Spontane Wireless LANs

Diplomarbeit

zur Erlangung des Akademischen Grades
Diplom-Informatiker

an der
Fachhochschule für Technik und Wirtschaft Berlin
Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

1. Betreuer: *Prof. Dr. Horst Hansen*
2. Betreuer: *Diplom-Verwaltungswirt Werner Heuser*

Eingereicht von *Andreas Gohr*
24. Mai 2004

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Mit der immer größeren Verbreitung mobiler Endgeräte wie Notebooks und Handheldcomputern ist auch der Wunsch nach der Vernetzung dieser Geräte entstanden. Herkömmliche kabelgebundene Vernetzung widerspricht jedoch dem Konzept der Mobilität dieser Geräte. So wurden zum einen Techniken zur Datenübertragung im Bereich des Mobilfunks entwickelt (GPRS, HSCSD und demnächst auch UMTS). Aber auch im providerunabhängigen privaten Bereich wurde eine Vielzahl von Techniken zum drahtlosen Datenaustausch geschaffen wie IRDA, BlueTooth und HomeRF. 1999 kam der Wireless LAN (WLAN) Standard 802.11 (siehe [IEE99a]) hinzu, welcher eine Vernetzung über Funkwellen vorsieht und erstmals größere Reichweiten zu erschwinglichen Preisen auch für den Endnutzer verspricht. WLAN ist heute die wohl verbreitetste unter den Funktechniken, so nutzen laut einer Studie von Mai 2003 (siehe [20]) 12,9% (5,44 Mio.) aller Internet-Nutzer (42,2 Mio.) WLAN oder planen dies zumindest. Innerhalb eines Jahres sei die Ausstattung der Endgeräte mit WLAN der Internet-Nutzer von 0 auf 9,9% (4,18 Mio.) gestiegen.

Im 802.11 Standard und seinen Nachfolgern 802.11a (siehe [IEE99b]), 802.11b (siehe [IEE99c]), 802.11d (siehe [IEE01]) und 802.11g unterscheidet man zwei Betriebsmodi.

Zum einen den *Infrastruktur-Modus* (siehe Kapitel 2.2.4), bei dem alle Teilnehmer des Funknetzes nur mit einer Basisstation — *Accesspoint* genannt — kommunizieren. Häufig ist die Basisstation über herkömmliche kabelgebundene Techniken mit anderen Netzen wie Firmenintranets oder dem Internet verbunden und arbeitet so für die mobilen Clients als Gateway in diese Netze. Durch den Einsatz mehrerer Accesspoints lassen sich relativ große Gebiete mit einem Funknetz abdecken. Die Mobilität der Clients bleibt letztenendes jedoch auf die Umgebung solcher „Hot Spots" beschränkt.

Eine einfachere Form der Funkvernetzung nach 802.11 stellt der *Ad-Hoc Modus* (siehe Kapitel 2.2.4) dar. Hier kommunizieren zwei oder mehr Clients direkt miteinander, auf einen zentralen Accesspoint wird verzichtet. Gateways in andere Netze sind hier jedoch normalerweise nicht vorgesehen, da der Modus — wie der Name schon andeutet — vor allem auf spontane Vernetzung an beliebigen Orten ausgelegt ist. Die Reichweite eines solchen Netzes ist, da hier jeder jeden erreichen können muss, auf die Funkreichweite des schwächsten Teilnehmers beschränkt.

Beide Modi haben also ihre Vor- und Nachteile. Der Infrastruktur-Modus bietet Möglichkeiten zum Verbinden herkömmlicher Netze mit den Funknetzen, während der Ad-Hoc Mode den

Aufbau kleinerer Funknetze ohne großen Aufwand ermöglicht. Beide Modi erfordern jedoch einen gewissen Konfigurationsaufwand und sind von ihrer Reichweite her beschränkt.

1.2 Aufgabenstellung

Sowohl eine Reichweitenvergrößerung als auch einen geringeren Aufwand bei der Konfiguration soll das Konzept der *Mobile Ad-Hoc Network*s, kurz MANETs (siehe [18]), schaffen. In diesen Netzen arbeitet jeder Teilnehmer als Router und leitet Datenpakete über seine Nachbarn weiter.

Wie MANETs auf Basis der heute erhältlichen WLAN-Technologie nach IEEE 802.11 aufgebaut werden können, soll diese Diplomarbeit zeigen. Dazu gibt es in Kapitel 2 zunächst eine Einführung in die Grundlagen von Wireless LAN und der spontanen Vernetzung, bevor dann auf die speziellen Anforderungen von MANETS eingegangen wird.

Damit Datenpakete über mehrere Teilnehmer des Netzes hinweg zum Ziel gelangen, ist der Einsatz eines Routingprotokolls nötig. Dieses muss in der Lage sein, auch bei sich ständig ändernder Topologie, wie es in mobilen Netzen der Fall ist, die benötigten Routen zu ermitteln. Herkömmliche, aus der drahtgebundenen Vernetzung bekannte Protokolle sind dazu nicht geeignet. In Kapitel 3.1 werden daher verschiedene, speziell für MANETs entworfene Routingprotokolle vorgestellt und verglichen. Zudem wird in Kapitel 3.2 deren Verhalten in unterschiedlichen mobilen Szenarien in einem Netzwerksimulator erprobt und bezüglich verschiedener Kenngrößen untersucht.

Um als Router fungieren zu können, benötigen alle Teilnehmer eine eindeutige Identifizierung. In modernen TCP/IP-Netzen, auf die sich diese Arbeit beschränken soll, werden dafür IP-Adressen eingesetzt. Auf Grund der oben genannten Forderung nach spontaner Vernetzung ohne zusätzlichen Konfigurationsaufwand, muss eine Methode zur automatischen Konfiguration von IP-Adressen in spontanen Netzen gefunden werden. Welche verschiedenen Lösungsansätze es dazu für die IP-Versionen 4 und 6 gibt und welche Probleme dabei gelöst werden müssen, wird in Kapitel 4 behandelt.

Der Aufbau eines MANETs unter dem Betriebssystem soll schließlich in Kapitel 5 prototypisch gezeigt werden.

2 Allgemeine Grundlagen

Dieses Kapitel gibt einen Überblick über die technischen und theoretischen Grundlagen zur spontanen Vernetzung, zeigt auf welche speziellen Probleme dabei zu lösen sind.

2.1 Computernetze

Wenn im folgenden von Netzwerken die Rede ist, so wird Tanenbaums (siehe [Tan03], Kapitel 1) *computer network* Definition zu Grunde gelegt:

> [...] the term ”computer network” [...] mean[s] a collection of autonomous computers interconnected by a single technology. Two computers are said to be interconnected if they are able to exchange information.

Die einzelnen Stationen innerhalb eines Netzes sollen im folgenden als *Knoten* oder *nodes* bezeichnet werden. Knoten die direkt — also ohne Weiterleitung über Zwischenstationen — erreicht werden können, heißen auch *Nachbarn*.

Zusätzlich beschränkt sich diese Arbeit auf die Betrachtung von auf der TCP/IP-Suite basierenden Netzwerken.

2.2 Der WLAN Standard IEEE 802.11

Wireless LAN oder kurz *WLAN* wird häufig in unterschiedlichen Zusammenhängen benutzt. Einerseits werden häufig alle drahtlosen Kommunikationstechniken mit denen sich Computernetzwerke aufbauen lassen so bezeichnet, andererseits auch Funknetze nach dem IEEE Standard 802.11. In dieser Arbeit wird der Begriff WLAN im letzteren Sinne verwendet.

Mit der Spezifikation des Wireless LAN-Standards wurde 1997 begonnen. Ursprünglich wurde der 802.11 Standard neben der Übertragung per Funkwellen auch für Infrarotübertragungen konzipiert. Auf Grund der Eigenschaften von Infrarotverbindungen wie kurzer Reichweite und hoher Störanfälligkeit gegenüber anderen (Infrarot-) Lichtquellen, spielt es als Übertragungsmedium allerdings heute keine Rolle mehr. So wird WLAN heute oft als Synonym für Funknetzwerke verwendet.

WLAN ist für zwei Frequenzbereiche spezifiziert: 2,4 GHz und 5 GHz. Hardware für den einen Bereich ist inkompatibel zur Hardware im anderen Bereich. Obwohl lange Zeit langsamer, hat sich in Europa weitgehend die Verwendung des 2,4 GHz-Bereiches durchgesetzt.

Während die erste WLAN-Spezifikation nur Datenraten bis zu 2 MBit/s vorsah (siehe [IEE99a]), wurden 1999 mit den Erweiterungen 802.11a (siehe [IEE99b]) und 802.11b (siehe [IEE99c]) Übertragungsraten mit bis zu 54 MBit/s im 5 GHz Bereich bzw. bis zu 11 MBit/s im 2,4 GHz Bereich eingeführt (siehe [Rot02]). Der 2003 verabschiedete Standard IEEE 802.11g bietet inzwischen auch im 2,4 GHz-Bereich Übertragungsraten von bis zu 54 MBit/s (siehe [19]).

2.2.1 Protokollschichten

Der IEEE 802.11-Standard ist eine weitere IEEE 802-Netzwerkspezifikation und bestimmt die Implementierung der OSI-Schichten eins und zwei (Physical Layer und Data Link Layer). Er besitzt daher den von anderen 802 Standards (wie beispielsweise den IEEE 802.3 für Ethernet-basierte Netzwerke) bekannten Aufbau. So wird die Sicherungsschicht in der Spezifikation noch einmal in zwei Ebenen unterteilt: In die *Logical Link Control*-Ebene (LLC) und die *Media Access Control*-Ebene (MAC) (vgl. Abb. 2.1). In der LLC Schicht wird die in IEEE 802.2 festgelegte und für alle 802-Standards verbindliche Definition übernommen, so kann der Networklayer (OSI-Schicht 3) immer über die selbe Schnittstelle zugreifen und die Kompatibilität zu allen anderen Netzwerkspezifikationen nach IEEE 802 wird so garantiert. Diese Transparenz gegenüber den höheren Schichten ist ein weiterer Vorteil gegenüber anderen drahtlosen Techniken wie beispielsweise BlueTooth und hat stark zur Verbreitung von WLAN beigetragen.

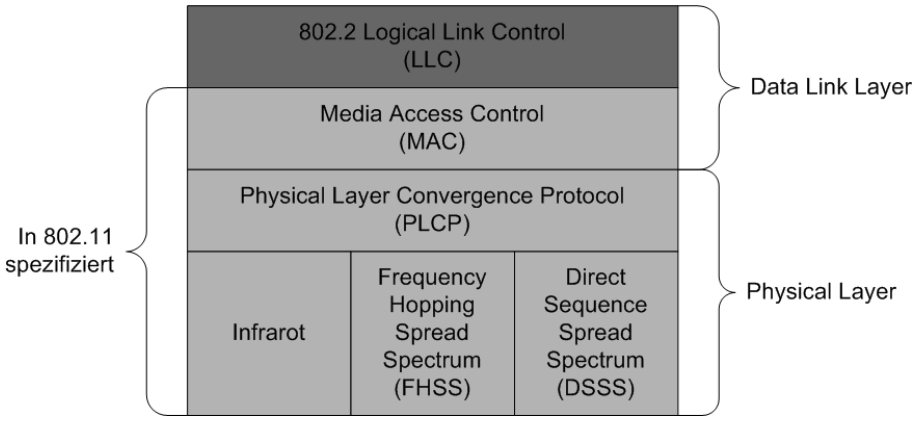

Abbildung 2.1: Protokollschichten von IEEE 802.11

2.2.2 Physical Layer

Die Bitübertragungsschicht (Physical Layer) des OSI-Modells wird in IEEE 802.11 in zwei Ebenen aufgeteilt: Eine übergeordnete *Physical Layer Convergence Protocol*-Ebene stellt eine einheitliche Schnittstelle zwischen der Sicherungsschicht und dem eigentlichen Funk- bzw. Infrarot-Medium zur Verfügung (vgl. Abb 2.1).

Für die Übertragung im Funkmedium stehen zwei Verfahren zur Verfügung — *Frequency Hopping Spread Spectrum* (FHSS) und *Direct Sequence Spread Spectrum* (DSSS). Die Übertragung findet entweder im Frequenzband um 2,4 GHz oder im 5 GHz-Bereich statt. Als Reichweitenrichtlinien sind 30 m in Gebäuden und 300 m im Freien angegeben (siehe [Rot02]).

Das FHSS Verfahren benutzt das Frequenzsprungverfahren (Frequency Hopping), wobei der verfügbare Frequenzbereich von 2400 MHz bis 2483,5 MHz in 79 Kanäle aufgeteilt wird und die Kanäle in schneller Folge gewechselt werden.

Bei DSSS wird eine Bandspreizung nach dem *Code Division Multiple Access* (CDMA) Verfahren durchgeführt. Damit können mehrere Sender eine Frequenz verwenden und das verfügbare Frequenzband wird nur auf 14 Kanäle aufgeteilt — dies ermöglicht eine wesentlich höhere Stabilität gegenüber Störungen des Funkmediums. DSSS hat sich inzwischen gegenüber dem FHSS-Verfahren durchgesetzt.

2.2.3 MAC Ebene

Der Zugriff auf das Funkmedium wird von der MAC-Schicht in IEEE 802.11 geregelt. Hauptaufgabe besteht hier in der Vermeidung von Kollisionen. Da ein Sender am Sendeort alle Signale eines anderen Senders überdeckt, kann bei Funkübertragung nicht das vom Ethernet bekannte CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) Kollisionserkennungsverfahren eingesetzt werden. Stattdessen müssen Kollisionen möglichst vermieden werden (siehe [Rot02]). Dafür spezifiziert IEEE 802.11 mehrere Zugriffsverfahren von denen allerdings nur eines verbindlich ist: Das *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA) Verfahren. Optional können zusätzlich noch CSMA/CA mit RTS/CTS (*Request to Send* und *Clear to Send*) sowie das *Point Coordination Function* (PCF) Verfahren unterstützt werden. Letzteres funktioniert jedoch nur im Infrastruktur-Modus (siehe auch 2.2.4).

CSMA/CA

Beim CSMA/CA-Verfahren lauscht eine sendewillige Station zunächst auf dem Funkmedium nach gerade stattfindenden Übertragungen. Ist das Medium belegt, so wird zunächst das Ende der Übertragung abgewartet, anschließend wird eine zusätzliche Zeit gewartet bevor mit der Übertragung begonnen wird. Diese zusätzliche Zeit ergibt sich zum einen aus einer konstanten Wartezeit, dem *DCF (Distributed Coordination Function) Interframe Space* oder kurz DIFS und einer zufällig gewählten Wartezeit. Erst wenn das Funkmedium nach Ablauf dieser Zeit immer noch frei ist, beginnt die Station mit dem Senden. Die zufällig gewählte Zeit ist dabei ein ganzzahliger Wert, der, während das Medium frei ist, ständig dekrementiert wird. Sendet eine andere Station während dieser Wartezeit, so wird der Zähler angehalten und erst weiter dekrementiert, wenn das Medium wieder frei ist — der Zähler wird also nicht nach jedem Warten neu initialisiert. Somit hat die Station eine statistisch bessere Chance, als nächste Zugriff auf das Medium zu erhalten. Jede Station muss aber nach jeder Belegung des Funk-

mediums mindestens für den Zeitraum des DIFS vor dem Senden warten — dieser wird also nicht dekrementiert.

Das Verfahren garantiert keine Kollisionsfreiheit, sondern verringert nur die Wahrscheinlichkeit einer Kollision. Direkten Einfluß auf diese Wahrscheinlichkeit hat die Größe des Bereichs, aus dem die Zufallszahlen gewählt werden — je größer der Bereich desto geringer die Wahrscheinlichkeit, dass zwei Stationen gleichzeitig mit dem Senden beginnen. Die maximale Zeit, die man beim Wählen der Zufallszahl erhalten kann, wird *Contention Window* genannt. Kleine Contention Windows erhöhen die Wahrscheinlichkeit für Kollisionen, haben aber durch die relativ kurzen Wartezeiten bei geringer Auslastung des Funkmediums einen höheren Datendurchsatz. Große Contention Windows hingegen bieten größeren Schutz vor Kollisionen, verringern aber den Datendurchsatz des Mediums. Um ein möglichst optimales Fenster zu finden, wird eine adaptive Technik namens *Exponential Backup* genutzt. Dabei wird mit einem kleinen Contention Window begonnen, welches bei auftretenden Fehlern automatisch vergrößert wird.

Da Kollisionen nicht direkt erkannt werden können, müssen alle Frames vom Empfänger quittiert werden. Beim Senden der Empfangsbestätigung werden sogenannte *Short Interframe Spaces* (SIFS) anstatt der DIFS beim Warten auf den Zugriff verwendet. Diese sind kürzer als die DIFS und geben den Quittungsframes somit eine höhere Priorität. Erhält ein Sender keine Quittung, so kann dies zwei Gründe haben: Entweder ist der gesendete Frame nicht angekommen oder die Quittung ist verloren gegangen. In beiden Fällen wird das Contention Window vergrößert und der Frame wird erneut versandt. Beim erneuten Versenden hat die Station jetzt jedoch keinen Vorteil gegenüber den anderen Teilnehmern und muss erst wieder mittels DIFS auf einen Sendeplatz im Medium warten. Erst wenn ein Frame erfolgreich zugestellt wurde, kann das Contention Window wieder auf die Ausgangsgröße zurückgesetzt werden.

CSMA/CA mit RTS/CTS

CSMA/CA geht davon aus, dass alle Teilnehmer des Netzes sich gegenseitig „sehen" können — nur so ist es möglich zu erkennen, ob das Funkmedium zur Zeit belegt ist. Dies ist aber nicht immer gegeben. Ist eine Station nicht mehr in Reichweite aller Teilnehmer, kommt es zum *Hidden-Terminal-Problem* — einige Stationen sind nicht mehr für alle sichtbar und die Kollisionsvermeidung wird uneffektiver (vgl. Abbildung 2.2).

Abhilfe soll hierbei das um *Request to Send* und *Clear to Send* erweiterte CSMA/CA-Verfahren bieten, welches jedoch lediglich optional und nicht verbindlich in IEEE 802.11 festgelegt ist. Bei diesem Verfahren sendet eine Station vor dem eigentlichen Senden einen RTS-Frame an den Empfänger, dieser bestätigt diesen mit einem CTS-Frame. Dieses Senden des CTS-Frames kann nun von den anderen Stationen empfangen werden, auch wenn ihnen der RTS-Frame entgeht. Beim Empfang eines RTS- bzw. CTS-Frames wird eine Belegung des Mediums für eine in diesen Frames angegebene Zeit angenommen, auch wenn die eigentliche Belegung nicht detektiert werden kann. Die Zeitdauer in den RTS- und CTS-Frames, welche den Wartezeitraum für die anderen Stationen angibt, wird *Net Allocation Vector* genannt. Die

Request- und Clear to Send-Frames werden dabei mit SIFS-Wartezeiten (vgl. Seite 9) — also höherer Priorität — übertragen.

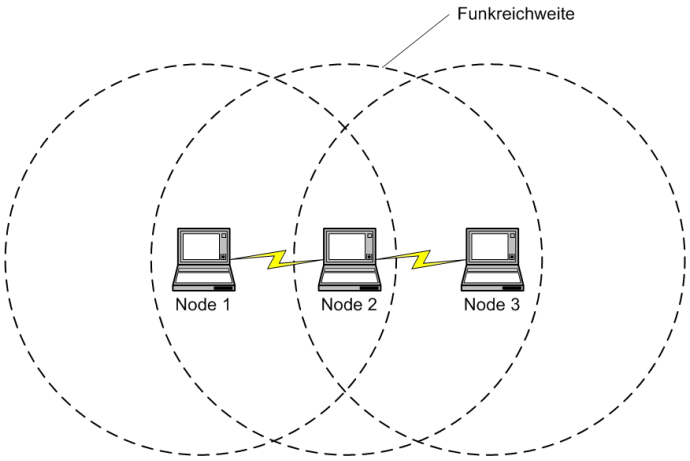

Abbildung 2.2: Hidden-Terminal-Problem

Abbildung 2.2 stellt das Hidden-Terminal-Problem dar. Node 1 sendet an Node 2, diese Funkübertragung geschieht jedoch außerhalb der Funkreichweite von Node 3. Für diesen sieht es aus, als währe das Funkmedium frei.

Point Coordination Function

Auch das PCF-Verfahren ist lediglich optional im IEEE 802.11-Standard enthalten. Im Unterschied zu den beiden bisher vorgestellten Verfahren benötigt PCF eine Master Station: einen sogenannten *Point Coordinator* (PC). Die Funktion des Masters wird üblicherweise von einem Accesspoint übernommen (vgl. 2.2.4).

Das PCF-Verfahren wechselt sich mit den Distributed Coordination Function-Verfahren (DCF)(also CSMA/CA) ab. Der Point Coordinator wartet zunächst auf ein freies Funkmedium, bevor er nach einer *PCF Interframe Space* (PIFS) Wartezeit allen Teilnehmern den Beginn der PCF-Phase signalisiert. Der PIFS ist kürzer als der DIFS, aber länger als die SIFS-Zeitspannen und hat damit zwar Priorität vor normalen Datenpaketen, muss aber Quittungen sowie RTS- und CTS-Frames abwarten. Der PCF-Startframe enthält einen *Net Allocation Vector*, der allen anderen Stationen für die Dauer der PCF-Phase das Senden untersagt. Anschließend fragt der Point Coordinator jede Station einzeln nach Sendewünschen ab (*poll*), welche diese durch das Senden ihrer Daten beantworten können. Wurden alle Teilnehmer abgefragt, wird ein End-Frame gesendet und der DCF-Modus tritt wieder in Kraft.

Der Vorteil des PCF-Verfahrens liegt darin, dass nur bei diesem Verfahren Garantien bezüglich Bandbreite und Verzögerungen gegeben werden können.

2.2.4 Betriebsmodi

WLAN nach IEEE 802.11 unterscheidet zwei Betriebsmodi: den *Infrastruktur-Modus* und den *Ad-Hoc Modus*, die im Folgenden genauer vorgestellt werden.

Infrastruktur-Modus

Im Infrastruktur-Modus kommunizieren die einzelnen Clients nicht direkt untereinander, sondern immer über eine feste Basisstation, *Accesspoint* (AP) genannt. IEEE 802.11 definiert Verbindungen von Teilnehmern allgemein als sogenannte *Service Sets*. Ein *Basic Service Set* (BSS) existiert bei der direkten Verbindung von zwei oder mehreren Teilnehmern (Nodes). Beim Infrastruktur-Modus ist genau einer dieser Nodes der Accesspoint (vgl. auch Abbildung 2.3).

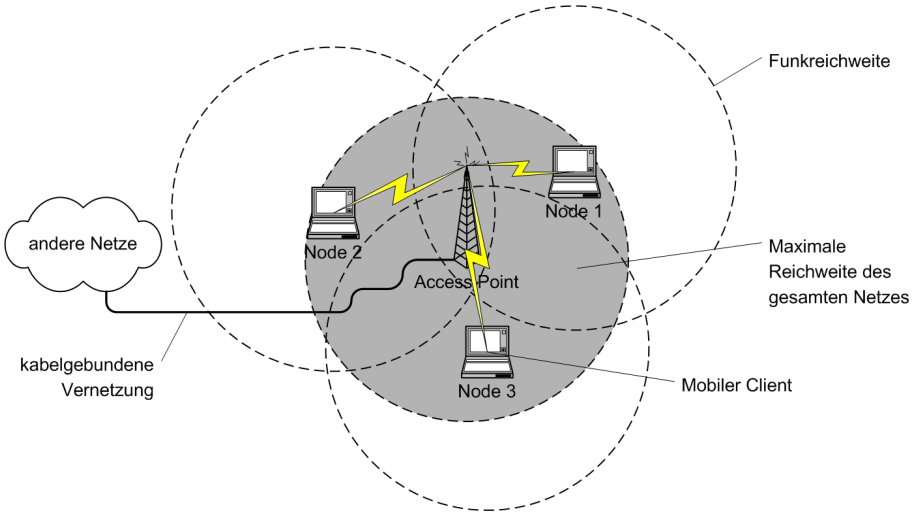

Abbildung 2.3: Infrastruktur-Modus

Abbildung 2.3 zeigt Kommunikation und Reichweiten im Infrastruktur-Modus. Alle drei Teilnehmer kommunizieren über den zentralen Accesspoint miteinander und haben über diesen auch Zugang zu anderen Netzen (*Portal*). Wie man sehen kann, müssen die einzelnen Nodes nicht in gegenseitiger Funkreichweite sein — lediglich die Kommunikation mit dem Accesspoint muss gewährleistet sein.

Der Accesspoint kann für die teilnehmenden Stationen erweiterte Funktionen wie Uhrensynchronisation und das Powermanagement der WLAN-Hardware übernehmen. Zudem lassen sich bei der Verwendung einer festen Basisstation auch effizientere Verfahren für den Zugriff auf das Funkmedium implementieren (vgl. Point Coordination Function Seite 10). Meist übernimmt der Accespoint noch zusätzliche Aufgaben für höhere Protokollebenen, wie die Verteilung von IP-Adressen per DHCP.

