klassen

Einsatz kommen.

Im Gegenzug dazu werden in hierarchischen Systemen Gruppen gebildet, in denen einzelne Knoten spezielle Aufgaben übernehmen. In der Abbildung der hierarchischen Systems übernehmen etwa die Clusterheads und die Gatewaynodes spezielle Aufgaben. Ein Beispiel für ein solches Routingprotokoll wäre das "Zone-based hierachical link state routing protocol" [Haas01].

In der dritten Kategorie von Routingprotokollen wird vorausgesetzt, dass jeder Knoten seine aktuelle Position kennt. Dadurch ist es möglich, Gebiete zu adressieren, sodass Nachrichten an alle Knoten, die sich in dem Zielgebiet befinden, weitergeleitet werden. Sollen auch solche Protokolle simuliert werden, ist von Seiten des Frameworks insofern Rücksicht darauf zu nehmen, als dass auch die Position des Knotens in der Simulation durch das jeweilige Routingprotokoll abfragbar ist.

Neben dem Routing gibt es noch einige weitere Bereiche, die durch die Eigenheiten dieser Netze besonders herausfordernd sind. Diese wären unter anderem die automatische Konfiguration der Netze, Sicherheit, Service discovery sowie Quality of service support [Labiod08].

Mobile Ad-Hoc Netze können in vielen verschiedenen Bereichen zum Einsatz kommen. Beispielsweise könnten Autos mit dem entsprechenden Equipment

ausgestattet werden, sodass Informationen über den Verkehr ausgetauscht werden können. Mit Hilfe eines solchen Systems könnten Autos, die sich am Ende eines Staus befinden, nachfolgende Autos über den Stau informieren, sodass diese in Ruhe abbremsen können. Auf diese Weise könnte versucht werden, Auffahrunfälle zu verhindern. Ein anderes Beispiel wären Konferenzen, wo spontan Netze zur Unterstützung der Kommunikation unter den Teilnehmern aufgebaut werden könnten. Ein klassischer Einsatzbereich von mobilen Ad-Hoc Netzen ist auch der militärische Bereich. So werden solche Netze eingesetzt, um Panzertruppen untereinander zu koordinieren.

Auffallend bei den verschiedenen Szenarien ist vor allem das unterschiedliche Bewegungsverhalten. Während sich die Besucher einer Konferenz recht frei bewegen können, fahren die Autos meist auf Straßen, was ein anderes Bewegungsmuster zur Folge hat. Panzer bewegen sich eventuell als Gruppe voran, was wieder einem anderen Bewegungsmuster entspricht.

2.2 Anforderungen an das Framework

Ziel bei der Entwicklung des im Rahmen dieser Arbeit entstanden Frameworks war es, mobile Ad-Hoc Netze zu simulieren, die zur Kommunikation das IEEE802.11 Protokoll benutzen. Das Framework soll vor allem zur Simulation von Routingprotokollen und später auch zur Simulation von Diensten auf höherer Ebene verwendet werden. Daher gab es folgende Anforderungen an das Framework:

Es soll auf dem bereits vorhanden Simulator IBKSim, der in Kapitel 3.2 beschrieben wird, aufsetzen und eine möglichst große Anzahl an Knoten unterstützen. Es soll dabei die Bewegung der Knoten und die Funkübertragung auf der physikalischen Schicht simulieren, sowie die für die Simulation von Routingprotokollen relevanten Teile des MAC Protokolls implementieren. Diese wären die Distributed Coordination Function sowie der Request to Send / Clear to Send Mechanismus. Die Point Coordination Function wird nicht benötigt, ebenso wie die Fragmentierungsfunktion oder Sicherheitsfunktionen.

Die einzelnen mobilen Knoten sollen im Weiteren einzelnen Knoten im IBK -Sim entsprechen, in denen bereits zumindest ein IBK-Layer Objekt vorhanden ist, über welches die MAC Schicht angesprochen werden kann. Weiters soll es möglich sein, auf einfache Weise neue Bewegungsmodelle in das Framework einfügen zu können, um das Verhalten von Protokollen und Applikationen in speziellen Bewegungsszenarien untersuchen zu können.

Das entwickelte Framework entspricht dieser Aufgabenstellung. Durch Tren-

nung von Layer 1 und 2 lassen sich die einzelnen Teile leicht erweitern und wieder verwenden. So könnten etwa durch einfache Änderungen der Radiomodelle andere Funksysteme simuliert werden. Dadurch, dass die Signalstärke zwischen den einzelnen Knoten berechnet wird, ist es auch möglich, das Framework zur Simulation von Positionierungssystemen zu verwenden, die die Signalstärke verwenden, um den Abstand zu einem anderen Knoten zu berechnen. Da intern jeder Knoten seine genaue Position in der Umgebung kennt, wäre es auch möglich Knoten zu simulieren, die mit GPS ausgestattet sind. Dies ist vor allem dann besonders nützlich, wenn Routingprotokolle simuliert werden sollen, die mit Positionsunterstützung arbeiten.

Kapitel 3

Simulation

Es ist nicht immer sinnvoll oder möglich Analysen beziehungsweise Experimente an einem realen System durchzuführen. Die Gründe dafür sind vielfältig. So kann es sein, dass das zu untersuchende System noch nicht existiert oder dass sich ein System nicht direkt beobachten lässt, wie es zum Beispiel bei Molekülen der Fall ist. Auch können Versuche an einem realen System zu gefährlich oder aus anderen Gründen nicht vertretbar sein. Weiters ist es manchmal aus Kosten- und Zeitgründen nicht möglich, Fragestellungen direkt an einem System zu beantworten.

Eine Möglichkeit trotzdem Fragen über ein System zu beantworten, bieten Modelle. Unter einem Modell versteht man eine von der Wirklichkeit abstrahierte Beschreibung eines dynamischen Systems [Fishwick95]. Grundsätzlich kann zwischen physikalischen und abstrakten Modellen unterschieden werden. Bei einem physikalischen Modell handelt es sich meist um einen vereinfachten Nachbau. So werden beispielsweise in der Flugzeugindustrie bei der Entwicklung neuer Flugzeuge kleinere Modelle des Flugzeugs in Windkanaltests verwendet. Bei einem solchen Modell ist vor allem die Modellierung der Form wichtig, nicht aber der Innenausstattung. Im Gegensatz dazu wird in einem abstrakten Modell ein System durch Worte, mathematische Gleichungen oder auf andere abstrakte Weise beschrieben [Bennett95].

Diese Modelle können nun verwendet werden, um Versuche an ihnen durchzuführen wobei die Ergebnisse dieser Versuche teilweise, unter anderem auch abhängig von der Qualität des verwendeten Modells, Rückschlüsse auf das ursprüngliche System zulassen. Die Entwicklung von Modellen, die Verwendung der Modelle und schließlich die Auswertung der Ergebnisse wird als Simulation bezeichnet [Fishwick95].

Ein Teilgebiet der Simulation ist die Computersimulation. Systeme werden dabei in abstrakte Modelle in der Form von Algorithmen umgesetzt und können auf Computern ausgeführt werden. Da im Rahmen dieser Diplomarbeit Computermodelle zur Simulation von mobilen Ad-Hoc Netzen erstellt wurden, wird im Weiteren die Computersimulation noch genauer behandelt.

3.1 Computersimulation

Je nach System und zu untersuchender Eigenschaft werden unterschiedliche Simulationsmodelle benötigt, die wiederum teilweise verschiedene Simulationsverfahren benötigen. Diese Verfahren lassen sich je nach ihrem Verhältnis zur Zeit in unterschiedliche Kategorien einteilen. Eine Übersicht der verschiedenen Kategorien ist in Abbildung 3.1 zu sehen.

Grundsätzlich kann zwischen analogen und digitalen Verfahren unterschieden werden, wobei man auch von analogen beziehungsweise digitalen Computersimulationen spricht. Analoge Verfahren werden zur Simulation von Systemen eingesetzt, die ein kontinuierliches Fortschreiten der Zeit benötigen. Daher werden analoge Verfahren auch als zeitkontinuierliche Computersimulation bezeichnet. Wie der Name schon verrät, eignen sich analoge Computersimulationen zur Simulation von analogen Vorgängen. Sie werden daher etwa bei Strömungsuntersuchungen oder bei mechanischen Systemen verwendet.

Im Gegensatz dazu stehen die digitalen oder auch zeitdiskreten Verfahren. Dabei wird das Simulationsmodel nur zu diskreten Zeitpunkten betrachtet.

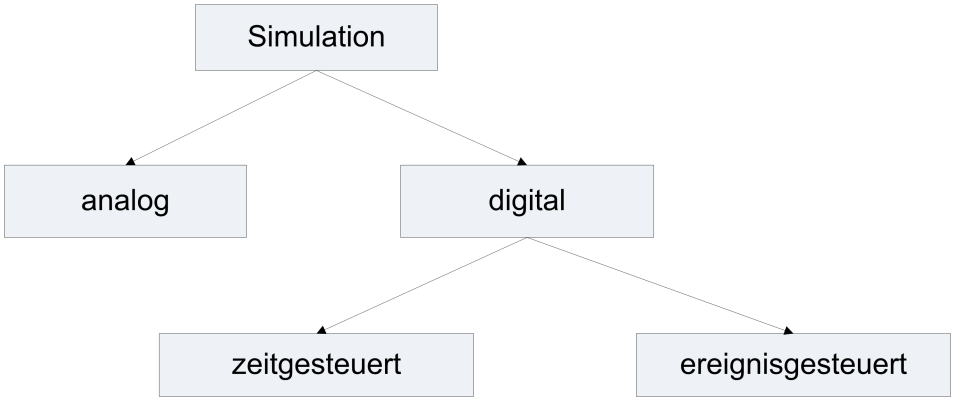

Abbildung 3.1: Simulatiosnverfahren nach Zeitsteuerung nach [Seibold02]

Diese Verfahren kommen bei der Simulation von digitalen Systemen zum Einsatz, wie das bei digitalen Kommunikationsnetzen der Fall ist.

Die digitalen Verfahren kann man wiederum in zwei Unterkategorien teilen. Je nachdem, wie die diskreten Zeitpunkte ausgewählt werden, unterscheidet man zwischen zeitgesteuerten und eventgesteuerten Verfahren.

In zeitgesteuerten Simulationen wird das Model in regelmäßigen Zeitabständen betrachtet. Zu diesen Zeitpunkten wird geprüft, ob seit dem letzten Zeitpunkt ein Ereignis aufgetreten ist. Ist dies der Fall, wird der Zustand des Modells entsprechend geändert. Wichtig in diesem Zusammenhang ist, dass ein Ereignis, welches zwischen zwei Zeitpunkten stattgefunden hat, erst beim nachfolgenden Zeitpunkt registriert wird [Sauerbier99]. Solche zeitgesteuerten Verfahren eignen sich vor allem zur Simulation von getakteten Systemen, da hier Ereignisse auf Grund ihrer Beschaffenheit nur zu diskreten Zeitpunkten auftreten. Tritt in einem System selten ein Ereignis auf, so ist dieses Simulationsverfahren nicht sonderlich effizient, weil auch wenn kein Ereignis aufgetreten ist, regelmäßig geprüft wird, ob das Model einen anderen Zustand einnehmen muss. In diesem Fall ist der Einsatz von einem eventgesteuerten System effizienter.

Ein eventgesteuerter Simulator überprüft nur beim Auftreten eines Events, ob ein Modell seinen Zustand verändern muss. Das Event wird also im Gegensatz zum zeitgesteuerten Verfahren sofort betrachtet. Der Simulator verwaltet dafür eine Eventliste, in der die Ereignisse nach ihrem zeitlichen Auftreten gereiht sind. Der eigentliche Simulationsvorgang besteht aus einer Schleife, in der jedesmal das nächste Event aus der Eventliste entfernt und bearbeitet wird. Dabei ändert sich der Zustand des Models und es werden neue Ereignisse generiert, die wieder in die Eventliste eingefügt werden. Durch die Zeitstempel der Ereignisse kann immer die momentane Simulationszeit berechnet werden. Die Simulation endet schließlich, sobald entweder ein gewisser Zeitpunkt überschritten wurde oder die Eventliste leer ist, da es dann zu keiner weiteren Veränderung des Modells mehr kommen kann [Seibold02].

Der im folgenden Abschnitt beschriebene Simulator des Instituts für Breitbandkommunikation ist ein solcher diskreter eventbasierender Simulator. Bei der Beschreibung des Simulationskerns in Abschnitt 3.2.1 findet sich das soeben beschriebene Konzept wieder.

3.2 IBKSim

Der Simulator des Instituts für Breitbandkommunikation ist wie bereits erwäht ein diskreter eventbasierender Simulator. Die erste Version wurde 2005 geschrieben, und im Jahr 2007 komplett überarbeitet.

Der Aufbau des Simulators ist in Abbildung 3.2 dargestellt. Er besteht aus einem kleinen generischen Kern, der die eigentliche Simulation übernimmt. Auf diesem Kern aufbauend gibt es eine Reihe von Klassen, die den Simulator um viele häufig benötigte Funktionen erweitern. Diese sind in Abbildung 3.2 als "IBKSIM Erweiterungen" dargestellt. Gerade diese Menge von Klassen macht den Simulator zu einem Netzwerksimulator, da viele dieser Klassen für Netze typische Konzepte und Funktionen implementieren.

Das Konzept, einen generischen Kern zu verwenden und die spezifischen Funktionen nicht direkt im Kern einzubetten, bietet einige Vorteile. So kann der Simulator auf einfache Weise erweitert werden, ohne dass der Kern geändert werden muss. Es können sowohl ein Teil der Klassen verwendet werden, die die eigentlich netzwerkspezifischen Funktionen beinhalten, als auch weitere direkt auf dem Kern aufbauende Klassen geschrieben werden. Von dieser Möglichkeit wurde bei der Entwicklung des in dieser Arbeit beschriebenen Frameworks Gebrauch gemacht. Es werden zwar einige der Erweiterungen verwendet, allerdings konnten einige Funktionen nicht auf diesen Erweiterungen aufgebaut werden. So konnte etwa das Konzept, dass einzelne Knoten fix miteinander verbunden sind, nicht weiter verwendet werden,

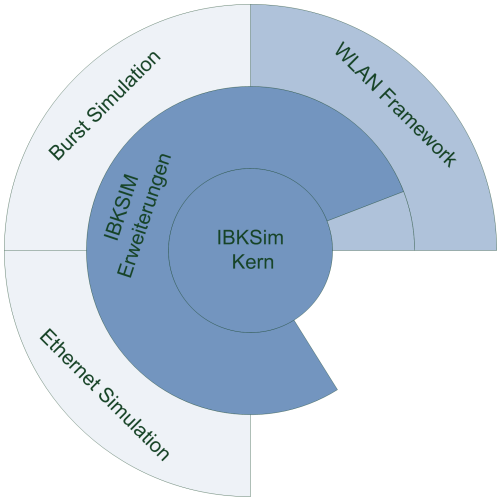

Abbildung 3.2: Aufbau des IBKSims

stattdessen wurde ein System eingeführt, welches dynamische Verbindungen zulässt. Daher reicht in Abbildung 3.2 das WLAN-Framework teilweise bis zum Kern. Andere bereits auf dem IBKSim aufbauende Frameworks hingegen, wie das zur Burst-Simulation in optischen Netzen, bauen nur auf den Erweiterungen des Kerns auf.

3.2.1 Simulationskern

Der Simulationskern ist in Hinsicht auf Aufbau und Benennung an [Hoff95] angelehnt. Es bietet sich an, anhand der Klassen *simEvent*, *simEntity*, *simControl* und *simQueue* die Funktionsweise der Simulationskerns zu erörtern.

Jedes Simulationsobjekt ist ein Objekt der Klasse *simEntity* oder im Weiteren ein Objekt einer von *simEntity* abgeleiteten Klasse. Die Klasse *simEntity* besitzt die Funktion *handleEvent()*, die die eigentliche Funktionalität des Objekts beinhaltet. Wenn ein Ereignis auftritt, wird diese Funktion vom simControl aufgerufen.

Objekte des Typs *simEvent* sind Ereignisse. Diese Objekte beinhalten unteranderem einen Eintrittszeitpunkt, einen Zeiger auf das Quellobjekt und einen auf das Zielobjekt. Sie werden von Simulationsobjekten erzeugt und an den Scheduler übergeben.

Die *simQueue* ist eine Queue die die einzelnen Ereignisse speichert. Diese wird vom Scheduler benötigt.

Die Klasse *simControl* implementiert den Scheduler. Jede Simulation besitzt einen solchen Scheduler, der die zentrale Komponente der Simulation ist. Er verwaltet die Simulationszeit und die Events. Simulationsobjekte erzeugen Events, die sie mit einem Zeiger auf sich selbst, einen Zeiger auf das Zielobjekt und einem Eintrittszeitpunkt versehen. Anschließend übergeben sie das Eventobjekt an den Scheduler. Sobald dieser ein Event erhält, überprüft er ob der Eintrittszeitpunkt den Wert Null besitzt, was bedeutet, dass das Event sofort eintritt. Ist dies der Fall, ruft er die *handleEvent*-Routine des Zielobjekts auf und übergibt diesem das Event-Objekt. Falls der Eintrittszeitpunkt größer als Null ist, bedeutet das, dass das Event in der Zukunft liegt. Der Scheduler ordnet das Event abhängig vom Zeitstempel in die Eventliste ein. Ein negativer Eintrittszeitpunkt ist nicht möglich, da dies zu Kausalitätsverletzungen in der Simulation führen würde. Ansonsten arbeitet der Scheduler die Events in der Liste nacheinander ab, und erhöht abhängig von den Eintrittszeitpunkten der Events die Simulationszeit.

Die Simulation läuft nun folgendermaßen ab: Beim Start des Programms

wird zuerst der Scheduler mit der *simQueue* erzeugt. Anschließend werden die Simulationsobjekte erzeugt, was über eine spezielle *build*-Funktion geschieht. Diese Funktion ruft gegebenenfalls nach Erzeugung des Objekts die *init*-Funktion desselbigen auf. Das Objekt hat damit die Möglichkeit, noch bevor die eigentliche Simulation startet, Events an den Scheduler zu übergeben. Diese Events benötigen allerdings einen Eintrittszeitpunkt größer Null, damit sichergestellt ist, dass alle anderen Simulationsobjekte erzeugt sind, bevor das Event ausgeführt wird. Der Scheduler ordnet ein solches Event in die Eventliste ein.

Wenn alle Simulationsobjekte erstellt wurden, wird die *loop*-Funktion des Schedulers aufgerufen. Dieser überprüft, ob sich ein Event in der SimQueue befindet. Ist dies nicht der Fall bedeutet das, dass die Simulation bereits wieder beendet ist. Der Grund dafür ist der, dass alle Simulationsobjekte auf ein Ereignis warten und diese erst bei Eintreffen eines Ereignisses neue Ereignisse erzeugen. Da aber kein Ereignis stattfinden wird, weil sich ja keines in der SimQueue befindet, können die Simulationsobjekte nie neue Ereigisse erzeugen. Befindet sich zumindest ein Event in der SimQueue, entfernt der Scheduler das Event aus der Queue, welches am nähesten in der Zukunft liegt. Er erhöht die Simulationszeit entsprechend und ruft die *handleEvent*-Funktion des Ziels des Events auf. Das so angesprochene Simulationsobjekt führt die Funktion aus und übergibt gegebenenfalls weitere Events an den Scheduler. Dieser führt sie wie bereits erwähnt entweder sofort aus, oder ordnet sie in die SimQueue ein. Wenn das Simulationsobjekt die Funktion ausgeführt hat, hat wieder der Scheduler die Kontrolle. Dieser entnimmt das nächste Event aus der Queue und setzt somit die Simulation fort. Das Ende der Simulation ist erreicht, wenn entweder die Queue leer ist oder wenn eine gewisse Simulationszeit erreicht wurde.

Damit ist die Grundfunktionalität für einen allgemeinen eventbasierenden Simulator gegeben. Um diesen Simulator effizient zur Simulation von Netzen zu verwenden, fehlt es allerdings noch an Funktionalität. Darauf wird im nächsten Abschnitt genauer eingegangen.

3.2.2 Konzepte des IBKSims zur Simulation von Netzen

Dass der IBKSim ursprünglich zur Simulation von fixen Netzen geschrieben wurde, zeigt sich an dem grundsätzlichen Konzept. Mit Hilfe eines XML-Files wird das Netz spezifiziert, welches simuliert werden soll. Abbildung 3.3 zeigt ein solches Netz. Das Netz besteht aus einer Reihe von Knoten, die über Links fix miteinander verbunden sind. Jeder dieser Knoten besteht aus einer Reihe von Simulationsobjekten, die von der Klasse *simEntity* abgelei-

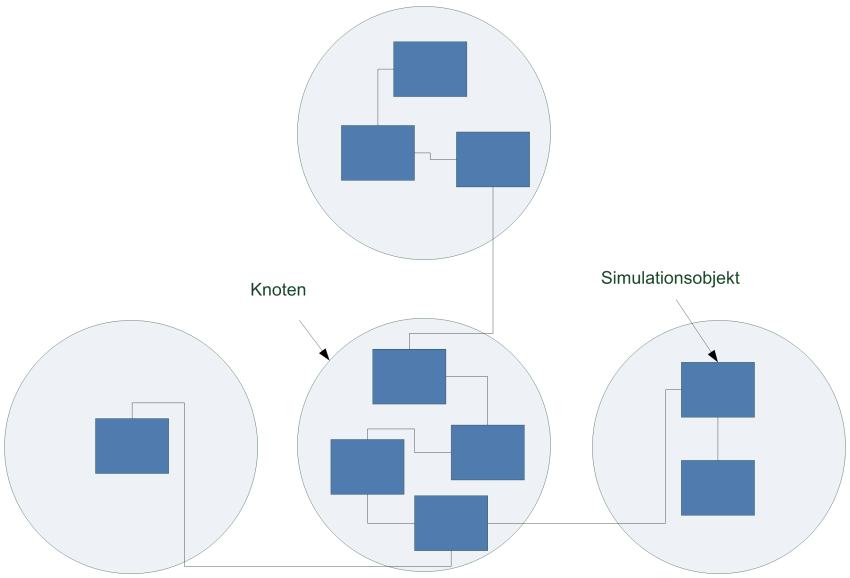

Abbildung 3.3: Netz-Konzept des IBKSims. Ein Netz besteht aus Knoten, die wiederum aus den eigentlichen Simulationsobjekten bestehen.

tet sind. Diese Simulationsobjekte sind ebenfalls untereinander verbunden. Das Verbinden der Knoten erfolgt durch das Verbinden zweier Simulationsobjekte, die sich in verschiedenen Knoten befinden, sofern kein Delay oder auch Paketverlust simuliert werden soll. Falls dies aber gewünscht wird, kann ein zusätzliches Simulationsobjekt in die Verbindung eingefügt werden, welches die Leitung simuliert.

Um dieses Konzept umzusetzen, wurde die Klasse *simEntity* um einige Funktionen erweitert. Abbildung 3.4 zeigt das UML Diagramm der Klasse *simEntity*. Neben Konstruktor und Destruktor besitzt sie die bereits erwähnte *handleEvent*-Funktion, die vom Scheduler aufgerufen wird, falls ein Event für dieses Simulationsobjekt vorliegt. Die *connect*-Funktion dient zum Erstellen fixer Verbindungen mit anderen Simulationsobjekten während der Initialisierung der Simulation. Diese Funktion musste allerdings beim Erstellen des Frameworks umgangen werden, da sich in einem mobilen Ad-Hoc Netz die Verbindungen laufend ändern können.

Weiters besitzt die Klasse *simEntity* sowie alle ihre Unterklassen die Funktion *getLogs*. Diese Funktion wird zum Sammeln von statistischen Daten benötigt. Die Logging-Funktionalität des IBKSims ist durch ein einfaches Simulationsobjekt realisiert, welches regelmäßig seine *handleEvent*-Funktion durch Senden von Ereignissen an sich selbst ausführt. In dem XML-File, in

dem das Netz konfiguriert wird, kann bei diesem Simulationsobjekt angegeben werden, von welchen anderen Simulationsobjekten es Daten sammeln soll. Wird die *handleEvent*-Routine des Log-Objekts aufgerufen, ruft es von allen Simulationsobjekten, von denen Daten gesammelt werden sollen, die *getLogs()*-Funktion auf. Es erhält als Antwort jeweils einen String mit den Log-Daten, die dann zum Beispiel in ein File geschrieben werden.

Die *simEntity*-Klasse besitzt noch einige weitere Funktionen, die allerdings für das Verständnis des Simulators nicht wichtig sind. Daher werden sie auch nicht weiter behandelt.

Eine wichtige Subklasse der Klasse *simEntity* ist der *ibkLayer* (siehe Abbildung 3.4). Häufig kommt es vor, dass die einzelnen Simulationsobjekte in einem Knoten unterschiedliche Schichten repräsentieren, sie also aufeinander aufsetzten. Diese Klasse besitzt eine ausprogrammierte *connect* -Funktion, sodass der darüber und darunterliegende Layer standardmäßig über den Zeiger *upwards* beziehungsweise *downwards* erreichbar ist. Weiters besitzt diese Klasse eine statische *build*-Funktion. Diese wird dazu verwendet, ein neues Objekt von diesem Typ zu erzeugen. Hat die *build*-Funktion ein Objekt erstellt, kann sie die *init*-Funktion von diesem aufrufen. Diese kann wie bereits erwähnt dafür verwendet werden, vor dem Start der Simulation bereits Events an den Scheduler zu übergeben.

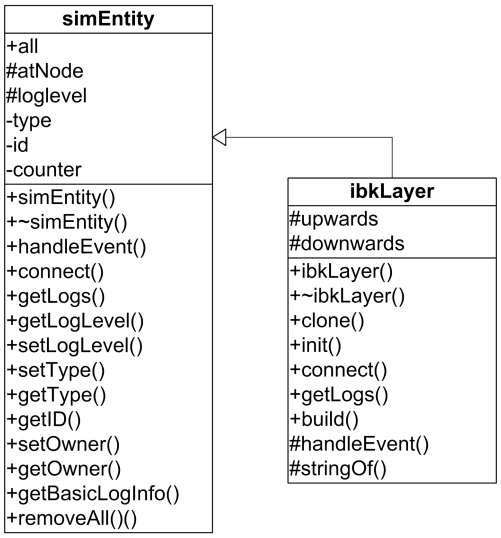

Abbildung 3.4: UML-Diagram der Klasse *simEntity* sowie der davon abgeleiteten Klasse ibkLayer

Bei der Konzeption des IBKSim wurde davon ausgegangen, dass der Austausch von Nachrichten zwischen verschiedenen Simulationsobjekten ein häufiges Ereignis ist. Daher wurde die Klasse *packet* erstellt. SimEvents beinhalten einen Pointer der auf ein Paket zeigen kann. So ist es möglich, Pakete zwischen den einzelnen Simulationsobjekten auszutauschen. Ein solches Paket enthält neben Informationen wie Absender, Ziel, Größe, Priorität, Alter auch wiederum einen Pointer auf ein weiteres Paket. So ist es möglich, weitere Pakete in das Paket einzubetten. An dieser Stelle sei auch erwähnt, dass die Paketklasse auch dafür verwendet wird, Frames zu implementieren. Wenn daher im Zusammenhang mit Frames von Paketen gesprochen wird, ist ein Objekt der Klasse *packet* gemeint und nicht ein Paket im Sinne eines Datenpakets.

3.2.3 Weitere Funktionalität

Neben den bereits erwähnten Klassen und Funktionen gibt es noch eine Vielzahl weiterer, deren genauer Aufbau allerdings für das Verständnis des Frameworks nicht weiter interessant ist. Trotzdem macht es Sinn die wichtigsten kurz zu erwähnen, da sie entweder direkt vom Framework verwendet werden oder bei möglichen Erweiterungen des Frameworks von Nutzen sein könnten.

Häufig kommt es vor, dass etwa die Zeit zwischen dem Auftreten zweier Ereignisse als statistische Verteilung modelliert wird. Ein anderes Beispiel wäre die Größe von Paketen, die man auch mit Hilfe einer Verteilung beschreiben kann. Daher gibt es eine Reihe von Klassen, die Zufallszahlen nach verschiedenen Verteilungen erzeugen.

Warteschlangen sind weitere Elemente, die häufig in Simulationen von Netzen benötigt werden. Auch hier bietet der IBKSim eine Auswahl an Warteschlangen an. Ebenso gibt es auch Simulationsobjekte, die als Server wieder Elemente aus den Warteschlangen entfernen und weiterleiten.

Viele andere Elemente wie etwa Splitter, Datenquellen und Senken können als Ausgangsbasis zur Erzeugung eigener Simulationen verwendet werden. Durch Anpassung der vorhanden Klassen lassen sich so gegebenenfalls rasch eigene Simulationsobjekte erstellen.

Erwähnenswert ist noch, dass der IBKSim bewusst keine Funktionen zur statistischen Auswertung der Daten besitzt. Der Grund dafür liegt darin, dass die *log*-Funktionen aller Simulationsobjekte so geschrieben sind, dass das Logfile, welches während einer Simulation erstellt wird, die Daten im XML-Format enthält. Daten in diesem Format lassen sich einfach

mit Programmen wie Excel oder auch mit Statistikprogrammen wie GNU R bearbeiten. Daher scheint es wenig sinnvoll, Statistikfunktionen in den Simulator einzubauen, wenn das Ergebnis problemlos mit mächtigen Statistikprogrammen ausgewertet werden kann.

3.2.4 Konfiguration

Damit eine Simulation überhaupt ablaufen kann, müssen zuerst einmal die einzelnen Simulationsobjekte erstellt und konfiguriert werden. Mit Hilfe eines XML Files kann ein zu simulierendes Netz beschrieben werden. So ein File könnte etwa so wie in Listing 3.1aussehen.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<simulation xsi:schemaLocation="http://www.ibk.tuwien.ac.at
  http://jupiter.ibk.tuwien.ac.at/IBKSimulator.xsd"
  xmlns="http://www.ibk.tuwien.ac.at"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <paramlist>
    <duration value="10000.0"/>
    <initialseed value = "2"/>
  </paramlist>

  <multinetwork>
    <network name="net1">
      <!-- source 0 is a node which creates traffic -->
      <node name="source">
        <component name="source0" type="source">
          <source prio="0">
            <timing start="0.0" stop="100000.0"/>
            <distribution purpose="pcksize" type="deterministic">
              <distconf value="1"/>
            </distribution>
            <distribution purpose="genrate" type="negexp">
              <distconf mean="95"/>
            </distribution>
          </source>
        </component>
      </node>

      <!--switch 2 -->
      <node name="mm1">
        <component name="queue0" type="fifoQueue">
          <size>100</size>
        </component>
        <component name="server0" type="singleServer">
          <distribution type="negexp">
            <distconf mean="100"/>
          </distribution>
        </component>
        <!-- connect all queues to the dequeue0 -->
        <clink src="queue0" dst="server0" direction="backward"
```

```
          port="inPort"/>
        <clink src="queue0" dst="server0" direction="forward"
          port="outPort"/>
      </node>

      <!-- sink 0 will accept all traffic -->
      <node name="sink">
        <component name="sink0" type="sink">
        </component>
      </node>

      <!-- sink drop will accept all traffic -->
      <node name="drop">
        <component name="sink0" type="sink">
        </component>
      </node>
    </network>
  </multinetwork>

  <linklist>
    <!--connect source with first switch-->
    <link name="src-queue" direction="forward" port="outPort">
      <src net="net1" node="source" cmp="source0"/>
      <dst net="net1" node="mm1" cmp="queue0"/>
    </link>
    <!--connect switch to default sink  (success)-->
    <link name="server-snk" direction="forward" port="outPort">
      <src net="net1" node="mm1" cmp="server0"/>
      <dst net="net1" node="sink" cmp="sink0"/>
    </link>
    <!--connect backup switch to default sink  (success)-->
    <link name="bS-snk" direction="forward" port="dropPort">
      <src net="net1" node="mm1" cmp="queue0"/>
      <dst net="net1" node="drop" cmp="sink0"/>
    </link>
  </linklist>


  <loglist>
    <logging type="file" interval="100">
      <config filename="logfile.xml"/>
      <lognetwork name="net1" v="1">
        <lognode name="source" v="2" displayNode="1">
          <logcmp name="source0" v="1"/>
        </lognode>
        <lognode name="drop" v="2" displayNode="1">
          <logcmp name="sink0" v="1"/>
        </lognode>
        <lognode name="sink" v="2" displayNode="1">
          <logcmp name="sink0" v="1"/>
        </lognode>
        <lognode name="mm1" v="2" displayNode="1">
          <logcmp name="queue0" v="1"/>
          <logcmp name="server0" v="1"/>
        </lognode>
      </lognetwork>
    </logging>
  </loglist>
```