Der große Vorteil des Infrastruktur-Modus ist die Möglichkeit mehrere Accesspoints über ein sogenanntes *Distribution System* miteinander zu verbinden und so die Reichweite des

gesamten Systems zu vergrößern. Das Distribution System kann dabei ebenfalls eine Funkverbindung zwischen den Accesspoints sein, aber auch eine herkömmliche kabelgebundene Verbindung ist häufig anzutreffen. Alle über ein Distribution System miteinander verbundene Basic Service Sets werden zu einem logischen *Extended Service Set* (ESS) zusammengefaßt. Innerhalb dieses Service Sets können sich die Teilnehmer beliebig bewegen und von einem Accesspoint zu einem anderen zu wechseln, ohne dass dabei die Netzwerkverbindung verloren geht. Diesen Vorgang nennt man, in Anlehnung an ein ähnliches Verfahren im Mobilfunk, *Roaming*.

Übernimmt der Accesspoint zusätzlich die Funktion eines Gateways in andere Netze, so wird diese Funktion im IEEE 802.11 als *Portal* definiert.

Ad-Hoc Modus

Der *Ad-Hoc Modus* ist die einfachste Form eines Service Sets — das *Independent Basic Service Set* (IBSS). Hier handelt es sich um mehrere Teilnehmer, die direkt über das Funkmedium miteinander verbunden sind (vgl. Abb. 2.4). Alle Teilnehmer sind dabei gleichberechtigt — es gibt also keine ausgezeichnete Station, die die Funktion eines Accesspoints übernimmt. Dies ist vergleichbar mit dem herkömmlichen kabelgebundenen Ethernet, bei dem alle an einem Bussegment angeschlossenen Rechner miteinander kommunizieren.

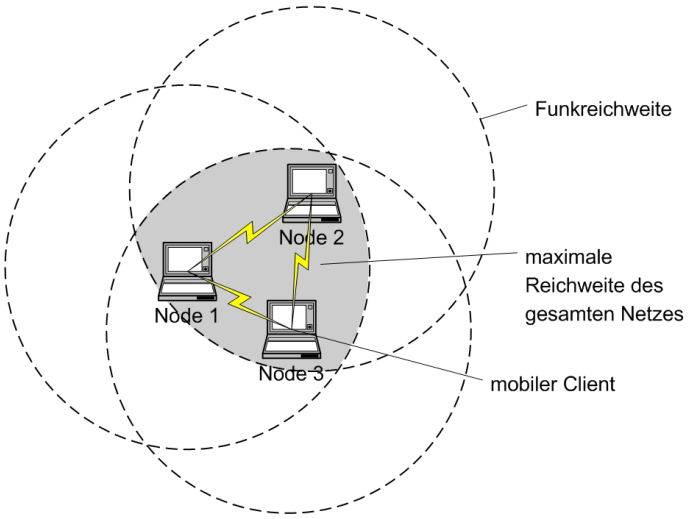

Abbildung 2.4: Ad-Hoc Modus

Im in Abbildung 2.4 dargestellten Ad-Hoc Modus kommunizieren alle Nodes direkt miteinander. Die maximale Reichweite (grau dargestellt) reduziert sich also auf den von allen Teilnehmern abgedeckten Bereich.

Neben den im IEEE 802.11-Standard definierten Ad-Hoc Modus implementieren einige Hersteller von WLAN-Hardware eigene proprietäre Verbindungsmodi, welche in deren Software häufig ebenfalls Ad-Hoc Modus genannt wird. Diese Modi sind nicht kompatibel mit dem

dann häufig als IBSS-Mode bezeichneten in IEEE 802.11 vorgesehenen Standard. Wenn im folgenden von Ad-Hoc Mode gesprochen wird, so meint dies immer den in IEEE 802.11 definierten.

Bei der hier vorgestellten Ad-Hoc-Vernetzung geht es jedoch immer um Verbindungen auf den untersten zwei Ebenen des OSI-Modells. Verbindungen auf IP-Ebene, bzw. die automatische Konfiguration von IP-Adressen sind nicht im IEEE 802.11-Standard definiert. Auch erweiterte Funktionen wie Portale für Ad-Hoc-Netze oder das Verbinden mehrerer Independent Basic Service Sets sind nicht im IEEE 802.11-Standard vorgesehen. Dies ist der Punkt, an dem das Konzept der mobilen Spontan-Netze ansetzt, welches im folgenden Kapitel vorgestellt wird.

2.3 Mobile spontane Netze (MANET)

Wenn im folgenden von mobilen spontanen Netzen (*Mobile Ad-Hoc Networks*) oder kurz *MANET*s die Rede ist, so sind damit sich selbstorganisierende (spontane) Netzwerke gemeint, deren Funktionalität über das Ad-Hoc-Konzept aus dem IEEE 802.11-Standard hinaus geht. In MANETs agieren alle Teilnehmer als Router. So können auch Teilnehmer ohne direkten (Funk-)Kontakt durch Paketweiterleitung miteinander kommunizieren. MANETs können dabei als eigenständige autonome Netze oder als Erweiterung bestehender, fester Netze fungieren. Solche Erweiterungsnetze werden auch als Stummel- bzw. *Stub*- Netzwerke bezeichnet (siehe [CM99]).

Obwohl das Konzept der MANETs theoretisch auf alle drahtlosen Übertragungstechniken anwendbar ist, wird im folgenden nur auf Netze eingegangen, die mittels WLAN-Technologie aufgebaut werden.

Mobile spontane Netze haben besondere Eigenschaften, welche sie gegenüber herkömmlichen Netzen auszeichnen.

So sind alle teilnehmenden Stationen mobil, das heißt sie sind frei, sich beliebig zu bewegen. So könnten Nodes beispielsweise Computer in Fahrzeugen, Schiffen oder Flugzeugen sein oder aber mobile Geräte wie PDAs oder Laptops. Auch der Einbau in Kleidung (*wearable computing*) ist denkbar. Diese Mobilität der einzelnen Nodes führt dazu, dass sich die Netzwerktopologie unvorhersehbar und plötzlich ändern kann.

Außerdem sind drahtlose Verbindungen heute und vorraussichtlich auch in Zukunft mit wesentlich geringerer verfügbarer Bandbreite ausgestattet als herkömmliche kabelgebundene Netze — die maximalen Übertragungsraten für WLAN liegen heute bei 54 MBit, in drahtgebundenen Netzwerken sind 100 MBit die Regel und Gigabitnetzwerke sind bereits verfügbar. Zudem liegt der reale Durchsatz der Verbindung durch Störungen meist weit unter dem angegebenen Maximalwert. Dies führt in solchen Netzen schnell zur Überlastung (siehe [CM99], Kapitel 3).

Oben genannte Geräte sind außerdem meist von autonomen Energiequellen wie Batterien oder Akkus abhängig, deren über die Zeit nachlassende Leistung ebenfalls zu einer Schwächung

der Signalstärke führen kann. Oberste Priorität für diese Geräte ist damit das Sparen von Energie.

Aus diesen Eigenschaften ergeben sich spezifische Anforderungen für MANETs.

Es muss ein verteiltes IP-Routing-Protokoll gefunden werden, welches effektiv auch größere Netze verwalten kann und schnell auf Änderungen der Netzwerktopologie reagieren kann. Die Effektivität bezieht sich dabei sowohl auf die Belastung des Netzes durch den Routingtraffic als auch auf das effektive Verwalten der Routinginformationen. Das Aufteilen eines größeren Netzes in zwei oder mehrere Teilnetze, sowie das Verschmelzen mehrerer Netze sollte ebenfalls möglich sein (*Split and Merge*). Zu diesem Zweck wurde innerhalb der IETF (*Internet Engineering Task Force*) eine Working Group zum Thema MANETs gegründet, welche sich der Evaluierung und Standardisierung geeigneter Routingprotokolle verschrieben hat (siehe [18]). Einige der vorgeschlagenen Routingprotokolle für mobile spontane Netze werden in Kapitel 3 vorgestellt und getestet.

Wichtig ist auch die Selbstkonfiguration der einzelnen Nodes — das Teilnehmen an einem MANET sollte ohne Interaktion seitens des Benutzers erfolgen. Eine manuelle Konfiguration ist vor allem bei vielen Nodes nicht nur unpraktikabel, sondern widerspräche auch der Anforderung einer spontanen Vernetzung. Wie in MANETs IP-Adressen ohne zentrale Verwaltung automatisch konfiguriert werden können, wird in Kapitel 4 gezeigt.

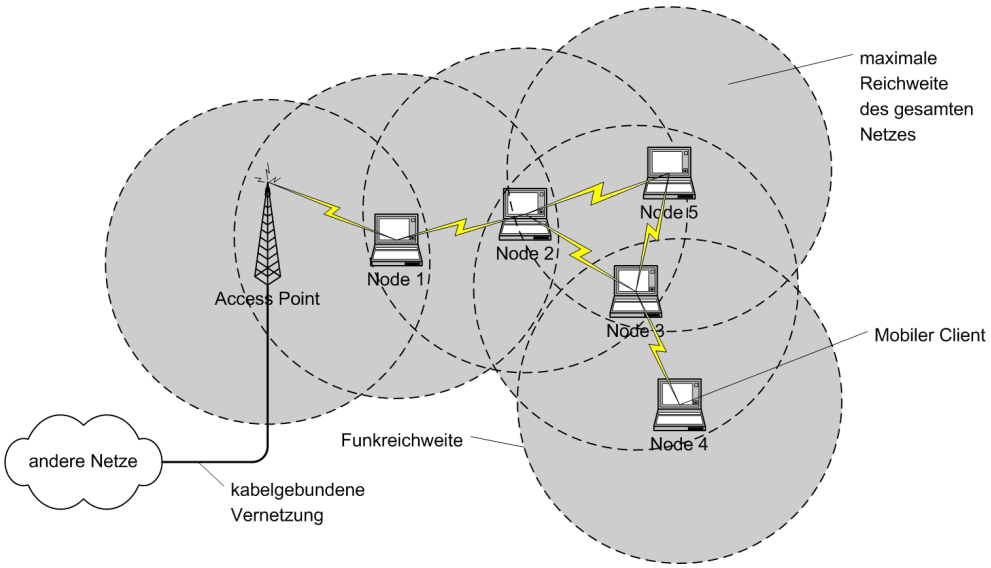

Abbildung 2.5: MANET

Abbildung 2.5 zeigt ein als Stub Network konfiguriertes MANET mit fünf Nodes und einem Accesspoint. Jeder Node fungiert hier gleichzeitig als Router und vergrößert so die Reichweite des Netzes. Beispielsweise kann Teilnehmer 4, obwohl er sich weit außerhalb der Reichweite des Accesspoints befindet, über die Nodes 3, 2 und 1 auf die, über den Accesspoint erreichbaren, anderen Netze zugreifen.

3 Routing

Erst durch den Einsatz von geeigneten Routingverfahren ist es möglich, MANETs aufzubauen. Jeder Teilnehmer des Netzes fungiert dabei als Router welcher Pakete weiterleitet (vgl. Abb. 2.5). MANETs stellen dabei besondere Anforderungen an das Routing. Da die Topologie des Netzes nicht bekannt ist und ständigen Änderungen unterworfen ist, muss die Verwaltung der Routinginformationen vollkommen automatisch durchgeführt werden. Das heißt, die Nodes müssen Nachbarn automatisch erkennen und ihre Routinginformationen entsprechend anpassen. Das verwendete Protokoll muss in der Lage sein, auf eine sich möglicherweise sehr schnell ändernde Netzwerkstruktur zu reagieren. Information über nicht mehr bestehende Verbindungen (*broken links*) müssen an die betroffenen Nodes zeitnah weiter verteilt und alternative Routen gefunden werden. Dabei darf die Belastung des Netzwerkes durch das Routingprotokoll selbst nicht zu hoch sein, da wie in Kapitel 2.3 beschrieben, Bandbreite und Leistungsfähigkeit heutiger WLAN-Technik begrenzt ist.

Herkömmliche aus drahtgebundenen Netzen bekannte Routingalgorithmen sind diesen Anfordeungen nicht gewachsen. Aus der Vielzahl der existierenden bzw. vorgeschlagenen, speziell für mobile Spontannetze konzipierten Routingalgorithmen (eine ausführliche Auflistung findet sich unter [2]), sollen im folgenden die wichtigsten vorgestellt und bewertet werden. Die Auswahl beschränkt sich dabei auf die bekannteren Protokolle, für die es bereits erste Implementationen gibt und die teilweise auch von Mitgliedern der MANET Working Group (siehe [18]) entwickelt wurden.

Um die einzelnen Verfahren besser beurteilen zu können, werden zunächst deren Prinzipien im einzelnen vorgestellt, um anschließend verschiedene Eigenschaften in einem Netzwerksimulator zu untersuchen und miteinander zu vergleichen.

3.1 Mobile Routingprotokolle

Routingverfahren werden im Allgemeinen in verschiedene Kategorien eingeteilt. Grob unterscheidet man zwischen *adaptiven* und *nicht-adaptiven* Verfahren. Während sich adaptive Verfahren automatisch an sich verändernde Netzwerktopologien anpassen, funktionieren nicht-adaptive Routing-Verfahren ausschließlich auf Basis fester Routingtabellen. Für MANETs kommen daher nur adaptive Verfahren in Frage. Eine weitere Einteilung erfolgt in *proaktive* und *reaktive* Verfahren. Erstere erstellen Routingtabellen für die gesamte Netzwerktopologie mit allen in dieser vorhandenen Nodes, selbst wenn nie ein Paket an einige dieser Rechner geschickt wird. Die reaktiven Verfahren erstellen Routingtabellen hingegen nur bei Bedarf, so

wird die Route zu einem Ziel erst berechnet, wenn ein Paket an dieses Ziel verschickt werden soll (siehe [Rot02]).

Grundsätzlich sind alle hier vorgestellten Verfahren nicht an die WLAN-Technologie oder den Einsatz des IP-Protokolls gebunden — sie wären auch mit beliebigen anderen Adressierungs- und Datenübertragungsverfahren einsetzbar. Im folgenden soll jedoch nur auf ihre Eigenschaften in Bezug ihre Einsatzfähigkeit in IP- und WLAN-basierten MANETs eingegeangen werden.

3.1.1 Destination-Sequenced Distance-Vector – DSDV

Das DSDV-Verfahren ist eines der ersten speziell für MANETs entwickelten Routingverfahren. Es handelt sich dabei um ein proaktives Verfahren, welches auf einem älteren, für stationäre Netze entwickelten Verfahren basiert — dem *Distributed Bellman-Ford* (DBF) Verfahren. Um das DSDV besser zu verstehen, soll zunächst kurz das DBF-Verfahren erläutert werden.

DBF

In *Distance Vector* Verfahren wie dem DBF werden Routingtabellen angelegt, die für jeden Knoten im Netz festlegen, über welchen Nachbarn dieser erreichbar ist und wie groß die Gesamtdistanz zum Ziel ist. Dabei wird der Nachbar als *next Hop* und die Entfernung zum Ziel als *Metrik* bezeichnet. Die Metrik wird dabei in der Regel in Anzahl der Paketweiterleitungen (*Hops*) bis zum Ziel angegeben. Dabei ist bei Distance Vector-Verfahren nicht der komplette Weg zum Ziel bekannt, sondern immer nur der nächste Schritt — der Weg des Pakets wird also erst auf der „Reise" ermittelt.

Das DBF-Verfahren legt fest, wie die Routingtabellen aufgebaut werden. Dies geschieht zum einen durch Messen der Entfernung zu den Nachbarn und zum anderen durch das Austauschen von *Distanz Vektoren*. Ein Distanz Vektor enthält drei Informationen: zum einen den Knoten, von dem er ausgesendet wurde, zum anderen einen Zielknoten und schließlich die Entfernung (Metrik) zu diesem Knoten. Auf Basis der erhaltenen Distanz Vektoren können die eigenen Routingtabellen um Einträge, die keine direkten Nachbarn sind, erweitert werden. Dabei ersetzen Einträge mit kleineren Metriken immer Einträge höherer Metriken — so wird immer der kürzeste Weg genommen. Durch das periodische Austauschen der Distanz Vektoren verbreitet sich die Distanzinformationen im gesamten Netzwerk (siehe [Rot02], Seite 199).

Das große Problem des DBF-Verfahrens ist die Reaktion auf unterbrochene Verbindungen. Abbildung 3.1 zeigt ein Netzwerk und die Routingtabellen der Nodes 1, 2 und 3. Wird jetzt die Kommunikation zwischen den Nodes 1 und 2 unterbrochen, so ändert Node 2 die Metrik für den Knoten 1 auf unendlich. Von Node 3 wird nun jedoch der Distanz Vektor mit der Information, das Node 1 mit 2 Hops von Node 3 aus erreichbar ist, übertragen. Node 2 weiss nicht, dass die Route von Node 3 über Node 2 selbst und die eigentlich nicht mehr verfügbare Route geht und ändert seinen eigenen Routing Eintrag für Node 1 entsprechend. Beim nächsten Austausch der Distanz Vektoren erfährt Node 3 nun jedoch, dass sich die Metrik zu

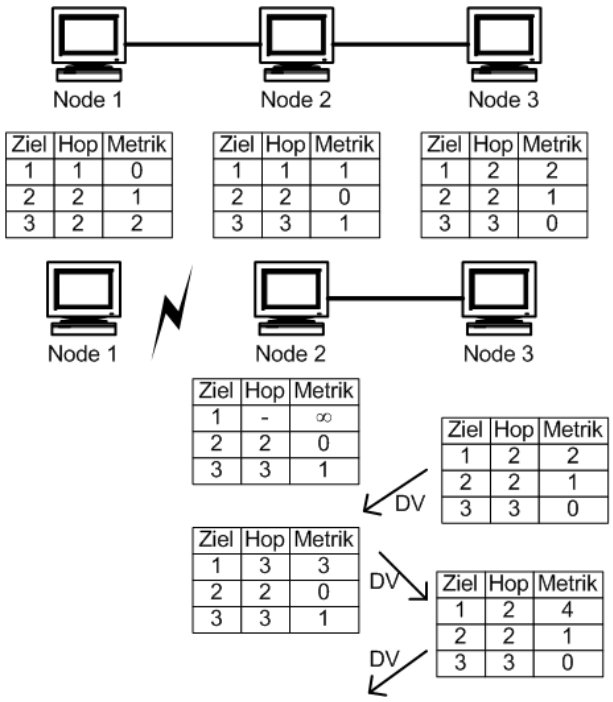

Abbildung 3.1: Das Count to Infinity Problem

Node 1 erhöht hat und Node 3 passt seine Routingtabelle entsprechend an. Diese Änderungen schaukeln sich immer weiter auf. Dieses Problem wird auch *Count to Infinity* genannt.

DSDV

Da Störungen in Ad-Hoc-Netzen relativ häufig auftreten, musste das Count to Infinity-Problem im DSDV-Verfahren gelöst werden.

Wie beim DBF-Verfahren werden auch bei DSDV periodisch oder immer dann wenn eine Topologieänderung erkannt wird, Distanzinformationen an alle Nachbarn versandt. Dabei können entweder nur die Änderungen (*incremental*) oder bei größeren Änderungen die kompletten Distanzinformationen (*full dump*) übertragen werden. Zusätzlich zu den Hop- und Metrikeinträgen speichert DSDV eine Sequenznummer zu jedem Routingeintrag, welche die Aktualität der Information kennzeichnet. Die Sequenznummer wird dabei immer vom Ziel der Route vergeben und bei jedem Verbreiten der Distanzinformation um zwei erhöht. Einträge werden nur aktualisiert, wenn sich entweder die Sequenznummer erhöht oder sich die Gesamtdistanz bei gleicher Sequenznummer verringert hat.

Um das Count to Infinity-Problem zu umgehen, gibt es in DSDV einen zusätzlichen Mechanismus um Netzwerkstörungen zu bearbeiten. Entdeckt ein Knoten eine Störung (*broken link*), entweder durch Ausbleiben von Distanzinformationen eines Nachbarn oder durch entsprechende Mechanismen in der Sicherungsschicht, so setzt er die Metrik aller Routen über

diese Hops auf unendlich und erhöht beim Versenden der Distanzinformationen die zugehörige Sequenznummer um eins. Dies ist der einzige Fall, bei dem die Sequenznummer von einem anderen Knoten als dem Ziel der Route vergeben wird.

Das Erhöhen der Sequenz um eins stellt zwei Dinge sicher: Zum einen sorgt die erhöhte Sequenznummer dafür, dass sich die Information des nicht erreichbaren Knotens im gesamten Netz verbreitet und dabei Vorrang vor älteren Einträgen hat. Zum anderen kann der als nicht erreichbar erkannte Knoten seine Anwesenheit bzw. eventuell neue Verbindungen selbst mit höherer Priorität (um zwei erhöhte Sequenznummer) bekannt geben (siehe [Rot02], Seite 200ff). Dies ist besonders wichtig für Ad-Hoc-Netze, da sich hier durch die Bewegung der mobilen Teilnehmer die Topologie häufig umstrukturiert.

Bei der beschriebenen Art Routingtabellen zu erneuern, kann es unter Umständen zu unerwünscht starken Fluktuationen kommen. Angenommen ein Knoten erhält zwei Routen zum selben Ziel mit der gleichen Sequenznummer, jedoch zuerst die Route mit der schlechteren (größeren) Metrik. Der entsprechende Knoten würde zuerst die schlechtere Route eintragen und verbreiten, um dann kurz darauf die Änderung mit der besseren Route bekannt zu geben. Diese Änderungen würden sich jedesmal im gesamten Netz fortsetzen (siehe [PB96], Seite 196).

DSDV enthält daher ein zusätzliches Verfahren zum Dämpfen der Fluktuationen. Erreicht wird dies durch Verzögern des Sendens der Routingupdates. Zusätzlich zu den Routinginformationen wie Hop und Metrik, wird zu jeder Zieladdresse abgespeichert wie lange die entsprechende Route bereits besteht (*settling time*) und wie lange Routen zu diesem Ziel durchschnittlich Bestand haben (*avarage settling time*). Anhand dieser Zeiten kann der Knoten nun entscheiden, wie lange er vor dem Verteilen der neuen Route wartet. Im Idealfall sollte er vor dem Ablauf dieser Zeit bereits die zweite Route mit der besseren Metrik empfangen haben und dann diese verteilen. Zu beachten ist, das Informationen zu broken links immer sofort und ohne Verzögerung verteilt werden (siehe [PB96], Seite 196ff).

Das DSDV-Verfahren basiert auf der Annahme, dass alle Verbindungen *bidirektional* sind. Dies ist gerade bei Funknetzen durch unterschiedliche Sendeleistungen nicht immer gegeben. So kann ein Knoten einen starken Sender möglicherweise noch empfangen, hat aber selbst nicht genügend Sendeleistung zum Antworten. DSDV sieht daher vor, dass Knoten nur Routinginformationen von Nachbarn akzeptieren, die zuvor gezeigt haben, dass sie in der Lage sind, ebenfalls Pakete des jeweiligen Knotens zu empfangen (siehe [PB96], Seite 189). Für MANETs basierend auf WLAN nach IEEE 802.11 spielt diese Überlegung jedoch keine Rolle, da alle Protokolle der 802-Familie nur bidirektionale Verbindungen zulassen.

3.1.2 Ad-Hoc On-demand Distance Vector Routing – AODV

Das AODV-Protokoll entstand aus den Erfahrungen mit dem DSDV-Protokoll (vgl. Kapitel 3.1.1). Systemweite Broadcasts sollen hier möglichst vermieden werden. Dementsprechend handelt es sich bei AODV um ein reaktives Verfahren (vgl. 3.1), bei dem nur bei Bedarf Routen berechnet werden (siehe [PR01], Seite 173).

Bei AODV speichert jeder Knoten neben Zieladresse, Metrik und dem Next Hop-Eintrag genau wie DSDV eine Sequenznummer. Zusätzlich wird zu jedem Eintrag eine Liste von Knoten geführt, welche den vollständigen Weg zum Ziel enthält. Um veraltete Einträge zu vermeiden, enthält jeder Routingeintrag eine Lebensspanne, die bei jeder Benutzung der Route erneuert wird. Wird eine Route längere Zeit nicht benutzt, wird sie aus der Routingtabelle gelöscht (siehe [PR01], Seite 176).

Will ein Knoten ein Paket an einen Teilnehmer verschicken, welcher sich nicht in direkter Nachbarschaft befindet, so versendet er zunächst eine *Route Request*-Nachricht (RREQ) per Broadcast an alle seine Nachbarn. Die RREQ-Nachricht enthält zum einen die IP-Adresse des Absenders, die aktuelle eigene Sequenznummer, sowie die IP-Adresse des Zielknotens und die letzte zu diesem Knoten bekannte Sequenznummer. Zusätzlich bekommt die Nachricht noch eine *Broadcast ID* welche vom Absender bei jedem Versenden eines Broadcasts erhöht wird. Zusammen mit der Absender-IP ergibt diese eine systemweit eindeutige Identifikation des Routing Requests (siehe [PR01], Seite 177).

Empfängt ein Knoten ein RREQ-Paket, so überprüft er zunächst, ob er bereits ein Paket mit gleicher Broadcast-ID und Absenderadresse empfangen hat. Dafür verwaltet jeder Knoten eine Liste empfangener Broadcasts für einen gewissen Zeitraum. Wurde das Paket bereits einmal empfangen wird es ignoriert — so werden im Kreis laufende Pakete vermieden. Ist das Paket jedoch neu, wird es wie folgt verarbeitet:

Zunächst wird ein *Reverse Routing*-Eintrag erstellt, der eine Route zum Absender des RREQ erstellt. So ist sichergestellt, dass der Knoten später eine mögliche Antwort auf den Request zurück an den Absender leiten kann. Vorraussetzung sind auch hier wieder die von IEEE 802 sichergestellte bidirektionale Verbindungen.

Um auf einen Routing Request antworten zu können, muss der Knoten eine nicht abgelaufene Route zum Ziel kennen, deren Sequenznummer größer oder gleich der im Routing Request enthaltenen Ziel-Sequenznummer ist. Ist keine solche Route bekannt, wird der im Routing Request enthaltene Hop-Count um eins erhöht und das Paket erneut gebroadcasted. Natürlich ist der Zielknoten immer in der Lage, auf das RREQ-Paket zu antworten (siehe [PR01], Seite 177f).

Um den Einfluss dieser Route Request Broadcasts auf das gesamte Netz möglichst gering zu halten, gibt es in AODV einen zusätzlichen *Expanding Ring Search* genannten Mechanismus. Dabei wird das RREQ-Paket mit einer Lebenspanne (*TTL*) versehen, die bei jeder Weiterleitung verringert wird. Wird mit der anfänglichen Lebenspanne keine Route gefunden, so wird diese erhöht und das RREQ-Paket erneut versandt. Muss später eine neue Route zu einem bereits bekannten Ziel gefunden werden, so wird der RREQ gleich mit der zuletzt erfolgreichen TTL versandt. Dieser Mechanismus schränkt die Broadcasts zunächst auf kleinere Teile des Netzes ein und vermeidet so unnötige systemweite Broadcasts (siehe [PR01], Seite 178).

Antworten auf Route Requests erfolgen in *Route Reply* (RREP) Paketen. Diese enthalten, analog zu den RREQ-Paketen, neben dem Absender des Route Requests und der Zieladresse auch die zum Ziel gehörenden Sequenznummer. Diese ist entweder die aktuelle Sequenznummer des Zielknotens, wenn dieser selbst den Request beantwortet oder die dem antwortenen

Knoten bekannte Sequenznummer des Ziels. Zusätzlich enthält der Reply auch die Lebensdauer der Route. Das Reply-Paket wird bis zum Absender des Requests zurückgeleitet. Dabei wird der im RREP-Paket enthaltene Hop-Count bei jeder Weiterleitung um eins erhöht und jeder weiterleitende Knoten trägt die im RREP-Paket enthaltene Route auch in seine eigene Routingtabelle ein. Es ist möglich, dass mehrere Nodes auf ein RREQ-Paket antworten. Die einzelnen Knoten leiten erneute Antworten auf den selben Request jedoch nur weiter, wenn dessen Hop-Count niedriger oder die Ziel-Sequenznummer größer ist (siehe [PR01], Seite 179).

Eine spezielle Art der Route Requests wird zur Nachbarerkennung eingesetzt. In einem bestimmten Interval versendet jeder Knoten sogenannte *HELLO*-Pakete. Dabei handelt es sich um Route Requests mit einer TTL von eins. HELLO-Pakete werden also niemals weitergeleitet, sondern sorgen lediglich dafür, dass die Lebensspannen der Routingeinträge für die Nachbarn erneuert werden bzw. neue Nachbarn eingetragen werden. HELLO-Pakete werden nur gesendet, wenn innerhalb des letzten Intervalls keine anderen RREQ-Pakete versendet wurden (siehe [PR01], Seite 182f).

3.1.3 Dynamic Source Routing – DSR

Auch DSR ist wie AODV ein reaktives Verfahren bei dem Routing-Informationen nur ausgetauscht werden, wenn es nötig ist.

Anders als im AODV-Verfahren, wird im Header eines jeden Datenpaketes auch der komplette Weg zum Ziel in einer Liste von Knoten mitgeschickt. Jeder Knoten leitet das Paket an den nächsten Hop in der Liste weiter.

Will ein Knoten ein Paket an einen Empfänger versenden, dessen Route noch unbekannt ist, so wird ein *Route Discovery* genanntes Verfahren durchgeführt. Dabei wird ähnlich wie bei AODV ein Route Request-Paket an alle Nachbarn verschickt. Das Request-Paket enthält neben Absender- und Zieladdresse auch eine *Request ID*, die wie bei AODV dafür sorgt, dass alle Route Requests im gesamten Netz eindeutig sind. Analog zu AODV werden also auch hier mehrfach empfangene Requests ignoriert. Ist der Knoten, welcher einen Routing Request empfangen hat, nicht der Zielknoten, so fügt er seine eigene Adresse in eine Liste der bereits passierten Knoten im Request-Paket hinzu und leitet das Paket an alle seine Nachbarn weiter (siehe [JM96], Seite 158ff).

Die Antwort des Zielhosts wird analog zu AODV in einem *Route Reply*-Paket verschickt. Dafür ist es nötig, dass eine Route vom Ziel zum Absender des Route Requests besteht. In Netzen nach dem IEEE 802-Standard mit bidirektionalen Verbindungen könnte diese Route durch invertieren der im Route Request enthaltenen Liste erzeugt werden. DSR setzt jedoch keine bidirektionalen Verbindungen vorraus, stattdessen sieht das Protokoll vor, das Route Reply-Paket mit einem neuen Route Request-Paket zum Absender zu verschicken (*piggyback*) (siehe [JM96], Seite 160).

Einmal bekannte Routen werden in, im DSR-Protokoll *Route Cache* genannten, Routingtabellen gespeichert. Dabei können Nodes auch sämtliche Routen speichern, die sie in weitergeleiteten Daten- und Route Reply-Paketen finden.

Der Route Cache erlaubt es Knoten, welche nicht das eigentliche Ziel eines Route Requests sind, diesen trotzdem zu beantworten, sofern sie eine Route zum Ziel kennen. Dabei wird die zurück zu meldende Route aus der im Request enthaltenen Liste der passierten Knoten und der bekannten Route zum Ziel zusammengesetzt. Bedingung ist jedoch, dass in dieser neuen Route keine Knoten doppelt vorhanden sind. Abbildung 3.2 zeigt ein Beispiel bei dem dies der Fall ist: Node 6 empfängt einen Route Request von Node 1 nach Node 5. Das Request-Paket enthält die Liste der bisherigen Knoten: (1,2,3,6) — die Route nach Node 5 enthält die folgenden Knoten: (3,4,5). Die daraus theoretisch entstehende Route (1,2,3,6,3,4,5) ist nach der oben genannten Bedingung jedoch nicht erlaubt. Diese Einschränkung verhindert unnötige Route Replies. Im gezeigten Szenario erhält auch Node 3 den Route Request und kann selbst mit einer besseren, weil kürzeren Route antworten (siehe [JMB01], Seite 149).

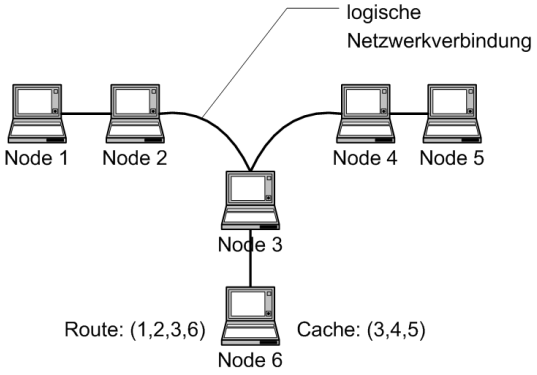

Abbildung 3.2: Vermeidung doppelter Knoten in Route Replies

Um veraltete Routingeinträge erkennen und zu entfernen zu können, stellt DSR ein Verfahren zur Routenpflege (*Route Maintainance*) zur Verfügung. Dabei handelt es sich im wesentlichen um *ERROR*-Pakete, welche an den Absender zurückgeschickt werden, wenn eine Verbindung in der im Datenpaketheader angegeben Route nicht mehr verfügbar ist. Bei der Weiterleitung des ERROR-Pakets löschen Nodes, welche dieses Paket sehen, alle Routen, die die angegebene unterbrochende Verbindung enthalten.

3.1.4 Optimized Link State Routing – OLSR

OLSR gehört zur Klasse der *Link State*-Verfahren. Bei Link State-Verfahren werden nicht nur Informationen zu den Abständenen einzelner Knoten (Distanz Vektoren), sondern auch Informationen über den Aufbau des gesamten Netzes ausgetauscht. OLSR gehört damit zu den proaktiven Verfahren.

Zum Erkennen der Netzwerktopologie wird bei allen Link State-Verfahren ähnlich vorgegangen. Zunächst werden Nachbarn wie beim AODV-Verfahren mittels HELLO-Paketen erkannt (vgl. Seite 20). Optional können für die Metrik statt der Hopanzahl auch Pingzeiten verwendet werden — ist dies der Fall, werden nach dem Erkennen eines Nachbarn *ECHO*-Pakete zur Zeitmessung verwendet. Zur Weitergabe der Nachbarinformationen wird ein Kontrollpaket (*Topology Control Message*, kurz TC) generiert, welches neben dem Absenderknoten und den Informationen zu den Nachbarn auch eine eindeutige Sequenznummer enthält. Dieses Kontrollpaket wird via Broadcast an alle Nachbarn versandt. Empfangene Kontrollpakete werden, sofern die Sequenznummer höher ist als in bisher empfangenen Kontrollpaketen des Absenderknotens, an alle Nachbarn weitergeleitet und verteilen sich so im gesamten Netz.

Über die in den, periodisch von allen Knoten des Netzes versandten, Kontrollpaketen enthaltenen Nachbarinformationen sind alle Knoten in der Lage, ein ein lokales Abbild der Netzwerktopologie zu erstellen. Mit diesem kann dann über geeignete Verfahren, wie beispielsweise dem *Dijkstra Algorithmus*, die beste Route für zu versendene Pakete bestimmt werden (siehe [Rot02], Seite 211-212).

OLSR verfolgt zwei Ansätze zur Optimierung des beschriebenen Link State-Verfahrens. Zum einen sollen die Kontrollpakete möglichst klein gehalten werden und zum anderen das Broadcasting möglichst minimiert werden.

OLSR führt dafür sogenannte *Multipoint Relays* ein. Diese Multipoint Relays sind Knoten, die bei der Weiterleitung der Kontrollnachrichten besondere Funktionen übernehmen. An Abbildung 3.3 soll demonstriert werden wie diese funktionieren. Gezeigt wird ein Netz mit 10 Nodes: N(N1) ist dabei die Menge der Nachbarn von Node 1, N2(N1) die Menge der Knoten, die exakt zwei Hops von Node 1 entfernt sind. Die Menge MPR(N1) definiert die Knoten, welche von Node 1 als Multipoint Relay ausgewählt wurden. Dabei kann MPR(N1) eine beliebige Untermenge von N(N1) sein, solange jeder Knoten der Menge N2(N1) eine Verbindung zu einem Multipoint Relay hat (siehe [Rot02], Seite 214). MPR(N1) muss dabei nicht minimal sein, je kleiner jedoch die Menge, desto größer ist der Effizienzvorteil gegenüber dem klassischen Link State-Verfahren.

OLSR sieht vor, die HELLO-Pakete des klassischen Link State-Verfahrens um die Menge der Nachbarn eines Knotens und seiner momentanen Multipoint Relays zu erweitern. Mit dieser Information kann jeder Knoten bereits über den Austausch von HELLO-Paketen ermitteln, welche Knoten über zwei Hops erreichbar sind. In Abbildung 3.3 kann Node 1 also neben der Menge N(N1) auch die Menge N2(N1) ermitteln. Ausserdem kann ein Knoten aus den HELLO-Paketen ermitteln, für welche Knoten er selbst ein Multipoint Relay darstellt. Sobald der Knoten eine Änderung in der Menge N(N1) oder N2(N1) erkennt, muss die Menge MPR(N1) neu aufgebaut werden.

Die Einführung der Multipoint Relays ermöglicht es, statt bisher alle Nachbarn in den Kontrollnachrichten zu verbreiten, nur noch die Menge der Nachbarn, welche den Knoten als Multipoint Relay gewählt haben zu versenden. Damit verkleinert sich die Größe der Kontrollnachrichten und das Netzwerk wird weniger belastet. Ist der Knoten kein Multipoint Relay,

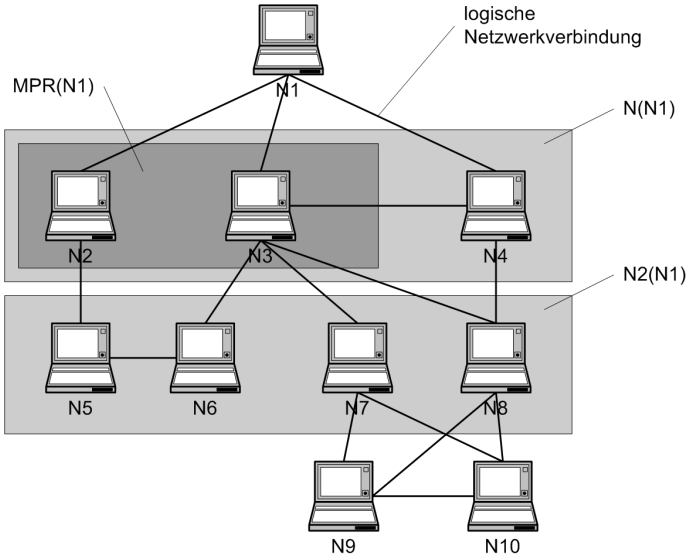

Abbildung 3.3: OLSR mit den Mengen N, N2 und MPR

versendet er keine Kontrollnachrichten. Zudem werden beim Broadcasten der Kontrollnachrichten nur noch Nachrichten weitergeleitet, die von Knoten kommen, welche den aktuellen Knoten als Multipoint Relay (MPRsel genannt) selektiert haben. Alle anderen Knoten verarbeiten zwar die Nachricht, senden sie jedoch nicht mehr weiter — Kontrollnachrichten werden also nur noch über die Multipoint Relays verteilt.

Um die Multipoint Relays zu finden, geht ein Knoten — in Abbildung 3.3 Node 1 — wie folgt vor: Begonnen wird mit einer leeren Menge MPR(N1), zu der alle Nodes aus N(N1) hinzugefügt werden, welche lediglich Verbindungen zu N2(N1) haben. Im Beispiel sind dies Node 2 und Node 3 — Node 4 hat neben der Verbindung zur Menge N2(N1) auch eine Verbindung zu anderen Knoten in N(N1). Solange noch Knoten in N2(N1) vorhanden sind, welche nicht über einen Knoten aus MPR(N1) erreichbar sind, wird das Verfahren fortgesetzt und der Knoten aus N(N1) zu MPR(N1) hinzugfügt, der die meisten noch nicht abgedeckten Knoten aus N2(N1) erreicht. Haben mehrere Knoten die gleiche Anzahl Verbindungen, so wird der Knoten mit der größten Anzahl von Nachbarknoten gewählt. Um die MPR(N1) möglichst klein zu halten, werden alle Knoten aus MPR(N1) entfernt, bei denen trotz des Entfernens noch alle Knoten aus N2(N1) über Multipoint Relays erreichbar sind (siehe [Rot02], Seite 215).

Um aus den Informationen der Kontrollnachrichten die kürzeste Route zu einem Ziel berechnen zu können, benötigt OSLR eine *Topologietabelle*, in der die Informationen der Kontrollnachrichten gespeichert werden. Jede Kontrollnachricht enthält, wie beschrieben, den Absender und die Menge der Knoten, welche diesen als Multipoint Relay gewählt haben (MPsel). Pro Zeile werden dabei der Knoten von dem die Nachricht kam und ein *Selektor* genannter Knoten aus der Menge MPRsel gespeichert. Für jeden Knoten in MPRsel wird also ein Eintrag in der Topologietabelle angelegt.

Zum Erzeugen der Routingtabelle mit Ziel, Next Hop und Metrik wird nun wie folgt vorgegangen:

Zunächst werden alle Nachbarn mit der Metrik 1 und ein Eintrag für den Knoten selbst (mit Metrik 0) angelegt und ein *Hop Counter* mit 1 initialisiert. Nun wird die Topologietabelle zeilenweise durchlaufen und alle Einträge, deren Selektor bereits als Ziel in der Routingtabelle vorhanden ist, entfernt. Ist die Topologietabelle nach diesem Schritt leer, so ist der Algorithmus abgeschlossen. Wenn nicht, wird die Topologietabelle erneut durchlaufen und für alle Knoten, für die es einen Zieleintrag in der Routingtabelle gibt und dessen Metrik dem Hop Counter entspricht, ein neuer Eintrag in der Routingtabelle angelegt. Als Ziel wird der zum Knoten gehörende Selektor verwendet, als Next Hop wird der Hop des gefundenen Knotens in der Routingtabelle verwendet und die Metrik entspricht dem um eins erhöhten Hop Counter. Dieses Verfahren wird nun wiederholt, wobei der Hop Counter um eins erhöht wird (vgl. Abbildung 3.4).

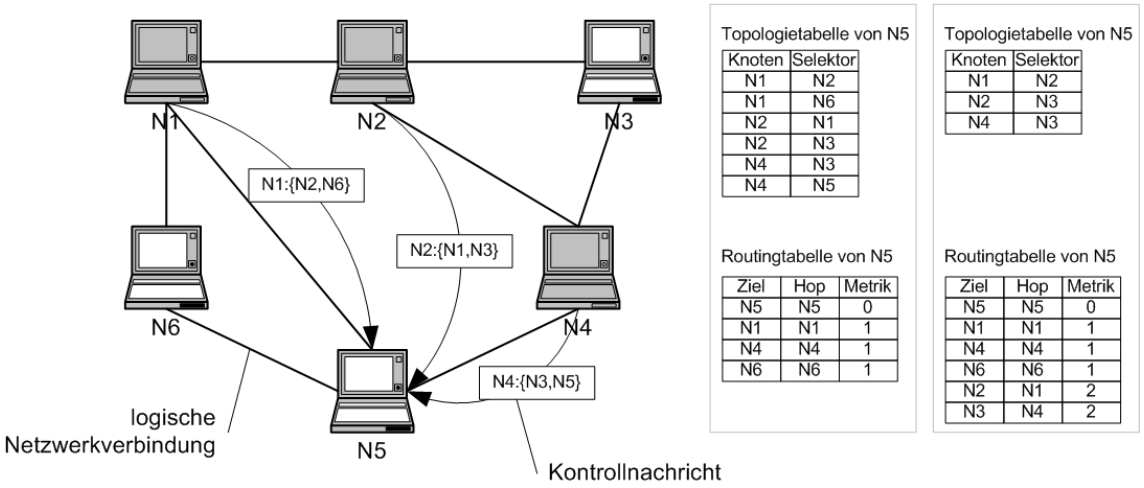

Abbildung 3.4: Erzeugen der Routingtabelle in OLSR

Abbildung 3.4 zeigt das Verfahren zur Erzeugung der Routingtabelle für den Knoten 5. Nur die Nodes 1, 2 und 4 sind Multipoint Relays und versenden Kontrollpakete, aus deren Informationen zunächst die oben links gezeigte Topologietabelle aufgebaut wird. Nach dem Eintragen der Nachbarn und des Knotens selbst (N5, N1, N4 und N6) werden alle Einträge, deren Selektor schon als Ziel in der Routingtabelle vorhanden ist, aus der Topologietabelle entfernt (oben rechts). Im anschließenden Schritt werden die Selektoren als Ziel in die Routingtabelle übernommen, für deren Knoten ein Ziel mit der Metrik 1 besteht (N2 und N3). Nach der anschließenden Bereinigung der Topologietabelle ist diese leer und das Verfahren abgeschlossen.

3.1.5 Temporally-Ordered Routing Algorithm – TORA

Das TORA Verfahren gehört zur Klasse der *Link-Reversal-Routing*-Protokolle (LRR) und geht damit einen anderen Weg als die bisher vorgestellten Verfahren. Während die bisherigen Rou-

tingalgorithmen versuchen, einen möglichst optimalen Weg zum Ziel zu finden, geht es bei den Link Reversal-Protokollen darum, überhaupt einen Weg zu finden: dieser muss nicht optimal sein. Um TORA besser zu verstehen, wird zunächst kurz auf die Link Reversal-Verfahren im Allgemeinen eingegangen.

Link Reversal Routing

Link Reversal-Verfahren nutzen die Graphen-Theorie, um den Weg zu einem gewünschten Ziel zu ermitteln. Dabei erzeugt jeder Knoten des Netzes je einen Graphen für jeden möglichen Zielknoten. Idee dabei ist es, eine Routingentscheidung durch einfaches „ablesen" aus dem zum Zielknoten gehörenden Graphen treffen zu können.

Um dies zu ermöglichen, werden sogenannte *directed acyclic graph*s (DAG) eingesetzt. Zum Erstellen eines solchen DAG werden alle Knoten des Netzes in den Graphen übertragen und Knoten in Kommunikationsreichweite durch gerichtete Kanten verbunden. Welche Richtung eine bestimmte Kante dabei hat, ist zunächst egal. Bedingung ist jedoch, dass der Graph keine Zyklen, also kreisförmig angeordnete Kanten, aufweist. Man beachte, dass es sich hierbei nur um eine Hilfe zur Wegfindung zu einem einzelnen Ziel handelt und nicht um den Aufbau des Netzes selbst. In diesem sind weiterhin alle Verbindungen nach IEEE 802 bidirektional.

Die Verwendung von DAGs für das Routing ermöglicht, dass beliebig viele verschiedene Routen zu einem Ziel verwaltet werden können. So kann flexibel auf den Wegfall einzelner Nodes reagiert werden.

Zunächst sollen noch zwei weitere Begriffe eingeführt werden: *Downstream* und *Upstream*. Diese bezeichnen die Kanten, die zu einem bestimmten Knoten hin- (Upstream) bzw. wegführen (Downstream).

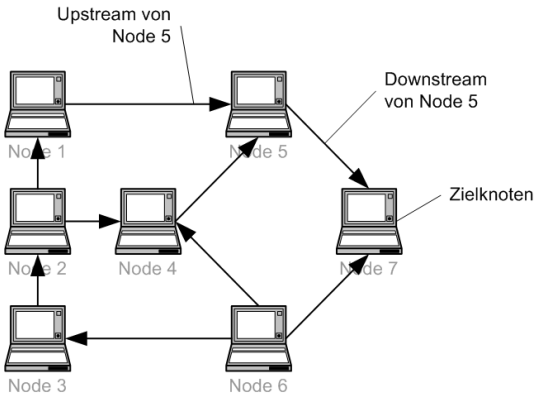

Abbildung 3.5: Beispiel für einen einfachen zielgerichteten DAG

Der hier gezeigte zielgerichtete DAG kann zum Finden einer Route zum Knoten 7 genutzt werden. Bei der Entscheidung an welchen Knoten das entsprechende Paket weitergeleitet werden soll, reicht es einen beliebigen Downstreamnachbarn auszuwählen.

Prinzip des Verfahrens ist, den Graphen zum Finden einer Route in einen *zielorientierten* DAG zu verwandeln. In einem zielorientierten DAG existiert nur ein einziger Knoten, der keine Downstreams besitzt: der *Zielknoten*. Wählt man nun einen DAG, dessen Zielknoten dem gewünschten Routingziel entspricht, lassen sich alle Routen zu diesem Ziel aus dem Graphen ablesen. Abbildung 3.5 zeigt einen solchen zielgerichteten DAG mit Node 7 als Zielknoten.

Die einzelnen Link Reversal-Verfahren unterscheiden sich nur in der Art wie ein zielorientierter DAG aufgebaut und bei Änderungen in der Netztopologie erhalten wird. In den meisten Fällen werden dafür Regeln zum Umkehren (Link Reversal) aller oder bestimmter Kanten verwandt, auf die hier nicht näher eingegangen werden soll. Letztendlich geht es dabei immer um die Frage wie ein Knoten behandelt wird, der nicht der Zielknoten ist und der all seine Downstreams verloren hat (siehe [Rot02], Seite 221).

TORA

Auch das TORA-Verfahren nutzt DAGs für die Wegfindung. Die Richtung der einzelnen Kanten wird dabei jedoch von einer für jeden Knoten festgelegten *Höhe* bestimmt. Nachbarknoten mit niedrigerer Höhe sind Downstream-Nachbarn und Knoten mit größerer Höhe sind Upstream-Nachbarn (vgl. Abbildung 3.6). Der Datenfluss von einem Knoten zum anderen erfolgt dabei immer „bergab" zum tiefstgelegenen Nachbarknoten — vergleichbar mit dem Fluss von Wasser.