```
</simulation>
```

Listing 3.1: Beispiel eines Konfigurationsfiles für den IBKSim

Dieses Konfigurationsfile erlaubt die Spezifikation von Netzen, die aus einzelnen Knoten bestehen. Ein Knoten besteht aus mindestens einer Komponente, bei der es sich um eine echtes Simulationsobjekt handelt. Die einzelnen Komponenten werden mit Links verbunden. So kann spezifiziert werden, welche Komponente welcher anderen Komponente Events und dadurch auch Pakete schicken kann. Weiters wird in diesem File spezifiziert von welchen Simulationsobjekten Loginformationen gesammelt werden. Ebenso werden Parameter wie zum Beispiel die Simulationsdauer gesetzt.

In Listing 3.1 wird ein einfaches Netzwerk mit vier Knoten spezifiziert. Nach dem Einlesen des Files überprüft der Simulator die Gültigkeit des Files, wofür die Informationen in Zeile 1 bis 4 benötigt werden.

Anschließend werden die Parameter eingelesen und über eine *build*-Funktion ein neues Multinetwork erzeugt. Der *build*-Funktion wird dabei der komplette Inhalt zwischen <multinetwork> und </multinetwork> übergeben.

In der *build*-Funktion des Multinetworks werden die einzelnen Knoten wiederum über die *build*-Funktion der Klasse node erzeugt wobei deim Aufruf der Funktion der Inhalt zwischen <node> und </node> an diese übergeben wird.

Nun werden durch diese Funktion die eigentlichen Simulationsobjekte erzeugt. Diese besitzen konsistenterweise ebenfalls eine *build*-Funktion, der auch wieder den Inhalt zwischen den "component"-Tags übergeben wird.

Durch dieses System, das auch als Fabrikmethode oder Factory method pattern [Gamma95] bekannt ist, können auf einfache Weise über das Konfigurationsskript Parameter an die Simulationsobjekte übergeben werden. Der große Vorteil liegt vor allem darin, dass beim Erstellen neuer Klassen von Simulationsobjekten die anderen Programmteile kaum geändert werden müssen. Das liegt daran, da die Konfigurationsdaten, die beim Erzeugen des Objekts benötigt werden, nur an die build-Funktion der jeweiligen Klasse übergeben werden müssen. Für die restlichen Programmteile ist es daher irrelevant, was die klassenspezifische Parameter überhaupt bedeuten.

Die Node-Komponente verlinkt nun die erzeugten und bereits konfigurierten Simulationsobjekte. Als nächstes wird die Linkliste abgearbeitet und somit auch die einzelnen Knoten untereinander verlinkt. Dabei werden jeweils bestimmte Simulationsobjekte, die sich in unterschiedlichen Knoten befinden,

miteinander verbunden.

Auf diese Weise werden aus dem Konfigurationsfile das in Listing 3.1 angeführt ist, vier Knoten erzeugt. Der Knoten mit der Bezeichnung "source" besteht aus einer Komponente vom Typ *source*. Diese Klasse, bei der es sich um eine von *simEntity* angeleitete Klasse handelt, erzeugt Events, die Pakete beinhalten. In dieser Simulation werden Pakete mit der Priorität Null erzeugt. Pakete werden nur im Zeitintervall von 0 bis 10000 erzeugt. Ihre Größe beträgt konstant Eins und ihre Erzeugungsrate ist negativexponentiell verteilt mit einem Mittelwert von 95.

Der zweite Knoten besteht aus zwei Komponenten, einer Warteschlange die nach dem First-in-First-out Prinzip arbeitet und maximal 100 Pakete aufnehmen kann sowie einem Server, der die Pakete aus der Warteschlange entfernt und weitersendet. Die Zeit, die vergeht, bis ein Paket weitergeleitet wird, ist wiederum negativexponentiell verteilt mit einem Mittelwert von 100.

Die beiden anderen Knoten beinhalten jeweils eine Komponente des Typs Sink. Diese Komponente zerstört die Pakete und sammelt Informationen wie etwa, die Anzahl der Pakete, deren Größe und so weiter.

Durch die Linkliste sind die Knoten so verbunden, dass die Source die erzeugten Pakete an die Queue liefert. Wenn diese überläuft, leitet sie die Pakete an den Knoten "drop" weiter. Ansonsten werden die Pakete durch die Server-Komponente entnommen, der sie an die Sink weiterleitet.

Ist das Netz erzeugt, wird auch ein spezielles Simulationsobjekt zum Loggen erstellt. Die angegebenen Parameter bedeuten in diesem Fall, dass dieser Knoten die gesammelten Informationen in ein File schreiben soll, welches den Namen logfile.xml trägt. Diese Komponente ruft alle 100 Zeiteinheiten die Daten von allen bereits erzeugten Komponenten ab.

Nachdem alle Komponenten erstellt worden sind, startet die eigentliche Simulation wenn, wie bereits beschrieben, zumindest eine der Komponenten bei der Erzeugung ein Event an den Scheduler übergeben hat. Wie dem Code der Klasse *source* zu entnehmen ist, hat diese bereits bei ihrer Erzeugung das erste Event generiert.

Sobald die Simulation beendet ist, können die Resultate, die sich im logfile.xml befinden, ausgewertet werden. Außerdem ist es sinnvoll, die Simulation einige weitere Male mit anderen Seed-Werten für die Zufallsgeneratoren durchzuführen.

Kapitel 4

IEEE 802.11

Der IEEE802.11 Standard beschäftigt sich wie jeder Standard der 802 Familie mit lokalen Netzen und spezifiziert Layer 1 und 2 des OSI Schichtmodels [OSI84] für ein Verfahren zur dratlosen Kommunikation. Im Standard selbst wird das Ziel des selbigen wie folgt beschrieben: "The purpose of this standard is to provide wireless connectivity to automatic machinery, equipment, or stations that require rapid deployment, which may be portable or hand-held, or which may be mounted on moving vehicles within a local area." [IEE99b] Es handelt sich also um einen Standard zur drahtlosen Kommunikation.

Inzwischen gibt es ein Reihe von Erweiterungen zum ursprünglichen Standard. Diese definieren etwa andere Codierungs- und Übertragungsverfahren ,um eine höhere Übertragungsrate zu erreichen. Dies ist etwa bei der 802.11b [IEE99a] und 802.11g [IEE03] Erweiterung der Fall. Andere wiederum liefern Zusätze für bessere Sicherheit (802.11i [IEE04]) oder Quality of Service Unterstützung (802.11e [IEE05]).

Eine genaue Beschreibung der 802.11 Familie würde den Umfang dieser Arbeit sprengen. Daher wird im Weitern nur auf die für das Framework relevanten Teile genauer eingegangen. So werden in der MAC-Schicht vor allem die Zugriffsprinzipien betrachtet. Auf andere Teile des Standards wie etwa Sicherheitssysteme oder die Unterstützung von Infrastruktur, wird bewusst nicht eingegangen.

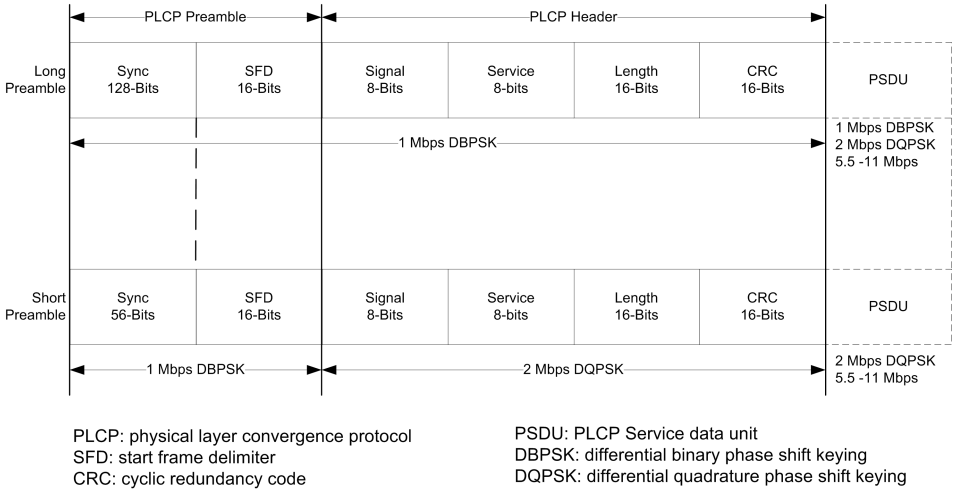

Abbildung 4.1: DSSS Frame Format

4.1 Physikalische Schicht

Der 802.11 Standard spezifiziert drei verschiedene Implementierungen der physikalischen Schicht [Prasad02]. Neben FHSS (frequency hopping spread spectrum) und DSSS (direct sequence spread spectrum) wird auch die physikalische Schicht zur Übertragung mittels Infrarot spezifiziert. Durch Erweiterungen kamen zusätzliche Implementierungen dazu, wie etwa mittels OFDM (orthogonal frequency division multiplexing) in der IEEE 802.11a Erweiterung.

Für die Simulation der physikalischen Übertragung mittels Funk sind nur wenige Teile der einzelnen Spezifikationen von Bedeutung. Gemeinsam ist all diesen Standards, dass der MAC-Frame um eine Präambel sowie einem zusätzlichen Header erweitert wird. Zur Simulation ist es lediglich notwendig zu wissen, mit welcher Geschwindigkeit und welcher Kodierung die einzelnen Teile des Frames übertragen werden sowie auf welcher Frequenz. Mit Hilfe der Geschwindigkeit kann die Übertragungsdauer berechnet werden, die Kodierung hat Einfluss auf die Bitfehlerwahrscheinlichkeit und die Frequenz auf die Berechnung der Signalstärke.

Exemplarisch wird die Implementierung der physikalischen Schicht nach dem IEEE 802.11b Standard genauer betrachtet. Wie in Abbildung 4.1 zu sehen, spezifiziert der Standard zwei Frame-Formate. Eines mit langer Präambel und eines mit kurzer. In der langen Form werden die Präambel und der Header mit einer Geschwindigkeit von 1Mbps übertragen. Nur die PSDU

Name	Beschreibung
SFD	der "start frame delimiter" markiert den Begin des Headers
Signal	gibt die Modulierung der PSDU an
Service	enthält verschiedene Informationen etwa zur Kodierung oder zum Signaltakt.
Length	gibt die Länge der PSDU in Mikrosekunden an
CRC	Prüfsumme, um Fehler im Header zu erkennen

Tabelle 4.1: Beschreibung der einzelnen Felder des DSSS Pakets

(physical layer convergence protocol service data unit), die den MAC-Frame enthält, wird je nach Signalstärke mit 1, 2, 5.5 oder 11 Mbps übertragen, wobei unterschiedliche Kodierungen zum Einsatz kommen. Die Übertragung des Headers und der Präambel entspricht dem ursprünglichem IEEE 802.11 DSSS PHY Standard.

Im kurzen Format wird nur die Präambel mit 1 Mbps übertragen, der Header hingegen mit 2Mbps und einer anderen Kodierung. Die PSDU kann mit 2, 5.5 oder 11 Mbps übertragen werden, wobei es auch hier wieder Unterschiede in der Kodierung gibt.

Auch wenn für die Simulation nur die PSDU von Bedeutung ist, ist in Tabelle 4.1 noch eine kurze Beschreibung der einzelnen Felder zu finden.

4.2 MAC-Schicht

Die Aufgabe der MAC-Schicht ist es, den Zugriff auf die physikalische Schicht zu kontrollieren. Im IEEE802.11 [IEE99b] Standard übernimmt sie zusätzlich eine Reihe weiterer Aufgaben. Diese umfassen etwa das Scannen nach anderen Netzen, Authentifizierung, Assoziation mit einem Access Point, Verschlüsselung, Roaming und einige weitere [Prasad02].

Für den Zugriff auf die physikalische Schicht definiert der Standard zwei Methoden. Zum Einen die so genannte "Distributed Coordinator Function", die verpflichtend in Geräte, die den Standard unterstützen sollen, implementiert werden muss. Diese verteilte Zugriffsmethode wird im Weiteren noch genauer behandelt. Zum Anderen wird die "Point Coordination Function" beschrieben, die allerdings optional ist. Sie wird von Access Points eingesetzt, um den Zugriff der Knoten zu koordinieren. Der Vollständigkeit halber sei noch erwähnt, dass die IEEE 802.11e Erweiterung eine weitere Zugriffsmethode, die "Hybrid Coordination Function", definiert, die aber für

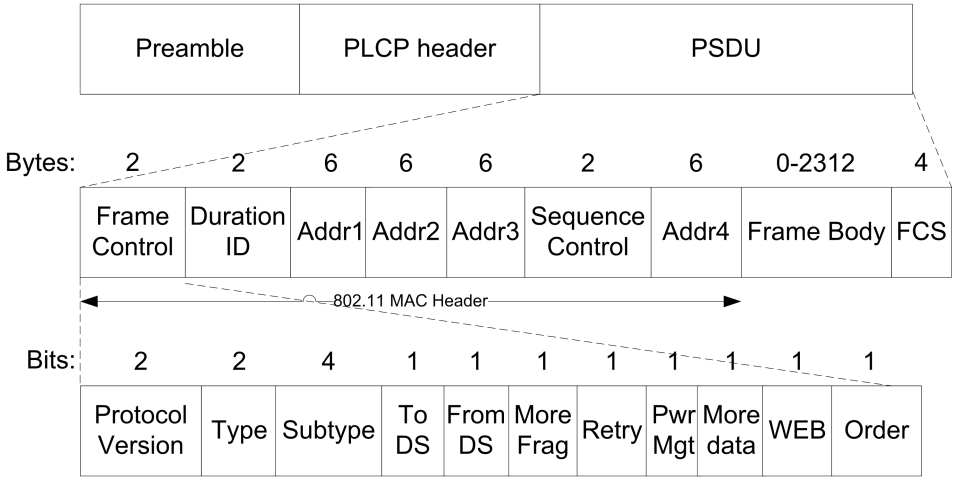

Abbildung 4.2: MAC-Frame Format basierend auf [Prasad02]

die vorliegende Arbeit nicht von Bedeutung ist.

4.2.1 Frameformat

In Abbildung 4.2 ist der Aufbau des MAC-Frames zu sehen sowie seine Einbettung in das aus Abbildung 4.1 bekannte Frameformat der physikalischen Schicht. Der Header des MAC-Frame besteht aus insgesamt sieben Feldern, wobei allerdings nicht alle zwangsweise in jedem Frame vorkommen. Grundsätzlich gibt es drei Klassen von Frames die jeweils für unterschiedliche Funktionen verwendet werden. So dienen Data Frames zur Übertragung von Daten. Control Frames wie etwa RTS, CTS und ACK, werden, wie später noch genauer beschrieben wird, zur Unterstützung beim Zugriff auf das Medium verwendet. Schließlich gibt es noch die Management Frames, die zum Austausch von Management-Informationen zwischen den MAC-Schichten der einzelnen Knoten dienen.

Die einzelnen Bits im Frame Control Feld haben unterschiedliche Bedeutungen. Die ersten Bits dienen dazu, die Protokollversion zu spezifizieren. Die anschließenden Bits geben an, zu welchem Typ der Frame gehört, also ob es sich, wie bereits erwähnt, etwa um einen Data-, einen Control-, oder einen Management Frame handelt. Der Subtyp spezifiziert den Typ noch genauer, also ob es sich beispielsweise um einen ACK oder einen RTS Frame handelt.

ToDS und FromDS werden im Infrastrukturmodus verwendet und geben

an, ob eine Nachricht über den Access Point weitergeleitet wird oder, wenn sie empfangen wurde, über einen Access Point geleitet worden ist. "More Fragments" wird verwendet, wenn eine Nachricht aufgrund der Größe nicht mit einem einzelnen Frame übertragen werden konnte, sondern auf mehrere Fragmente zerteilt wurde. Ist dieses Bit auf 1 gesetzt, bedeutet das, dass noch weitere Fragmente zu diesem Frame gehören.

Das Retry Bit wird verwendet, um Frames, die erneut gesendet wurden, zu kennzeichnen. Das Power Management Bit gibt an, in welchem Power Management Modus sich der Knoten nach der Übertragung des Frames befinden wird. Ebenfalls zum Power Management gehört das More Data Bit. Es wird vom Access Point verwendet, um einem Knoten mitzuteilen, dass noch weitere Frames für ihn zur Zeit im Access Point gepuffert sind. Das Order Bit gibt schließlich an, dass die "Strictly-Ordered service class" verwendet wird [IEE99b].

Das auf das Frame Control Feld folgende Feld ist die Duration ID, die, mit Ausnahme bei der Verwendung in einer Power-Save Poll Message, für die Übermittlung der Daten, die für den Network allocation vector benötigt werden, verwendet wird.

Weiters enthält der MAC-Header eine Reihe von Adressfeldern. Adresse 1 ist immer die Adresse des Empfängers des Paktes, Adresse 2 die des Senders. Die anderen Adressfelder sind nicht immer vorhanden und enthalten gegebenenfalls die Adresse des ursprünglichen Senders beziehungsweise die Ziel Adresse.

Das Sequence Control Feld wird bei fragmentierten Frames verwendet. Mit den darin enthaltenden Informationen kann die ursprüngliche Reihenfolge der Frames festgestellt sowie Duplikate erkannt werden.

Im Frame Body befinden sich schließlich die eigentlich zu übertragenden Daten. Am Ende des Frames befindet sich noch eine Prüfsumme, um fehlerhafte Frames zu erkennen.

4.2.2 Zugriff mittels Distributed Coordinator Function

In Abbildung 4.3 ist das grundsätzliche Zugriffsschema, welches im IEEE802.11 Standard definiert wird, zu sehen. Bevor ein Knoten zu senden beginnen kann, muss das Medium für eine gewisse Zeit frei sein. Wenn die Übertragung abgeschlossen ist und ein weiterer Frame übertragen werden soll, muss das Medium wiederum für eine gewisse Zeit frei sein, bevor der Backoff-Algorithmus ausgeführt wird. Der Backoff-Algorithmus dient zur

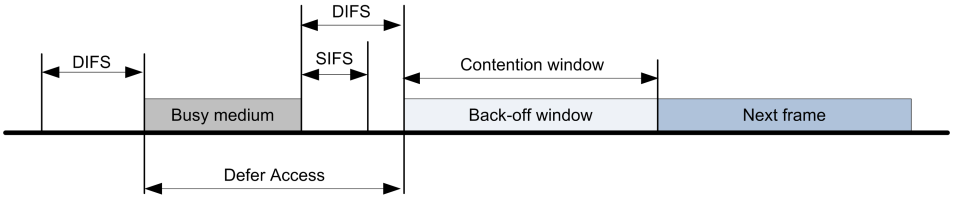

Abbildung 4.3: Darstellung der Sendepausen

Lösung von Konflikten, die entstehen, wenn zwei Knoten gleichzeitig eine Nachricht übertragen wollen. Nach einer Übertragung wählt ein Knoten zufällig eine Backoffzeit aus, die einer gewissen Anzahl an Zeitslots entspricht. Er wartet anschließend solange, bis die Zeit abgelaufen ist, wobei laufend geprüft wird, ob das Medium noch frei ist. Ist das Medium wieder belegt, wird der Backoff Timer gestoppt. Erst wenn das Medium wieder eine Zeit lang frei ist, läuft der Timer weiter. Da verschiedene Knoten wahrscheinlich unterschiedliche Backoff-Zeiten gewählt haben, wird der Knoten mit der geringeren Backoff-Zeit früher auf das Medium zugreifen, während ein etwaiger anderer Knoten mit einer längeren Backoff-Zeit noch wartet. Damit wird die Wahrscheinlichkeit, dass es zu einem Konflikt kommt, verringert. Kommt es doch zu einem Crash, berechnen die beteiligten Knoten eine neue Backoffzeit, wobei der mögliche Wertebereich größer ist [Gast05].

Wichtig sind in diesem Zusammenhang auch die verschiedenen Zeiten, die, nachdem das Medium belegt war, abgewartet werden müssen. Wie später noch zu sehen ist, ist es durch diese Wartezeiten möglich, priorisierte Nachrichten zu senden. So erfolgt etwa die Bestätigung, dass ein Frame erfolgreich übertragen wurde, nach der kleinsten definierten Wartezeit. Da andere Knoten, die eine normale Nachricht senden wollen, eine längere Zeit abwarten müssen, kann die Antwort prompt erfolgen.

Es gibt vier definiere Wartezeiten, die als interframe spacing, kurz IFS, bezeichnet werden. Die da wären in ansteigender Reihenfolge: Short interframe space (SIF), PCF interframe space (PIFS), DCF interframe space (DIFS) sowie der in der Grafik nicht eingezeichnete Extended interframe space (EIFS) [Prasad02].

Der Short interframe space wird für priorisierte Kontrollnachrichten verwendet, wie etwa RTS, CTS oder ACK. Der PCF interframe Space findet Verwendung, wenn ein Accesspoint die Koordination zwischen den Knoten übernimmt. Der nächstgrößere Interframe Space ist der DCF interframe space, der etwa zum Austausch von Datenframes unter Einsatz der Dis-

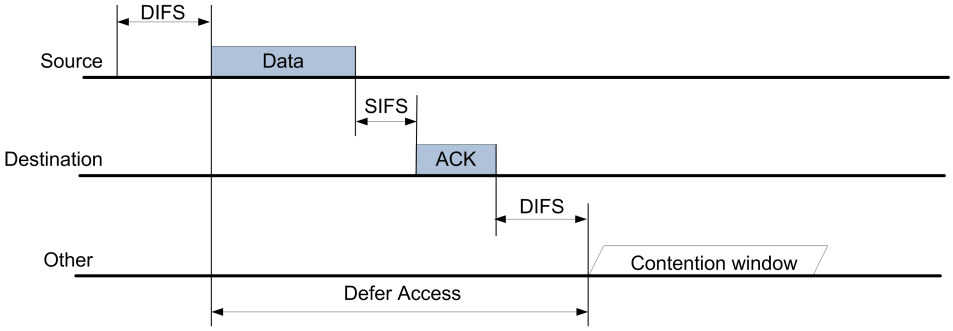

Abbildung 4.4: CSMA/CS Zugriff

tributed coordinaton function verwendet wird. Der Extended interframe space kommt zum Einsatz, wenn es zu einer Kollision gekommen ist. Sobald nach einer Kollision das Medium wieder frei ist, müssen die beteiligten Stationen den Extended interframe space abwarten, bevor sie ihren Backoff-Algorithmus durchführen.

In Abbildung 4.4 ist der Ablauf einer kompletten Übertragung eines MAC-Frames zu sehen. Die Station, die senden will, wartet so lange, bis das Übertragungsmedium für die Dauer eines DCF interframe spaces nicht belegt ist. Anschließend übermittelt sie die Nachricht. Die Station, die die Nachricht empfängt, quittiert den Empfang durch das Senden eines Kontrollframes vom Typ ACK (Acknowledge). Wichtig in diesem Zusammenhang ist, dass auf der Empfänger-Seite zwischen Empfang der Nachricht und der Übermittlung der Bestätigungsnachricht nur ein Short interframe space liegt. Dadurch hat keine andere Station, die beide Stationen empfängt, die Möglichkeit, vor dem Senden der Bestätigungsnachricht einen Datenframe zu senden. Der Grund dafür liegt darin, dass vor Übermittlung eines Datenframes der Kanal für die Dauer eines DCF interframe spaces frei sein muss, wobei der DCF interframe spaces, wie bereits erwähnt, länger als der Short interframe space ist [Gast05].

Da die Empfangsstation die Bestätigungsnachricht nur sendet, wenn ein Frame korrekt empfangen wurde, kann aufgrund des Ausbleibens der Nachricht vom Sender erkannt werden, dass die Übertragung nicht erfolgreich war. Da die Antwort im Normalfall prompt erfolgt, ist auch die Zeit bekannt, ab wann der Sender davon ausgehen kann, dass es zu einem Fehler gekommen ist. Der Sender versucht darauf, die Nachricht erneut zu übermitteln. Kam nach einer gewissen Anzahl an Versuchen keine Übermittlung zu Stande, wird der Frame verworfen.

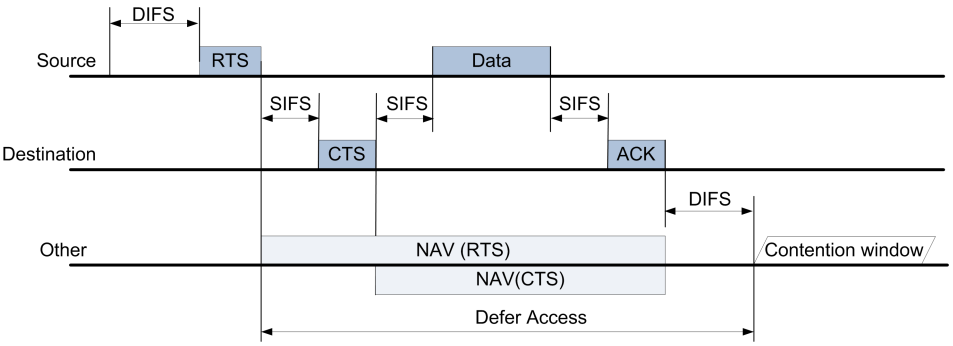

Abbildung 4.5: CSMA/CS Zugriff mit RTS/CTS Erweiterung

Wie bereits beschrieben, wird durch das Scannen des Mediums vor dem Senden versucht die Wahrscheinlichkeit einer Kollision zu minimieren. Es gibt allerdings die Möglichkeit, dass ein dritter Knoten nur im Sende- beziehungsweise Empfangsbereich des Empfänger Knotens liegt, und daher nicht erkennt, dass ein anderer Knoten, den er nicht empfängt, gerade eine Nachricht an den Empfangsknoten übermittelt. Dieser Knoten erkennt nur, dass gerade in seinem Empfangsbereich keine Übertragung stattfindet und beginnt ebenfalls mit seiner Übertragung, was zur Folge hat, dass der Empfänger keine der beiden Nachrichten empfangen kann. Dieses als "Hidden Station" [Tobagi75] bekannte Problem wird in Kapitel 7 noch genauer beschrieben.

Dieses Problem ist besonders schwerwiegend, wenn lange Nachrichten erneut gesendet werden müssen. Daher wurde ein virtueller Carrier Sense Mechanismus in das IEE802.11 Protokoll aufgenommen, der in Abbildung 4.5 dargestellt wird [IEE99b]. Wie in der Abbildung zu erkennen ist, werden vor der eigentlichen Datenübertragung zwei zusätzliche Frames ausgetauscht. Die Funktionsweise lässt sich anhand des folgenden Szenarios gut beschreiben. Es gibt vier Knoten: Knoten A will Knoten B eine Nachricht zukommen lassen. Zwei weitere Knoten H und I befinden sich in der Nähe. Knoten H befindet sich in Sende- und Empfangsreichweite von Knoten B und will diesem ebenfalls etwas später eine Nachricht zukommen lassen. Allerdings ist er von Knoten A so weit entfernt, dass er ihn nicht empfängt und auch nicht erreicht. Knoten I befindet sich in Reichweite von Knoten A, seine Position zu den anderen Knoten ist im Weiteren nicht von Belang.

Ohne den virtuellen Carrier Sense Mechanismus würde Knoten A mit der Übertragung zu Knoten B beginnen. Knoten I würde die Übertragung von Knoten A erkennen, und daher selbst nicht versuchen, eine eigene Übertra-

gung zu starten. Knoten H hingegen würde nicht erkennen, dass Knoten A bereits zu Knoten B sendet. Er würde einfach mit der Übertragung starten und so eine Kollision erzeugen.

Wird der virtuelle Carrier Sense Mechanismus verwendet, übermittelt Knoten A zuerst einen speziellen Kontrol-Frame, welcher Request to Send oder kurz RTS-Frame genannt wird. Dieser enthält Informationen über die Dauer der geplanten Übertragung. Knoten B und I empfangen diesen Frame. Knoten I berechnet daraus seinen Network Allocation Vector, der angibt, wann die komplette Übertragung beendet sein wird. Solange dieser Vektor aktiv ist, gilt das Medium als belegt. Erst wenn die entsprechende Zeitspanne vorbei ist, beginnt Knoten I wieder das Medium zu scannen. In Abbildung 4.5 entspricht Knoten I dem oberen Verlauf in der Zeile "Other". Der Empfänger-Knoten, Knoten B, antwortet auf den Request to Send Frame mit einem Clear to Send Frame. Dieser enthält ebenfalls Informationen wie lange die Übertragung noch dauern wird. Wichtig ist nun, dass dieser Frame nicht nur wieder von Knoten A empfangen wird, sondern auch von Knoten H, der so von der bevorstehenden Übertragung erfährt. Dieser berechnet sich ebenfalls wie Knoten I den Network Allocation Vector und verschiebt seine Übertragung entsprechend.

Da Knoten H erst von der geplanten Übertragung erfährt, wenn er den Clear to Send Frame empfängt, ist es natürlich davor möglich, das Knoten H die Übertragung des Request to Send Frames unterbricht. Da dieser Frame allerdings im vergleich zu einem Datenframe klein ist, dauert eine erneute Übertragung nicht so lange, als wenn der dazugehörige komplette Datenframe neu übertragen werden müsste. Ist der Datenframe allerdings relativ klein, rentiert sich eventuell der durch RTS und CTS erzeugte Overhead nicht. Daher können kleine Datenframes auch ohne Verwendung des virtuellen Carrier Sense Mechanismus übertragen werden [Prasad02].

Neben der soeben beschriebenen Distibuted Coordinator Funktion spezifiziert der Standard noch die Point Coordinator Funktion. Diese wird bei Verwendung von Infrastruktur verwendet und entsprechend im Weiteren nicht mehr genauer behandelt. Eine Beschreibung dieser Funktion findet sich etwa in [Gast05] sowie in [Prasad02].

Kapitel 5

Modelle

Wie bereits in Kapitel 3 beschrieben, ist das Erstellen eines Modells von einem System bei der Simulation von großer Bedeutung. Von der Qualität der Modelle ist der Realismus der Simulation abhängig. Allerdings lassen sich nicht alle Systeme beliebig genau modellieren. Je genauer ein Model ist, umso genauer stellt es ein konkretes System wieder dar. Kann man jedoch mit einem solchen System allgemeine Aussagen treffen? Weiters haben die Modelle auch Auswirkungen auf die Dauer einer Simulation. In vielen Fällen ist es so, dass komplexere Modelle mehr Rechenzeit und mehr Speicher benötigen. Konkret bedeutet das, dass etwa das Modell eines WLAN-Knotens Einfluss auf die maximale Größe eines simulierbaren Netzwerk hat, da die Simulationsplattform nur eine bestimmte Speichermenge besitzt. Die Schwierigkeit liegt bei der Modellierung also darin, den richtigen Abstraktionslevel zu erreichen, um die Fragestellungen beantworten zu können [Heidemann01].

In diesem Kapitel wird auf die verschiedenen in dem Framework implementierten Modelle eingegangen, deren Realismus diskutiert, sowie weitere Alternativen erörtert. Weiters werden auch die Ansätze in Simulationsumgebungen, vor allem Omnet++ und NS2, beschrieben und mit dem Framework des IBKSims verglichen. Aufgrund der Komplexität liegt dabei der Schwerpunkt vor allem auf der Modellierung der physikalischen Welt beziehungsweise der Bitübertragungsschicht.

5.1 Bitübertragungsschicht

Man stelle sich ein aus mehreren Knoten bestehendes mobiles Ad-Hoc Netz in einer Stadt vor. Um die Performance eines gewissen Routing-Algorithmus zu testen, versucht ein Knoten regelmäßig eine Nachricht an einen anderen mittels WLAN zu senden. Sind die Knoten in Reichweite oder befinden sich andere Knoten dazwischen, die die Nachricht weiterleiten können, wird die Nachricht übertragen, wobei allerdings nicht sicher gestellt ist, dass die Übertragung auch erfolgreich ist, also ein unbeschädigtes Paket ankommt.

Zur Simulation eines solchen Szenarios werden in vielen Fällen zwei Modelle für die Modellierung der Bitübertragungsschicht verwendet. Zum Einen ein Bewegungsmodel, mit dessen Hilfe die Bewegung der einzelnen Knoten simuliert wird, zum Anderen werden Funkmodelle verwendet, die den Datenaustausch per Funk modellieren. Die Einflussfaktoren auf die beiden Modelle, wie etwa Hindernisse bei Bewegungsmodellen, die die Bewegung einschränken, werden dabei in das jeweilige Modell integriert. Auch das vorgestellte Framework verwendet diese beiden Modelltypen, die im Weiteren separat betrachtet werden.

5.1.1 Radio Modelle

Im Weiteren werden die implementierten Radiomodelle beschrieben. Es wird deren Realismus diskutiert sowie auf die Ansätze anderer Simulatoren eingegangen.

5.1.1.1 Implementierte Radiomodelle

In dem Framework wurden eine Reihe von Radiomodellen implementiert. Sie unterscheiden sich vor allem durch ihre Komplexität.

Das einfachste Modell entspricht der Klasse der in [Kotz04] beschriebenen “Flat Earth”-Modelle. Ein Knoten kann die Nachricht eines anderen Knoten genau dann empfangen, wenn die Distanz zwischen beiden Knoten unter einem gewissen Wert liegt. Wenn der Knoten in Reichweite ist, kann er den Sender perfekt empfangen. Lediglich wenn sich der Empfänger in Sendereichweite eines anderen Knotens befindet, kann die Übertragung durch eine Kollision gestört werden, sofern der andere Knoten ebenfalls versucht, eine Nachricht zu übertragen.

In dem beschriebenen Modell spielt die Signalstärke keine Rolle. Durch die Einfachheit des Modells ist es besonders performant. Es eignet sich außerdem besonders zum Testen der Implementierung höherer Schichten.

Bei den anderen implementierten Modellen wurde eine andere Herangehensweise gewählt. Jeder Knoten hat einen Einflussbereich, der beliebig konfiguriert werden kann. Sendet ein Knoten eine Nachricht, wird für alle Knoten, die sich im Einflussbereichs des Knotens befinden, die Signalstärke der Übertragung berechnet. Dies erfolgt anhand verschiedener Modelle die im Weiteren noch erörtert werden. Überschreitet die Signalstärke bei einem empfangenen Knoten einen gewissen Wert, wird die Sendung als solche erkannt. Anderenfalls wird nur ein Rauschen wahrgenommen, das zum aktuellen Rauschpegel hinzugefügt wird. Wird eine Sendung erkannt, wird der höchste Rauschpegel während der Übertragung ermittelt. Auch das Empfangen eines zweiten Signals, welches aufgrund der Signalstärke als Signal erkannt werden würde, wird während des Empfangs dem Rauschen zugeordnet. Mit Hilfe des so ermittelten Rausch- und Interferenz-Pegels sowie der Signalstärke kann die Wahrscheinlichkeit, dass es zu einem Übertragungsfehler kam, ermittelt werden. Die Berechnungen hierzu werden etwas später in diesem Kapitel erörtert.

Für die Berechnung der Signalstärke beim Empfänger wurden drei Modelle implementiert. Es ist allerdings problemlos möglich weitere Modelle, wie etwa in [Parsons00] beschrieben, in das Framework einzufügen.

Das einfachste Modell dabei ist das "free space propagation model", auch Friis-Modell genannt [Rappaport99]. Die von der Distanz zwischen Sender und Empfänger abhängige Empfangsstärke wird dabei wie folgt berechnet:

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L} \tag{5.1}$$

In dieser Formel beschreibt P_t die Signalstärke des Senders. G_r beschreibt den Gewinn durch die Antenne des Empfängers, G_t den des Senders. Der Systemverlust fließt durch L in die Gleichung ein, λ ist die Wellenlänge in Metern. Diese Formel beschreibt die Empfangsstärke bei direkter, ungehinderter Sicht zwischen Sender und Empfänger. Reflexionen oder andere Einflüsse auf die Empfangsstärke werden in diesem Modell nicht einbezogen. Trotzdem liefert dieses Modell eine erste Annäherung.

Ein etwas genaueres Modell ist das "two-ray ground reflection model". In diesem Modell wird sowohl die direkte Empfangslinie wie im Friis-Model als auch ein indirekter Pfad, und zwar über die Reflexion am Boden, mit eingerechnet. Die in [Rappaport99] beschriebene Formel lautet wie folgt:

$$P_r(d) = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4} \tag{5.2}$$

P_t ist die Sendestärke des Senders, G_t und G_r sind wiederum der Antennengewinn des Senders beziehungsweise des Empfängers. Der Parameter h_t gibt die Höhe der Antenne des Senders über Grund an, h_r die Höhe der Antenne des Empfängers. Die Distanz zwischen Sender und Empfänger fließt durch d in die Formel ein.

Um mit dem Friis-Model konsistent zu sein, wurde die ursprüngliche Formel durch den Systemverlust L erweitert. Das eigentliche implementierte Modell sieht also folgendermaßen aus:

$$P_r(d) = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4 L} \tag{5.3}$$

Da die Distanz nicht quadratisch sondern biquadratisch in die Formel eingeht, ist der Pfadverlust bei zunehmender Distanz höher als im Friis-Modell.

Das "two-ray ground reflection model" liefert aufgrund von Oszillationen, die durch konstruktive und destruktive Überlagerung der beiden Strahlen entstehen, für geringe Distanzen eher schlechte Ergebnisse [Wei08]. Es ist allerdings möglich das "two-ray ground reflection model" mit dem Friis-Modell zu kombinieren indem man einen Kreuzungspunkt zwischen beiden Funktionen berechnet. Dies geschieht mit folgender Gleichung:

$$d_c = \frac{4\pi h_t^2 h_r^2}{\lambda} \tag{5.4}$$

Dabei ist d_c die Distanz, bei der sich die beiden Gleichungen überschneiden. Ist der Abstand kleiner als diese Distanz wird das Friis-Modell verwendet, anderenfalls dass "two-ray ground reflection model".

Bei beiden Modellen ist anzumerken, dass es zu Problemen kommen würde, falls sich Sender und Empfänger auf der gleichen Position befinden. Daher wurden die Modelle noch so erweitert, dass falls die Knoten eine Mindestdistanz ϵ unterschreiten, ϵ als Distanz angenommen wird.

Bei beiden Modellen handelt es sich um deterministische Modelle bei denen die Reichweite einen perfekten Kreis bilden. Wie später noch diskutiert wird, ist dass in der Wirklichkeit nicht der Fall. Einen anderen Ansatz verfolgt das "shadowing model" [Rappaport99]. Dieses empirische Model besteht aus zwei Teilen, einer Pfadverlust- und einer "shadowing"-Komponente.

Die Pfadverlust-Komponente berechnet die mittlere Signalenergie, abhängig von der Distanz der beiden Knoten und der Umgebung. Folgende Gleichung beschreibt diese Komponente:

$$\frac{P_r(d_0)}{P_r(d)} = \left(\frac{d}{d_0}\right)^{\beta} \tag{5.5}$$

Umgebung	Exponent β
freie Sicht	2
urbanes Gebiet	2.7 bis 3.5
abgeschattetes urbanes Gebiet	3 bis 5
direkte Sicht in Gebäude	1.6 bis 1.8
verstellte Sicht in Gebäude	4 bis 6
verstellte Sicht in Fabriken	2 bis 3

Tabelle 5.1: Pfad-Verlust Exponent in verschiedenen Umgebungen nach [Rappaport99]

Die von der Distanz d abhängige mittlere Energie wird in der Gleichung durch $\overline{P_r(d)}$ repräsentiert. Sie wird relativ zu $P_r(d_0)$ berechnet, wobei dieser Term die Energie im Abstand d_0 darstellt. Diese Energie wird beispielsweise durch das Friis-Model berechnet.

Der interessante Teil in der Gleichung ist der Exponent β. Dieser Term wird auch Pfadverlust-Komponente genannt, und wird empirisch über Feldmessungen ermittelt. Tabelle 5.1 gibt einige typische Werte an. Es ist zu erkennen, dass je mehr Hindernisse es in einer Umgebung gibt, um so größer die Pfadverlust-Komponente wird, was einem stärkeren Abfall der Signalstärke in Abhängigkeit von der Distanz entspricht. Da der Pfadverlust meist in Dezibel angegeben wird, hier auch noch vollständigkeitshalber die Formel in entsprechender Form:

$$\left[\frac{\overline{P_r(d)}}{P_r(d_0)}\right]_{dB} = -10\beta log\left(\frac{d}{d_0}\right) \tag{5.6}$$

Die zweite Komponente, die "shadowing"-Komponente, dient zur Modellierung der Varianz der Empfangsstärke an einem gewissen Punkt. In der nachfolgenden Formel wird sie durch X_{dB} dargestellt. Sie ist eine gaussverteilte Zufallsvariable mit einer Standardabweichung von σ_{dB}. Diese Standarabweichung ist wiederum ein empirischer Wert. Einige typische Werte sind in Tabelle 5.2 zu sehn.

Durch Zusammenfügen der beiden Komponenten erhält man die komplette Formel zur Berechnung der Signalstärke an einem Punkt:

$$\left[\frac{\overline{P_r(d)}}{P_r(d_0)}\right]_{dB} = -10\beta log\left(\frac{d}{d_0}\right) + X_{dB} \tag{5.7}$$

Zu den wichtigsten Unterschieden von diesem Modell zu den beiden vorherigen gehört die Möglichkeit, durch die empirischen Werte das Umfeld

Umgebung	σ_{dB} (dB)
Freien	4 bis 12
Büro, stark unterteilt	7
Büro, leicht geteilt	9.6
direkte Sicht in Fabriken	3 bis 6
verstellte Sicht in Fabriken	6.8

Tabelle 5.2: Typische Standartabweichungen für verschiedene Umgebungen nach [Wei08]

besser in die Simulation mit einzubeziehen sowie die Tatsache, dass der Empfangsbereich keinen idealen Kreis mehr bildet.