Wie bei allen Link Reversal-Verfahren, muss auch hier definiert werden wie mit Knoten umgegangen wird, die alle Downstreams verloren haben. Im Höhenmodell von TORA stellen solche Knoten sozusagen ein lokales Tal dar, in welchem sich das Wasser sammelt, anstatt zum tiefergelegenen Zielknoten weiterzufließen. Folgerichtig muss die Höhe des entsprechenden Knoten so angepasst werden, dass es zumindest einen Downstream-Nachbarn gibt (siehe [Rot02], Seite 224).

Um das Verständnis zu erleichtern, soll in diesem Kapitel der physikalische Rechner, auf dem das TORA-Protokoll ausgeführt wird, *Host* genannt werden, während sich die Bezeichnung *Knoten* auf einen beliebigen Knoten im DAG bezieht.

Die von TORA genutzte Höheninformation eines Knotens setzt sich aus fünf Werten ($HEIGHT = (\tau, oid, r, \delta, i)$) mit folgenden Bedeutungen zusammen(siehe [33], Kapitel 4.4):

τ	Zeit, zu der die Referenzhöhe erzeugt wurde
oid	ID des Knotens, der die Referenzhöhe erzeugt hat
r	reflection flag (0 oder 1)
δ	Delta-Höhe
i	ID des Hosts

Dabei bilden die ersten drei Werte die sogenannte Referenzhöhe, die nach jedem Ändern der Höhe eines Knotens an alle Nachbarn in einem UPD-Paket verteilt wird. Um die Gesamthöhen miteinander zu vergleichen, wird ein lexikalischer Vergleich aller fünf Werte zusammen

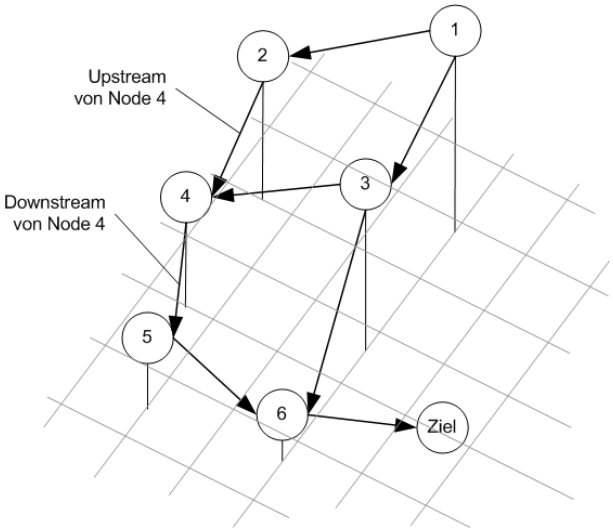

Abbildung 3.6: Ein zielgerichteter DAG mit höhenbasierten Kantenrichtungen

durchgefürt. Wird im folgenden vom höchsten oder niedrigsten Nachbarn gesprochen, so ist diese lexikalische Gesamthöhe gemeint.

Ein Problem bringt die Verwendung eines Zeitstempels zur Erzeugung der Höhen mit sich: Alle Teilnehmer des Netztes müssen synchron laufende Uhren haben — eine Anforderung, die für spontane Netze unter Umständen nur schwer zu erfüllen ist.

TORA definiert zwei besondere Höhen: zum einen die *NULL*-Höhe ($HEIGHT = (-,-,-,-,i)$), die sozusagen einen unendlich großen Wert symbolisiert und die *Zielhöhe* ($HEIGHT = (0,0,0,0,i)$), die minimale Höhe repräsentiert und den Zielknoten identifiziert. Verliert ein Knoten nicht nur alle Downstreams, sondern auch alle Upstreams so bedeutet dies, dass er vom restlichen Netz getrennt wurde und seine Höhe auf NULL setzt.

Jeder Host verwaltet für jeden Zielknoten einen eigenen DAG. Die im folgenden vorgestellten Verfahren beziehen sich jeweils nur auf einen Zielknoten bzw. den dazugehörigen DAG.

Zum Erstellen der Höheninformationen und daraus resultierenden DAGs sieht TORA vier verschiedenen Modi vor: Das Erstellen, Pflegen, Löschen und Optimieren von Routen (siehe [33], Kapitel 3), die im folgenden näher vorgestellt werden.

Anders als die bisher vorgestellten Algorithmen, die entweder reaktive oder proaktive Verfahren sind, lassen sich in Tora beide Arten kombinieren. Normalerweise erfolgt das Erstellen von Routen reaktiv, also auf Anfrage, für ausgewählte Ziele ist es jedoch auch möglich diese proaktiv also im voraus zu erstellen.

Im reaktiven Modus wird das Finden der Route zu einem Ziel durch das Aussenden eines Query-Paketes (*QRY*) an alle Nachbarn initiiert. Dieses Paket wird solange weitergeleitet bis

ein Knoten erreicht wird, der eine Route zum Ziel kennt. Alle Knoten, die das Query-Paket weiterleiten, setzen für eine bestimmte Zeit eine *route requested flag* für den Zielknoten - weitere Requests für das selbe Ziel werden so erkannt und verworfen.

Hosts, die eine Route zum Ziel kennen, antworten durch das Versenden eines Update-Paketes (*UPD*) an alle Nachbarn. Dieses Paket enthält die Refrenzhöhe des antwortenden Hosts. Knoten mit gesetztem route request flag, die dieses UPD-Paket erhalten, passen ihren DAG entsprechend der erhaltenen Information an und versenden ihrerseits ein UPD-Paket mit iher neuen Referenzhöhe. Die ID des Zielknotens, die in allen TORA-Routingpaketen enthalten ist, identifiziert dabei den jeweils zu bearbeitenden DAG.

Der proaktive Mode von TORA lässt sich für einzelne Zieladressen einschalten und parallel zum reaktiven Modus betreiben. Hosts, für die Routen proaktiv berechnet werden sollen, initieren die Routenerzeugung selbstständig durch das Versenden von Optimierungspaketen (*OPT*) an ihre Nachbarn. Diese enthalten neben der obligatorischen Zieladresse, die Referenzhöhe und den Operationsmodus des Knotens — also die Information, dass es sich um ein proaktiv zu berechnendes Ziel handelt — sowie eine Sequenznummer, die das Paket eindeutig identifiziert. Auch dieses Paket wird wie UPD-Pakete nach dem justieren der Höhe und dem Setzen des proakiven Modus an alle Nachbarn weitergeleitet (siehe [33], Kapitel 3.1).

Die Pflege der Routen wird nur für Knoten, deren Höhe nicht der NULL-Höhe entspricht, durchgeführt — Nachbarknoten mit der Höhe NULL werden auch in der Neuberechnung von Höhen nicht berücksichtigt.

Um auf das Wegfallen aller Downstreams zu reagieren, definiert TORA fünf verschiedene Aktionen (siehe [33], Kapitel 3.2), um eine neue Höhe zu erstellen, die dann mittels UPD an alle Nachbarn verteilt wird. Welche der möglichen Aktionen wann ausgeführt wird, soll hier kurz erklärt werden. Auch hier wird immer nur ein einzelner DAG für einen einzelnen Zielknoten betrachtet.

- *Generate*: Erkennt der Host selbst durch die Detektion einer unterbrochenen Verbindung, dass es für einen Knoten keine Downstreams mehr gibt, so definiert er die neue Referenzhöhe des Knotens selbst: $(\tau, oid, r, \delta, i) = (t, i, 0, 0, i)$, wobei t der Zeitpunkt der Unterbrechung ist.

- *Propagate*: Gibt es für einen Knoten keine Downstreams mehr, da sich die Höhe durch ein UPD-Paket geändert hat und ist die Bedingung erfüllt, dass nicht alle Nachbarknoten die selbe Referenzhöhe haben, so wird die Referenzhöhe des höchsten Nachbarn ($(\tau, oid, r) = max(\tau[k], oid[k], r[k])$ für alle Nachbarn k) verwendet. δ ergibt sich durch den niedrigsten Nachbarn mit der eben ermittelten Referenzhöhe (m): $(\delta, i) = (delta[m] - 1, i)$. Grob gesprochen wird also eine Gesamthöhe gewählt, die unterhalb des höchsten Nachbarn aber über allen anderen Nachbarn liegt.

Im Regelfall reichen die beiden oben beschriebenen Verfahren zur Pflege der Routen aus, da sich ein beliebiger DAG immer in einen zielgerichteten Graphen transformieren lässt. Wird

jedoch das Netz getrennt, so ist dies nur noch für den Teil möglich, in dem sich der Zielknoten befindet (siehe [Rot02], Seite 225). Um einen solchen Fall zu erkennen, nutzt TORA das in der Höhe enthaltene reflection flag (r).

- *Reflect*: Fallen alle Downstreams eines Knotens auf Grund eines UPD-Paketes weg und haben jedoch alle Nachbarn die selbe Referenzhöhe, so hat möglicherweise eine Partitionierung des Netzes stattgefunden. Um dies zu erkennen, setzt der Host nun die Referenzhöhe des Knotens auf die der Nachbarknoten(k) und aktiviert zusätzlich das *reflection flag* — damit erhöht sich zunächst die Gesamthöhe des Knotens: $(\tau, oid, r, \delta, i) = (\tau[k], oid[k], 1, 0, i)$. Die Nachbarknoten würden nun die Höhe wie bei der Routenerzeugung anpassen und die neue Höhe mit gesetztem reflection flag versenden.

- *Detect*: Empfängt nun der Host die soeben versendete Höhe erneut — diesmal mit gesetztem Reflection Flag, so wurde eine Partitionierung erkannt und die entsprechenden Routen müssen gelöscht werden (siehe nächster Abschnitt). Die Höhe des Knotes wird auf NULL gesetzt. Das Empfangen der Höhe mit gesetzem reflection flag muss nicht zwangsläufig auf eine Partitonierung hindeuten, deshalb wird der Detect-Fall nur ausgeführt, wenn es sich um eine vom Host selbst definierte Referenzhöhe handelt ($oid[k] = i$).

- *Generate*: Ist die Bedingung jedoch ($oid[k] = i$) nicht erfüllt, so wird für den Knoten analog zum ersten Fall vom Host selbst eine neue Höhe erstellt. Im Falle einer wirklichen Partitionierung würde auch dies wieder zu einer Reflektion führen, die dann den Detect-Fall auslösen würde.

Das Löschen von Routen wird beim Erkennen einer Partitionierung des Netzes (Fall 4 im Route Maintainance) eingeleitet. Dazu setzt der Host seine Höhe sowie die Höhe aller Nachbarn auf NULL. Anschließend versendet er ein Clear-Paket (*CLR*) an alle Nachbarn, welches wieder die Zieladresse, auf die sich die Information bezieht und die reflektierte Referenzhöhe enthält ($(\tau, oid, 1)$). Empfängt ein Knoten dieses CLR-Paket und stimmt die empfangene Referenzhöhe mit der selbst gespeicherten Referenzhöhe überein, so setzt auch er seine Höhe und die seiner Nachbarn auf NULL und versendet das CLR-Paket erneut. Stimmt die Referenzhöhe nicht mit der selbst gespeicherten überein, so werden nur die Nachbarhöhen, für die die empfangene Referenzhöhe übereinstimmt, auf NULL gesetzt. Auf diese Weise werden die Höhen im abgetrennten Teil des Netzes auf NULL gesetzt und somit im Sinne des Routingprotokolls unerreichbar gemacht (siehe [33], Kapitel 3.3).

Anders als die bisher vorgestellten Protokolle definiert TORA kein eigenes Verfahren zur Erkennung von Nachbarn oder broken links, sondern setzt auf ein beliebiges darunterliegendes Protokoll auf, welches diese Funktionalitäten zur Verfügung stellt. In der Praxis kommt dabei nur das ebenfalls von den TORA-Entwicklern stammende *Internet MANET Encapsulation*

Protokol (IMEP) (siehe [26]) zum Einsatz. Dieses Protokoll wurde eigentlich als Basis für unterschiedlichste Ad-Hoc-Routingprotokolle geplant, wird aber letztendlich nur von TORA verwendet. Auf das IMEP-Protokoll soll hier nicht weiter eingegangen werden, da es ähnlich wie die bisher vorgestellten Protokolle Nachbarn und ausgefallende Links über das Versenden von HELLO-Paketen erkennt.

3.1.6 Zusammenfassung

Die folgende Tabelle stellt noch einmal die grundlegenden Eigenschaften der vorgestellten Protokolle gegenüber.

Protokoll	**Verfahren**	**Aktiv**	**Nachbar-erkennung**	**Besonderheiten**
DSDV	Distance Vector	proaktiv	HELLO Pakete/ Link Layer Feedback	Modifizierter Bellman-Ford-Algorithmus, periodischer Austausch der kompletten Routing-Tabellen
AODV	Distance Vector	reaktiv	HELLO Pakete/ Link Layer Feedback	Erweiterung von DSDV, Route Requests können über Expanding Ring Search eingeschränkt werden
DSR	Source Routing	reaktiv	HELLO Pakete/ Link Layer Feedback	Routen werden im Voraus ermittelt und in den Datenpaketen mitgeschickt
OLSR	Link State	proaktiv	HELLO Pakete	Broadcastbeschränkung über die Definition von Multipoint Relays
TORA	Link Reversal	beides	HELLO Pakete	Mehrere Routen zum selben Ziel werden unterstützt, Nodes müssen synchrone Uhren haben

3.2 Netzwerksimulation

Um die Performance und Eigenschaften der unterschiedlichen Protokolle besser untersuchen zu können, sollen diese in einem Netzwerksimulator getestet und auf die folgenden Kenngrößen hin untersucht werden:

- Das Verhältnis von verworfenen Paketen zu den Paketen, die ihr Ziel tatsächlich erreichten (*Packet Delivery Fraction*). Dieser Wert zeigt in wie weit das Protokoll in der Lage ist, die Route zu einem gewünschten Ziel zu finden.
- Die durchschnittliche Dauer zur Übermittlung eines Pakets vom Sender zum Empfänger. Berücksichtigt werden dabei nur die Pakete, die auch tatsächlich ihr Ziel erreichen. Aus diesem Wert kann geschlossen werden in wie weit das Routingprotokoll die Latenzzeit des Netzes beeinflusst.

- Die Anzahl der Weiterleitungen von Datenpaketen im Vergleich zur kürzesten Route. Hier zeigt sich in wie weit die Protokolle in der Lage sind, die kürzesten Wege zum Ziel zu finden.

- Die Menge der für das Routing versendeten Daten. Dieser Wert gibt Aufschluß über den vom Routingprotokoll erzeugten Overhead.

3.2.1 Anwendungszenarien

Um die Routingprotokolle hinsichtlich ihrer Tauglichkeit im Echteinsatz besser testen zu können, werden zunächst einige Szenarien entworfen, in denen Ad-Hoc-Protokolle zum Einsatz kommen könnten. Anhand dieser können dann die Parameter für die verschiedenen Simulationen ausgewählt werden.

Zunächst einmal ist festzuhalten, dass der Einsatz eines MANETs erst Sinn macht, wenn zum einen genügend Knoten beteiligt sind, um Pakete weiterzuleiten (das Minimum wären hier also drei) und zum anderen, dass diese weit genug von einander entfernt sind, um das Weiterleiten überhaupt nötig zu machen — ein MANET in einem Büro einzurichten würde also beispielsweise keinen Sinn machen. Die folgenden Anwendungen konzentrieren sich also auf Szenarien mit mehr als 10 Knoten und Gebieten von mehr als 1000 m^2.

Als erstes Anwendungsszenario soll ein Kongress dienen. Verschiedene Teilnehmer bewegen sich mit ihren Laptops oder PDAs zwischen den verschiedenen Vortragsräumen hin und her. Sowohl die Anzahl der Teilnehmer, als auch der Mobilitätsgrad und die Geschwindigkeiten sind bei diesem Szenario eher gering.

Eine zweite Anwendung, bei der ähnliche Geschwindigkeiten, jedoch mehr Knoten und höhere Mobilität anzutreffen ist, wäre der Einsatz auf dem Campus einer Universität.

Ein drittes Einsatzgebiet könnte die Anwendung in einem Krisengebiet, bei dem ein Großteil der Kommunikationsinfrastruktur zerstört wurde, sein — beispielsweise in einem Erdbebengebiet. Verschiedene Hilfsorganisationen könnten über spontane Netze ihre Einsätze koordinieren und Daten austauschen. In einem solchen Szenario können leicht 50 bis 100 Knoten in Gebieten von mehreren Quadratkilometern zusammenkommen . Zudem könnten hier Laptops und PDAs auch in Fahrzeugen oder gar Hubschraubern installiert sein — es sind also auch höhere Geschwindigkeiten in der Simulation zu berücksichtigen.

Als viertes und letztes Szenario wollen wir annehmen, dass in Zukunft auch Kraftfahrzeuge mit MANET-Technologie ausgestattet sind und so untereinander Informationen beispielsweise über Strassenverhältnisse, Verkehrsaufkommen und ähnliches austauschen können. Für die folgenden Simulationen soll ein kleineres Gebiet einer Stadt simuliert werden. Die Geschwindigkeit beschränkt sich also auf die zugelassene Höchstgeschwindigkeit von 50km/h, anders als bei den anderen Szenarien sind hier jedoch fast alle Fahrzeuge in ständiger Bewegung.

Die folgende Tabelle gibt noch einmal die verschiedenen Szenarien mit den für den Simulator relevanten Werten wieder:

Szenario	Kongress	Uni-Campus	Erdbebenhilfe	Stadtverkehr
Anzahl der Nodes	15	40	80	100
max. Geschwindigkeit	3 m/s (10,8 km/h)	3 m/s (10,8 km/h)	25 m/s (90 km/h)	14 m/s (50,4 km/h)
Mobilität	10%	30%	50%	80%
Größe des Gebiets	300x300 m	500x500 m	1000x1000 m	1000x1000m

Um neben den durch die Szenarien festgelegten Parametern, auch untersuchen zu können, wie die einzelnen Protokolle auf die Änderung einzelner Parameter reagieren, werden zusätzlich zwei weitere Simulationsreihen durchgeführt:

In der ersten Versuchsreihe wird ein MANET mit 50 Nodes in einem Gebiet von 500 mal 500 Metern simuliert. Die maximale Geschwindigkeit der einzelnen Knoten beträgt 15 m/s. Untersucht werden soll der Einfluss der Bewegung der einzelnen Nodes auf die Effizienz der Routingprotokolle. Die Versuchsreihe wird daher mit unterschiedlichem Mobilitätsgrad, also mit Pausen von 0 bis 100 Sekunden durchgeführt.

Die zweite Versuchsreihe simuliert Netze unterschiedlicher Größe. Auch hier ist die maximal Geschwindigkeit der Nodes auf 15 m/s festgelegt. Simuliert werden Netze mit 20 Nodes (350x350 m), 50 Nodes (450x450 m), 80 Nodes (800x800 m) und 100 Nodes (1000x100 m). Protokolle, die auch in Netzen dieser Größe noch funktionieren, werden zusätzlich in einem Netz mit 200 Knoten simuliert.

3.2.2 Netzwerksimulator ns2

Für die Simulation der verschiedenen Anwendungsszenarien kommt der Netzwerksimulator *ns2* (siehe [12]) zum Einsatz. ns2 erlaubt die Beschreibung und Simulation von beliebigen Netzwerken. Dabei ist ns2 derzeit der einzige freie Netzwerksimulator, der auch drahtlose Netzwerke unterstützt.

ns2 unterstützt bereits die Routingprotokolle DSR, DSDV, TORA und AODV. Für die folgenden Simulationen wurde zusätzlich noch die AODV-Implementation der Universität Uppsala (siehe [3]) (im folgenden auch AODVUU abgekürzt) sowie die OLSR-Implementation vom US Naval Research Laboratory (siehe [13]) (NRLOLSR) für ns2 kompiliert und installiert. AODVUU wurde mit in die Simulationen aufgenommen, da diese Version, die vor allem für große Netze interessante Expanding Ring Search-Methode implementiert.

Während der Simulation werden alle Ereignisse und Pakete innerhalb des simulierten Netzwerkes in sogenannten *trace files* protokolliert. ns2 ermöglicht dabei verschiedene *trace level* ein- und auszuschalten und so die Menge der gespeicherten Ereignisse zu beeinflussen. Als ein Ereignis wird dabei jedes Paket, das von einem Knoten gesendet, empfangen oder weitergeleitet wird betrachtet.

Die Beschreibung der Netzwerktopologie und dem darauf laufenden Verkehr erfolgt in TCL-Skripten (siehe [27]), die die dafür vom Simulator bereitgestellte API nutzen. Das TCL-Script für die folgenden Simulationen, sowie alle weiteren für die Simulation verwendeten Skripte finden sich auf der beiliegenden CD (Anhang Seite 75).

3.2.3 Simulationsparameter

Bei allen Simulationen wurden für alle Nodes das von ns2 zur Verfügung gestellte Wireless LAN-Interface verwendet. Dieses Interface simuliert die Eigenschaften einer Lucent WaveLAN DSSS-Karte mit einer omnidirektionalen Antenne. Die Reichweite der Nodes beträgt dabei 250 Meter. Die Ausbreitung der Funkwellen wird in ns2 zur Zeit nur zweidimensional simuliert. Auch Dämpfungen oder Reflektionen von Funkwellen, wie sie in der realen Welt vorkommen, werden in ns2 zur Zeit nicht berücksichtigt.

Alle simulierten Nodes nutzen eine priorisierte Warteschlange für das Netzwerkinterface, die maximal 50 Pakete zwischenspeichert. Beim Versenden werden Routingpakete bevorzugt behandelt.

Alle simulierten Routingprotokolle wurden mit ihren von den Entwicklern vorgesehenden Standardoptionen simuliert. So nutzen die AODV-Protokolle sowie DSR und DSDV das Link-Layer-Feedback zur Nachbarerkennung anstatt HELLO-Pakete einzusetzen.

Traffic- und Bewegungsmodelle

Zum Erzeugen von Netzwerkverkehr zwischen den einzelnen Nodes werden sogenannte *traffic models* generiert. Ein Script dafür wird mit ns2 mitgeliefert und liegt unter `ns2/indep-utils/cmu-scen-gen/cbrgen.tcl`, wobei `ns2` das Verzeichnis ist, in dem die Quellen ausgepackt wurden. Allerdings hat das Script mehrere Bugs, wodurch teilweise Netzwerkverkehr außerhalb der Simulationszeit oder Verbindungen zu nicht existenten Nodes erzeugt wird. Statt dem mitgelieferten Script wurde für die Simulationen ein eigenes Perlscript verwendet, welches zuverlässigere Modelle generiert.

Generiert wird eine vorgegebene Anzahl von Verbindungen zwischen zufällig ausgewählten Nodes. Das Script kann sowohl TCP-Datenquellen unter Verwendung des ns2-*FTP Agent*s als auch Datenquellen mit konstanter Bitrate — sogenannte *Constant Bit Rate*-Agents (CBR) — generieren.

Auch die Bewegung der Nodes wird generiert. Dazu wird das mitgelieferte Tool `ns2/indep-utils/cmu-scen-gen/setdest/setdest` verwendet. Generiert wird dabei ein quadratisches, flaches Areal in dem die einzelnen Nodes zufällig verteilt werden. Nach einer konfigurierbaren Pause bewegen sich die Nodes zu einem neuen zufällig gewählten Punkt innerhalb dieses Areals, an welchem sie erneut die definierte Pausenzeit abwarten, bevor sie den nächsten Punkt aufsuchen. Über die Definition unterschiedlicher Pausen kann der Mobilitätsgrad des simulierten Netzes beeinflusst werden. Bei einer Gesamtsimulationszeit von 100 Sekunden lässt sich so über die unterschiedliche Konfiguration der Pausen die

Mobilität der Teilnehmer von 0% (100 s) bis 100% (0 s) beeinflussen. Neben der Angabe der Pausen kann auch die maximale Geschwindigkeit, mit der sich die Nodes fortbewegen definiert werden — die Nodes wählen dabei eine zufällige Geschwindigkeit zwischen 0 und der maximal erlaubten Geschwindigkeit.

Bei der Wahl der Größe des Simulationsgebietes ist darauf zu achten, dass möglichst keine Nodes aus der Reichweite des gesamten Netzes geraten, da dies die Bewertung des Routingprotokolls erschweren würde — es wäre beispielsweise nicht mehr ersichtlich, ob ein Paket das Ziel nicht erreicht, weil das Routingprotokoll die neue Route nicht rechtzeitig ermitteln konnte oder weil der Zielknoten außerhalb der Reichweite des Gesamtnetzes lag.

3.2.4 Simulation

Um das Durchführen einer großen Anzahl an Testserien zu automatisieren, werden Shellscripte verwendet, welche die benötigten Bewegungs- und Trafficszenarien erzeugen, die eigentliche Simulation mit den verschiedenen Routingprotokollen durchführen und aus den Tracefiles die benötigten Daten zur grafischen Darstellung erzeugen. Zum extrahieren der auszuwertenen Daten aus den Tracefiles wird ein Perl Script verwendet, welches nach jedem Simulationslauf ausgeführt wird. Die erzeugten Tracefiles werden aus Platzgründen nach der Analyse wieder gelöscht.

Um zusätzlich die durch die zufällig generierten Bewegungs- und Trafficmodelle eventuell entstehenden Unregelmäßigkeiten auszugleichen, werden alle Simulationen mit unterschiedlich initialisiertem Zufallsgenerator mehrfach durchgeführt und für die Auswertung die Durchschnittswerte aller Durchläufe verwendet.

3.2.5 Ergebnisse

Zunächst ist darauf hinzuweisen, dass die Ergebnisse der Simulation mit Vorsicht in Hinsicht ihrer Relevanz für den Echteinsatz zu betrachten sind. Die Implementierungen der mobilen Routingprotokolle sind noch relativ neu und erst vor kurzem in den Netzwerksimulator eingeflossen. Zudem unterstützen einige der Implementationen noch nicht den vollen Umfang des Protokolls - einige implementierungsspezifische Fehler lassen sich also nicht ausschließen. Vor allem die Daten von TORA und NRLOLSR sind nur bedingt aussagekräftig in Bezug auf die Eigenschaften der Protokolle, da die Implementationen innerhalb des Simulators nicht sehr stabil liefen und hin und wieder den Simulator mit einem Speicherzugriffsfehler zum Absturz brachten. So konnten für NRLOLSR keine Werte für die TCP-Simulationen mit den Erdbeben- und Stadtverkehrszenarien ermittelt werden. Für das Stadtverkehrsszenario waren auch UDP-Simulationen mit NRLOLSR nicht möglich — die entsprechenden Werte fehlen in den folgenden Diagrammen.

Trotz der genannten Probleme lassen sich aus den Simulationen qualitative und quantitative Aussagen über das Verhalten der Protokolle in ihrer vorliegenden Implementation treffen. Um den Einfluss der genannten Störfaktoren zumindest abzuschwächen, wurden alle Simulationen

mehrmals durchgeführt. Im folgenden werden die Mittelwerte aller Simulationen ausgewertet.

Packet Delivery Fraction

Zunächst wird untersucht, inwieweit die verschiedenen Protokolle in der Lage sind, überhaupt eine Route zum gewünschten Ziel zu finden. Dafür wurden Constant Bitrate-Datenquellen verwedet, die per UDP einen konstanten Datenstrom an einen Zielknoten senden. Findet ein Paket den Weg zum Ziel nicht, so wird es verworfen. Das Verhältnis zwischen gesendeten und tatsächlich am Ziel angekommenen Datenpaketen wird auch *Packet Delivery Fraction* genannt.

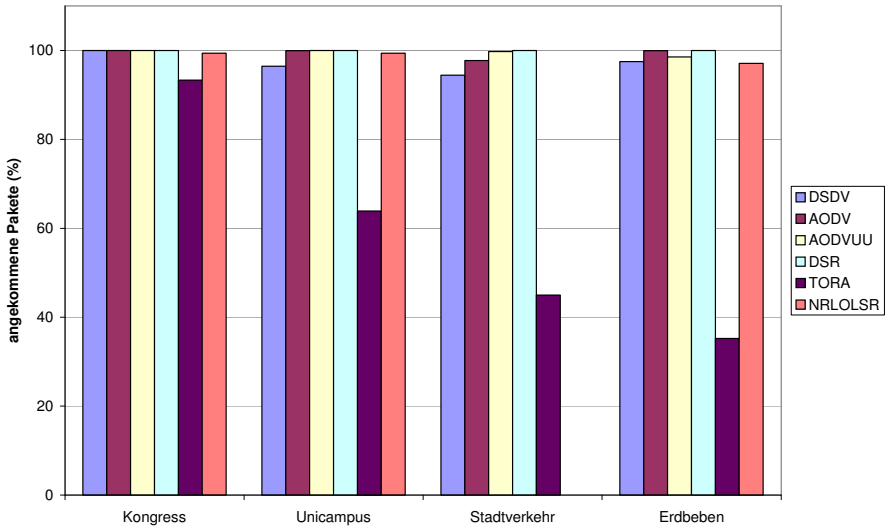

Abbildung 3.7: Packet Delivery Fraction für die verschiedenen Szenarien

Die Simulation der definierten Szenarien (Abbildung 3.7) zeigt, dass fast alle Protokolle in der Lage sind, für nahezu 100% der gesendeten Pakete eine Route zum Ziel zu finden. Lediglich das TORA-Protokoll hat bei größeren Geschwidigkeiten und Netzen Probleme.

In der Simulation verschiedener Mobilitätsgrade (Abbildung 3.8) zeigt sich, dass, wie zu erwarten, die reaktiven Protokolle AODV und DSR besser mit sehr hoher Mobilität umgehen können, aber auch bei OLSR wird die Packet Delivery Fraction nur wenig von kürzeren Pausenzeiten beeinflusst. Offenbar ist es durch den von Multipoint Relays beschränkten Austausch von Kontrollpaketen schneller möglich, auf Topologieänderungen zu reagieren als beim klassischen Link State-Verfahren DSDV.

Werden die Netze größer, ist der Unterschied zwischen proaktiven und reaktiven Protokollen deutlich zu sehen (Abbildung 3.9). Je mehr Nodes am Netz teilnehmen, desto mehr Routen müssen bei proaktiven Protokollen gefunden und ausgetauscht werden. Während AODV und

DSR nahezu unabhängig von der Netzgröße arbeiten, sinkt die Anzahl der empfangenen Pakete für OLSR deutlich. Dass TORA trotz reaktivem Verfahren so schlecht abschneidet, liegt vor allem am erzeugten Routing Overhead, der im nächsten Abschnitt untersucht wird.

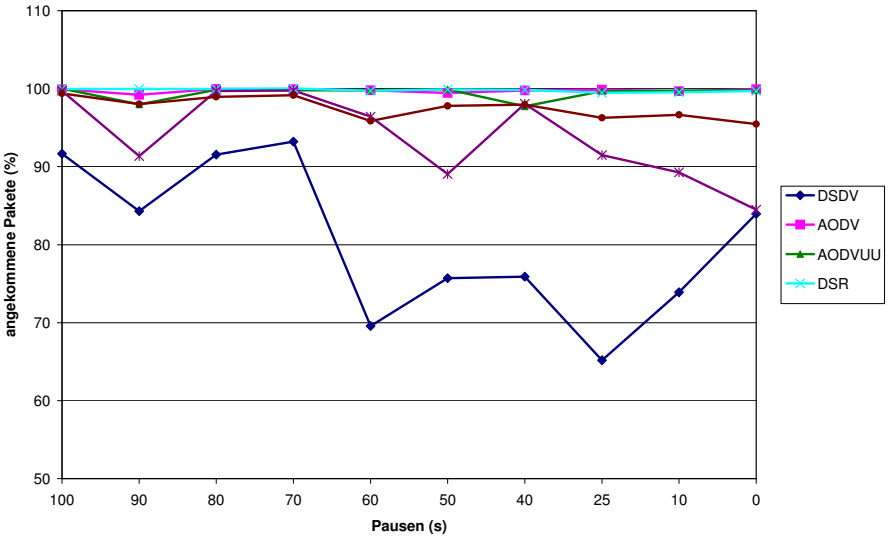

Abbildung 3.8: Verhältnis von empfangenen zu gesendeten Paketen bei unterschiedlich starker Nodebewegung

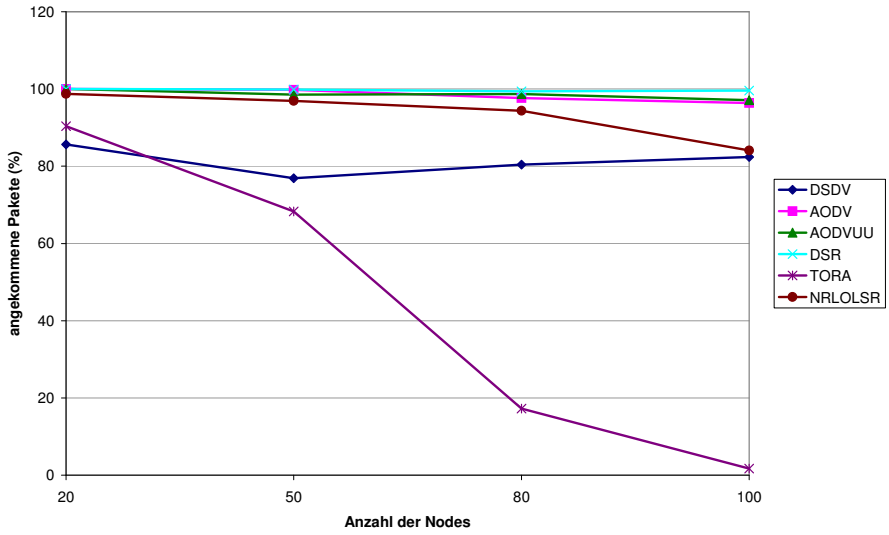

Abbildung 3.9: Verhältnis von empfangenen zu gesendeten Paketen bei unterschiedlich großen Netzen

Routing Overhead

Um den von den Protokollen erzeugten Overhead zu untersuchen, wurden die Größen aller gesendeten oder weitergeleiteten Routingpakete addiert. Da alle Protokolle in den selben Netzen mit den selben Verbindungsmodellen simuliert wurden, lassen sich die von den Routingalgorithmen produzierten Datenmengen direkt miteinander vergleichen. Da hier erhebliche Unterschiede auftraten, wurden für die folgenden Graphiken logarithmische Skalen gewählt.

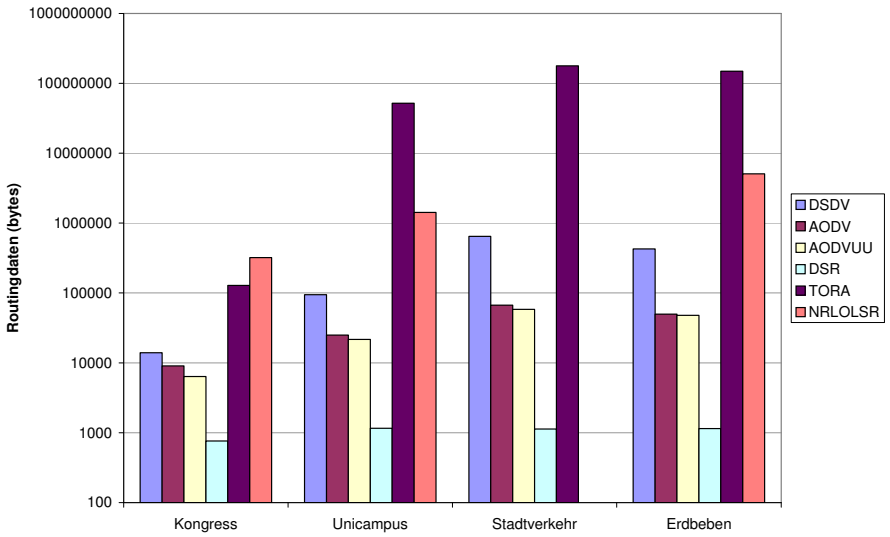

Abbildung 3.10: Menge der insgesamt für das Routing versendeten Daten für die verschiedenen Szenarien

Beim Testen der verschiedenen Szenarien (Abbildung 3.10) fällt sofort auf, dass das DSR-Protokoll nahezu unabhängig von den Parametern einen sehr geringen Overhead produziert. Hier kommt vor allem das Cachen von Routinginformationen zum Tragen. Zwar vergrößert sich die Größe der gesendeten Datenpakete, da die Route zum Ziel im Header mitgeschickt wird, beim Vergleich der durchschnittlichen Paketgrößen der verschiedenen Simulationen war dieser Unterschied jedoch nicht signifikant.

Zu beachten sind hier auch die Möglichkeiten der verschiedenen Protokolle zur Erkennung von Nachbarn und unterbrochenen Verbindungen. AODV und DSR können dafür, anstatt der im RFC beschriebenen HELLO-Pakete, auch auf die Informationen aus der Sicherungsschicht zurückgreifen und somit völlig auf HELLO-Pakete verzichten. Bei OLSR ist dies nicht möglich, da in den HELLO-Paketen auch Nachbarinformationen ausgetauscht werden müssen. Auch TORA bzw. das darunterliegende IMEP-Protokoll sieht keine Möglichkeit vor, auf HELLO-Pakete zu verzichten.

Abbildung 3.11 zeigt wie die verschidenen Protokolle bei unterschiedlicher Netzgröße skalieren. Hier wird noch einmal deutlich, dass TORA vor allem für große Netze nicht geeignet ist. Die Anzahl der ausgetauschten Routingpakete ist um ein Vielfaches höher als bei allen ande-

ren Protokollen. Aber auch OLSR muss bei großen Netzen, bedingt durch die Notwendigkeit von HELLO-Paketen, besonders viele Daten austauschen.

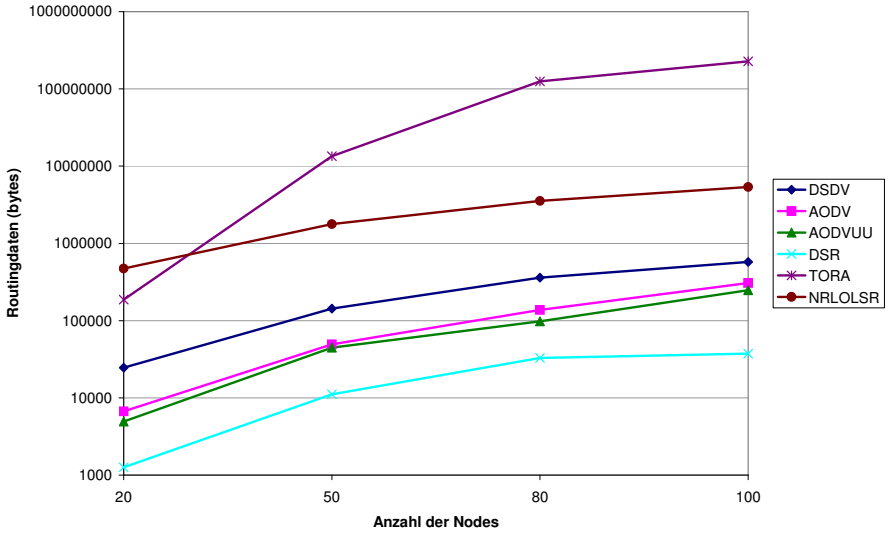

Abbildung 3.11: Menge der für das Routing versendeten Daten bei unterschiedlich großen Netzen

Durchschnittliche Paketübertragungszeit

Als nächstes soll die durchschnittliche Übertragungszeit von Paketen untersucht werden. Dieser Wert liefert einen Hinweis darauf, wie lange es dauert bis eine Route zu einem neuen Ziel ermittelt wurde.

Während alle Protokolle (außer TORA) für alle Szenarien Pakete innerhalb 0,1 Sekunden zum Ziel brachten, ist die Übertragungszeit bei DSR deutlich höher — ungefähr um das Dreifache. Dies ist möglicherweise ein Ergebnis des Source Routings: vor dem Versenden des Pakets ermittelt der Absender die komplette Route zum Ziel und schickt diese im Header des Datenpakets mit. Wird diese im Voraus bestimmte Route unterbrochen, so wird dies dem Absender mitgeteilt, welcher dann erneut eine Route bestimmen muss. Bei allen anderen Protokollen übernimmt der jeweils weiterleitende Knoten das Finden des nächsten Hops — dies ist in der Regel schneller. Der bessonders auffällige Wert des TORA-Protokolls im Stadtverkehrszenario hängt vor allem wieder mit der großen Anzahl der Nodes zusammen. Abbildung 3.13 zeigt, dass die Mobilität der Nodes hier kaum Einfluss auf das Verhalten der Protokolle hat.

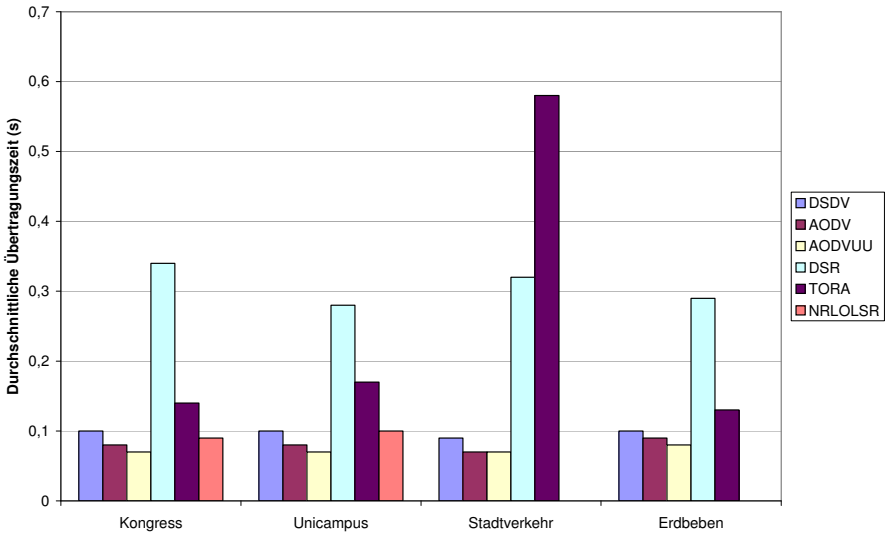

Abbildung 3.12: Durchschnittliche Paketübertragungszeit für die verschiedenen Szenarien

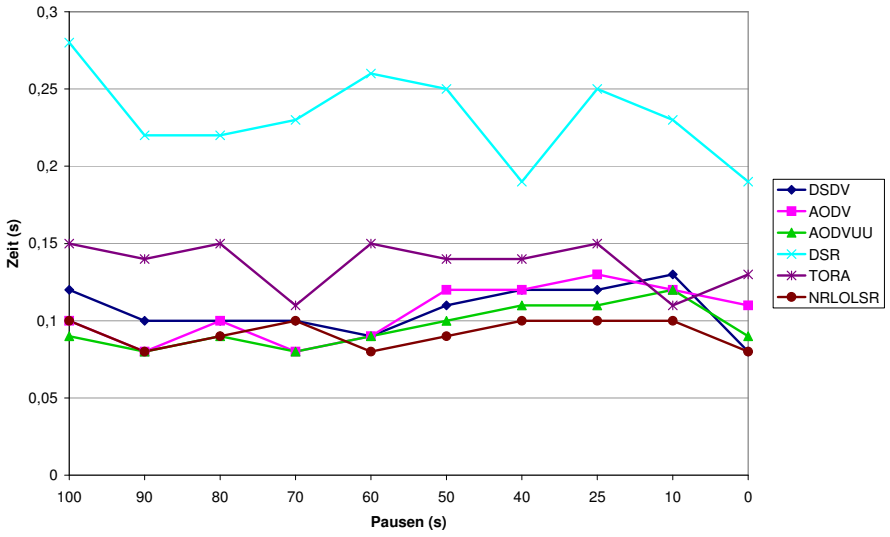

Abbildung 3.13: Durchschnittliche Paketübertragungszeit bei unterschiedlich starker Nodebewegung

Finden von optimalen Routen

ns2 speichert zu jedem Paket, das sein Ziel erreicht, wie oft dieses Paket weitergeleitet wurde und wieviele Hops die kürzeste Route gehabt hätte. Das Verhältnis beider Werte gibt Aufschluß darüber, wie optimal die von den Protokollen gefundenen Routen sind.

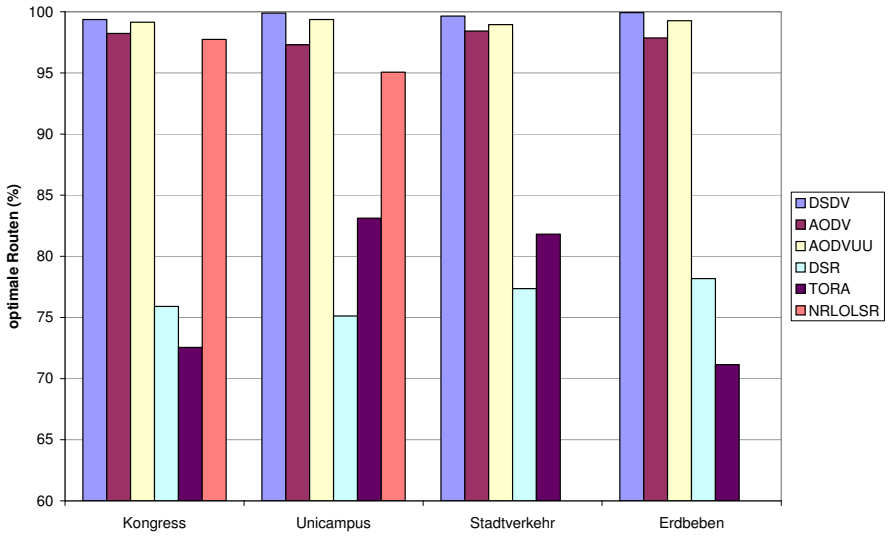

Abbildung 3.14: Optimale Routen für die verschiedenen Szenarien

Bei der Betrachtung der Abbildung 3.14 fällt auf, dass sowohl TORA als auch DSR hier nur mittelmäßig abschneiden. Beide Protokolle liefern um 20 bis 30 Prozent längere Routen, als nötig wären, um das entsprechende Ziel zu erreichen. Bei TORA war dies zu erwarten, da das Protokoll nicht vorsieht, optimale Routen zu finden, sondern lediglich einen beliebigen Weg zum Ziel sucht (vgl. 3.1.5). Der Grund für das schlechte Abschneiden des DSR Protokolls ist wohl auch hier in der Vorausberechnung der Routen zu suchen. In einem mobilen Netz kann der so ermittelte Weg bereits veraltet sein, wenn das Paket versandt wird. So werden teilweise Routen verwendet, die zwar noch bestehen, aber inzwischen durch kürzere Routen hätten ersetzt werden können.

Alle anderen getesteten Protokolle hingegen finden nahezu zu 100% optimale Wege zum Zielknoten.

Skalieren bei großen Netzen

Um zu sehen wie sich die Protokolle bei noch größeren Netzen verhalten, beispielsweise wenn man das Stadtverkehrsszenario weiter ausbauen würde, wurde eine weitere Simulation mit den beiden AODV-Implementierungen und dem DSR-Protokoll durchgeführt (Abbildung 3.15).

Auffällig ist hier das Verhalten der zwei AODV-Versionen: während sich beide in den bisherigen Versionen noch relativ gleich verhielten, zeigt jetzt die Implementierung der Universität von Uppsala eine wesentlich geringeren Routingoverhead. Dies liegt vor allem daran, dass AODVUU die Expanding Ring Search-Methode (vgl . Kapitel 3.1.2) verwendet und so wesentlich effektiver Route-Requests versendet. Wie man sehen kann, ist das DSR-Protokoll bei 200 teilnehmenden Knoten nicht mehr in der Lage, Routen für die gewünschten Ziele zu

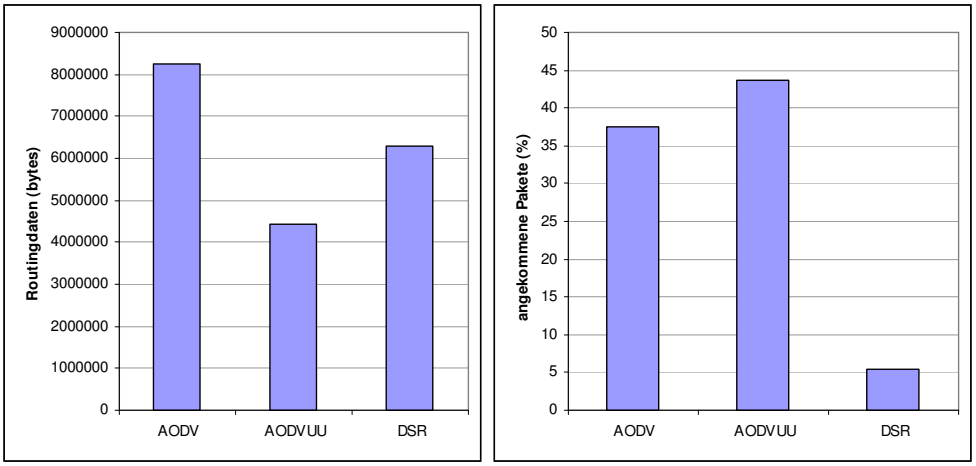

Abbildung 3.15: Routingoverhead und Packet Delivery Fraction bei 200 Nodes

finden. Das Vorausermitteln von Routen wird bei zunehmender Hopanzahl bei sich ständig ändernen Netzen zunehmend ineffektiver.

Zusammenfassung

In allen durchgeführten Simulationen waren die Protokolle AODV und DSR immer in der Lage, ein zuverlässiges Routing für das simulierte MANET zur Verfügung zu stellen. OLSR ist für kleinere Netze, wie im Kongress- oder Campus-Szenario beschrieben, ebenfalls geeignet, obwohl gerade in diesen Bereichen DSR klar vorne liegt. Die Ergebnisse des TORA-Protokolls hingegen waren vor allem für große Netze ungeeignet.