Durch die verschiedenen Radiomodelle kann die Stärke des empfangenen Signals ermittelt werden. Ebenso kann auch die Interferenz durch die Signale anderer Knoten berechnet werden. Da auch das Kodierungsverfahren beziehungsweise das Modulationsverfahren bekannt ist, kann auf einfache Weise die Wahrscheinlichkeit, dass ein Bitfehler bei der Übertragung aufgetreten ist, also dass die Übertragung nicht erfolgreich war, berechnet werden.

Für die Phase-shift keying modulation, kurz PSK, wird folgende Gleichung verwendet:

$$P_e = \frac{1}{2} exp\left(-\frac{E_b}{N_0}\right) \tag{5.8}$$

Sie wurde von [Rappaport99] übernommen und beschreibt die Wahrscheinlichkeit eines Fehlers, abhängig vom Verhältnis der Signalstärke zum Rausch- und Interferenzsignal.

Das Complementary Code Keying Modulations Schema wurde durch die M-Quadraturamplitudenmodulation modelliert. Deren Fehlerberechnung lautet wie folgt:

$$P_e = 4\left(1 - \frac{1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{2E_{min}}{N_0}}\right) \tag{5.9}$$

mit

$$Q(x) = 0.5 erfc\left(\frac{x}{\sqrt{2}}\right) \tag{5.10}$$

Der Term M gibt dabei die Anzahl der Phasen an. Für genauere Informationen zu dieser Berechnung sei auf [Proakis01] verwiesen.

Diese Kodierungsverfahren kommen im IEEE802.11b Standard zum Einsatz. Für andere Kodierungsverfahren gibt es die entsprechen Modelle für

die Bitfehlerberechnung in der Literatur, die dann auf einfache Weise in den Simulator eingefügt werden können.

Mit den beschriebenen Modellen lässt sich die Stärke der Signale sowie mit deren Hilfe die Fehlerwahrscheinlichkeit bei der Übertragung abhängig von der Position der Knoten berechnen. Es stellt sich allerdings die Frage, wie realistisch die verwendeten Modelle zur Simulation der Radio-Übertragung sind.

5.1.1.2 Realismus der Modelle

Wie etwa in [Heidemann01] oder in [Stepanov05] gezeigt wird, hat die Qualität des Models Einfluss auf die Aussagekraft einer Simulation. Mit Aussagekraft ist hierbei der Unterschied zwischen den Ergebnissen von Simulationen und Messungen gemeint. Die Schwierigkeit bei der Wahl der Modelle liegt vor allem darin, die Modelle zu wählen, die für die durch die Simulation zu klärende Fragestellung die passende Detailliertheit bieten. Sind die Modelle zu genau, dauert die Simulation unnötig lange. So macht es wenig Sinn, wenn Routingprotokolle in mobilen Ad-Hoc Netzen untersucht werden sollen, die komplette Hardware des Knotens zu simulieren. Umgekehrt können zu einfache Modelle zu Ergebnissen führen, die stark von der Realität abweichen. Daher werden im Weiteren die Annahmen bei der Modellierung der Funkübertragung in Hinblick auf ihren Realismus und auf mögliche Auswirkungen zu überprüfen.

In [Kotz04] werden sechs Axiome formuliert, die in vielen Radiomodellen in der einen oder anderen Form zu finden sind. So auch in den Modellen die im Rahmen dieser Arbeit implementiert wurden.

Die Axiome lauten wie folgt:

1. Die Welt ist flach.
2. Das Sendegebiet eines Funksenders ist rund.
3. Alle Sender haben die gleiche Reichweite.
4. Wenn ich dich höre, kannst du auch mich hören. (Symmetrie)
5. Wenn ich dich höre, dann höre ich dich perfekt.
6. Die Signalstärke ist nur von der Distanz abhängig.

Dass diese Axiome nicht der Wirklichkeit entsprechen, ist leicht einzusehen. Die Welt ist im Allgemeinen nicht flach. Befinden sich etwa zwei WLAN-Knoten in einer Stadt, so könnten sich eine Vielzahl an Hindernissen, wie etwa Mauern, Autos oder Personen zwischen ihnen befinden. Sie werden sich vielleicht auch nicht in derselben Ebene befinden. All diese Dinge haben Ein-

fluss auf die Ausbreitung der Radiowellen und dadurch letztendlich auf die Datenübertragung. Natürlich könnte versucht werden, ein spezifisches Szenario zu modellieren, in dem dann auch Hindernisse einberechnet werden, allerdings würde sich auch hier wieder die Frage stellen, ob sich die Ergebnisse einer solchen Simulation verallgemeinern lassen. Von den vorgestellten Radiomodellen wird auf solche Gegebenheiten nur vom "shadowing"-Model etwas Rücksicht genommen, indem versucht wird, Hindernisse durch die statistische Komponenten in die Simulation einfließen zu lassen. Allerdings kann auch bei diesem Model nicht behauptet werden, dass die Umgebung in irgendeiner Weise genau simuliert wird. So kann etwa nicht das Szenario simuliert werden, dass ein Knoten eine bessere Reichweite durch seine aktuelle Position hat.

Sowohl im Friis-Modell als auch im "two-ray ground reflection" Modell bilden die Äquipotentialinien der Signalstärke einen perfekten Kreis. Dass dies in Wirklichkeit nicht zwangsweise der Fall ist, ergibt sich aus der Tatsache, dass Hindernisse das Signal etwa dämpfen können. In diesem Zusammenhang ist auch das dritte Axiom, dass alle Sender die gleiche Reichweite besitzen, zu betrachten. Auch wenn alle Sender in einem Szenario mit der gleichen Leistung senden, kann die Reichweite aufgrund von äußeren Bedingungen variieren. Diese Tatsache fließt in das "shadowing"-Modell durch die "shadowing"-Komponente ein.

Eine direkte Folge daraus ist auch, dass das Axiom, dass wenn ein Knoten einen anderen Knoten empfängt, dieser auch ihn empfangen kann, so im allgemeinen nicht stimmt. Aber auch wenn alle Knoten die gleiche Reichweite besäßen, und diese die Form eines exakten Kreises hätte, ist dieses Axiom nicht zwangsweise richtig. Der Grund dafür ist die Zeit. Knoten A könnte eine Nachricht an Knoten B senden und sich, noch bevor B darauf antworten kann, aus dessen Sendebereich entfernen. Dies hat allerdings direkte Auswirkungen auf die Wahrscheinlichkeit, dass eine Übertragung erfolgreich war, da etwa im 802.11b Protokoll der Empfänger den korrekten Empfang bestätigen muss, was im Fall, dass sich der Sender nicht mehr in Reichweite befindet, nicht möglich ist [IEE99a]. Asymmetrische Übertragungen können aber auch ganz andere Einflüsse auf die Aussagekraft der Simulation haben. Wird ein MAC-Protokoll verwendet, welches mit asymmetrischen Verbindungen zurecht kommt und werden in einem solchen System zwei Routingalgorithmen getestet, wobei einer dabei mit asymmetrischen Links umgehen kann und der andere nicht, kann das Ergebnis stark von der Wirklichkeit abweichen, wenn auf physikalischer Ebene nur symmetrische Übertragungen simuliert werden. Die in dem Framework implementierten Radiomodelle können zur Simulation asymmetrischer Verbindungen verwendet werden,

da die einzelnen Knoten mit unterschiedlicher Sendeleistung senden können, beziehungsweise ihre Reichweite verändert werden kann.

Dass wenn ein Knoten eine Übertragung empfängt, diese perfekt empfangen wird, davon wird, wie später in Kapitel 6.2.3.5 zu sehen sein wird, nur in dem Flat-Earth Modell ausgegangen, welches allerdings kein Radiomodell implementiert. Jedoch wird in diesem Model auf Kollisionen Rücksicht genommen. Würden zur Untersuchung von Protokollen Modelle verwendet werden die keine Störungen simulieren können, könnten keine Aussagen über die Toleranz von Übertragungsfehlern der Protokollen getätigt werden.

Auf das letzte Axiom wird nur im "shadowing"-Modell Rücksicht genommen, wiederum durch die Zufallswerte. Bei den anderen Modellen ist die Signalstärke, wie im Axiom beschrieben, nur von der Distanz abhängig.

Dadurch, dass mehrere Modelle zur Simulation der Funkübertragung vorhanden sind, kann der Benutzer entscheiden, welches Modell für das gewünschte Szenario akkurat ist. Wichtig ist, dass er sich im klaren ist, dass das verwendete Modell Einfluss auf das Ergebnis hat und daher im vorhinein abzuschätzen ist, ob die verschiedenen Ungenauigkeiten des jeweiligen Modells tolerierbar sind.

5.1.1.3 Vergleich mit anderen Simulatoren

Der IBKSim ist natürlich nicht der einzige Simulator, der zur Simulation von mobilen Ad-Hoc Netzen verwendet werden kann. Interessant sind allerdings die Unterschiede zwischen dem hier beschriebenen Framework und anderen Simulatoren, die mobile Ad-Hoc Netze unterstützen. Zu den bekanntesten Simulatoren zählen sicherlich NS2 [Inf08] sowie OMNeT++ [OMN08] in Zusammenhang mit dem Mobility Framework für OMNeT++ [Tel08]. Das in dieser Arbeit vorgestellte Framework implementiert dieselben Modelle wie NS2 in der ursprünglichen Form. Trotzdem gibt es einige Unterschiede. Wie in [Xiuchao04] beschrieben, verwendet NS2 zur Ermittlung, ob ein Frame korrekt übertragen wurde, lediglich einen Schwellwert und nicht wie das vorgestellte Framework Modelle zur Berechnung der Fehlerwahrscheinlichkeit.

Auch bei den Radio-Modellen gibt es Unterschiede. In NS2 wirkt sich die Übertragung zwischen zwei Knoten auf alle anderen Knoten aus, indem das Übertragungssignal in das Störsignal, welches aus dem Grundrauschen und den Interferenzsignalen besteht, einberechnet wird. Dies wird auch in dem vorliegenden Framework gemacht. Allerdings ist es möglich, eine maximale Entfernung anzugeben, innerhalb der die Übertragung eines Knotens Ein-

fluss auf die anderen Knoten hat. Es ist also möglich, das Framework so zu konfigurieren, dass die geringe Störung, die das Übertragungssignal bei weit entfernten Knoten bewirkt, vernachlässigt wird. Dadurch eignet sich das Framework besser zur Simulation von großen, weit verteilten mobilen Ad-Hoc Netzen.

In NS2 ist es möglich, dass sich zwei oder mehr Knoten an derselben Stelle befinden, was zu unrealistischen Ergebnissen führt. Diese Möglichkeit wurde bereits in den Radiomodellen unterbunden. Befinden sich zwei Knoten an derselben Stelle, werden die Knoten so behandelt, als wenn sie sich sehr nahe nebeneinander befinden würden.

Einen weiteren Unterschied gibt es noch bei der Verwendung des "shadowing"-Modelles. In NS2 wird für jede Übertragung ein neuer Empfangswert berechnet, auch wenn sich die Knoten nicht bewegt haben und sich auch sonst keine Umgebungsvariable geändert hat. Die Signalstärke springt also laufend zwischen verschiedenen Werten. Das im Rahmen dieser Arbeit entwickelte Framework berechnet den Wert nur bei jeder Änderung einer Umgebungsvariablen oder Positionsänderung des Knotens neu. Das wirkt sich positiv auf die Performance aus.

Das Mobility Framework für OMNeT++ bietet zwei Modelle zur Simulation der physikalischen Schicht. Im einfacheren Modell können Knoten, die innerhalb einer gewissen Reichweite sind, Nachrichten austauschen, die mit einer gewissen Wahrscheinlichkeit übertragen werden. Auf Interferenzen oder Kollisionen wird dabei keine Rücksicht genommen [Tel08]. Ein weiteres Modul implementiert ein simples Free-Space Model und ermittelt aus der so erhaltenen Signalstärke und Interferenz mit Hilfe von Bitfehler-Modellen die Übertragungswahrscheinlichkeit. Das Framework des IBKSims bietet im Gegensatz dazu mehrere Radiomodelle an.

Es gibt natürlich noch eine Vielzahl weiterer Simulatoren die mobile Ad-Hoc Netze unterstützen. Eine gute Übersicht über die jeweils verwendeten Ansätze zur Simulation der physikalischen Schicht bietet [Khosroshahy07].

5.1.2 Bewegungsmodelle

Wie im letzten Abschnitt beschrieben, spielt die Distanz zwischen den einzelnen Knoten bei der Simulation der Signalübertragung eine wichtige Rolle. Es ist daher nötig, die Bewegung der Knoten zu simulieren. Dies geschieht mit Hilfe von Bewegungsmodellen. Wie in Kapitel 6.2.1 beschrieben, sind bereits einige Bewegungsmodelle in das Framework implementiert. Es wurde jedoch vor allem darauf geachtet, dass auf einfache Weise weitere Modelle

in den Simulator eingefügt werden können. Damit kann der Simulator je nach gewünschten Szenario erweitert werden.

In der Literatur finden sich eine Reihe von Bewegungsmodellen. Gute Übersichten dazu bieten unter anderem [Camp02] sowie [Jardosh03]. Wichtig in diesem Zusammenhang ist vorallem, dass beim Vergleich verschiedener Protokolle die Knoten genau den gleichen Bewegungsablauf durchführen sollten [Sánchez01]. Auf diese Weise besteht für die zu vergleichenden Protokolle die gleiche Ausgangsbedingung beziehungsweise die Protokolle können im selben Szenario überprüft werden. Dies wird durch die im IBKSim implementierten Zufallsfunktionen gewährleistet, die bei gleichem Seedwert jeweils die gleichen Zufallswerte liefern.

5.2 Modellierung der MAC Schicht

Um das MAC-Protokoll simulieren zu können, reicht es aus, dieses einfach zu implementieren. Da es von Grund auf abstrakt ist, muss nicht extra ein abstraktes Modell davon geschaffen werden. Allerdings stellt sich die Frage, in welchem Umfang und mit welcher Genauigkeit das Protokoll implementiert werden muss beziehungsweise welche Funktionen wirklich benötigt werden.

Zur Simulation mobiler Ad-Hoc Netze wird beispielsweise die Unterstützung von Infrastruktur nicht benötigt. Ebenso ist die Unterstützung der "Wi-Fi Protected Access"-Funktion zur Untersuchung von Routing Protokollen in mobilen Ad-Hoc Netzen nicht von wesentlicher Bedeutung. Daher wurden einige nicht benötigte Funktionen auch nicht implementiert.

Eine direkte Folge davon ist, neben dem Performance-Gewinn, dass auch in den Frames nicht alle Datenfelder benötigt werden. Diese wurden entsprechend in der aktuellen Version auch nicht in die Datenstruktur der Frames aufgenommen. Da allerdings die Größe des Frames separat gespeichert und nicht direkt aus der Datenstruktur berechnet wird, hat das Nichtvorhandensein des entsprechenden Datenfeldes keine Auswirkung auf die Größe des Frames und damit auf die simulierte Übertragungszeit. Falls das Framework für andere Simulationen verwendet werden soll, in denen weitere Funktionen des Protokolls benötigt werden, können sie jederzeit implementiert werden.

Auch in anderen Frameworks, die zur Simulation mobiler Ad-Hoc Netze dienen, kommt es häufig vor, dass nicht die komplette MAC-Schicht implementiert wurde. So unterstützt das Mobility Framework für OMNeT++ keine Infrastruktur und auch keine Fragmentierung [Tel08]. An dieser Stelle sei auch auf [Khosroshahy07] verwiesen, wo unter anderem auch die implementierte Funktionalität des MAC Protokolls bei verschiedenen Simulatoren

beschrieben wird.

Kapitel 6

Realisierung

In diesem Kapitel wird das im Rahmen der Diplomarbeit entwickelte Framework beschrieben. Dabei wird zuerst eine Übersicht über alle Klassen, die im Weiteren auch Module genannt werden, und ihr Zusammenwirken gegeben. Anschließend werden die einzelnen Klassen genauer beschrieben sowie ihre Schnittstellen und gegebenenfalls ihre Konfigurationsparameter dokumentiert. Das Kapitel schließt mit einer exemplarischen Konfiguration für eine WLAN-Simulation um die Unterschiede zu der in Kapitel 3.2.4 beschriebenen Konfiguration deutlich zu machen.

6.1 Übersicht über die Module

Grundsätzlich lassen sich die Klassen des Frameworks in drei Kategorien aufteilen: Schicht-, Paket- und Bewegungsmodule. Jedes Modul entspricht dabei einer Klasse in dem Framework. Eine Übersicht über alle Module beziehungsweise Klassen des Frameworks ist in Tabelle 6.1 zu sehen.

Die Schichtmodule sind von *ibkLayer* abgeleitete Klassen. Im Framework wurde die logische Trennung der Funktionalität auf die verschiedenen Schichten beibehalten und entsprechend repräsentieren die einzelnen Schichtmodule jeweils eine Schicht des OSI-Modells. Abbildung 6.1 gibt eine Übersicht über die einzelnen Schichten.

In der physikalischen Schicht sind zwei Klassen angesiedelt und zwar *WlanPhys* und *WlanPhysAdvanced*. Diese zwei Module sind für die Simulation der physikalischen Welt in dem Framework zuständig. Dies beinhaltet sowohl die Übertragung über die Radioschnittstelle, als auch die Bewegung

Name	Typ
ibkmobilityFactory	Fabriks-Klasse für die Bewegungsmodelle
ibkmobility	Bewegungsmodel
ibkmobilityBounce	Bewegungsmodel
ibkmobilitySquare	Bewegungsmodel
Ieee80211MacFrame	Paket-Modul
WlanFrame	Paket-Modul
DummyPacket	Paket-Modul
TimePacket	Paket-Modul
DummySend	Schicht-3-Modul
DummyReceive	Schicht-3-Modul
WlanPing	Schicht-3-Modul
Ieee80211Mac	Schicht-2-Modul
WlanPhys	Schicht-1-Modul
WlanPhysAdvanced	Schicht-1-Modul

Tabelle 6.1: Auflistung aller Module des Frameworks

der Module, wobei die Bewegung, wie noch später genauer beschrieben wird, mit Hilfe von Bewegungsmodulen simuliert wird. Der Unterschied zwischen den beiden Modulen liegt in der Genauigkeit wie die Umwelt simuliert wird. Das *WlanPhysAdvanced*-Modul bezieht Interferenzsignale sowie den Abstand zwischen zwei Knoten genauer in die Berechnung ein. Dies geht allerdings auf Kosten der Rechenzeit. Das *WlanPhys*-Modul ist daher besonders für Simulationen, die möglichst schnell grobe Resultate liefern sollen

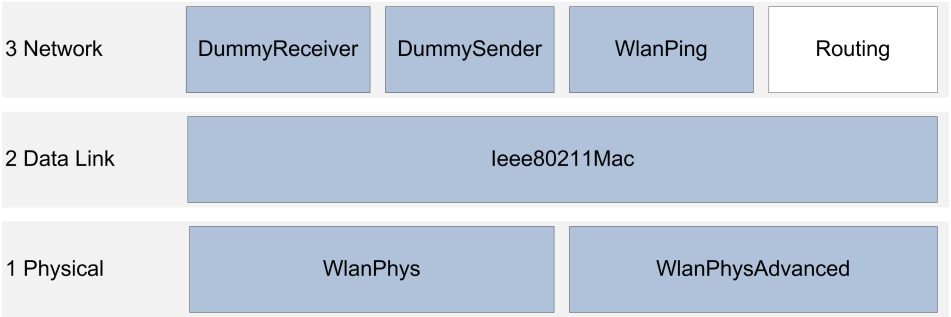

Abbildung 6.1: Übersicht über die Module des Frameworks

oder auch zum Testen gut geeignet. Wichtig in diesem Zusammenhang ist, dass diese Module nicht wie im Simulator vorhergesehen fix miteinander vernetzt sind, sondern untereinander dynamisch Verbindungen aufbauen können.

Auf die physikalische Schicht setzt die Datenübertragungsschicht auf. In dieser Schicht ist das *Ieee80211Mac*-Modul angesiedelt, welches das bereits beschriebene IEEE 802.11 Protokoll implementiert.

In der Netzschicht sind mehrere Module vorhanden, die zum Testen der unteren Schichten dienen. In dieser Schicht sind auch die Routingprotokolle angesiedelt, die allerdings nicht Teil dieser Arbeit sind. Entsprechend sind sie daher etwas blasser in Abbildung 6.1 eingezeichnet.

Die einzelnen Schichten kommunizieren untereinander mit Hilfe von Events in denen Pakete eingebettet sind. Daher sind in dem Framework einige Pakettypen definiert, wobei es sich jeweils um von der Klasse *cPacket* abgeleitete Klassen handelt. *WlanPhys* und *WlanPhysAdvanced* verwenden Objekte der Klasse *WlanFrame*, um miteinander zu kommunizieren. Sie werden sowohl zur Übertragung andere Pakete verwendet, die sie in sich einbetten können, als auch zum Austausch von Systeminformationen, wie etwa Informationen über das Positionsupdate einzelner Knoten.

Objekte der Klasse *Ieee80211MacFrame* repräsentieren die Frames der MAC-Schicht. Sie werden in Objekte der Klasse *WlanFrame* eingebettet und dienen zum Datenaustausch zwischen *Ieee80211Mac*-Modulen. Auch sie können weitere Instanzen der Klasse *cPacket* oder davon abgeleitete Klassen aufnehmen. Für Testzwecke gibt es noch die Paket-Klasse *DummyPaket*, deren Instanzen dazu verwendet werden können, um Strings zu transportieren. Als vierte Paket-Klasse wurde *TimerPacket* eingeführt, die für die Verwendung von Timern praktisch ist. Objekte dieser Klasse werden allerdings nicht zum Datenaustausch zwischen Modulen verwendet, sondern von den Simulationsobjekten an sich selbst gesendet.

Bei der dritten Kategorie von Modulen, den Bewegungsmodulen, handelt es sich um Klassen, die die Bewegung der Knoten simulieren. Ein Objekt der Klasse *mobilityFactory* erzeugt dabei die einzelnen Bewegungsmodule, die anschließend von den *WlanPhys*-Objekten verwendet werden. Alle Bewegungsmodule sind von der Klasse *ibkmobility* abgeleitet. Auf diese Weise können beliebige neue Bewegungsklassen entwickelt werden, die neue Bewegungsmuster enthalten, ohne dass die *WlanPhys*-Klasse beziehugsweise die davon abgeleitete *WlanPhysAdvances*-Klasse angepasst werden muss.

Abbildung 6.2 zeigt den möglichen Aufbau zweier WLAN-Knoten. Auf un-

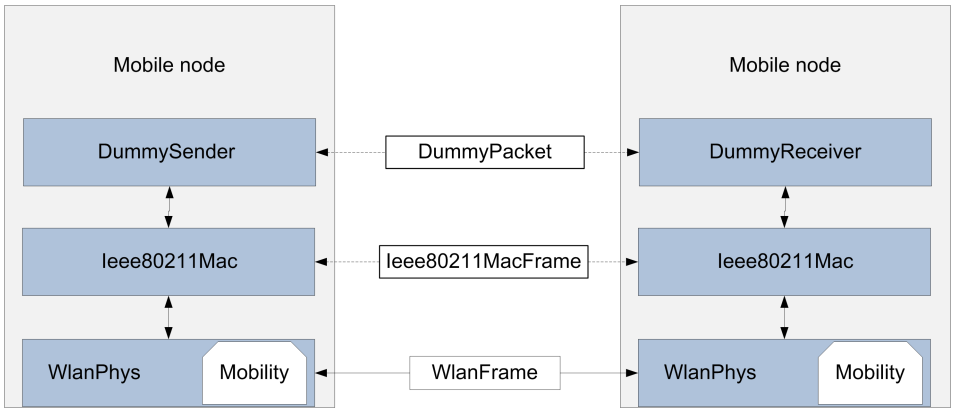

Abbildung 6.2: Aufbau zweier Knoten

terster Ebene bestehen beide aus den einfacheren *WlanPhys*-Objekten, die jeweils auch ein Bewegungsmodul beinhalten. In der Ebene darüber befindet sich jeweils ein *Ieee80211Mac*-Objekt. *Ieee80211Mac*-Objekte kommunizieren durch Verwendung von Objekten der Klasse *MacFrame* miteinander. Diese werden dabei nicht direkt übertragen, sondern zuerst an die darunterliegende physikalische Schicht weitergereicht, die den Transport zwischen den Knoten übernimmt. In der Vermittlungsschicht befindet sich in einem dargestellten Knoten ein *DummySend*- und im anderen ein *DummyReceive*-Objekt. Diese Module kommunizieren mit Hilfe der Klasse *DummyPacket* deren Instanzen wiederum nicht direkt, sondern mit Hilfe der darunterliegenden Schichten ausgetauscht werden.

6.2 Beschreibung der Module

In diesem Abschnitt werden die einzelnen Klassen genauer beschrieben. Es wird ihr Aufbau erklärt, ihre Schnittstellen erörtert sowie ihre Konfigurationsmöglichkeiten dokumentiert. Dabei werden zuerst die Bewegungs- sowie die Paketmodule beschrieben und erst zuletzt die Schichtmodule, da die Schichtmodule die anderen Klassen verwenden.

6.2.1 Bewegungsmodule

In dem entwickelten Framework existieren insgesamt vier Klassen, die für die Simulation der Bewegung der Knoten zuständig sind. Die da wären:

- ibkmobilityFactory
- ibkmobility
- ibkmobilityBounce
- ibkmobilitySquare

Die Klasse *ibkmobilityFactory* dient zum Erzeugen eines Bewegungsobjekts. Sie liefert ein Objekt des Typs *ibkmobility* zurück. Dabei kann es sich allerdings auch um ein Objekt des Typs *ibkmobilityBounce* oder *ibkmobilitySquare* handeln, da es sich bei diesen Klassen um Unterklassen von *ibkmobility* handelt. Bei *ibkmobility* sowie den entsprechenden Unterklassen kommt die in [Gamma95] beschriebene Fabrikmethode zum Einsatz. Nachfolgend werden die einzelnen Klassen genauer beschrieben.

6.2.1.1 ibkmobilityFactory

Die Aufgabe dieser Klasse ist es, Objekte des Typs *ibkmobility* zu erzeugen. Dafür besitzt es eine Funktion mit dem Namen *build()*, der ein String übergeben wird. Beim ersten Aufruf der Funktion wird der Klasse *ibkmobility* sowie jeder davon abgeleiteten Subklasse ein String zugeordnet. Dies geschieht mit Hilfe der Containerklasse *map*, wobei die Paare jeweils aus dem Namen und einem Zeiger auf die *build()*-Funktion der entsprechenden Klasse besteht. In Tabelle 6.2 sind die jeweiligen Zuordnungen von einem Namen zur einer Klasse zu sehen.

Mit Hilfe dieser Map kann nun je nach String die *build()*-Funktion der gewünschten Klasse aufgerufen werden, die darauf ein Objekt des Typs *ibkmobility* zurückliefert. Um ein neues Bewegungsmodell in das Framework einzufügen, muss dieses von *ibkmobility* oder einer der von dieser Klasse abgeleiteten Klassen abgeleitet sein und als neues Paar in die Map der Factory eingetragen werden.

Name	**Klasse**
none	ibkmobility
square	ibkmobilitySquare
bounce	ibkmobilityBounce

Tabelle 6.2: Zuordnung von Namen zu Klassen

6.2.1.2 ibkmobility

Die Klasse *ibkmobility* ist die Ursprungsklasse aller Klassen, die ein Bewegungsmodell implementieren. In Abbildung 6.3 ist das UML-Diagramm dieser Klasse zu sehen. Neben Konstruktor und Destruktor gibt es wieder eine *build()*-Funktion die eine neue Instanz dieser Klasse erstellt. Mit Hilfe der Funktion *setArea()* kann anschließend ein Gebiet festgesetzt werden, in dem sich der Knoten bewegen kann, sowie seine Startposition in diesem Gebiet mit Hilfe der *setPosition()*-Funktion.

Abbildung 6.4 verdeutlicht dieses Bereichskonzept der Bewegungsklassen. Jeder Knoten hat einen Bereich in dem er sich bewegen kann. Dieser Bereich ist durch die Koordinaten seines Ursprungs im Bezug auf den Ursprung des Gesamtsystems sowie seiner Höhe und Breite gegeben. Die Bereiche verschiedener Knoten können sich überlappen oder ident sein, sie müssen es aber nicht. Dieses Konzept gilt auch für alle von *ibkmobility* abgeleiteten Klassen.

Die Informationen über den Ursprung des Bewegungsbereichs des Knoten werden in den Variablen *areauX* und *areauY* gespeichert, die Länge und Breite in *areaX* und *areaY*. Schließlich kann noch die Anfangsgeschwindigkeit mit setSpeed gesetzt werden.

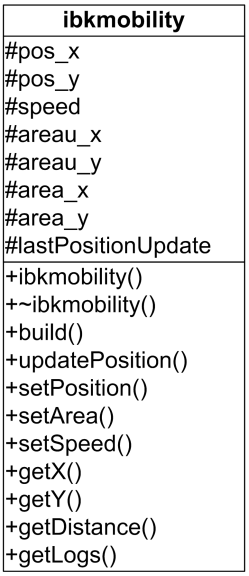

Abbildung 6.3: UML-Diagram von ibkMobility

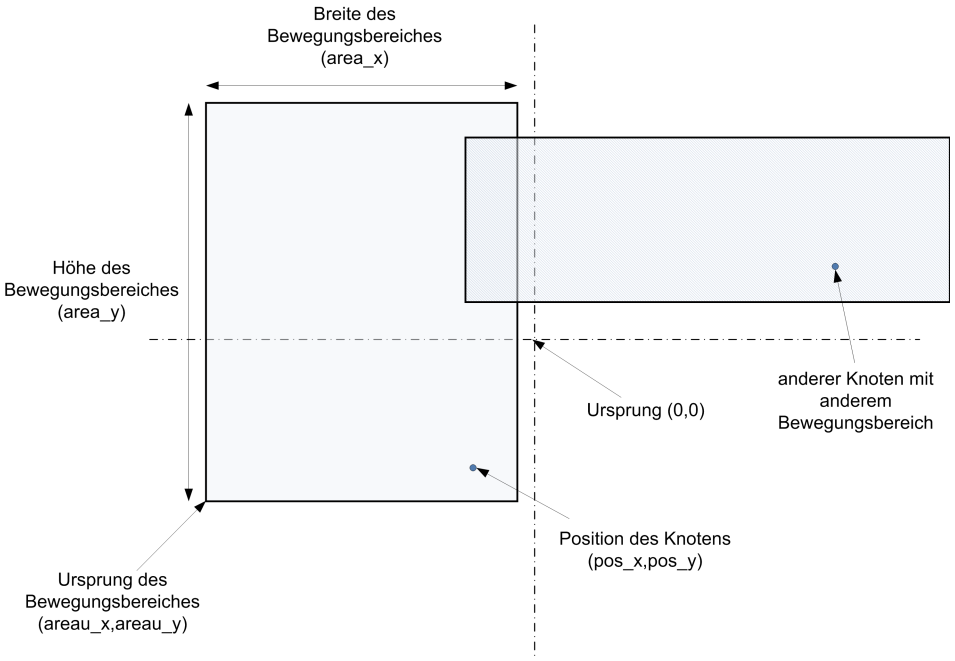

Abbildung 6.4: Konzept des Bewegungsbereiches

Mittels der Funktionen *getX()* und *getY()* kann die Position des Knotens abgefragt werden. Die aktuelle Position des Knotens wird allerdings erst durch das Ausführen der Funktion *updatePosition()* neu berechnet. Diese berechnet aufgrund des in dieser Funktion implementierten Bewegungsalgorithmus und anhand der Differenz der Simulationszeit die neue Position des Knotens. Wichtig in diesem Zusammenhang ist, dass die Bewegung in diesem Model nicht als kontinuierliche Funktion umgesetzt ist, obwohl sie das eigentlich wäre. Die Knoten springen stattdessen von einem Punkt zum nächsten. Wie später noch beschrieben wird, spielt die Position der Knoten für die Übertragung von Events zwischen zwei Knoten durch Objekte der Klasse *WlanPhys* oder *WlanPhysAdvanced* eine wichtige Rolle. Daher müsste korrekterweise bei jedem Event zwischen zwei Knoten, welches nicht zur gleichen Simulationszeit wie das vorherige stattfindet, ein Positionsupdate durchgeführt werden.

Betrachtet man allerdings ein mobiles Netz mit einigen tausenden Knoten, ist es schnell ersichtlich, dass diese Vorgangsweise zu Problemen führen kann. Es müssten, sobald zwei Knoten ein Event austauschen, welches auch nur minimal weiter in der Zukunft liegt als das vorherige, alle Knoten ihre

Position neu berechnen. Es stellt sich die Frage, ob eine so exakte Simulation der Bewegung überhaupt benötigt wird. Da viele Störgrößen sowieso nicht in die Simulation einbezogen werden können, wurde beschlossen, einen geringen Positionierungsfehler zugunsten einer wesentlich besseren Performance in Kauf zu nehmen. Die Position der Knoten wird also nicht bei jedem Event neu berechnet, sondern, wie noch bei der Beschreibung der Klassen *WlanPhys* und *WlanphysAdvanced* zu sehen sein wird, in regelmäßigen Abständen. Die Zeit zwischen zwei Updates ist dabei von der Geschwindigkeit der Knoten abhängig. So kann sichergestellt werden, dass die Ungenauigkeit einen gewissen Grenzwert nicht überschreitet. Dieser Ansatz hat zur Folge, dass die Knoten sich nicht mehr gleichmäßig bewegen sondern in regelmäßigen Abständen auf eine neue Position springen.

In der *ibkmobility*-Klasse ist keine Funktionalität in die *updatePosition()*-Funktion implementiert. Knoten, die dieses Bewegungmodell verwenden, bewegen sich also nicht von der Stelle. Dieses Modell ist daher für WLAN-Knoten mit fixer Position gedacht.

Weitere Funktionen der Klasse sind die Berechnung der Distanz zwischen zwei Knoten sowie die Ausgabe von Informationen für die Loggingfunktion des IBKSims. Die Funktion zur Berechnung der Distanz gibt dabei den euklidische Abstand zurück.

6.2.1.3 ibkmobilitySquare

Die Klasse *ibkmobilitySquare* ist von *ibkmobility* abgeleitet. Abbildung 6.5 zeigt das UML-Diagramm dieser Klasse. Sie unterscheidet sich von der Oberklasse dadurch, dass die *updatePosition()*-Funktion einen einfachen Bewegungsalgorithmus implementiert. Ein Knoten mit diesem Bewegungmodell bewegt sich mit konstanter Geschwindigkeit schräg nach unten bis er eine Grenze des Bewegungsbereiches erreicht. Ist dies der Fall springt er zur gegenüberliegenden Grenze und setzt die Bewegung fort.

6.2.1.4 ibkmobilityBounce

Auch die Klasse *ibkmobilityBounce* ist von *ibkmobility* abgeleitet, das UML-Diagramm ist in Abbildung 6.5 zu sehen. Der implementierte Bewegungsalgorithmus ist ähnlich wie bei *ibkmobilitySquare*. Der Unterschied liegt darin, dass der Knoten bei Erreichen einer Grenze nicht auf die gegenüberliegende Seite springt, sondern dass er von ihr abprallt. Der Knoten ändert seine Bewegungsrichtung, wobei der Winkel, mit dem der Knoten die Grenze erreicht auch dem Winkel entspricht, mit dem sich der Knoten von der Wand

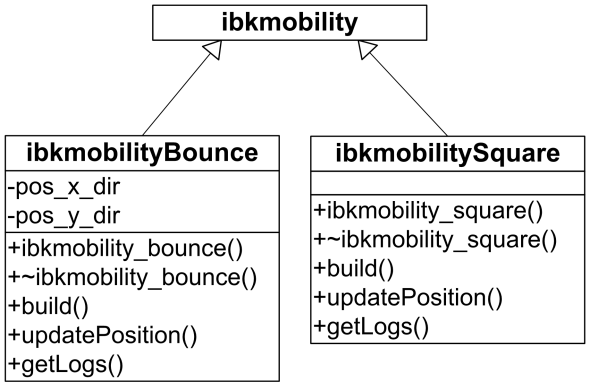

Abbildung 6.5: UML-Diagram von *ibkMobilityBounce* und *ibkMobilitySquare*

anschließend wieder fort bewegt.

6.2.2 Paketmodule

Paketobjekte dienen zum Informationsaustausch zwischen verschieden Simulationsmodulen. Wie bereits in Kapitel 3.2.2 beschrieben wurde, wird ein Zeiger auf ein Paketobjekt in einem Objekt der Klasse *simEvent* von einem Simulationsobjekt zu einer anderen weitergereicht. Im Weiteren werden die in dem Framework vorhandenen Pakettypen beschrieben. Ihnen ist gemeinsam, dass sie alle von der Klasse *cPacket* abgeleitet sind.

6.2.2.1 Ieee80211MacFrame

Wie in Abbildung 6.2 zu sehen ist, dient dieses Objekt zur Kommunikation zwischen zwei Simulationsobjekten vom Typ *Ieee80211Mac*. Es repräsentiert das in [IEE99b] beschriebene MAC-Frame Format.

Da das Framework nicht die komplette Funktionalität des IEEE802.11 Standards umsetzt, werden auch nicht alle Felder des MAC-Frames benötigt. Aus dem "FrameControl"-Feld wird für das Framework nur die Variablen Type und Subtype benötigt. Diese wurden zusammengefasst in der Variable *type*. Zusätzlich wurde aus dem "FrameControl"-Feld auch das "more fragments"-Feld in das Packet aufgenommen, auch wenn es in der aktuellen Version des Frameworks nicht verwendet wird.

Die Adressfelder 1 bis 4 sind vorhanden, ebenso wie das Feld "duration". Das "SequenceControl"-Feld ist auch vorhanden, obwohl es in der aktuellen

Version nicht verwendet wird. Der Framebody wird durch einen Zeiger auf ein weiteres Packet implementiert.

Falls weitere Felder benötigt werden, können sie problemlos noch im Nachhinein in die Klasse hinzugefügt werden. Das restliche Framework muss dafür nicht verändert werden.

6.2.2.2 WlanFrame

Diese Paketklasse dient zur Kommunikation zwischen den Objekten der Klasse *WlanPhys* beziehungsweise *WlanPhysAdvanced*. Sie werden nicht nur verwendet um die physikalische Datenübertragung zu simulieren, sondern auch, um andere Knoten über ein Positionsupdate zu informieren. Diese Klasse erweitert die Oberklasse um die Variable *flag*, die angibt, welche Bedeutung das Paket hat.

6.2.2.3 DummyPacket

Die Klasse *DummyPacket* wird vorallem zum Testen eingesetzt. Sie enthält im Vergleich zur Oberklasse eine Stringvariable, so dass Texte zwischen verschiedenen Simulationsobjekten ausgetauscht werden können.

6.2.2.4 TimePacket

Dieses Paketobjekt repräsentiert kein Paket im eigentlichen Sinne. Es wird verwendet, um Timerfunktionen zu implementierten. Ein Timer in einer eventbasierenden Computersimulation kann realisiert werden, indem ein Simulationsobjekt ein Event, welches in der Zukunft liegt, an sich selbst sendet. Um verschiedene Timer zu implementieren, besitzt dieses Paket eine Variable, mit der der Timer-Typ angegeben werden kann.

Interessant ist das Problem, wie so ein Timer wieder gestoppt wird. Eine Möglichkeit ist, das Event aus der SimQueue zu entfernen, so dass es nie aktiv werden kann. Allerdings muss das Event dazu in der SimQueue gesucht werden, was sich schlecht auf die Performance auswirkt. Daher wurde ein anderer Weg gewählt. Das Paket besitzt die Funktion *kill()*. Wird diese ausgeführt, wird der Timer-Typ auf einen speziellen Wert gesetzt. Dieser Wert repräsentiert einen deaktivierten Timer. Wird das Event nun ausgeführt, erkennt das Zielobjekt, dass der Timer deaktiviert wurde und kümmert sich nicht weiter darum.

6.2.3 Schichtmodule

Im nachfolgenden Abschnitt werden die einzelnen von der Klasse *ibkLayer* abgeleiteten Module erörtert. Dabei werden zuerst die Klassen der Schicht 3 abgehandelt, gefolgt von dem MAC-Modul der Schicht 2. Als letztes werden auch noch die Klassen der Bitübertragungsschicht beschrieben.

6.2.3.1 DummySend

Die auf der Vermittlungsschicht angesiedelte Klasse *DummySend* arbeitet eng mit der Klasse *DummyReceive* zusammen. Die beiden Klassen dienen vor allem zum Testen der darunter liegenden Schichten. Dazu sendet ein Objekt der Klasse *DummySend* ein *DummyPacket*-Objekt an ein *DummyReceive*-Objekt eines anderen Knotens.

In Listing 6.1 ist die Konfiguration eines *DummySend*-Objekts zu sehen. In der ersten Zeile wird eine neue Komponente der Klasse *DummySend* deklariert, was einen Aufruf der *build()*-Funktion von *DummySend* zur Folge hat. In dieser Funktion wird ein neues Objekt erstellt. Anschließend wird das Debug-Level gesetzt, in diesem Fall auf den Wert 1, was zur Folge hat, dass während der Simulation zusätzliche Informationen auf der Konsole ausgegeben werden. Dabei wird vorausgesetzt, dass das Programm auch im Debuggingmode compiliert wurde, da im Release Mode aus Performancegründen die Debugging-Funktionalität deaktiviert ist. In der zweiten Zeile des Listings wird die MAC-Adresse des Knotens angegeben, an den die Nachrichten gesendet werden sollen. Wird diese Zeile weggelassen oder die MAC-Adresse auf 0 gesetzt, hat das zur Folge, dass die Nachrichten gebroadcasted werden.

Ist schließlich die neue Instanz erzeugt und konfiguriert, wird die *init()*-Funktion aufgerufen. In dieser Funktion sendet das *DummySend*-Objekt ein Event an sich selbst, welches eine Zeiteinheit in der Zukunft liegt.

Wird vom Scheduler schließlich das Event ausgeführt, ruft dieser die *handle -Event()*-Funktion des *DummySender*-Objekts auf. In dieser wird nun ein neues *DummyPacket*-Objekt mit der Nachricht "Hello World from $macaddress" erzeugt und an das durch die Konfiguration gesetzte Ziel gesendet. Anschließend sendet das Objekt ein weiteres Event an sich selbst, so dass

```
<component name="Sender" type="dummysender" debug="1">
        <destination value="3" />
</component>
```

Listing 6.1: Konfiguration von dummy_sender

```
<component name="Empfänger" type="dummyreceive" debug="1">
</component>
```

Listing 6.2: Konfiguration von dummy_receive

60 Zeiteinheiten später die *handelEvent()*-Funktion wieder aufgerufen wird. Das *DummySend*-Modul macht daher nichts anderes, als alle 60 Zeiteinheiten ein neues *DummyPacket*-Objekt zu versenden.

6.2.3.2 DummyReceive

Die Klasse *DummyReceive* ist das Gegenstück zu *DummySend*. Ihre Aufgabe ist es, *DummyPacket*-Objekte entgegen zu nehmen und falls der entsprechende Debuglevel gesetzt ist, bei Erhalt der Nachricht eine Meldung über die Konsole auszugeben.

Entsprechend einfach ist die in Listing 6.2 dargestellte Konfiguration. Zur Erzeugung des Objekts wird die *build()*-Funktion der Klasse aufgerufen, die eine neue Instanz erzeugt und das Debuglevel der Instanz entsprechend der Konfiguration setzt.

6.2.3.3 WlanPing

Diese Klasse beinhaltet die Funktionalität der Klassen *DummySend* und *DummyReceive* und erweitert diese. Zur Instanzierung wird die *build()*-Funktion der Klasse aufgerufen, die ein neues Objekt erzeugt und entsprechend der Daten aus dem Konfigurationsfile konfiguriert. Der relevante Ausschnitt aus dem Konfigurationsfile ist in Listing 6.3 zu sehen. Neben der Angabe einer Zieladresse kann sowohl für die Größe der Pakete als auch für die Rate, mit der die Pakete erzeugt werden, eine Verteilungsfunktion verwendet werden.

Ansonsten gibt es keine Unterschiede zu den beiden bereits beschriebenen Dummy Modulen. Durch die *build()*-Funktion wird ebenfalls die *init()*-Funktion aufgerufen, die wiederum das erste Event generiert. Dadurch wird immer wieder die *handleEvent()*-Funktion durch den Scheduler ausgeführt, die jedesmal ein *cPacket*-Objekt erzeugt und ein neues Event an sich selbst schickt. Empfängt das Modul ein Event welches ein *cPacket* beinhaltet, wird, sofern das entsprechende Loglevel gesetzt ist, eine Nachricht auf der Konsole ausgegeben.

6.2.3.4 Ieee80211Mac

Diese Klasse implementiert das IEEE 802.11 MAC-Protokoll. Die Aufgabe dieses Moduls ist es, den Zugriff auf die Bitübertragungsschicht zwischen den Knoten zu koordinieren und so den Datenaustausch zwischen den Knoten zu unterstützen.

Objekte dieser Klasse erhalten von dem darüber liegenden Layer *Ieee80211-MacFrame*-Objekte, welche weitere *cPaket*-Objekte eingebettet haben können. Da es nicht immer möglich ist, diese Objekte sofort zu übertragen, werden alle *Ieee80211MacFrame*-Objekte, die von der oberen Schicht übergeben werden, in einem Puffer zwischengespeichert.

Zentraler Teil des Moduls ist eine Zustandsmaschine, die versucht nach dem MAC-Protokoll die einzelnen *Ieee80211MacFrame*-Objekte aus dem Puffer zu übertragen. Eng damit verbunden ist die *handleEvent()*-Funktion sowie verschiedene weitere Funktionen, die von dieser Funktion verwendeten werden. Diese können mittels Flags und Eventnachrichten Einfluss auf die Zustandsmaschine nehmen. Bevor jedoch genauer darauf eingegangen wird, zuerst noch zur Konfiguration.

Die aktuelle Version dieser Klasse bietet so gut wie keine Einstellungsmöglichkeiten. Lediglich der Debuglevel kann angegeben werden. Nachdem eine neue Instanz der Klasse mit Hilfe der *build()*-Funktion erstellt und der Debuglevel konfiguriert wurde, wird automatisch eine MAC-Adresse für diesen Knoten generiert. Konstanten, die sich je nach verwendetem Standard der 802.11-Serie [IEE99b] und eingesetztem Codierungsverfahren gegebenenfalls unterscheiden können, lassen sich nicht über das Skript konfigurieren, sondern sind direkt im Header-File zu ändern. Die Werte sind standardmäßig auf die des 802.11b Protokolls [IEE99a] unter Verwendung des "High rate Direct Sequence Spread Spectrum" Verfahrens (DSSS/HR) gesetzt. Listing 6.4 zeigt die Konfiguration des MAC-Moduls.