Vor allem in Hinblick auf die Ergebnisse in dem mit 200 Nodes sehr großen Netz ist AODV von den getesteten Algorithmen das für MANETs am besten geeignete Protokoll.

4 IP-Vergabe

Moderne Computernetze basieren auf der TCP/IP-Protokollsuite. Ein spontanes WLAN sollte ebenfalls auf diese Technik setzen, um bestehende Netzwerkapplikationen und Protokolle auch in diesen Netzen einsetzen zu können.

Problem bei einem spontanen Peer-to-Peer-Netz ist dabei die Vergabe von IP-Adressen. Da es in einem spontanen Netz keine zentrale Stelle zur Verwaltung der Adressen gibt, welche die IPs verteilt und darauf achtet, dass keine Adressen doppelt vergeben werden, muss hier eine andere Lösung gefunden werden. Anders als bei den mobilen Routingprotokollen hat man sich hier erst seit kurzem des Problems angenommen. Daher sind alle hier vorgestellten Lösungen noch nicht als RFCs festgeschrieben, sondern lediglich als Internetdrafts vorgeschlagen worden.

Zunächst sollen die möglichen Szenarien aufgezeigt werden, die eine Lösung zur automatischen Konfiguration der IP-Adressen behandeln können sollte.

Im einfachsten Fall kommt ein neuer Knoten zu einem bestehenden MANET hinzu, bekommt eine IP-Adresse zugewiesen und verlässt das MANET später wieder, wobei die zugewiesene IP-Adresse wieder frei wird. Tritt der selbe Knoten jedoch später wieder dem MANET bei, sollte er — wenn möglich — die selbe IP-Adresse erhalten. Das eben beschriebene Verhalten gilt natürlich nicht nur für einen einzelnen Knoten. Möglich ist es auch, dass eine Gruppe von Nodes vom Rest des MANETs getrennt wird und somit zwei unabhängige Netze entstehen, man spricht hier von *Partitionierung* oder *split*. Auch das wieder Zusammenführen der einzelnen Partitionen (*merge*) sollte möglich sein. Schließlich sollte auch das Verschmelzen zweier zuvor unabhängig konfigurierter MANETs unterstützt werden.

Der Algorithmus zur Vergabe der IP-Adressen muss also folgende Ereignisse behandeln können:

- Ein einzelner Knoten tritt dem Netz bei oder verlässt es
- Ein MANET wird partitioniert und später wieder zusammengefügt (*split and merge*)
- Zwei zuvor unabhängige MANETs verschmelzen miteinander

4.1 IPv4

Die IP-Version 4 ist derzeit noch die weitverbreiteste Form der Adressierung in heutigen TCP/IP-Netzen. Der relativ kleine Bereich verfügbarer Adressen macht es jedoch schwierig,

automatisch Adressen zuzuweisen, ohne dabei Adressen doppelt zu vergeben. Verschiedene Lösungsansätze zur automatischen IP-Konfiguration in IPv4-MANETs sollen hier vorgestellt werden.

4.1.1 Zeroconf

Die *Zeroconf Working Group* (siehe [17]) der IETF arbeitet an der Standardisierung von Lösungen zur Vergabe von IP-Adressen ohne eine zentrale Verwaltung, wie beispielsweise DHCP, aber auch an Möglichkeiten zur Auflösung von Namen in IP-Adressen, dem Finden von Services in Netzen und der Zuweisung von Multicastadressen ohne zentrale Verwaltungsserver. Wenn auch bei der Entwicklung der Spezifikationen MANETs nicht im Vordergrund standen, sind all diese Lösungen besonders für spontane Netze interessant.

In dieser Arbeit soll zunächst nur die von der IETF vorgeschlagene Lösung zur IP-Adressvergabe vorgestellt werden. Implementationen dieses Protokolls finden sich bereits in den neueren Betriebssystemen von Apple (siehe [14]) und Microsoft (siehe [4]).

Bei der vorgeschlagenen Lösung (siehe [25]) wird der von der IANA für *Link-Local*-Adressen reservierte Adressbereich 169.254.0.0/16 für die Zuweisung von IP-Adressen verwendet. Die Verwendung dieses Class B-Netzes ermöglicht die Verwendung von maximal 65024 Adressen. Für die Vergabe der IPv4-Adressen in Ad-Hoc-Netzen wird eine pseudozufällige Auswahl mit anschließender Adresskollisionsdetektierung (siehe [24]) verwendet. Um unnötigen Netzwerkverkehr zu vermeiden, werden Adresskollisionen nur in den folgenden Fällen überprüft:

- Reboot und Rechnerstart
- Aktivieren eines vorher inaktiven Interfaces (zum Beispiel beim Aufwachen aus den Sleepmodus)
- Erkennen eines *Link-state change*: also beispielsweise dem Einstecken eines Netzwerkkabels u.ä.
- Beim Verbinden mit einem AccessPoint

Die Adressen werden zufällig generiert, so dass eine statistische Gleichverteilung über den gesamten verfügbaren Adressbereich gewährleistet ist. Dabei wird der Zufallsgenerator mit der Hardwareadresse des Knotens initialisiert. Dies ermöglicht, dass ein Host nach einem reboot — wenn möglich — die selbe IP-Adresse erhält.

Um eventuelle Doppelvergaben von IP-Adressen zu vermeiden, wird das *Address Resolution Protocol* (ARP) verwendet. Soll eine IP-Adresse vergeben werden, so werden sogenannte *ARP probes* versandt. Dabei handelt es sich um ARP-Requests, deren Absender-IP-Adresse auf 0.0.0.0, die Ziel-Hardware-Adresse auf 00:00:00:00:00:00:00:00 und die Ziel-IP-Adresse auf die zu prüfende IP gesetzt wird. Die Absender-Hardware-Adresse ist dabei die tatsächliche Hardware-Adresse des Interfaces. Hat ein Host bereits die entsprechende IP-Adresse, so würde

er auf den ARP-Request mit einem Paket an die Absender-Harware-Adresse antworten. IP-Adressen können so auf ihre Verwendung geprüft werden.

Das Zeroconf Protokoll sieht vor, dass ein Host, der eine bestimmte IP verwenden will, diese zunächst über zwei ARP Probes testen muss. Erhält er keine Antwort auf diese ARP-Requests, so kann er die Adresse verwenden und muss diese Verwendung über einen ARP-Broadcast allen Teilnehmer des Netzes mitteilen. Ist die Adresse bereits vergeben, so muss der Host eine andere pseudozufällige IP-Adresse generieren und diese erneut testen.

Je mehr Adressen vergeben sind, um so größer wird die Wahrscheinlichkeit einer Adresskollision. Die Zeroconf-Workinggroup gibt jedoch für ein Netz mit 1300 Nodes eine Warscheinlichkeit von 99,96% für das Finden einer freien IP-Adresse mit zwei Versuchen an.

Neben der Kollisonsvermeidung während der Vergabe von IP-Adressen definiert das Zeroconf-Protokoll einen Mechanismus zur „Verteidigung" einer IP-Adresse.

Empfängt ein Knoten ein ARP-Paket — egal ob Request oder Reply — welches als Absender-IP-Adresse die Adresse des Knotens selbst, jedoch eine andere Hardware-Adresse enthält, so wird dies als Konflikt betrachtet. Der Knoten kann darauf mit zwei möglichen Antworten reagieren: Entweder er gibt seine eigene IP-Adresse auf und versucht eine neue, unbenutzte IP zu finden, oder er „verteidigt" seine Adresse. Dazu versendet er ein einziges ARP-Paket mit seiner eigenen IP- und Hardware-Adresse als Broadcast an das gesamte Netz. Er kann dann die IP-Adresse ohne weitere Aktionen seinerseits weiter verwenden. Um jedoch zu vermeiden, dass zwei Knoten die selbe IP-Adresse verteidigen, ist eine Verteidigung nur erlaubt, wenn keine Konflikte für diese IP innerhalb einer bestimmten Zeit aufgetreten sind. Ist dies der Fall, so muss die IP-Adresse unverzüglich freigegeben werden.

Durch die Nutzung des ARP-Protokolls ist das Zeroconf-Protokoll jedoch nicht direkt für die automatische IP-Vergabe geeignet. ARP arbeitet auf der MAC Ebene und funktioniert, da MAC-Pakete nicht weitergeleitet werden, nicht über mehrere Hops hinweg. Eine Adresskollison würde so nur detektiert werden, wenn die störende IP-Adresse einem direkten Nachbarn des Hosts gehören würde.

Der Zeroconf-Draft (siehe [25], Kapitel 1.3) erlaubt auch andere Mechanismen zur Erkennung von Adresskonflikten, spezifiziert diese jedoch nicht weiter.

4.1.2 Duplicate Address Detection

Ein zum Zeroconf-Protokoll sehr ähnliches Verfahren, welches jedoch ohne ARP-Requests auskommt, wurde in [36] vorgeschlagen und in [28] erweitert. Verwendet wird auch hier der für *Link-Local*-Adressen reservierte Bereich 169.254.0.0/16, wobei der 65024 Adressen große Bereich in zwei Teilbereiche aufgeteilt wird. Der erste Bereich von 0 bis 2048 wird für temporäre Adressen verwendet und nur der zweite Bereich ist Konfiguration der mobilen Nodes verfügbar.

Statt dem ARP-Protokoll verwendet die vorgeschlagene Lösung modifizierte ICMP-Pakete. Tritt ein Knoten dem MANET bei, so wählt er eine zufällige Adresse aus dem ersten Bereich, um sein Netzwerkinterface zu konfigurieren. Anschließend versendet er ein *Address Request*-Paket (AREQ) mit einer zufällig gewählten IP aus dem oberen Adressraum an seine Nachbarn. Empfangene AREQ-Pakete werden von allen Nodes protokolliert und mehrfach empfangene Pakete verworfen. Ist das Paket jedoch neu, überprüft der Empfänger, ob die darin enthaltene IP-Adresse mit seiner eigenen übereinstimmt. Falls ja, so antwortet er dem Sender mit einem *Address Reply*, andernfalls leitet er das empfangene Paket an seine Nachbarn weiter (siehe [36], Kapitel 6.2).

Empfängt der Absender des Requests keine Antwort innerhalb einer bestimmten Zeit, so wiederholt er analog zum Zeroconf-Protokoll die Anfrage. Zuvor wählt er jedoch eine andere temporäre Adresse aus dem unteren Bereich. Erhält er auf drei Anfragen keine Antwort, so kann der Knoten die gewählte IP-Adresse verwenden und die temporäre Addresse freigeben. Ist die Adresse jedoch bereits in Verwendung, so wählt er eine neue temporäre Adresse und sendet erneut ein AREQ-Paket für eine zufällig gewählte Adresse aus dem oberen Bereich (siehe [36], Kapitel 6.3).

Empfängt ein Knoten nach dem Konfigurieren ein AREP-Paket mit seiner eigenen neuen IP-Adresse, so muss er diese unverzüglich freigeben und mit der IP-Konfiguration erneut wie beschrieben beginnen.

Um doppelt vergebene IP-Adressen auch nach der initialen Konfiguration erkennen zu können (etwa beim Verschmelzen zwei MANETs oder Partitionen), nutzt das vorgeschlagene Verfahren Informationen des Routingprotokolls. Dafür muss zu jeder Ziel-Adresse ein vom Ziel erstellter Schlüssel gespeichert werden, der dann mit jedem Routingpaket verbreitet wird. Jeder Knoten untersucht nun permanent empfangene Routingpakete auf Pakete mit den gleichen IP-Adressen, aber unterschiedlichen Schlüsseln. Wird ein solcher Fall erkannt, wird ein *Address Error*-Paket (AERR) an die entsprechende IP-Adresse geschickt, wobei der zusätzliche Schlüssel bei der Routingentscheidung verwendet wird. Bei einigen mobilen Routingprotokollen, wie zum Beispiel DSDV und AODV, ist es zusätzlich möglich die enthaltenen Sequenznummern zur Adresskonfliktdetektion zu verwenden. Enthält ein Knoten beispielsweise ein Routingpaket, das seine eigene IP-Adresse enthält, jedoch mit einer höheren Sequenznummer als der von ihm geführten, so liegt hier möglicherweise ein Adresskonflikt vor (siehe [28], Kapitel 5.1).

Bei der Erkennung eines Adresskonfliktes zum Beispiel durch das Empfangen eines AERR-Pakets wird die Adresskonfiguation erneut durchgeführt, jedoch wird anstatt eine neue zufällige Adresse zu wählen, zunächst die alte Adresse über das oben beschriebene Verfahren geprüft.

4.1.3 Prophet Adress Allocation

Die *Prophet Allocation*-Methode (siehe [40]) nutzt eine zustandsbehaftete Funktion $f(n)$, die, mit einem *seed* initialisiert, eine bestimmte Folge von Ganzzahlen innerhalb eines vorgegebe-

nen Bereichs generiert. Bei der Auswahl der Funktion muss darauf geachtet werden, dass der Abstand zwischen zwei gleichen Zahlen in dieser Folge möglichst groß ist. Zudem sollte die Wahrscheinlichkeit möglichst gering sein, dass die selbe Zahl von einer begrenzten Anzahl von Folgen, die mit unterschiedlichen Werten erzeugt wurden, generiert wird.

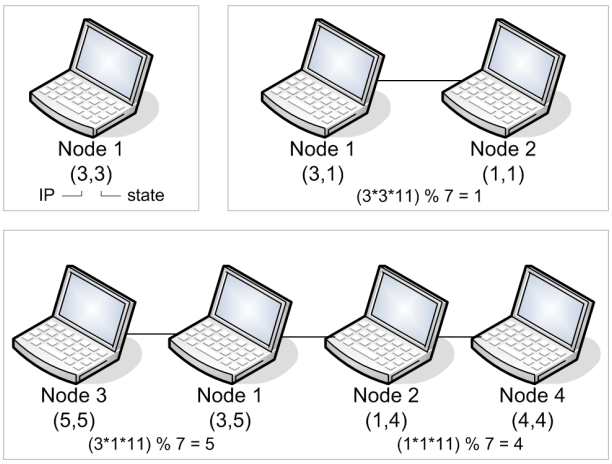

Abbildung 4.1: Prophet Allocation Beispiel

Wie diese Funktion zur Konfiguration von IP-Adressen genutzt werden kann, soll an einem einfachen Beispiel (Abbildung 4.1) erklärt werden: Der Einfachheit halber werden die IP-Adressen als einfache Ganzzahlen dargestellt, als Zustand (state) wird jeweils die neu generierte IP-Addresse verwendet. Als Funktion soll $f(n) = (ip * state * 11) mod 7$ dienen und die zu vergebenen IPs werden auf das Interval $R = [0, 6]$ eingeschränkt.

Node 1 ist der erste Knoten des MANETs und wählt den zufälligen Wert 3 aus R für seine IP und den seed der Funktion $f(n)$. Im zweiten Schritt kontaktiert Node 2 Node 1 und fragt nach einer IP-Adresse. Node 1 benutzt die Funktion $f(n)$, um diese zu erzeugen. Sowohl Node 1 als auch Node 2 setzen jetzt den state von $f(n)$ auf den neu gewonnenen Wert. Node 2 ist jetzt in der Lage mit der Funktion $f(n)$ selbst IP-Adressen zu verteilen. Im dritten Schritt vergeben die Nodes 1 und 2 unabhängig voneinander neue IP-Adressen für die Nodes 3 und 4. Dieses simple Beispiel würde auf Grund des sehr kleinen Adressbereichs bereits in einem vierten Schritt doppelte IP-Adressen erzeugen, dennoch zeigt es wie mit einer geeignet gewählten Funktion und einem genügend großen Adressbereich automatisch IP-Adressen verteilt werden können.

Node 1 ist in diesem Beispiel der Erzeuger des seeds und kann so voraussagen, welche IP-Adressen vergeben werden und so bereits Konflikte vorhersehen. Dieser erste Node wird auch *Prophet* genannt. Bei der Wahl des Seeds kann der Prophet häufige Wiederholungen von Zahlen in der entstehenden Sequenz vorhersehen und einen anderen geeigneteren seed wählen (siehe [40], Kapitel III.A).

Im Falle einer Partitionierung des MANETs sind die in den einzelnen Partitionen erzeugten IPs auch für das Gesamtnetz weiterhin eindeutig, das heißt auch bei einer späteren Wiedervereinigung treten keine Konflikte auf. So hätte im gezeigten Beipiel der dritte Schritt auch

ohne eine weiterhin bestehenden Verbindung zwischen Node 1 und Node 2 ablaufen können. Bei einer anschließenden Vereinigung aller vier Nodes wären trotzdem keine Adresskonflikte aufgetreten.

Um das Verschmelzen zweier zuvor unabhängiger MANETs zu ermöglichen, wird neben dem state zusätzlich eine *Netzwerk-ID* (NID) zwischen den Nodes ausgetauscht. Die NID wird vom Propheten zufällig gewählt und bleibt für das gesamte MANET gleich. Ist der Bereich, aus dem die NID gewählt wird, ausreichend groß, sollten zwei seperat konfigurierte MANETs nie die selbe NID besitzen. Idee ist es, die NID zwischen Nachbarnodes periodisch auszutauschen — beispielsweise könnte sie in den HELLO-Paketen, die einige Routingprotokolle austauschen (Siehe 3.1), mitgesendet werden.

Empfängt nun ein Knoten ein Paket mit einer NID, die größer als die eigene ist, so dekonfiguriert er seine bisherige IP-Adresse und fordert vom Sender des Pakets mit der höheren ID eine neue IP-Adresse an. Dabei würde er natürlich auch die neue Netzwerk-ID übermittelt bekommen und verwenden, was wiederum dazu führen würde, dass seine Nachbarn nun ebenfalls neue IP-Adressen und Netzwerk-IDs anfordern würden. Das Verschmelzen zweier MANETs bedeutet also, dass eines der beiden Netze seine IP-Adressen aufgibt und sich Adressen aus dem anderen Netz zuteilen lässt (siehe [40], Kapitel III.B).

4.1.4 Zusammenfassung

Zwei Probleme bestehen bei allen IPV4-basierten Verfahren.

Der für das Ad-Hoc-Netz gewählte IP-Bereich kann sich beim Aufbau von *stub networks* mit dem IP-Bereich hinter einem Gateway überschneiden und so zu Problemen bei der Adressierung führen.

Zudem kann sich die IP bei der Bewegung durch ein größeres Netz ändern: So kann es beispielsweise beim Vereinigen zweier vorher getrennter Netze zu IP-Konflikten kommen, die nur durch das umkonfigurieren bisher vergebener IPs behoben werden können. Jedes Ändern der IP-Adressen hat zum einen jedoch auch Änderungen in der Routingstruktur zur Folge und lässt zum anderen auch möglicherweise bestehende Verbindungen auf Applikationsebene unbrauchbar werden.

4.2 IPv6

1998 wurde mit der Version 6 ein Nachfolger zum bis heute gebräuchlichen IP-Protokoll Version 4 eingeführt. IP-Version 6 — kurz IPv6 — nutzt Adressen mit einer Länge von 128 Bit und bietet damit einen wesentlich größeren Adressraum (siehe [DH98]).

Um IPv6-Adressen in MANETs automatisch zu konfigurieren, könnte man die für IPv4 vorgestellten Lösungen anpassen und analog verwenden. IPv6 bietet jedoch eine eigene, *stateless address autoconfiguration* genannte, Möglichkeit zur automatischen Konfiguration von IP-Adressen (siehe [TN98]). Der große Adressraum ermöglicht es, direkt aus der Hardware-

Adresse (MAC) eine theoretisch weltweit eindeutige IP-Adresse zu generieren (siehe [23], Kapitel 3.3.1).

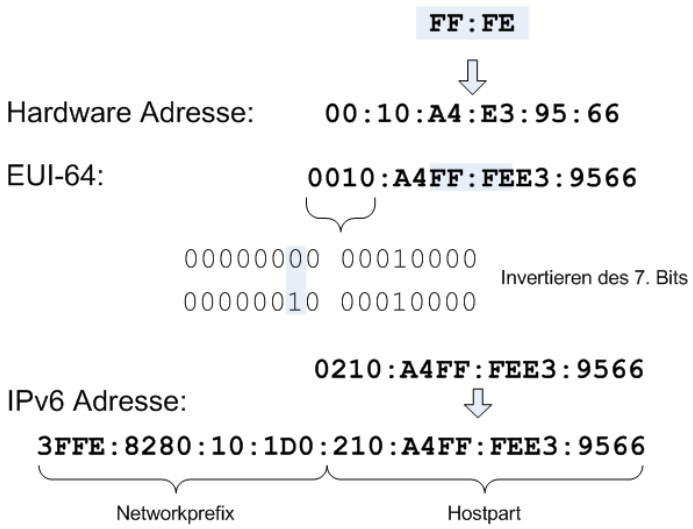

Abbildung 4.2: IPv6 Autokonfiguration Beispiel

Abbildung 4.2 zeigt, wie die IPv6-Adresse erzeugt wird. Zunächst muss aus der 48 Bit langen Hardware-Adresse ein 64 Bit *Extended Universal Identifier* (EUI-64) gebildet werden. Dazu werden die Oktets `FF` und `FE` zwischen dem dritten und vierten Byte der Adresse eingefügt (siehe [HD98], Anhang A). Der so erzeugte Interface-Identifier wird nun zur Bildung des Hostparts der Adresse verwendet, wobei das siebente Bit invertiert wird (siehe [HD98], Kapitel 2.5.1).

Die Adresse wird dann aus einem geeigneten Netzwerkpräfix und dem Hostpart gebildet. Für Adressen mit globalem Scope wird das Präfix `3FFE:8280:10:1D0` verwendet.

Diese Methode generiert jedoch nur global eindeutige IPs, wenn auch die verwendeten Hardware-Adressen eindeutig sind. Es gibt jedoch Netzwerkkartenhersteller, die nicht registrierte Hardware-Adressen verwenden oder es ermöglichen, die Hardware-Adresse der Karte per Software zu ändern, so dass die Eindeutigkeit in sehr großen Netzen nicht mehr gewährleistet ist.

Das Protokoll sieht daher vor, dass die automatisch erzeugte Adresse vor der Verwendung auf Konflikte überprüft werden muss. Die dabei in IPv6 verwendete *Duplicate Address Detection* (DAD) ist der im Zeroconf-Verfahren verwendeten Vorgehensweise relativ ähnlich (vgl. Kapitel 4.1.1). Statt dem in IPv4 verwendeten ARP-Protokoll werden bei IPv6 sogenannte *Neighbor Solicitation Message*s (NS) und *Neighbor Advertisement Message*s (NA) genutzt (siehe [NNS98], Kapitel 4.3 und 4.4). Diese sind jedoch, ähnlich den ARP-Paketen, auf den *Link Local Scope*, also die direkten Nachbarn beschränkt.

Eine entsprechende Erweiterung, um NS- und NA-Nachrichten auch über mehrere Hops weiterzuleiten, wurde in [32] vorgeschlagen. Um unnötige netzweite Broadcasts einzuschränken,

wurde zusätzlich eine dem im OLSR-Protokoll verwendeten Verfahren ähnliche Methode vorgeschlagen, um sogenannte *Leader Nodes* auszuwählen, welche dann die DAD-Nachrichten weiterleiten (siehe [WZ02]). Keine der beiden Methoden ist jedoch zur Zeit Bestandteil des IPv6-Standards.

Größter Nachteil der Verwendung von IPv6 für MANET-Lösungen ist die derzeit fehlende Unterstützung durch Anwendungprogramme. Durch die Nutzung von IPv4- zu IPv6-Proxies wie 6tunnel (siehe [1]) können jedoch auch IPv4-Programme innerhalb einer IPv6-Infrastruktur genutzt werden.

5 Entwurf, Implementation, Test

In diesem Teil der Arbeit wird beispielhaft gezeigt, wie mit heute vorhandener Technologie sich selbstorganisierede WLANs aufgebaut werden können. Entwickelt werden zwei Beispiel-Implementationen: eine, welche auf der zur Zeit noch üblichen IP-Version 4 basiert und eine, bei der die neue IP-Version 6 verwendet wird. Ziel ist es, ein kleines Netz von drei bis vier Nodes aufzubauen und Daten zwischen den einzelnen Nodes bei sich änderner Topologie, auszutauschen.

Die bisherigen Kapitel haben gezeigt, welche Probleme bei MANETs zu lösen sind:

Zum spontanen Aufbau der Netze ist es nötig, dass die IP-Adressen der am Netz teilnehmenden Nodes automatisch und netzweit eindeutig zugewiesen werden können. In Kapitel 4.1 wurden dafür verschiedene Lösungen vorgestellt. Jedoch existiert derzeit für keinen der vorgeschlagenen Algorithmen eine Implementation, weshalb für den hier zu zeigenden Prototyp zunächst eine eigene Lösung entwickelt werden muss. Dass diese Eigenentwicklung nötig werden würde, war nicht von vornherein klar, sondern ergab sich erst während der Arbeit.

Um eine Selbstorganisation zu ermöglichen, muss das verwendete Routingprotokoll in der Lage sein, die entsprechenden Routen auch bei wechselnder Netzwerktopologie ohne große Netzwerkbelastung zu managen. Ein entsprechendes Protokoll ist auf Basis der in Kapitel 3.2 durchgeführten Simulationen auszuwählen.