```
<component name="ping0" type="wlanping" debug="0">
  <distribution purpose="pcksize" type="uniform">
   <low value="1"/>
    <high  value="1500" />
  </distribution>
  <distribution purpose="genrate" type="negexp">
    <distconf mean="50"/>
  </distribution>
  <destination value="3" />
</component>
```

Listing 6.3: Konfiguration von WlanPing

```
<component name="mac" type="ieee802_11mac" debug="1">
</component>
```

Listing 6.4: Konfiguration von Ieee80211Mac

Wie bei allen Simulationsobjekten wird die *handleEvent()*-Funktion vom Scheduler aufgerufen, wenn ein Event an das Simulationsobjekt gesendet wurde. Die *Ieee80211Mac*-Klasse unterscheidet hierbei zwischen drei Eventtypen: *Timer1* [1], *Down* und *Up*.

Bei Ereignissen vom Typ *Timer1* handelt es sich um Events, die das Simulationsobjekt an sich selbst schickt, um verschiedene Timer-Funktionalitäten zu implementieren. Dabei kommt das bereits beschriebene *TimePacket*-Objekt zum Einsatz, welches in den Events eingebettet ist. Durch dieses Objekt werden insgesamt sechs Timertypen realisiert. Bis auf die Typen *Tnone* und *Tnav* lösen alle Timertypen ein Event in der Zustandsmaschine aus, was durch das Ausführen der Funktion, die den aktuellen Zustand darstellt, sowie durch die Übergabe einer entsprechenden Flag erfolgt. In Tabelle 6.3 werden die einzelnen Timer-Typen beschrieben.

Events vom Typ *Down* sind Ereignisse, die von der darüber liegenden Schicht stammen. Diese Ereignisse erhalten ein Paket-Objekt vom Typ *Ieee80211 -MacFrame*, welches weitere Paket-Objekte beinhalten kann. Wird ein solches Event erkannt, ruft die *handleEvent()*-Funktion die *down()*-Funktion auf, der das Simulationsevent übergeben wird. Diese extrahiert das *Ieee80211 -MacFrame*-Objekt, setzt die Adresse der Quelle auf den Wert der eigenen MAC-Adresse sowie den Typ des Frames auf *PtData*, wodurch der Frame von nun an als Datenframe gekennzeichnet ist. Anschließend wird der Frame in eine Queue gereiht und das *EnewPacket*-Event bei der Zustandsmaschine ausgelöst. Falls allerdings eine gewisse Anzahl an Paket-Objekten bereits in der Queue vorhanden ist, wird das *Ieee80211MacFrame*-Objekt einfach verworfen.

Die dritte Art von Events, die auftreten können, sind Events von einem sich unter dem MAC-Layer befindenden Simulationsobjekt, welches Events nach oben weiterreicht. Daher auch die Bezeichnung "Up". Diese Events beinhalten Paket-Objekte vom Typ *WlanFrame*. Die Verarbeitung dieser Events wird durch die Funktion *up()* übernommen, die die *WlanFrame*-Objekte aus den Events extrahiert und anschließend, je nachdem auf welchen Wert die Flag des Objekt gesetztes ist, unterschiedlich reagiert. Mittels der *Wlan*

[1] Zur besseren Lesbarkeit werden die Typen im Weiteren nicht komplett groß geschrieben, auch wenn im Code selbst etwa "Timer1" "TIMER1" heißt

Typ	Beschreibung
Tnone	Dieser Typ repräsentiert einen deaktivierten Timer. Daher ist keine weitere Aktion nötig.
Tnav	Dieser Timer-Typ gibt an, dass die durch den Network Allocation Vector (NAV) gegebene Zeit abgelaufen ist. Es wird die *resetNAV*-Funktion ausgeführt, die das virtuelle Carrier-Sensing beendet.
Tdifs	Wird verwendet um die Zeitspanne von der Dauer eines DCF interframe spaces zu warten. Es löst ein *EtimerDifs*-Event an der Zustandsmaschine aus.
Tack	Gibt an, dass die Zeitspanne für ein ACK abgelaufen ist. Es löst ein *EtimerAck*-Event aus.
Tbackoff	Wird verwendet um die Backoff-Zeit abzuwarten. Löst ein *EtimerBackoff*-Event aus.
Tcts	Dieser Timer löst ein *EtimerCts*-Event aus, welches angibt, dass es zu einem Timeout beim Warten auf die CTS-Nachricht gekommen ist.

Tabelle 6.3: Beschreibung der Timer Typen

-*Frame*-Klasse informiert das Objekt der Bitübertragungschicht das MAC-Objekt, ob eine Übertragung begonnen wurde oder ob der Kanal wieder frei ist. Anhand der Flag kann die Bedeutung des *WlanFrame*-Objekts erkannt werden. Dabei lässt sich noch genauer differenzieren, ob die letzte Übertragung eine eingehende oder ausgehende Übertragung war, beziehungsweise ob ein Paket empfangen wurde oder nicht.

Ist die Flag des *WlanFrame*-Objekts auf *StartTm* gesetzt bedeutet dies, dass der Übertragungskanal ab diesem Zeitpunkt belegt ist, also dass gerade mit einer Übertragung begonnen wurde. Diese Information wird verwendet, um den aktuellen Status des Übertragungskanals intern zu speichern. Weiters wird die Zustandsmaschine über dieses Event informiert.

Ein *WlanFrame*-Objekt mit der Flag *StopSelf* bedeutet, dass die von dem Knoten gestartete Übertragung beendet wurde. Allerdings bedeutet das nicht automatisch, dass der Kanal dadurch wieder frei ist. Es ist nämlich denkbar, dass eine andere Station zwischenzeitlich zu senden begonnen hat, wodurch der Kanal weiterhin belegt ist. Daher wird in diesem Fall nur die Zustandsmaschine darüber informiert, dass die eigene Übertragung beendet wurde. Ist nach Beendigung der Übertragung der Kanal wirklich frei, trifft gleichzeitig ein weiteres *WlanFrame*-Objekt ein, dessen Flag den

Wert *StopTm* besitzt. Auch darüber wird die Zustandsmaschine mit einem *EfreeChannel*-Event informiert. Die Änderung des Zustandes des Kanals wird wieder intern gespeichert.

Wenn in einem *WlanFrame*-Objekt das Flag auf *StopTmInvalid* gesetzt wurde, bedeutet das, dass vom Empfänger zwar etwas empfangen wurde, das Signal allerdings gestört war. Gründe dafür sind etwa Kollisionen oder sonstige Störungen bei der Übertragung. Das Verhalten der Zustandsmaschine ist die selbe wie bei Erhalt der Flag *StopTm*, allerdings wird zusätzlich intern vermerkt, dass die letzte Übertragung fehlerhaft war.

Schließlich gibt es auch die Möglichkeit, dass eine Übertragung mit dem erfolgreichen Empfang eines Pakets endet. Dies wird mit der Flag *StopTmValid* im *WlanFrame*-Objekt vermerkt. Ein solches Objekt hat auch das empfangenen *Ieee80211MacFrame*-Objekt eingebettet. Im Weiteren wird unterschieden, um welchen Typ von *Ieee80211MacFrame*-Objekt es sich handelt. Unabhängig davon wird vermerkt, dass es eine erfolgreiche Übertragung gegeben hat und dass der Kanal wieder frei ist.

Handelt es sich bei dem empfangenen *Ieee80211MacFrame*-Objekt um einen Datenframe, ist er also vom Typ *PtData*, wird unterschieden ob dieser Frame gebroadcasted wurde oder nicht. Ist dies der Fall, wird das *Ieee80211 -MacFrame*-Objekt einfach an die darüber liegende Schicht weitergereicht und die Zustandmaschine über das Eintreffen eines Datenframes mittels eines *Edata*-Events informiert. Ist der Frame an einen speziellen Knoten gerichtet, wird anhand der MAC-Adresse überprüft, ob der aktuelle Knoten überhaupt das Ziel ist. Ist dies nicht der Fall, kann der Frame einfach verworfen werden. Ansonsten wird gemäß des 802.11-Protokolls ein Acknowledge etwas später zurückgesendet.

Ein weiterer Typ ist *PtAck*, ein Acknowledge Frame. Wird ein Acknowledge empfangen, welches an den aktuellen Knoten adressiert ist, wird nur die Zustandsmaschine mittels eines *Eack*-Events darüber informiert.

Wird ein RTS-Frame, also ein *Ieee80211MacFrame*-Objekt vom Typ *PtRts*, empfangen, wird wiederum unterschieden, ob er an den aktuellen Knoten geschickt wurde oder nicht. Wenn das der Fall ist, wird ein *Ieee80211MacFrame*-Objekt des Typs *PtCts* zurück geschickt, wobei der Wert im Duration Feld entsprechend gesetzt wird (vgl. Kapitel 4.2.2). Andernfalls wird intern der "Network allocation vector" auf einen aus dem Wert des Duration Feldes berechneten Wert gesetzt, was auch der Fall ist, wenn ein CTS-Frame empfangen wurde, der nicht an den Knoten adressiert war. Ist der CTS-Frame an den Knoten adressiert, wird das *Ects*-Event in der Zustandsmaschine

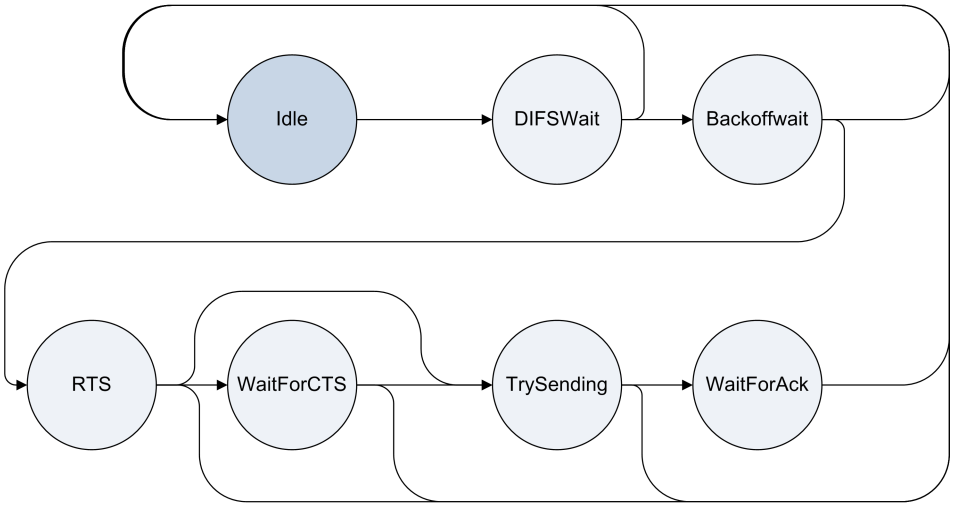

Abbildung 6.6: Zustandsmaschine der Klasse IEEE802_11MAC

ausgelöst.

Im Weiteren wird nun auf die Zustandsmaschine des Moduls eingegangen, die den Kern der Klasse bildet. Sie besteht aus einer Reihe von Funktionen, die alle ein Event als Argument übernehmen. Dabei handelt es sich allerdings nicht um ein Simulationsevent, sondern nur um einen Aufzählungstyp. Der momentan aktive Zustand wird durch einen Funktionszeiger markiert. Dadurch ist es möglich, auch wenn eine Funktion nicht weiß, welche Funktion zur Zeit die aktive Funktion der Zustandsmaschine ist, über den Funktionszeiger Events wie etwa *Eack* oder *Ects* an die Funktion zu senden. Dies geschieht durch einen einfachen Aufruf der Funktion über den Pointer mit dem Event als Argument.

Der Aufbau der Zustandsmaschine ist in Abbildung 6.6 dargestellt. Der Ursprungszustand ist der Zustand *Idle*. Falls es während der Übertragung zu irgendeinem Fehler kommt, kehrt die Zustandsmaschine in diesen Zustand zurück. Falls ein Frame in dem Puffer vorhanden ist oder die Backoffzeit nach dem Senden abgewartet werden muss und gewisse Ereignisse auftreten, wechselt die Zustandsmaschine in den Zustand *Difs Wait*. Ereignisse die diesen Wechsel auslösen sind etwa Ereignisse, die darauf hindeuten, dass der Kanal zur Zeit frei ist, oder dass ein neues Paket-Objekt eingetroffen ist.

Im Zustand *Difs Wait* wartet die Maschine den "DCF interframe space" oder gegebenenfalls den "Extended interframe space" ab. Sobald der Übergang zu

diesem Zustand stattfindet, was durch ein *Etransfer*-Event gekennzeichnet wird, wird überprüft, ob der Kanal frei ist. Ist dies nicht der Fall, kehrt die Zustandsmaschine in den Ursprungszustand zurück. Falls er aber frei sein sollte, wird ein Timer gestartet, so dass sich das Simulationsobjekt nach Ablauf der Zeitspanne von selbst wieder aktiviert. Ist der Timer abgelaufen, wird ein weiteres Mal geprüft, ob der Kanal frei ist. Auf diese Weise kann sichergestellt werden, dass in der Zwischenzeit kein anderer Knoten, der sich in Sendereichweite befindet, mit einer Übertragung begonnen hat. Wenn der Kanal noch immer frei ist, wird in den Zustand *Backoffwait* gewechselt.

Auch in diesem Zustand wird nach dem Transfer ein Timer gestartet, der den Knoten nach Ablauf der Backoffzeit reaktiviert. Falls in der Zwischenzeit der Kanal belegt wird, erfährt der Zustand davon durch das *Ebusy-Channel*-Event. Darauf wird der Timer gelöscht und eine neue Backoffzeit berechnet, die von der bereits vergangenen Wartezeit abhängig ist. Anschließend, kehrt die Maschine in den Ursprungszustand zurück um wieder auf das Freiwerden des Kanals zu warten. Wird die Wartezeit jedoch nicht unterbrochen und der Simulationsknoten durch den Timer aktiviert, erfolgt, falls ein Paket-Objekt in dem Puffer vorhanden ist, ein Wechsel in den Zustand *Rts*. Ist die Putter leer, was bedeutet, dass der Backoff nach einer Übertragung statt fand, wird wieder in den Ursprungszustand gewechselt.

In diesem Zustand wird anfangs entschieden, ob es nötig ist, das RTS/CTS-Verfahren zu verwenden oder ob das Paket-Objekt sofort übertragen werden kann. Dies ist abhängig von der Größe des Frames. Wenn er direkt übertragen werden kann, erfolgt ein Zustandswechsel nach *TrySending*. Andernfalls wird der Network Allocation Vector berechnet und ein RTS-Frame mit dem Vektor an den Zielknoten geschickt. Nachdem die Zustandsmaschine das Event erhält, dass der Sendevorgang beendet wurde, wird in den Zustand *WaitForCTS* gewechselt.

Hier wird auf die Antwort des Knotens, an den der RTS-Frame gesendet wurde, gewartet. Dazu wird ein Timer aktiviert, der nach Ablauf des Zeitfensters, in dem die Antwort erwartet wird, den Knoten reaktiviert. Entsprechend wird, nachdem der Timer gestartet wurde, auf das Eintreffen des CTS-Frames gewartet. Wird die Zustandsmaschine über das Eintreffen des Frames mittels des *Ects*-Events informiert, kann in den *TrySending*-Zustand gewechselt werden, wo die Übertragung des Daten-Frames stattfindet. Bleibt die Antwort aus, reaktiviert der Timer die Zustandsmaschine. Das Ausbleiben der Antwort wird vermerkt, damit, falls der Zielknoten nicht in Reichweite ist, das Paket-Objekt nach einiger Zeit verworfen werden kann. Danach erfolgt wieder ein Wechsel in den *idle*-Zustand.

In *TrySending* erfolgt die Übertragung des Daten-Frames. Ist die Übertragung zu Ende, erfolgt ein Wechsel zu *WaitForAck*, wo auf die Empfangsbestätigung des Empfängers gewartet wird. Dazu wird wieder ein Timer aktiviert, der das Simulationsobjekt nach Ablauf des Zeitfensters über das Ausbleiben der Empfangsbestätigung informiert. Bleibt die Bestätigung aus, wird dies vermerkt, was unter anderem eine Auswirkung auf die Dauer des Backoffs hat. Wird die Empfangsbestätigung empfangen, wird der Frame aus der Warteschlange entfernt, und vermerkt, dass anschließend der Backoffalgorithmus ausgeführt werden soll. Abschließend wird wieder in den Zustand *idle* gewechselt, wo der Vorgang von Neuem beginnt.

6.2.3.5 WlanPhys

Diese Klasse dient zur Simulation der physikalischen Datenübertragung. Die einzelnen *WlanPhys*-Objekte tauschen dazu Objekte der Klasse *WlanFrame* aus. Diese Objekte werden sowohl zur Simulation der physikalischen Datenübertragung verwendet, als auch zur Konfiguration der Knoten. Diese Konfigurationsnachrichten entsprechen keinen Nachrichten, die in Wirklichkeit in einem solchen System vorkommen. Entsprechend interferieren sie nicht mit der eigentlichen Datenübertragung, werden ohne Zeitverlust übertragen und kommen immer an. Ihre genaue Verwendung wird etwas später beschrieben.

Zur Simulation der physikalischen Datenübertragung wird ein sehr einfaches Model verwendet. Jeder Sender besitzt eine gewisse Reichweite. Soll nun ein *Ieee80211MacFrame*-Objekt übertragen werden, wird dieses Frame-Objekt in einen *WlanFrame*-Objekt eingebettet. Eine Kopie dieses *WlanFrame*-Objekts wird an jeden Knoten gesendet, der sich in Reichweite des Senders befindet. Dieses Model entspricht dem in [Kotz04] beschrieben Flat Earth Model. Da die Übertragung in Wirklichkeit einige Zeit dauert, werden zwei *WlanFrame*-Objekte übertragen. Zu Beginn der Übertragung wird ein *WlanFrame*-Objekt gesendet, in dem die Flag auf *StartTm* gesetzt ist. Empfängt ein Knoten einen solchen Frame, erkennt er, dass der Kanal ab diesem Zeitpunkt belegt ist. Ist die Übertragung zu Ende, wird ein weiterer *WlanFrame* übertragen, der das eigentliche *Ieee80211MacFrame*-Objekt enthält. Bei diesem ist die Flag auf *StopTm* gesetzt. Wie in Abbildung 6.7 zu sehen ist, erkennt ein Empfänger, wenn er zwei *WlanFrame*-Objekte hintereinander empfängt, die beide den Start einer Übertragung markieren, dass er im Moment zwei Übertragungen gleichzeitig empfangen würde, was einer Kollision entspricht. Der Trick, mittels zwei Events die Übertragung zu simulieren, dient also vor allem dazu auch Kollisionen simulieren zu können.

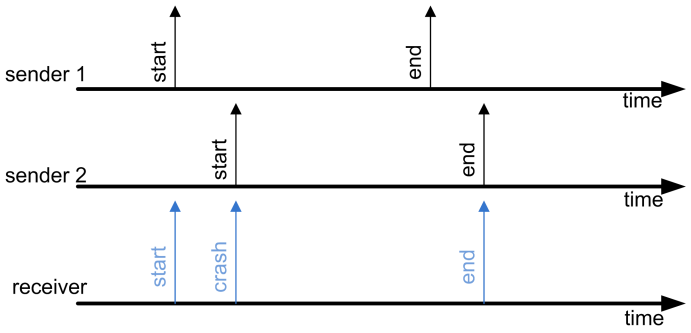

Abbildung 6.7: Erkennung eines Crashs durch die *WlanPhys* Klasse

Empfängt also ein Knoten ein *WlanFrame*-Objekt, welches den Start einer Übertragung markiert, wird ein interner Zähler, der die Anzahl der momentanen Übertragungen zählt, inkrementiert. Sofern zuvor keine Übertragung stattgefunden hat, wird das *WlanFrame*-Objekt mit einer auf *StartTm* gesetzten Flag an die darüber liegende MAC-Schicht gesandt. Wie bereits beschrieben, wird diese dadurch darüber informiert, dass ab diesem Zeitpunkt der Kanal belegt ist. Findet mehr als eine Übertragung statt, wird die MAC-Schicht nicht darüber informiert, da ein Empfänger in Wirklichkeit nicht erkennen kann, wie viele Stationen gleichzeitig versuchen eine Nachricht zu ermitteln. Er kann in so einem Fall einfach nicht das Signal decodieren. Intern wird ein solcher Crash allerdings vermerkt.

Wird ein *WlanFrame*-Objekt mit einer auf *StopTm* gesetzten Flag empfangen, wird der Übertragungszähler wieder dekrementiert. Ist der Absender der Knoten selber, bedeutet das, dass der aktuelle Knoten zu senden aufgehört hat, was allerdings nicht gleichbedeutend damit ist, dass der Kanal deswegen frei sein muss. Das Objekt, welches die MAC-Schicht simuliert, wird allerdings darüber informiert, dass der Knoten selbst nicht mehr sendet, was wieder durch ein *WlanFrame*-Objekt geschieht. Bei diesem ist das Flag auf *StopTmSelfe* gesetzt. Erreicht der Übertragungszähler Null, bedeutet das, dass der Kanal wieder frei ist. Es wird nun unterschieden, ob der Kanal durch das eigene Senden belegt war, oder ob ein Signal empfangen wurde. Wurde ein Signal empfangen, wird überprüft, ob es zu einer Kollision gekommen ist oder nicht. Je nachdem welcher Fall zutrifft, wird ein *WlanFrame*-Objekt mit einem für jeden Fall anders gesetzten Flag an die MAC-Schicht übertragen, wobei im Fall eines erfolgreichen Empfangs ein *Ieee80211MacFrame*-Objekt im *WlanFrame*-Objekt eingebettet ist.