Zudem sollte eine MANET-Lösung in der Lage sein, auf eine Aufteilung größerer Netze und die Vereinigung von Subnetzen zu reagieren. Auch das Zusammenführen unabhängig voneinander konfigurierter Netze sollte möglich sein.

Um die Implementationen mit mehreren Rechnern testen zu können, ohne auf jedem die benötigte Software installieren zu müssen, wird eine bootfähige CD-ROM mit allen erforderlichen Treibern und der benötigten Software erstellt. Die erforderlichen Schritte zur Teilnahme an einem MANET, also dem Konfigurieren der Hardware und dem Starten der entsprechenden Software, werden dabei über Skripte automatisiert.

5.1 Entwurf

Die prototypische Implementation erfolgt unter dem freien UNIX-Derivat Linux, welches gegenüber anderen Betriebssystemen mehrere Vorteile zum Aufbau von MANETs bietet: Zum einen sind durch die Verfügbarkeit des Quellcodes Anpassungen an die eigenen Bedürfnisse jederzeit möglich, zum anderen erleichtert das offene Konzept das Erkennen von Zusammenhängen zwischen den verschiedenen beteiligten Komponenten. Dazu kommt, dass die ersten

Implementationen für neue Netzwerkprotokolle oft als erstes für UNIX-Plattformen erscheinen, bevor eine Portierung auf andere Systeme erfolgt. Im speziellen Fall der mobilen Routingprotokolle beispielsweise existieren bis jetzt ausschließlich Implementierungen für Linux bzw. FreeBSD. Auch die erst seit kurzem für Microsoft Betriebssysteme erhältliche Unterstützung des IPv6-Protokolls ist schon seit längerem ein Bestandteil des Linux-Kernels.

Sowohl für die IPv4- als auch die IPv6-Version kommt das AODV-Protokoll zum Einsatz. Zum einen lieferte es in den Netzwerksimulationen neben DSR für alle Szenarien sehr gute Ergebnisse, zum anderen wurde dieses Protokoll bereits vollständig implementiert.

Da es zur Zeit keine verfügbaren Implementationen der in Kapitel 4.1 vorgestellten Verfahren zur automatischen Konfiguration von IPv4-Addressen in MANETs gibt, musste eine eigene Implementation entwickelt werden. Die Wahl fiel dabei auf das Prophet Allocation-Verfahren (vgl. Kapitel 4.1.3), da dieses relativ einfach zu implementieren ist und dennoch eine gute Möglichkeit für die Zuweisung eindeutiger IP-Adressen bietet.

Lediglich auf die Erkennung des Verschmelzens zweier zuvor unabhängiger Netze wird aus Zeitgründen verzichtet. Um dies zu realisieren wäre der periodische Austausch von Netzwerk-IDs zwischen den Nodes notwendig. Dies wäre zum einen durch das Anpassen der Quellen für den Routing-Daemon möglich, so dass die Netzwerk-ID, wie in [40] vorgeschlagen, in den Routing-Kontrollpaketen ausgetauscht wird oder durch zusätzliches Austauschen von HELLO-Paketen nur für die Übermittlung der Netzwerk-ID. Für die in diesem Prototyp vorgestellten sehr kleinen Netze und dem sehr grossen Adressbereich 10.0.0.0/8 ist jedoch auch für unabhängig konfigurierte Netze die Wahrscheinlichkeit doppelt vergebener IPs nur sehr gering.

Als Basis der zu erstellenden Boot-CD dient „Timo's Rescue CD Set" (siehe [21]) — eine auf Debian basierende Minidistribution, welche sich durch gute Handhabbarkeit und einfache Erweiterungsmöglichkeiten auszeichnet.

5.1.1 Nicht behandelte Themen

Diese Arbeit zeigt, welche Probleme es grundsätzlich beim Aufbau von spontanen Funknetzen gibt und wie diese zu lösen sind. Nicht untersucht wird jedoch, welche Änderungen auf Applikationsebene nötig sind oder wie bestimmte Services in spontanen Netzen gefunden werden können. Auch das Thema der Sicherheit in spontanen Netzen wird hier nicht eingehend behandelt. Dennoch sollen dazu kurz einige Überlegungen vorgestellt werden.

Servicelokalisierung

In nicht spontanen, kabelgebundenen Netzen werden bestimmte Services wie der Zugriff auf Datenbanken, Verzeichnis- oder Maildienste von immer den selben, durch ihren DNS-Namen identifizierte Servern zur Verfügung gestellt. Für spontane Netze, die sowohl ständige Topologieänderungen als auch Änderungen in den IP-Adressen einzelner Nodes erfahren, ist das Finden eines Netzteilnehmers, der einen bestimmten Service anbietet, weitaus schwieriger.

Eine Möglichkeit zur Service-Vermittlung wäre der Einsatz des *Service Location Protocol* SLP (siehe [VGPK97] und [GPVD99]). Hier suchen sogenannte *User Agents* über Multicast-Anfragen nach *Service Agent*s, die Informationen über die angebotenen Dienste liefern können. SLP ist je nach Netzgröße skalierbar, so können die User Agents entweder direkt nach den Dienste anbietenden Knoten suchen oder mehrere Service Agents können von *Directory Agents* verwaltet werden (siehe [Rot02], Seite 234ff).

Sun bietet mit *Java Intelligent Network Infrastructure* (JINI) (siehe [7]) eine Möglichkeit der Dienste-Vermittlung für Java-basierte Anwendungen. Auch hier gibt es einen den Directory Agents ähnlichen Dienste-Verwalter, der durch Discovery Aufrufe gefunden und nach aktiven Diensten befragt bzw. über neue Dienste informiert werden kann (siehe [Rot02], Seite 238ff). Bei solchen zentralen Serviceverwaltungen muss jedoch die Frage geklärt werden, welche Nodes im MANET diese Aufgabe übernehmen. Ein Mechanismus ähnlich dem von OLSR (vgl. Kapitel 3.1.4) verwendeten zum Ermitteln von Multipoint Routern bietet sich hier an.

Anwendungen, die keiner Anpassung für die Verwendung in MANETs bedürfen, sind typische *Peer2Peer*-Anwendungen wie Filesharing- oder Chat-Programme. Als Beispiel wäre hier das Gnutella-Protokoll (siehe [5]) zu nennen, das ohne zentralen Server einen Austausch direkt zwischen den einzelnen Nodes ermöglicht. Ein Verfahren zur Bildung von Hubs und Supernodes hilft auch hier bei der Skalierung für besonders große Netze.

Applikationsebene

Ein Problem für heutige Applikationen stellt eine sich während einer laufenden Netzwerkkommunikation änderne IP-Adressen dar. Die meisten Netzwerkanwendungen, abgesehen von denen, die auf zustandslose Protokolle wie HTTP aufsetzen, gehen davon aus, dass die IP-Adresse des Kommunikationspartners während der gesamten Sitzung gleich bleibt — eine Annahme, die bei spontanen Netzen keineswegs gegeben ist.

Bei der Verwendung von weltweit eindeutigen IPv6-Adressen entfällt das Problem natürlich. Für IPv4 wurde bereits ein „IP Handoff" (siehe [41]) genanntes Verfahren vorgeschlagen. Das Verfahren basiert auf einer Kombination von mehreren IP-Adressen auf einem Netzwerkinterface und *Network Address Translation* (NAT). So könnte eine tiefere Schicht dafür sorgen, dass bestehende Kommunikationsverbindungen höherer Schichten durch Umschreiben der IP-Adressen erhalten bleiben.

Sicherheit

Nicht behandelt wurde in dieser Arbeit das Thema Sicherheit in MANETs. Einen umfassener Überblick zu Texten über MANET-Sicherheit findet sich auf [42]. Da sich die Sicherheitsprobleme auf Netzwerk- und Anwendungebene nur wenig von denen nichtspontaner Netze unterscheiden, sollen hier nur einige der möglichen Angriffspunkte auf die Architektur von MANETs vorgestellt werden.

Da jeder Netzteilnehmer auch als Router fungiert, vereinfacht sich das Abhören fremder Kommunikation für eventuelle Angreifer. Durch sich ständig änderne Routen wird jedoch das gezielte Abhören bestimmter Verbindungen wiederum schwerer. Dennoch sind vor allem die Algorithmen zur IP-Zuweisung und die Routingprotokolle angreifbar.

So wären unter anderem folgende Angriffe auf mobile Routingprotokolle denkbar (vgl. auch [22] und [31]):

- Versenden falscher Route-Requests von verschiedenen Positionen aus, um so die verfügbare Bandbreite durch unnötigen Routingverkehr zu belasten
- Denial of Service-Attacken durch das wiederholte Erzeugen von Route-Kontrollpaketen oder Datenpaketen
- Nichtantworten eines Knotens auf Route-Requests
- Weiterleiten von Daten an andere Knoten außerhalb des Netzes, bzw. Knoten, die nicht auf der eigentlichen Route liegen
- Versenden falscher Route-Replies, um fremde Pakete abzufangen (Man-in-the-Middle-Attacken)

In der Literatur werden dazu vor allem auf Public Key-Verfahren basierende Lösungen zur Authentifizierung einzelner Nodes und der Validierung von Kontrollpaketen vorgeschlagen.

5.2 Implementation

5.2.1 Vorraussetzungen

Hier soll zunächst auf die Voraussetzungen eingegangen werden, die den Aufbau eines WLAN-basierten MANETs unter dem Linux Betriebssystem erst möglich machen.

Konventionen

Im Folgenden werden Beispiele zur Installation und Konfiguration von Daemons und Protokollen vorgestellt. Zur besseren Lesbarkeit und besserem Verständnis sollen dabei folgende Konventionen eingehalten werden:

Befehle, die als normaler User auf der Kommandozeile ausgeführt werden, beginnen mit $>.

```
$> make
```

Werden für Befehle, die auf der Kommandozeile ausgeführt werden, *root*-Rechte benötigt, so wird dies durch #> gekennzeichnet.

```
#> make install
```

Falls nicht anders erwähnt, werden alle Programme mit dem GNU C-Compiler (*gcc*) bzw. dem GNU C++-Compiler (*g++*) in der Version 2.95.4 kompiliert. Als Distribution kommt ein Debian Linux mit einem ungepatchten Standard-Kernel von `http://www.kernel.org` in der Version 2.4.24 zum Einsatz.

WLAN Support

Der Intersil Prism-Chipsatz (Version 1 und 2) ist der verbreitetste in den erhältlichen WLAN-Karten. Er wird momentan unter Linux von verschiedenen Treibermodulen unterstützt. Zum einem vom im Standard-Kernel enthaltenen `orinoco` Treiber, zum anderen vom `prism2` Treiber, welcher im Linux-WLAN-Project (siehe [10]) entwickelt wird.

Der `prism2`-Treiber befindet sich jedoch noch in der Entwicklungsphase und hat daher noch Probleme mit einigen Karten, weshalb in den folgenden Implementierungen nur der `orinoco`-Treiber eingesetzt wird. Als WLAN-Karten kamen eine *Netgear MA401* PCMCIA Karte, eine *Cabletron RoamAbout 802.11 DS* PCMCIA Karte sowie eine *Netgear MA311* PCI Karte zum Einsatz.

Karten mit anderem, von Linux unterstütztem Chipsatz, sollten analog zum hier Gezeigten funktionieren, solange sie den Ad-Hoc-Mode unterstützen.

Viele moderne Laptops sind jedoch mit Intels Centrino-Technologie ausgestattet — dazu gehört auch ein *Intel PRO/Wireless Lan 2100* genannter WLAN-Chipsatz (siehe [6]), der bisher von Linux nicht direkt unterstützt wird. Um auch auf diesen Laptops die WLAN-Hardware nutzen zu können, kann auf das OpenSource-Projekt *NdisWrapper* (siehe [11]) zurück gegriffen werden. Die Software besteht aus einem Kernelmodul, das die Hardware über die NDIS-Schnittstelle der Windowstreiber anspricht und einem zusätzlichen Userspacetool. Nach dem Kompilieren und Installieren der Software mittels dem mitgelieferten Makefile, muss zunächst der passende Windowstreiber für die Intel-Hardware herunter geladen und ndiswrapper bekannt gemacht werden:

```
$> unzip intel2100b.zip
#> ndisdriver -i WINXP/w70n51.inf
```

Anschließend kann der entsprechende Treiber geladen werden:

```
#> modprobe ndiswrapper
```

Um eine einheitliche Schnittstelle zur Konfiguration der WLAN-Karten unter Linux zu haben, wurden von den für den WLAN-Support verantwortlichen Kernel-Entwicklern die *Linux Wireless LAN Extensions* (siehe [37]) definiert, die von allen WLAN-Kartentreibern unter Linux implementiert werden — auch der NdisWrapper unterstützt die Wireless Extensions. Über diese Schnittstelle können die Karten mit Hilfe der *wireless tools* (siehe [16]) angesprochen und konfiguriert werden. Hauptwerkzeug ist dabei das Programm `iwconfig`, mit dem die verschiedenen Einstellungen des Kartentreibers beeinflusst werden.

Für den Einsatz in einem MANET muss die Karte zunächst in den Ad-Hoc-Modus versetzt werden. Zudem muss eine *Extended Service Set ID* (ESSID) (vgl. Kapitel 2.2.4) definiert werden, welche das momentane Funknetz kennzeichnet. WLAN-Karten nutzen diese ID, um nur den zu einem bestimmten Netz gehörigen Funkverkehr zu emfangen. Empfangene Daten mit ESSIDs anderer Netze werden von der Karte automatisch verworfen. So ist es möglich, mehrere sich flächenmäßig überlagernde, aber von einander unabhänige Funknetze zu betreiben. Für spontane Netze ist dies jedoch nicht erwünscht - einige Karten bieten daher die Möglichkeit, sie im *ESSID promiscous mode* zu betreiben, bei den jeder empfangene Frame unabhängig von seiner ESSID verarbeitet wird. Da dies jedoch nicht von allen WLAN-Karten unterstützt wird, müssen alle am MANET teilnehmenden Rechner mit der selben ESSID konfiguriert werden. Für die folgenden Implementationen soll der String `MANET` als ESSID dienen.

Der folgende Befehl versetzt die über `eth1` angesprochene WLAN-Karte in den Ad-Hoc-Modus und setzt die ESSID auf `MANET`.

```
#> iwconfig eth1 essid MANET mode ad-hoc
```

IPV6 Support

Der IPv6-Support ist im Linux-Kernel momentan noch als experimentell deklariert. Bei der Konfiguration des Kernels müssen daher auch die experimentellen Features (`Code maturity level options --> Prompt for development and/or incomplete code/drivers`) eingeschaltet werden, um die IPv6-Unterstützung auswählen zu können (`Networking options --> The IPv6 Protokoll`). Wird der IPv6-Code als Modul kompiliert, anstatt ihn direkt in den Kernel zu integrieren, kann der IPv6-Support je nach Bedarf nachgeladen werden:

```
#> modprobe ip6
```

5.2.2 IPv4

Das folgende Kapitel zeigt eine beispielhafte Konfiguration eines MANETs auf Basis der IP-Version 4.

Prophet Address Allocation

Für den entwickelten Prototyp des Prophet Adress Allocation-Algorithmus (vgl. Kapitel 4.1.3) wird die in [40] vorgeschlagene Primzahlzerlegung zur Generierung von neuen IP-Adressen verwendet. Zur Berechnung wird dabei folgende Funktion verwendet:

$newip = \langle\langle ip + p_1^{s_1} \cdot p_2^{s_2} \cdot ... \cdot p_8^{s_8}\rangle \bmod range\rangle + 1$

Bei p_1 bis p_8 handelt es sich um acht verschiedene im Quellcode fest definierte Primzahlen — es müssen also nicht immer neue Primzahlen gefunden werden. Als state der Funktion kommt ein Feld von acht Zahlen (s_1 bis S_8) zum Einsatz. Anders als in der sehr einfachen Funktion aus Kapitel 4.1.3 erzeugt hier nicht die Funktion selbst den neuen Status, sondern dieser wird zusätzlich durch das Hochzählen eines der acht Statusfelder generiert. Welches Feld dazu verwendet wird, speichert ein zusammen mit dem Status ausgetauschter Index. $range$ ist die Anzahl aller möglichen Adressen, also 16777215 bei dem verwendeten Adressbereich 10.0.0.0/8.

Zur Verdeutlichung des Algorithmus hier der der entsprechende C-Code zum Erzeugen einer neuen IP-Adresse:

```
u_long ip;        // holds my own address
u_long address=1; // holds the new address
int i = 0;

ip = libnet_get_ipaddr4(_libnet);

//calc new address
for(i=0; i<K; i++){
    address *= _prime[i]^_prophet.state[i];
    address %= RANGE;
}
address += ip;
address %= RANGE;
address += 1;
address += BASENET;        //add 10.0.0.0
address  = ntohl(address); //convert byte order

//increase state at index_pos
_prophet.state[_prophet.index]++;
```

Der erste Knoten des Netzes wählt seine eigene Adresse zufällig aus dem verfügbaren Addressbereich, wobei der Zufallsgenerator mit der Hardware-Adresse der Netzwerkkarte initialisiert wird. Dieser erste Knoten initialisiert auch alle Felder des Status mit 1 und setzt den Index auf das erste Statusfeld. Generiert der Knoten nun eine Addresse für einen anderen Node, so erhöht er das erste Statusfeld und überträgt, die neue IP-Adresse, den Status und den Index an den anfragenden Knoten. Dieser erhöht nun seinerseits den Index um 1 und nutzt fortan das zweite Statusfeld. Bekommt ein Node den Index des achten Feldes, beginnt wird dieser auf Null zurückgesetzt.

Eine Simulation mit einem dafür entwickelten Perl-Script zeigte, dass mit den hier verwendeten Werten durchschnittlich 5200 eindeutige Adressen im verwendeten Adressbereich erzeugt werden können bis es zu einer Doppelvergabe kommt. Durch eine geeignetere Wahl der Funktion ist dieser Wert sicherlich noch zu verbessern — in Anbetracht der in Kapitel 3.2 simulierten Netzgrößen von maximal 200 Nodes jedoch vorerst durchaus ausreichend.

Wie IP-Anfragen und -Antworten ausgetauscht werden sollen, ist in der Beschreibung des Prophet-Algorithmus nicht definiert. Für diese Implementation werden modifizierte ARP-Pakete verwendet (vgl. Abbildung 5.1). ARP-Pakete enthalten eine *opcode* genannte ID, die die Art des Pakets kennzeichnet. Gebräuchlich sind die opcodes 1 und 2 für ARP-Requests und -Replies. In verschiedenen RFCs sind opcodes von 1 bis 10 definiert. Für den hier implementierten Prophet-Algorithmus werden zusätzlich zwei neue definiert: Anfragen nach einer neuen IP-Addresse werden als Broadcasts mit dem opcode 55 (`ARP_PROPHET_REQ`) versendet. Replies werden durch den opcode 56 (`ARP_PROPHET_REP`) gekennzeichnet.

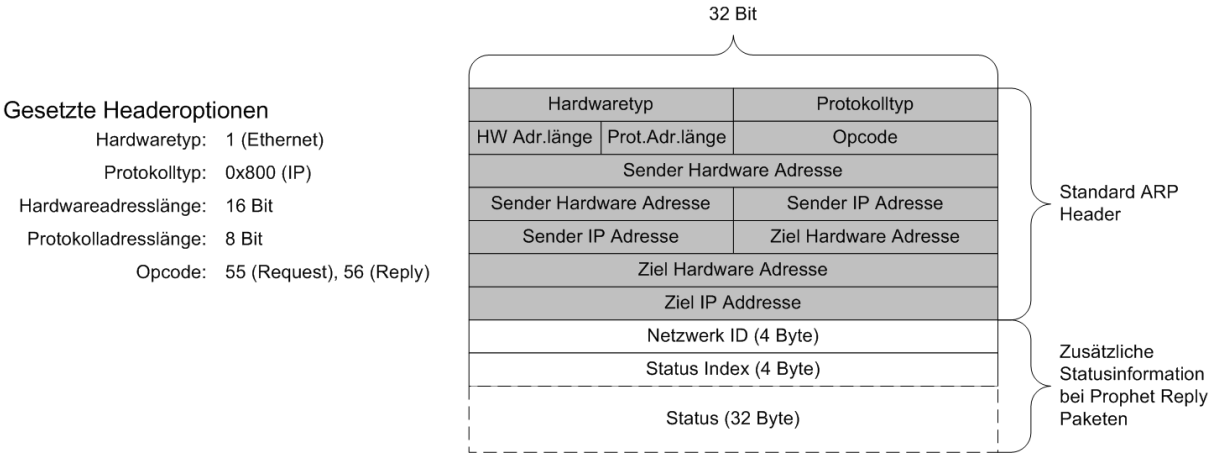

Abbildung 5.1: Aufbau der für Prophet modifizierten ARP Pakete

Prophet-Requests unterscheiden sich nur durch den opcode von normalen ARP-Requests. Absender- und Empfänger-IP-Adresse werden auf `0.0.0.0` gesetzt, die Empfänger-Hardware-Adresse wird mit der Hardware-Broadcast-Adresse `FF:FF:FF:FF:FF:FF` gefüllt und die Absender-Hardware-Adresse mit der Adresse der benutzten WLAN-Netzwerkkarte.

Prophet-Replies werden direkt an die Hardware-Adresse des anfragenden Knotens geschickt. Absender-IP- und -Hardware-Adressen enthalten die Daten der Netzwerkkarte des Senders. In der Ziel-IP-Addresse wird die neu erzeugte IP-Adresse versandt. Anders als Standard-ARP-Pakete enthalten die Prophet-Antwort-Pakete zusätzlichen Payload, in welchem die State-Informationen und die Netzwerk-ID ausgetauscht werden.

Die hier vorgestellte Implementation erzeugt zwar eine Netzwerk-ID und versendet diese auch in den Reply-Paketen, jedoch werden keine zusätzlichen periodischen HELLO-Pakete mit der Netzwerk-ID ausgetauscht. Eine Erkennung von zwei verschmelzenden unabhängig konfigu-

rierten MANETs, wie in Kapitel 4.1.3 beschrieben, ist daher mit dieser beispielhaften Implementation noch nicht möglich.

Eine Simulation mit dem erwähnten Perl-Script zeigt, dass ohne den Austausch der Netzwerk-ID die Gefahr einer Kollision stark ansteigt. Das Script simuliert zwei unabhängige Netze, zu denen abwechselnd Nodes hinzugefügt werden und anschließend auf gemeinsame IPs untersucht werden. Durchschnittlich kam es hier bereits nach ca. 100 IPs pro Netz zu in beiden Netzen vorhandenen IPs, die bei einer Verschmelzung der Netze zu einer Kollision führen würden.

Zum Erzeugen und Versenden der ARP-Pakete wird die libnet-Bibliothek (siehe [8]) genutzt. Das Filtern empfangener ARP-Pakete erfolgt über die libpcap-Bibliothek (siehe [9]). Beide Bibliotheken müssen dementsprechend vor dem Kompilieren installiert sein. Das Kompilieren selbst kann dann über das mitgelieferte Makefile erfolgen:

```
$ make
```

Das erzeugte Programm `prophet` muss mit root-Rechten und der zu konfigurierenden Netzwerkkarte als Parameter aufgerufen werden.

```
# ./prophet eth1
```

Das Programm versucht dann über das Versenden von ARP-Requests eine IP-Adresse von einem der Nachbarn zu empfangen und die Netzwerkkarte mit dieser zu konfigurieren. Wurde nach fünf, im Abstand von drei Sekunden versendeten, Requests immer noch keine Antwort empfangen, so inititialisiert es die Karte mit einer zufälligen Adresse aus dem Adressbereich 10.0.0.0/8.

Nach der Konfiguration der Adresse beginnt das Programm auf IP-Requests zu lauschen und diese zu beantworten. Zu beachten ist, dass es sich bei der Implementation von der Funktionsweise her zwar um einen Daemon handelt, das Programm jedoch nicht automatisch im Hintergrund läuft.

Während libnet und libpcap für verschiedene Plattformen erhältlich sind, ist der Code zur Konfiguration der Netzwerkkarte Linux-spezifisch und müsste für die Portierung auf andere Betriebssysteme entsprechend angepasst werden.

AODV-UU

Das AODV-Protokoll (vgl. 3.1.2) ist unter den mobilen Routingprotokollen derzeit das am besten implementierteste. So existieren derzeit mehrere Implementationen für Linux, wobei der an der Universität von Uppsala (siehe [3]) entwickelte Userspace-Daemon (`aodvd`) die zur Zeit am weitesten entwickelte und daher hier eingesetzte Implementierung des AODV-Protokolls ist. `aodvd` liegt aktuell in der Version 0.8 vor.

AODV-UU lässt sich nur auf einem Kernel mit Netfilter-Support kompilieren. Der Daemon versucht beim Start die Netfilterkomponente `ip_queue` als Modul nachzuladen. Liegt diese jedoch nicht als Modul vor, sondern wurde direkt in den Kernel kompiliert, schlägt das Starten des Daemons fehl. Der Kernel sollte also entsprechend konfiguriert sein.