```
<component name="wlan" type="wlanphys" debug="0">
  <mobilitymodel type="square" />
  <speed value="1" />
  <range value="200" />
  <location x="13.0" y="99.0" />
  <area x="10" y="20" width="100" hight="100" />
</component>
```

Listing 6.5: Konfiguration von WlanPhys

Es macht wenig Sinn bei jedem Sendeversuch für jeden Knoten zu eruieren, ob er sich in Reichweite befindet. Stattdessen verwaltet jeder Knoten eine interne Liste mit allen Knoten, die sich in Reichweite befinden. Ändert der Knoten die Position, wird die Liste neu erstellt, indem er zu allen Knoten den Abstand berechnet und die, die in Reichweite sind, in die Liste einträgt. Gleichzeit informiert er alle Knoten darüber, dass er eine Positionsänderung vorgenommen hat, so dass auch sie die neue Distanz zu ihm berechnen können um ihn gegebenenfalls in die eigene Liste einzutragen oder ihn aus dieser zu entfernen. Dies geschieht durch das Senden eines speziellen *WlanFrame*-Objekt, wobei , wie bereits erwähnt, das Senden dieses Objekts nicht mit der Datenübertragung interferiert. Um die Bewegung zu simulieren, bindet jedes *WlanPhys*-Objekt bei seiner Initialisierung ein Bewegungsobjekt ein, welches auch sofort konfiguriert wird. Abhängig von der Bewegungsgeschwindigkeit, wird die in Kapitel 6.2.1 beschriebene *updatePosition()*-Funktion regelmäßig ausgeführt. Ist die neue Position bekannt, werden, wie bereits beschrieben, die Listen mit den Knoten in Reichweite aktualisiert. Da sich die Verbindungen laufend ändern können, musste das Konzept des IBKSims, dass die Knoten fix verbunden sind, fallen gelassen werden.

Da diese Klasse auch die Konfiguration des verwendeten Bewegungsmodels übernimmt, gibt es relativ viele Konfigurationsparameter. Prinzipiell kann dieses Modul zur Simulation verschiedener Funkstandards verwendet werden. Allerdings ist es defaultmäßig so konfiguriert, dass es dem IEEE802.11b Standard bei 11 Mbps entspricht [IEE99a]. Soll ein anderer Standard verwendet werden, müssen die entsprechenden Konstanten im Code angepasst werden.

Listing 6.5 zeigt eine mögliche Konfiguration. Neben dem üblichen Debuglevel wird angegeben, welches Bewegungsmodell verwendet werden soll, welche Reichweite der Sender besitzt, wie schnell sich der Knoten bewegt, wo seine Startposition ist und in welchem Bereich er sich bewegen kann.

6.2.3.6 WlanPhysAdvanced

Diese Unterklasse von *WlanPhys* erweitert die Oberklasse in Hinblick auf die Simulation der physikalischen Datenübertragung. Dazu werden die drei in Kapitel 5.1.1 beschriebenen Radiomodelle verwendet. Das Senden von Daten funktioniert genau gleich wie im *WlanPhys*-Modul, es wird ein Start- und ein Ende-Event, welches auch das eingebetteten *Ieee80211MacFrame*-Objekt enthält, an alle Knoten, die sich innerhalb eines gewissen Radius um den Knoten befinden, gesendet. Auf alle Knoten außerhalb dieses Radius hat die Datenübertragung keinen Einfluss.

Beim Eintreffen eines *WlanFrame*-Objekts, in dem die Flag auf *SartTm* gesetzt ist, wird zuerst die Empfangsstärke berechnet, wobei eines der drei Radiomodelle zum Einsatz kommt. Anschließend wird überprüft, ob das Signal über dem Sensibilitätslevel des Empfängers liegt. Ist dies nicht der Fall, wird es zum Rausch- und Interferenzwert addiert. Falls es empfangen werden kann und zur Zeit keine andere Übertragung empfangen wird, wird der MAC-Layer über den belegten Kanal informiert und es wird vermerkt, dass das Signal von dem Knoten, der das *WlanFrame*-Objekt gesendet hat, im Moment empfangen wird. Tritt ein weiteres Signal über dem Sensibilitätslevel auf, wird auch dieses Signal zum Rausch und Interferenzwert addiert. Dies hat später zur Folge, dass aufgrund des geringen Signal-Rauschabstandes das Paket nicht empfangen werden konnte.

Wird ein *WlanFrame*-Objekt mit einer auf *StopTm* gesetzten Flag empfangen, wird die Signalstärke vom aktuellen Rauschpegel abgezogen, da ja das Signal nicht mehr gesendet wird. Gehört das *WlanFrame*-Objekt zu der aktuell empfangenen Sendung, wird der minimale Rausch-Signalabstand, den es während des Empfangs gegeben hat berechnet, außer das Objekt stammt von einer eigenen Übertragung des Knotens. In diesem Fall wird die MAC-Schicht über das Ende der Übertragung unterrichtet. Gibt es schließlich kein Signal mehr, welches über dem Sensibilitätslevel des Empfängers liegt, wird aufgrund der Signalstärke und des Rauschabstandes, sowie einiger anderer Parameter, wie in Kapitel 5.1.1 beschrieben, die Bitfehlerwahrscheinlichkeit berechnet. Abhängig von dieser Wahrscheinlichkeit wird schließlich ein gültiges oder ein fehlerhaftes *Ieee80211MacFrame*-Objekt an die darüber liegende Schicht weitergereicht.

An dieser Stelle sei noch auf eine Besonderheit bei der Berechnung der Signalstärke sowie auf die Interferenz der Signale hingewiesen. Durch die Verwendung eines zusätzlichen Radius kann ein maximaler Wirkbereich einer Übertragung festgesetzt werden. Für Knoten außerhalb dieses Radius wird die Interferenz nicht mehr eingerechnet, was sich positiv auf die Performan-

ce der Simulation auswirkt. Damit die Simulation realistisch bleibt, muss ein entsprechend großer Radius gewählt werden, so dass diese Ungenauigkeit vernachlässigt werden kann. Die Signalstärke zwischen zwei Knoten wird im Gegensatz zu anderen Simulatoren nur bei der ersten Übertragung berechnet und solange zwischengespeichert, bis einer der beiden beteiligten Knoten seine Position verändert hat. Das wirkt sich sowohl positiv auf die Performance aus, als auch ist dieses Verfahren realistischer, da sich die Signalstärke zwischen zwei fixen Punkten nicht bei jeder Übertragung komplett ändert.

Auch die Berechnung der Bitfehlerrate ist in diesem Modul für den IEEE802.11b Standard implementiert. Das Modul kann aber, durch Verwendung der entsprechenden Berechnung, für andere Übertragungsstandards verwendet werden.

Die Konfigurationsmöglichkeiten des Moduls sind recht umfangreich und abhängig vom verwendeten Radiomodell. Zusätzlich zu den Parameter, die auch im *WlanPhys*-Model vorhanden sind, kommen noch verschiedene Werte für die Radiomodelle hinzu, die da wären die Sensibilität des Empfängers, der Empfangsgewinn der Sende und der Empfangsantenne, die Höhe der Antenne, sowie eine Reihe weiterer Parameter, die bei der Beschreibung der Radiomodelle in Kapitel 5.1.1 erörtert werden. In Listing 6.6 ist eine mögliche Konfiguration zu sehen. Im Falle, dass irgendwelche Parameter nicht angegeben werden, werden die Werte angenommen, die der Standardeinstellung von NS2 entsprechen. An dieser Stelle sei noch darauf hingewiesen, dass es meistens wenig Sinn macht, in einem Netz sowohl *WlanPhys* als auch *WlanPhysAdvanced*-Modelle zu mixen oder unterschiedliche Radiomodelle zu verwenden.

6.3 Beispielkonfiguration

Nachdem die einzelnen Module erörtert wurden, bietet es sich an, anhand eines kompletten Konfigurationsbeispiels die Konfiguration eines WLAN-Netzes zu zeigen. Die in Listing 6.7 zu sehende Konfiguration erstellt ein Netz mit zwei WLAN-Knoten. Ein Knoten ist dabei als Sender konfiguriert, der regelmäßig Pakete aussendet, die gegebenfalls vom anderen Knoten, der als Empfänger fungiert, empfangen werden. Beide besitzen gleiche Sendemodule mit gleicher Reichweite. Sie unterscheiden sich allerdings in Hinblick auf ihr Bewegungsmodell sowie auf ihre Bewegungsgeschwindigkeit. Man kann erkennen, dass das DummySender- beziehungsweise DummyReceiver-Modul mit dem MAC-Modul des jeweiligen Knotens verbunden wird, welches wiederum mit dem WlanPhys-Modul verlinkt wird. Natürlich kann ein

```
<component name="wlan2" type="wlanphysadvanced" debug="3">
  <sensitivity value="1"/>
  <powermodel value="1"/>
  <powertx value="0.28183815"/>
  <gaintx value="1"/>
  <gainrx value="1"/>
  <loss value="1"/>
  <freq value="914000000"/>
  <height value="1.5"/>
  <pathlossexponent value="2"/>
  <variance value="4"/>
  <mobilitymodel type="square" />
  <speed value="1" />
  <range value="200" />
  <location x="13.0" y="99.0" />
  <area x="10" y="20" width="100" hight="100" />
</component>
```

Listing 6.6: Konfiguration von WlanPhysAdvanced

Netz aus wesentlich mehr Knoten bestehen. Das Framework unterstützt Netze mit mehreren tausend Knoten sofern der Rechner genügend Speicher verfügt.

Interessant in der Konfiguration ist auch die Linkliste. Diese bleibt im Gegensatz zu der in Listing 3.1 gezeigten Konfiguration leer, da sich die Knoten automatisch verbinden, wenn sie in Reichweite sind. Das ist der große Unterschied zwischen den bisherigen Simulationsmodellen des IBKSims und denen des in dieser Arbeit vorgestellten Frameworks. Anhand der Konfiguration des Log-Objekts lässt sich erkennen, dass von beiden Knoten nur von der physikalischen und von der MAC-Schicht Daten während der Simulation gesammelt werden.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<simulation xsi:schemaLocation="http://www.ibk.tuwien.ac.at
  http://jupiter.ibk.tuwien.ac.at/IBKSimulator.xsd"
  xmlns="http://www.ibk.tuwien.ac.at"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <paramlist>
    <duration value="5.0"/>
    <initialseed value = "111"/>
  </paramlist>

  <multinetwork>
    <network name="net0">
      <node name="node0">
```

```
        <component name="Sender0" type="dummysender" debug="7">
        </component>
        <component name="mac0" type="ieee802_11mac" debug="15">
        </component>
        <component name="wlan0" type="wlanphysadvanced" debug="3">
          <sensitivity value="1"/>
          <powermodel value="2"/>
          <powertx value="0.28183815"/>
          <gaintx value="1"/>
          <gainrx value="1"/>
          <loss value="1"/>
          <freq value="914000000"/>
          <height value="1.5"/>
          <pathlossexponent value="2"/>
          <variance value="6"/>

          <mobilitymodel type="bounce"/>
          <speed value="2"/>
          <range value="200" />
          <location x="13.0" y="99.0" />
          <area x="10" y="20" width="100" hight="100" />
        </component>
        <clink src="wlan0" dst="mac0" direction="both"/>
        <clink src="mac0" dst="Sender0" direction="both"/>
      </node>

      <node name="node1">
        <component name="Receiver1" type="dummyreceiver debug="7">
        </component>
        <component name="mac1" type="ieee802_11mac" debug="15">
        </component>
        <component name="wlan1" type="wlanphysadvanced" debug="3">
          <sensitivity value="1"/>
          <powermodel value="2"/>
          <powertx value="0.28183815"/>
          <gaintx value="1"/>
          <gainrx value="1"/>
          <loss value="1"/>
          <freq value="914000000"/>
          <height value="1.5"/>
          <pathlossexponent value="2"/>
          <variance value="6"/>

          <mobilitymodel type="square"/>
          <speed value="4"/>
          <range value="200" />
          <location x="35.0" y="35.0" />
          <area x="10" y="20" width="150" hight="100" />
          </component>
        <clink src="wlan1" dst="mac1" direction="both"/>
        <clink src="mac1" dst="Receiver1" direction="both"/>
      </node>
    </network>
  </multinetwork>

  <linklist>
<!-- empty -->
  </linklist>
```

```
  <loglist>
    <logging type="file" interval="0.1">
      <config filename="wlan_logfile.xml"/>
      <lognetwork name="net0" v="3">
        <lognode name="node0" v="3" displayNode="1">
          <logcmp name="wlan0" v="1"/>
          <logcmp name="mac0" v="1"/>
        </lognode>
        <lognode name="node1" v="3" displayNode="1">
          <logcmp name="wlan1" v="1"/>
          <logcmp name="mac1" v="1"/>
        </lognode>
      </lognetwork>
    </logging>
  </loglist>
</simulation>
```

Listing 6.7: Konfiguration eines Wlan-Netzes

Kapitel 7

Didaktischer Wert

Simulation eignet sich nicht nur um neue Fragestellungen zu beantworten, sondern sie kann auch in der Lehre als didaktisches Instrument eingesetzt werden um Wissen zu vermitteln. Neben den Einsatzmöglichkeiten in verschiedenen naturwissenschaftlichen Fächern, wie etwa in [Simon80] beschrieben, gibt es auch verschiedene Verwendungsmöglichkeiten im Informatikunterricht.

7.1 IBKSim als Unterrichtswerkzeug

Der Simulator des Instituts für Breitbandkommunikation in Verbindung mit dem in dieser Arbeit vorgestellten Framework eignet sich besonders, um Schülern und Studenten WLAN-spezifische Themen näher zu bringen. Dabei kann das Framework auf zwei verschiedene Arten verwendet werden. Zum einen kann es dazu genutzt werden um Themen, die die MAC- oder die physikalische Schicht betreffen, besser zu vermitteln. Ein Beispiel dafür wäre das "Hidden Station" Problem. Wie dieses Thema mit dem hier vorgestellten Framework vermittelt werden kann, darauf wird etwas später in diesem Kapitel noch genauer eingegangen.

Neben der bereits implementierten Funktionalität kann der Simulator auch als Versuchsfeld für Protokolle auf höheren Schichten dienen. So könnten verschiedenen Routingprotokolle implementiert werden und die Schüler können anhand der ausgetauschten Nachrichten und internen Tabellen genau beobachten, wie etwa Informationen über die Topologie gesammelt und ausgetauscht werden. Auch können die Auswirkungen, die der Ausfall eines einzelnen Knotens bewirkt, veranschaulicht werden. Ebenso kann der Simulator als Versuchsfeld dienen um eigene Protokolle zu testen.

7.2 Beispiel Hidden Station Problem

Das Hidden Station Problem gehört zu den klassischen Phänomenen im Gebiet der drahtlosen Kommunikation. Bereits 1975 wird auf die Probleme und auf die Auswirkungen, die solche "versteckten" Knoten haben, aufmerksam gemacht [Tobagi75].

Abbildung 7.1 veranschaulicht dieses Phänomen. In dem dargestellten Szenario verwenden alle drei Knoten dieselbe Sendefrequenz und das selbe Modulationsverfahren. Knoten B ist in Reichweite von Knoten A und C, die jedoch beide keinen direkten Funkkontakt zueinander haben. Die Knoten verwenden ein Carrier Sense Multiple Access Schema, um sich gegenseitig möglichst nicht zu stören. Dieses Zugriffschema versagt allerdings in der vorliegenden Konstellation. Wenn Knoten A eine Nachricht an Knoten B senden will, prüft er zuerst, ob zur Zeit keine Übertragung stattfindet. Dieses Schema setzt voraus, dass alle Knoten die selbe Reichweite besitzen. Erkennt Knoten A, dass der Sendekanal frei ist beginnt er mit seiner Übertragung zu Knoten B. Jetzt kann es passieren, dass auch Knoten C eine Nachricht an Knoten B senden will. Auch er überprüft ob der Kanal frei ist. Da er sich allerdings außerhalb der Reichweite von Knoten A befindet, kann er nicht erkennen, dass der Zielknoten, Knoten B, gerade eine Nachricht empfängt, also dass der Kanal beim Empfänger nicht frei ist. Knoten B beginnt also ebenfalls mit der Übertragung und stört damit die Übertragung von Knoten A. Ein so genannter Crash tritt auf.

Wie bereits in Kapitel 4 beschrieben wird im IEEE802.11 [IEE99b] versucht, die Wahrscheinlichkeit eines Crashes zu minimieren, indem in diesem

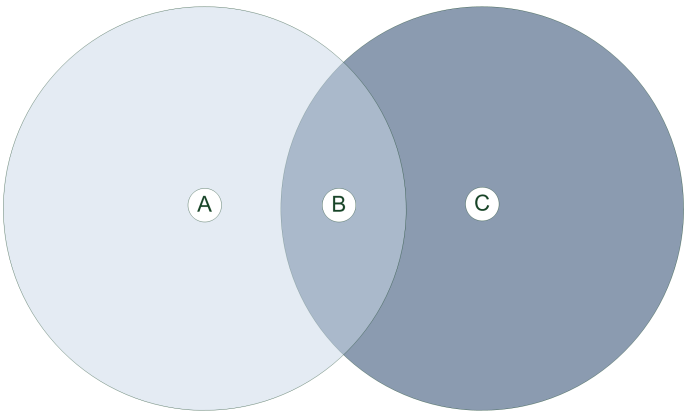

Abbildung 7.1: Darstellung des Hidden Station Problems

Szenario Knoten A Knoten B zuerst eine Anfrage sendet, ob er senden darf worauf Knoten B eine Bestätigung zurück sendet, die Informationen über die Dauer der Übertragung enthält. Da auch Knoten C diese Bestätigung empfängt, weiß dieser nun, dass das Medium für eine gewisse Zeit belegt ist, auch wenn er selbst die Übertragung nicht erkennen kann.

Dieses Problem lässt sich schön mit dem IBKSim in Kombination mit dem hier vorgestellten Framework darstellen. Dazu wird der Simulator mit Hilfe eines Konfigurationsskipts dazu veranlasst, drei Knoten zu erzeugen, die den gleichen Empfangs- beziehungsweise Senderadius besitzen. Diese werden so zueinander positioniert, dass ein Knoten, der im Weiteren als Knoten B bezeichnet wird, im Empfangsbereich der beiden anderen liegt. Die beiden anderen Knoten, Knoten A und C, können sich allerdings nicht gegenseitig empfangen. Dies entspricht wiederum dem bereits beschriebenen Szenario. Knoten A und C versuchen in regelmäßigen Abständen Nachrichten an Knoten B zu senden. Die Intervalle sind dabei so gewählt, das es manchmal zu Überlappungen kommt, also beide Knoten gleichzeitig senden. Dadurch kommt es zu einem Crash bei der Übertragung.

Wird die Logfunktion des Simulators entsprechend konfiguriert, kann aus diesem herausgelesen werden, dass es zu einem Crash kam und der Grund dafür die Konstellation der beteiligten Knoten ist. Allerdings ist es nicht sonderlich komfortabel, dieses Verhalten durch das Lesen eines langen Logfiles zu beobachten. Wünschenswert wäre eine grafische Darstellung, am besten eine Animation.

Auch dies ist mit dem Simulator des IBKs möglich. Wie in Kapitel 3.2 beschrieben, verwendet der IBKSim XML-Files, sowohl für die Konfiguration als auch für das Logfile. Ein auf XML basierendes Bildformat, welches auch Animationen unterstützt ist SVG [Eisenberg02]. Mit XSLT-Prozessoren wie etwa Xalan [Apa08] ist es möglich, ein XML-Format mit Hilfe von so genannten Stylesheets in ein anderes zu transformieren. Es ist also möglich, wenn die entsprechenden Informationen im Logfile vorhanden sind, direkt eine Animation aus dem Logfile zu erstellen. Einige Module dieses Frameworks unterstützen allerdings auch schon direkt das SVG-Format.

In Abbildung 7.2 ist ein Screenshot von so einer Animation zu sehen. Es werden die einzelnen Knoten mit ihren Senderadien dargestellt. Anhand der Farbe der Senderadien ist zu erkennen, ob ein Knoten gerade den Funkkanal als belegt (rot eingezeichnet) oder als frei (grün) einstuft. Die Farbe des Knotens gibt darüber Auskunft, ob der Knoten gerade sendet (rot), er auf Empfang geschalten ist (grün) oder ob er eine Kollision aufgetreten ist (blau). Durch Änderungen im Stylesheet ist es natürlich auch möglich, die

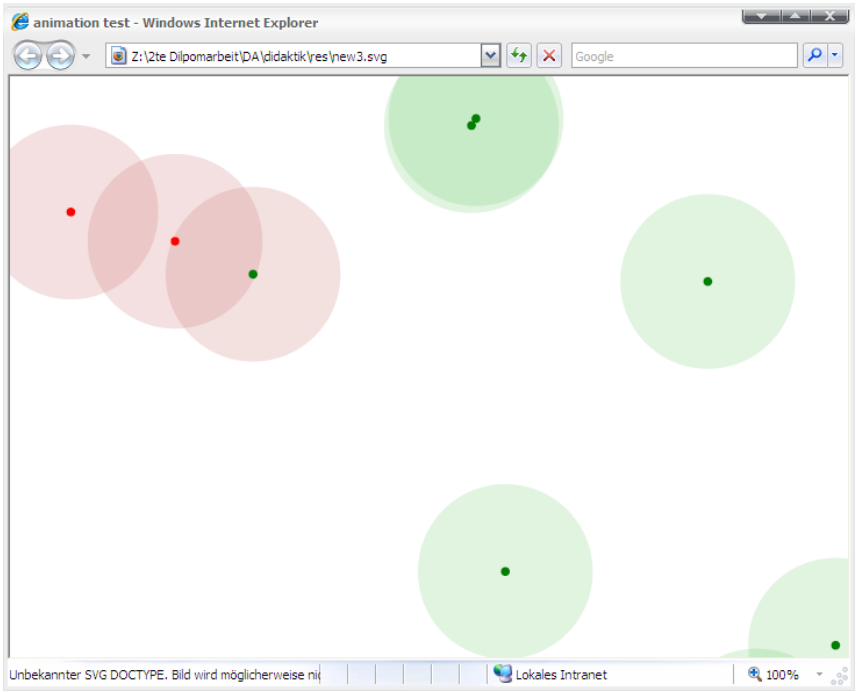

Abbildung 7.2: Visualisierung des Logfiles

Art, wie das Logfile dargestellt wird, zu verändern.

Diese einfache Möglichkeit, das Logfile zu visualisieren kann im Weiteren auch dafür verwendet werden, den in IEEE802.11 [IEE99b] beschriebenen virtuellen Carrier Sense Mechanismus zu veranschaulichen. So könnte etwa der Senderadius der Knoten, die aufgrund des Network Allocation Vectors zur Zeit den Kanal als belegt registrieren, in einer weiteren Farbe eingefärbt werden. In einer solchen Animation lässt sich dann schön erkennen, wie zuerst Knoten A zu B eine Nachricht sendet, Knoten B darauf antwortet, worauf Knoten C erkennt, das das Medium für eine gewisse Zeit reserviert ist und dies durch einen entsprechend eingefärbten Senderadius darstellt.

Trotz dieser Möglichkeit, die Ausgabe des Simulators zu visualisieren, muss gesagt werden, dass der Einsatz im Unterricht nicht das vorrangige Ziel bei der Entwicklung des Simulators war. So ist es nicht möglich interaktiv in die Simulation einzugreifen. Gerade eine solche Funktionalität wäre jedoch beim Einsatz im Unterricht wünschenswert. In der vorliegenden Version kann die Simulation nur durch Veränderung des Konfigurationsfiles beeinflusst werden was das Experimentieren mit dem Simulator erschwert. Zusätzlich

kommt noch hinzu, dass die Konfiguration nicht über ein grafisches Tool geschieht. Es ist also nötig zuerst die Konfigurationssprache zu lernen bevor man wirklich den Simulator bedienen kann. Zusammenfassend kann man sagen, dass sich der Simulator zur Darstellung von verschiedenen Abläufen eignet, ein spielerisches Lernen allerdings kaum möglich ist.

Kapitel 8

Fazit und Ausblick

Ziel dieser Arbeit war die Entwicklung eines Frameworks zur Simulation von Mobilen Ad-Hoc Netzen. Das Framework ist eine Erweiterung des Simulators des Instituts für Breitbandkommunikation. Neben der Möglichkeit, auf einfache Weise mobile Ad-Hoc Netze zu simulieren, war die Erweiterbarkeit sowie die Unterstützung von großen Ad-Hoc Netzen von besonderer Wichtigkeit.

Angelehnt an das OSI-Schichtmodel [OSI84] wurde die Bitübertragungsschicht getrennt von der Sicherungsschicht implementiert. Des Weiteren wurden einige Module, die auf der Sicherungsschicht aufbauen, entwickelt, um die Verwendbarkeit des Frameworks zu demonstrieren.

Auf der Ebene der Bitübertragungsschicht stehen verschiedene Modelle zur Verfügung, sodass je nach Szenario das am besten geeignete Model verwendet werden kann. Das einfachste implementierte Model ist ein einfaches Flat Earth Model, in welchem die Übertragung sicherlich erfolgreich ist, sofern sich die Knoten im jeweiligen Empfangsbereich befinden. Nur durch das Auftreten von Kollisionen kann es zu einer Störung der Übertragung kommen. Dieses Model ist zwar sehr performant, allerdings nicht sonderlich genau.

Ein weiteres, komplexeres Modul verwendet verschiedene Radiomodelle um die Signalstärke beim Empfänger zu berechnen, sowie die Interferenz durch andere Knoten. Durch dem Abstand der Signalstärke zur Interferenz durch die anderen Knoten und zu dem Grundrauschen, wird unter Einbeziehung von kodierungsspezifischen Bitfehlerraten die Übertragungswahrscheinlichkeit berechnet. Durch Einbettung weiterer Modelle lässt sich dieses Modul auch zur Simulation anderer Funksysteme verwenden, beziehungsweise bie-

tet es eine gute Ausgangsbasis für noch komplexere Modelle. Erwähnenswert ist außerdem, dass in diesem Modul ein maximaler Wirkungsbereich der Knoten festsetzbar ist. Außerhalb dieses Bereiches interferiert eine Übertragung nicht mehr mit anderen Knoten. Dadurch lässt sich die Simulation von großen, weit verteilten Netzen beschleunigen.

Die Bewegungsmodelle der einzelnen Knoten sind in eigenen Modulen gekapselt, die von den Modellen der Bitübertragungsschicht verwendet werden. Dadurch ist es auf einfache Weise möglich, weitere Bewegungsmodelle zu den bereits vorhandenen hinzuzufügen, ohne dass die Funktionalität der Modelle der Bitübertragungsschicht bekannt sein muss.

Auf der Sicherungsschicht wurden die für Ad-Hoc Netze relevanten Teile des IEEE802.11 MAC Protokolls implementiert. Durch Erweiterungen in diesem Modul, ließen sich auch Netze mit Infrastruktur simulieren, was allerdings für die Simulation von mobilen Ad-Hoc Netzen irrelevant ist. Ebenso könnten Erweiterungen der IEEE802.11 Standards, wie etwa IEEE802.11e [IEE05], in diesem Modul oder einem davon abgeleiteten implementiert werden, wobei keine Veränderungen an den Modellen der Bitübertragungsschicht notwendig wären.

Das Framework besitzt alle nötigen Funktionen um verschiedene Fragestellungen im Bezug auf mobile Ad-Hoc Netze zu beantworten. Konkret wird es in naher Zukunft zur Untersuchung von Routingprotokollen in Ad-Hoc Netzen eingesetzt werden.

Im theoretischen Teil der Arbeit, wurde allgemein das Thema Simulation, im speziellen die Simulation mobiler Ad-Hoc Netze, behandelt. Es wurden verschiedene Modelle vorgestellt, die Vor- und Nachteile derselbigen erörtert, sowie allgemein die Herausforderungen bei der Simulation solcher Systeme diskutiert. Des Weiteren wurde auf die Grundlagen des IEEE802.11 Standards eingegangen, da dieser für das vorgestellte Framework von besonderer Bedeutung ist. Als didaktischer Teil wurden die Anwendungsmöglichkeiten des Simulators im Unterricht erörtert.

Anhang

Abkürzungsverzeichnis

ACK	Acknowledgement
CRC	Cyclic Redundancy Code
CTS	Clear to send
DBPSK	Differential Binary Phase Shift Keying
DCF	Distributed Coordination Function
DIFS	DCF Interframe Space
DQPSK	SIfferential Quadrature Phase Shift Keying
DSSS	Direct Sequence Spread Spectrum
DSSS/HR	High Rate Direct Sequence Spread Spectrum
EIFS	Extendet Interframe Space
FHSS	Frequency Hopping Spread Spectrum
GNU	GNUs Not Unix
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IFS	Interframe Space
LAN	Local Area Network
MAC	Media Access Control
Mbps	Mega Bit per Second
OFDM	Orthogonal Frequency Division Multiplexing
PCF	Point Coordination Function
PIFS	PCF Interframe Space
PLCP	Physical Layer Convergence Protocol
PSDU	PLCP Service Data Unit
PSK	Phase Shift Keying

RTS	Request to send
SIF	Short Interframe Space
SFD	Start Frame Delimiter
SVG	Scalable Vector Graphics
UML	Unified Modeling Language
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[Apa08] Apache Software Foundation. *Xalan-C++ version 1.10*, `http://xml.apache.org/xalan-c/index.html`, August 2008.

[Bennett95] B. Bennett. *Simulation Fundamentals.* Prentice Hall, 1995.

[Camp02] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002.

[Clausen03] T. Clausen, P. Jacquet (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L.Viennot. Optimized link state routing protocol (OLSR). RFC 3626, October 2003. Network Working Group.

[Eisenberg02] J. David Eisenberg. *Svg Essentials.* O'Reilly Media, 2002.

[Fishwick95] Paul A. Fishwick. *Simulation Model Design and Execution.* Prentice Hall, 1995.

[Gamma95] Erich Gamma and Richard Helm. *Design Patterns, Elements of Reusable Object-Oriented Software*, pages 107–116. Addison-Wesley, 1995.

[Gast05] Matthew Gast. *802.11 Wireless Networks: The Definitive Guide.* O'Reilly, 2005.

[Haas01] Z.J. Haas and M.R. Pearlman. The performance of query control schemes for the zone routing protocol. *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 4, pp. 427–438, 2001.

[Heidemann01] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, K. Lan, Y. Xu, W. Ye, D. Estrin, and R. Govindan. Effects of detail in wireless network simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, pp. 3–11, 2001.

[Hoff95] Simon Hoff and Otto Spaniol. *Ereignisorientierte Simulation.* Thomson Publishing, 1995.

[Hong02] X. Hong, K. Xu, M. Gerla, and CA Los Angeles. Scalable routing protocols for mobile ad hoc networks. *Network, IEEE*, vol. 16, no. 4, pp. 11–21, 2002.

[IEE99a] IEEE. *IEEE Std 802.11b-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band.* statut : « Supplement to IEEE Std 802.11-1999 ».

[IEE99b] IEEE. *IEEE Std 802.11-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* statut : « Adopted by the ISO/IEC and redesignated as ISO/IEC 8802-11:1999(E) ».

[IEE03] IEEE. *IEEE Std 802.11b-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band.*

[IEE04] IEEE. *IEEE Std 802.11b-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements.*

[IEE05] IEEE. *IEEE Std 802.11b-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements.*

[Inf08] Information Sciences Institute. *The Network Simulator - ns-2*, `http://nsnam.isi.edu/nsnam/index.php/Main_Page`, August 2008.

[Jardosh03] A. Jardosh, E.M. Belding-Royer, K.C. Almeroth, and S. Suri. Towards realistic mobility models for mobile ad hoc networks. *Proceedings of the 9th annual international conference on Mobile computing and networking*, pp. 217–229, 2003.

[Johnson03] D.B. Johnson, D.A. Maltz, Y.C. Hu, and J.G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). In *IETF Mobile Ad Hoc Networks Working Group, Internet Draft, work in progress*, volume 15. 2003.

[Khosroshahy07] M. Khosroshahy, T. Turletti, and K. Obraczka. Snapshot of MAC, PHY and Propagation Models for IEEE 802.11 in Open-Source Network Simulators. In *Rapport de recherche n° 6310, Institut National de Recherche en Informatique et en Automatique*. 2007.

[Kotz04] D. Kotz, C. Newport, R.S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *Dartmouth Computer Science Technical Report TR2004-507*. 2004.

[Labiod08] Houda Labiod. *Wireless Ad Hoc and Sensor Networks*. Wiley, 2008.

[Macker98] Joseph P. Macker and M. Scott Corson. Mobile ad hoc networking and the ietf. In *ACM Mobile Computing and Communications Review*, volume 2. 1998.

[Ogier04] R. Ogier, F. Templin, and M. Lewis. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). RFC 3684, 2004.

[OMN08] OMNeT++ Community. *OMNeT++*, `http://www.omnetpp.org/`, August 2008.

[OSI84] OSI. Information technology – Open Systems Interconnection – Basic reference model. Technical Report 7498-1, ISO/IEC, 1984. ITU-T Rec. X.200.

[Parsons00] J. David Parsons. *The Mobile Radio Propagation Channel*. Wiley; 2 edition, 2000.

[Perkins01] Charles E. Perkins. *Ad Hoc Networking.* Addison-Wesley, 2001.

[Perkins03] C. Perkins, E.M. Royer, and S.R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing, 2003.

[Prasad02] Neeli Prasad. *WLAN Systems and Wireless IP for Next Generation Communications.* Artech House, 2002.

[Proakis01] John G. Proakis. *Digital communications.* LinkMcGraw-Hill series in electrical and computer engineering. LinkMcGraw-Hill, Boston, Mass. [u.a.], 4 edition, 2001.

[Rappaport99] Theodore S. Rappaport. *Wireless Communications: Principles and Practice.* Upper Saddle River, NJ : Prentice Hall PTR, 1999.

[Sánchez01] M. Sánchez and P. Manzoni. ANEJOS: a Java based simulator for ad hoc networks. *Future Generation Computer Systems*, vol. 17, no. 5, pp. 573–583, 2001.

[Sauerbier99] Thomas Sauerbier. *Theorie und Praxis von Simulationssystemen.* Vieweg, 1999.

[Seibold02] Wolfram Seibold. *Ein Mehrprozessor mit rekonfigurierbarer Verbindungsstruktur für die parallele und veteilte ereignisgesteuerte Simulation - 81. Bericht über verkehrstheoretische Arbeiten.* Universität Stuttgart, 2002.

[Simon80] Hartmut Simon. *Computer-Simulation und Modellbildung im Unterricht.* Oldenbourg, 1980.

[Stepanov05] I. Stepanov, D. Herrscher, and K. Rothermel. On the impact of radio propagation models on manet simulation results. In *Proceedings of the 7th IFIP International Conference on Mobile and Wireless Communication Networks (MWCN 2005), Marrakech, Morocco, september.* 2005.

[Tel08] Telecommunication Networks Group (TKN), Technische Universitaet Berlin. *Mobility Framework for OMNeT++*, `http://mobility-fw.sourceforge.net/`, August 2008.

[Tobagi75] F. Tobagi and L. Kleinrock. Packet Switching in Radio Channels: Part II–The Hidden Terminal Problem in

Carrier Sense Multiple-Access and the Busy-Tone Solution. *Communications, IEEE Transactions on [legacy, pre-1988]*, vol. 23, no. 12, pp. 1417–1433, 1975.

[Wei08] Ye Wei. *The ns Manual (formerly ns Notes and Documentation)*, chapter Radio Propagation Models. 2008.

[Xiuchao04] W. Xiuchao. Simulate 802.11 b channel within ns2. In *Relatorio tecnico, National University of Singapore*. 2004.