Die Installation beschränkt sich auf das Auspacken und Kompilieren der Quellen über das mitgelieferte Makefile:

```
$> tar -xzvf aodv-uu-0.7.2.tar.gz
$> cd aodv-uu-0.7.2
$> make
#> make install
```

`make install` installiert ein Kernelmodul in das entsprechende Verzeichnis unter `/lib/modules/`, sowie den eigentlichen Routing-Daemon nach `/usr/sbin/aodvd`.

Vor der Nutzung des AODV-Daemons muss das entsprechende WLAN-Device hoch gefahren und mit einer IPv4-Adresse versehen sein. AODV-UU verlangt dabei, dass alle IP-Adressen im selben Subnetz liegen — im konkreten Fall also im Bereich 10.0.0.0/8. Eine Nutzung mehrerer verschiedener Bereiche wie beispielsweise dem zusätzlichen Nutzen des 192.168.0.0/16 Bereichs ist nicht möglich.

Anschließend kann der Daemon mit `#> aodvd` gestartet werden. Er lädt automatisch die benötigten Kernel-Module, bindet sich an das erste gefundene WLAN-Device und beginnt mittels HELLO-Paketen nach Nachbarn zu suchen.

Das folgende Beispiel zeigt das Starten des Daemons und das Finden eines Nachbarn mit der IP-Adresse `10.0.0.5`:

```
#> aodvd
14:15:19.365 host_init: Attaching to eth1, override with -i <if1,if2,...>.
14:15:19.483 aodv_socket_init: Receive buffer size set to 262144
14:15:19.483 main: In wait on reboot for 15000 milliseconds. Disable with "-D".
14:15:19.483 hello_start: Starting to send HELLOs!
14:15:34.491 wait_on_reboot_timeout: Wait on reboot over!!
14:15:44.715 rt_table_insert: Inserting 10.0.0.5 (bucket 5) next hop 10.0.0.5
14:15:44.716 rt_table_insert: New timer for 10.0.0.5, life=2100
14:15:44.716 hello_process: 10.0.0.5 new NEIGHBOR!
```

Der gefundene Nachbar wird dann als Host-Eintrag (`UH`) in die Routingtabelle des Kernels übernommen:

```
#> route -n
Kernel IP routing table
Destination  Gateway       Genmask         Flags Metric Ref  Use Iface
```

```
10.0.0.5     0.0.0.0       255.255.255.255 UH    1      0      0 eth1
192.168.1.0  0.0.0.0       255.255.255.0   U     0      0      0 eth0
10.0.0.0     0.0.0.0       255.0.0.0       U     0      0      0 eth1
0.0.0.0      192.168.1.10  0.0.0.0         UG    0      0      0 eth0
```

5.2.3 IPv6

AODV6

Die AODV-Implementierung der Universität von Uppsala unterstützt nur IPv4. Jedoch existiert ein auf der Version 0.5 basierender Zweig für IPv6 (siehe [30]). Da die Implementierung jedoch auf einer älteren Version basiert, wird hier nicht der komplette in RFC 3561 (siehe [PBRD03]) beschriebene Umfang des Protokolls, sondern nur die im AODV-Draft 10 (siehe [34]) beschriebenen Funktionen unterstützt.

Zum Kompilieren des Daemons muss nach dem Auspacken der Sourcen zunächst ein von der selben Seite (siehe [30]) erhältlicher Patch eingespielt werden, bevor mit dem mitgelieferten Makefile die Übersetzung und Installation angestoßen werden kann:

```
$> tar -xzvf AODV6-R0.2
$> cp -r AODV6-R0.2 AODV6-R0.2.org
$> patch -p0 <AODV6-R0.2-ishizuka.diff
$> rm -rf AODV6-R0.2.org
$> cd AODV6-R0.2
$> make
#> make install
```

Zu beachten ist, dass die IPv6-Version des AODV-Daemons zur Zeit nur mit Site Local-Adressen funktioniert. Statt dem globalen Präfix `3FFE:8280:10:1D0` wird daher für diese Arbeit der Präfix `FEC0` mit Site Local Scope verwendet.

5.2.4 Live CD

Die im Rahmen dieser Arbeit erstellte Live-CD (Anhang Seite 75) enthält ein direkt von der CD bootfähiges Linux-System, sowie die in diesem Kapitel vorgestellte Software zur Teilnahme an einem MANET. Die CD installiert keinerlei Daten auf der Festplatte des Rechners.

Bei der auf „Timo's Rescue CD Set" (siehe [21]) basierenden CD wurde der ursprüngliche 2.4.20er Kernel gegen einen eigenen, angepassten 2.4.24er Kernel mit PCMCIA- und WLAN-Support ausgetauscht und die zusätzlich benötigte Software in das CD-Image integriert.

Für das Kompilieren der Software für die Live-CD mussten teilweise Makefiles angepasst werden. Auf der CD finden sich alle verwendeten Quellen mit den entsprechenden Änderungen.

Die CD stellt neben den im Kernel enthaltenen WLAN-Karten-Treibern und den AODV-Daemons für IPv4 und IPv6 zwei Scripte zur Verfügung, die das Laden und Konfigurieren der benötigten Software übernehmen. Nachdem der benötigte Treiber für die WLAN-Karte geladen wurde (bei PCMCIA-Karten geschieht das automatisch beim Einstecken der Karte), kann mit dem Script `manet-ipv4.sh` bzw. `manet-ipv6.sh` der Rechner für die Teilnahme an einem MANET konfiguriert werden. Dabei wird die IP-Adresse automatisch konfiguriert und der AODV-Daemon gestartet. Die Meldungen der Daemons lassen sich auf den virtuellen Konsolen 10 (prophet) und 11 (aodvd) mitverfolgen.

5.3 Tests

In diesem Kapitel soll die Funktionsfähigkeit der in Kapitel 5.2 vorgestellten MANET-Lösungen getestet werden. Dabei wird die dafür entwickelte Live-CD (vgl. Kapitel 5.2.4) auf drei Rechnern eingesetzt und Daten zwischen diesen ausgetauscht.

Für den Datenaustausch kommen verschiedene Protokolle zum Einsatz. Zunächst wird die generelle Erreichbarkeit mittels ICMP-Pings überprüft. Zudem werden interaktive Verbindungen über SSH aufgebaut. Untersucht werden soll, ob die entsprechenden Routen wie erwartet gefunden werden und inwieweit sich Änderungen in der Routingstruktur auf bestehende Verbindungen auswirken.

Um verschiedene Verbindungskonstellationen testen zu können, ohne die beteiligten Rechner jeweils aus gegenseitiger Reichweite bewegen zu müssen, werden Pakete mittels der im Linux-Kernel integrierten Firewall auf MAC-Ebene gefiltert und so ein Verlassen der Funkreichweite simuliert. Zur Konfiguration der Firewall kommt das Werkzeug `iptables` zum Filtern von IPv4-Paketen bzw. `ip6tables` für IPv6-Pakete zum Einsatz.

```
#> iptables -A INPUT -m mac ---mac-source 00:09:5B:4C:2F:E1 -j DROP
#> ip6tables -A INPUT -m mac ---mac-source 00:09:5B:4C:2F:E1 -j DROP
```

5.3.1 IPv4

Zunächst werden alle drei verwendeten Rechner in gegenseitiger Reichweite von der Live-CD gestartet und nach dem Laden der Netzwerkkartentreiber das Initialisationsscript gestartet. Die drei Rechner konfigurieren mittels Prophet Allocation-Verfahren ihre IP-Adressen und starten den AODV-Daemon. Im Test wurden die Knoten mit den Adressen `10.5.142.124` (Node 1), `10.61.169.140` (Node 2) und `10.21.5.181` (Node 3) initialisiert.

Da alle Rechner in gegenseitiger Reichweite sind, erzeugt der AODV-Daemon für alle Nachbarn Einträge in der Routingtabelle mit der Metrik 1:

```
Destination     Gateway         Genmask         Flags Metric Iface
10.61.169.140   0.0.0.0         255.255.255.255 UH    1      eth0
10.21.5.181     0.0.0.0         255.255.255.255 UH    1      eth0
10.0.0.0        0.0.0.0         255.0.0.0       U     0      eth0
```

Unterbindet man nun Verbindungen direkt zwischen Node 1 und Node 3, so wird der Eintrag für Node 3 aus der Routingtabelle von Node 1 entfernt. Da AODV ein reaktives Protokoll ist (vgl. Kapitel 3.1.2), wird die Route erst bei Bedarf ermittelt. Bei einem Ping auf die IP von Node 3 ermittelt der AODV-Daemon die korrekte Route über Node 2 und legt den entsprechenden Eintrag in der Kernel-Routingtabelle an:

```
#> route -n
Destination     Gateway         Genmask         Flags Metric Iface
10.61.169.140   0.0.0.0         255.255.255.255 UH    1      eth0
10.21.5.181     10.61.169.140   255.255.255.255 UGH   2      eth0
10.0.0.0        0.0.0.0         255.0.0.0       U     0      eth0
```

Um zu testen wie lange AODV braucht, um eine neue geeignete Route bei einer Topologieänderung zu finden, wird Node 3 von Node 1 aus zunächst bei bestehender direkter Verbindung angepingt und dann während des Pingens die Verbindung unterbrochen.

```
64 bytes from 10.21.5.181: icmp_seq=7 ttl=64 time=1.9 ms
64 bytes from 10.21.5.181: icmp_seq=8 ttl=64 time=2.5 ms
64 bytes from 10.21.5.181: icmp_seq=9 ttl=64 time=2.9 ms
64 bytes from 10.21.5.181: icmp_seq=10 ttl=64 time=2.1 ms
64 bytes from 10.21.5.181: icmp_seq=13 ttl=64 time=508.5 ms
64 bytes from 10.21.5.181: icmp_seq=14 ttl=64 time=34.2 ms
64 bytes from 10.21.5.181: icmp_seq=15 ttl=64 time=27.6 ms
64 bytes from 10.21.5.181: icmp_seq=16 ttl=64 time=34.0 ms
```

Die Ausgabe zeigt, dass die Antworten auf zwei Pings (11 und 12) komplett verloren gehen bevor der AODV-Daemon die neue Route erstellt hat. Deutlich zu sehen ist auch die Verzögerung, die durch den längeren Weg über Node 2 entsteht — von durchschnittlichen Antwortzeiten von 3 Millisekunden auf durchschnittlich 30 Millisekunden nach der Routenänderung.

Bei AODV bleiben Routen nur solange bestehen wie sie genutzt werden. Die hier eingesetzte Implementation verwendet einen Timeout von 15 Sekunden. Werden also über eine bestehende Route keine Daten mehr ausgetauscht, so wird diese nach 15 Sekunden gelöscht und muss bei Bedarf neu erstellt werden. Ein Ping auf Node 3 bei nicht bestehender Route zeigt die benötigte Zeit zum Finden der Route:

```
64 bytes from 10.21.5.181: icmp_seq=0 ttl=64 time=1053.3 ms
64 bytes from 10.21.5.181: icmp_seq=1 ttl=64 time=97.3 ms
64 bytes from 10.21.5.181: icmp_seq=2 ttl=64 time=32.6 ms
64 bytes from 10.21.5.181: icmp_seq=3 ttl=64 time=29.5 ms
64 bytes from 10.21.5.181: icmp_seq=4 ttl=64 time=31.9 ms
```

Bei interaktiven Verbindungen wie bei der hier getesteten SSH-Verbindung macht sich diese Eigenschaft der reaktiven Protokolle besonders bemerkbar. Arbeitet man kurze Zeit nicht in der offenen Sitzung, geht die Route verloren und muss bei der nächsten Eingabe zunächst neu erstellt werden, was eine merkbare Latenz von eben der, mit dem Ping ermittelten, Sekunde erzeugt.

Um dies zu vermeiden, könnte entweder auf ein proaktives Protokoll zurückgegriffen werden (vgl. Kapitel 3.1) oder die Route mittels im Hintergrund ausgetauschter „Keep-Alive" Pakete erhalten werden. Bei nicht interaktiven Protokollen wie Streaming- oder Dateiübertragungsprotokollen spielt dieses Problem keine Rolle, da bei diesen in der Regel kontinuierlich Daten ausgetauscht werden.

5.3.2 IPv6

Beim Aufbau eines MANETs unter Verwendung des IPv6-Protokolls werden die IP-Adressen mit Site Local Scope aus der Hardware-Adresse generiert (vgl. Kapitel 4.2). Im Test waren dies die Adressen `fec0::209:5bff:fe67:f0b2` (Node 1), `fec0::209:5bff:fe91:82d5` (Node 2) und `fec0::209:5bff:fe4c:2fe1` (Node 3).

Wie schon im IPv4-Test erzeugt der AODV-Daemon zunächst Einträge für alle gefundenen Nachbarn in der Kernel-Routingtabelle, welche mit dem Programm `ip` angezeigt werden können.

```
#> ip -6 route show dev eth1
fe80::/64  metric 256  mtu 1500 advmss 1440
fec0::209:5bff:fe4c:2fe1  proto redirect  metric 2  mtu 1500 advmss 1440
fec0::209:5bff:fe91:82d5  proto redirect  metric 2  mtu 1500 advmss 1440
fec0::/64  metric 256  mtu 1500 advmss 1440
ff00::/8  metric 256  mtu 1500 advmss 1440
```

Die direkten Nachbarn von Node 1 werden in der Routingtabelle mit der Metrik 2 angelegt und haben damit Vorrang vor den Standard-Routingeinträgen für die zu den IP-Adressen gehörigen Netze mit der Metrik 256.

Bei der Unterbrechung der direkten Verbindung zwischen Node 1 und Node 3 wurde auf Node 1 die Routingtabelle entsprechend angepasst, jedoch schlägt das Löschen des alten Routingeintrags auf Node 3 mit der Meldung „`aodvd6: rt_table_invalidate: Could not delete kernel route!: no such process`" fehl. Statt dessen finden sich zwei Routingeinträge für Node 1: Der alte, direkte Eintrag mit Metrik 2 und die neue Route über Node 2 mit Metrik 3. Da der Eintrag mit der Metrik 2 den Vorrang hat, diese Route jedoch nicht mehr existiert, ist ein Antworten auf die von Node 1 gesendeten Pings nicht möglich.

Startet man den AODV-Daemon auf Node 3 bei unterbrochener Verbindung zwischen Node 1 und 3 neu, so wird eine korrekte Routingtabelle aufgebaut. Der Daemon hat also offenbar in der eingesetzen Version einen Fehler und ist damit nicht in der Lage auf eine sich ändernde Topologie entsprechend zu reagieren.

Bei einer statischen Netzwerktoplogie werden die Routen je nach Bedarf jedoch korrekt auf- und wieder abgebaut. Der Aufbau der Route erfolgte sogar wesentlich schneller als bei IPv4. Wartete man bei IPv4 noch über 1000 Millisekunden länger auf die Antwort des ersten Pings als auf die folgenden, so beträgt der Unterschied in der Antwortzeit bei IPv6 gerade 10 Millisekunden:

```
64 bytes from fec0::209:5bff:fe4c:2fe1: icmp_seq=0 ttl=64 time=17.2 ms
64 bytes from fec0::209:5bff:fe4c:2fe1: icmp_seq=1 ttl=64 time=7.01 ms
64 bytes from fec0::209:5bff:fe4c:2fe1: icmp_seq=2 ttl=64 time=8.23 ms
64 bytes from fec0::209:5bff:fe4c:2fe1: icmp_seq=3 ttl=64 time=9.61 ms
64 bytes from fec0::209:5bff:fe4c:2fe1: icmp_seq=4 ttl=64 time=9.38 ms
```

Auch die Pingzeiten bei bestehender Route sind mit rund 9 Millisekunden wesentlich kürzer als bei IPv4 (30 Millisekunden). Ursache hierfür könnte zum einen eine effizientere Implementation in derIPv6-Version des AODV-Daemons sein, zum anderen aber vor allem die Optimierungen die speziell für das Routing im IPv6-Header eingeführt wurden. So hat der IPv6-Header eine feste Größe von 40 Bytes und enthält keine Checksumme, die bei jedem Empfang geprüft werden muss. Zusätzlich wurden die von einem weiterleitenden Router zu überprüfenen Felder im Header von sechs auf vier reduziert (siehe [Dav03], Kapitel 4). Insgesamt muss das Betriebssystem also wesentlich weniger Schritte für eine Routingentscheidung durchführen, woraus die beobachtete Performance-Verbesserung resultieren könnte.

5.3.3 Zusammenfassung

Der hier verwendete AODV-Daemon der Universität Upsala war in der IPv4-Version in der Lage, Routen auch bei wechselner Topologie zuverlässig herzustellen. Verbunden mit der prototypischen Implementation der Prophet Address Allocation konnte ein funktionierendes MANET implementiert werden.

Der Einsatz des IPv6-Protokolls zeigte zwar eine deutliche bessere Routingperformance, der auf einer älteren Version der des Upsala-Daemons basierende IPv6-Version, war jedoch nicht in der Lage, korrekt auf wechselne Routen einzugehen und ist damit für den Einsatz noch nicht geeignet.

Ein wirklich funktionsfähiges MANET war damit nur unter IPv4 realisierbar.

6 Fazit

6.1 Zusammenfassung und Bewertung

In der vorliegenden Arbeit wurde gezeigt, wie spontane Wireless LANs im Allgemeinen aufgebaut werden können und wie die automatisierte IP-Vergabe und das Routing in spontanen Netzen zu realisieren sind.

Von der großen Anzahl der in der Fachwelt vorgeschlagenen mobilen Routingprotokolle wurden in dieser Arbeit die Verfahren DSDV, AODV, DSR, OLSR und TORA auf Grund ihres Bekanntheitsgrades ausgewählt und näher untersucht. Neben der theoretischen Vorstellung (siehe Kapitel 3.1) wurden die einzelnen Protokolle im Netzwerksimulator ns2 in verschiedenen Anwendungszenarien simuliert (Kapitel 3.2). Neben kleineren Szenarien wie dem Einsatz zur Vernetzung von Teilnehmern eines Kongresses und dem Einsatz auf einem Universitäts-Campus wurden auch anspruchsvollere Simulationen, wie die Nutzung eines MANETs zur Kommunikation zwischen Rettungskräften in einem Erdbebengebiet oder die spontane Vernetzung von Fahrzeugen im innerstädtischen Verkehr untersucht (siehe Kapitel 3.2.1). Neben der Simulation dieser vordefinierten Szenarien, wurden auch Netze unterschiedlicher Größe, also größerer Nodeanzahl und Ausbreitungsfläche getestet.

Untersucht wurde, wie sich die verschiedenen Kennwerte des Netzwerkverkehrs je nach simuliertem Szenario und eingesetztem Routingprotokoll ändern. Verglichen wurde dabei die Anzahl der erfolgreich zum Ziel gelieferten Pakete, die durchschnittliche Paketübertragungszeit, der vom Routingprotokoll erzeugte Overhead und die Fähigkeit der getesteten Protokolle optimale Routen zu finden.

Bei allen Simulationen zeigten die Protokolle DSR und AODV eine deutliche Überlegenheit gegenüber den anderen Verfahren. In besonders großen Netzen (simuliert wurde ein Netz mit 200 Teilnehmern) war nur noch AODV in der Lage, die gewünschten Routen zuverlässig zu finden (siehe Kapitel 3.2.5).

Neben den in Kapitel 4.1 vorgestellten verschiedenen Lösungsansätzen zur automatischen Vergabe von IPv4-Adressen in spontanen Netzen, wie der *Duplicate Address Detection* und des *Prophet Address Allocation*-Verfahrens, wurde in Kapitel 4.2 die in der IPv6-Spezifikation vorgesehene *stateless address autoconfiguration* zur automatischen IP-Konfiguration vorgestellt.

Im praktischen Teil der Arbeit (Kapitel 5) wurde unter Verwendung des AODV-Protokolls ein kleines, aus drei Nodes bestehendes Netz aufgebaut. Dafür wurde eine bootfähige Linux-CD entwickelt, welche alle benötigte Software und entsprechende Initialisationsskripte enthält (siehe Kapitel 5.2.4).

Im Praxisteil zeigte sich auch die Aktualität des Themas. So gibt es derzeit noch keine Implementation der vorgeschlagenen Methoden zur automatischen IP-Vergabe mit IPv4. Daher wurde im Rahmen der Arbeit eine prototypische Implementation des Prophet Allocation-Algorithmus erstellt. Dabei wurden jedoch nicht alle in Kapitel 5.1 genannten Anforderungen erfüllt: durch das fehlende periodische Versenden der Netzwerk-ID kann ein Verschmelzen zweier unabhängig gebildeter Netze nicht erkannt werden. Durch die Wahl des relativ großen Adressbereichs führt dies jedoch in der Regel, wie in Kapitel 4.1.3 gezeigt, erst bei größeren Netzen mit mehr als 100 Nodes zu Kollisionen.

Auch beim Aufbau des MANETs unter Verwendung des IPv6-Protokolls mussten Einschränkungen hingenommen werden. So musste auf die eigentlich im Standard vorgesehende Duplicate Address Detection (vgl. Kapitel 4.2) verzichtet werden, da diese nicht für Netze mit mehreren Hops spezifiziert ist und entsprechende vorgeschlagenene Erweiterungen des IPv6-Protokolls noch nicht implementiert sind. Beim Einsatz ordnungsgemäßer Hardware sind die so generierten IP-Adressen jedoch trotzdem, wie in Kapitel 4.2 ausgeführt, weltweit eindeutig.

Während das IPv4-basierte Netz zuverlässig auch mit wechselner Netztopologie arbeitete (vgl. Kapitel 5.3.1), zeigte der für die IPv6-Implemenation verwendete AODV-Daemon noch Fehler bei der Behandlung von wechselnen Routen (vgl. Kapitel 5.3.2). Der Aufbau eines IPv6-basierten MANETs war daher nicht unter Berücksichtigung aller Anforderungen möglich.

Weiterführende Probleme wie Servicelokalisierung oder Sicherheit in spontanen WLANs wurden in dieser Arbeit nicht ausführlich behandelt, sondern nur in Kapitel 5.1.1 kurz vorgestellt.

Es wurde gezeigt, dass MANETs mit den heute zur Verfügung stehenden Technologien bereits realisierbar sind, wenn auch durch unvollständige Implementationen Einschränkungen hinzunehmen sind. Für spezialisierte Anwendungen mit homogenem Benutzerkreis, wie beispielsweise die spontane Vernetzung von Kongressteilnehmern oder Arbeitsgruppen sind MANETs, wie der praktische Teil der Arbeit gezeigt hat, schon heute realisierbar und sinnvoll anwendbar. Für den breiten Einsatz von spontanen drahtlosen Netzen ist jedoch noch einige Entwicklungsarbeit zu leisten.

6.2 Ausblick

Nicht alle Facetten spontaner WLANs konnten in dieser Arbeit behandelt werden. So ist auf dem Gebiet der MANETs noch einge Arbeit zu leisten, bevor sie größere Verbreitung finden werden.

Für den breiten Einsatz von MANETs sind vor allem die in Kapitel 5.1.1 kurz angeschnittenen Themen wie Sicherheit, Servicelokalisierung und die Auswirkungen auf Applikationsebene weiter zu erforschen und eventuelle Probleme zu lösen.

Für den Aufbau spontaner Netze mit dem IPv4 Protokoll muss vor allem ein Verfahren zur automatischen IP-Konfiguration vollständig implementiert werden. Der in Kapitel 5.2.2 vorgestellte Prototyp ist wegen des Fehlens der Übermittlung der Netzwerk ID nur bedingt in mehreren Netzen einsetzbar. Auch die zur Vergabe der IP Adressen gewählte Funktion kann hinsichtlich der Ausnutzung des zur Verfügung stehenden Adressbereichs noch verbessert werden.

Die für MANETs in Zukunft interessantere Lösung ist jedoch sicherlich der Einsatz des IPv6 Protokolls. Die bereits vom Protokoll vorgesehene automatische Zuweisung von IP Adressen (siehe Kapitel 4.2) und der große Adressbereich, der eine weltweit eindeutige Zuweisung von IP Adressen ermöglicht, machen IPv6 zum prädestinierten Protokoll für spontane Netze. Für den breiten Einsatz sollte jedoch die in der IPv6 Spezifikation vorgesehene Duplicate Address Detection auch über mehrere Netze hinweg implementiert werden. Auch das in Kapitel 5.3.2 beobachtete effizientere Routingverhalten spricht für den Einsatz von IPv6 in MANETs. Somit könnte die stärkere Verbreitung von spontanen Netzen im Rahmen des Wunsches nach der „Überallverfügbarkeit" (*ubiquitous computing*) auch zu einem breiteren Einsatz des IPv6 Protokolls insgesamt beitragen.

Obwohl jedoch die vorgestellten Routingalgorithmen selbst unabhängig von der verwendeten IP Version funktionieren, gibt es zur Zeit so gut wie keine Implementationen für IPv6. Auch der in dieser Arbeit verwendete AODV Daemon zeigte noch Fehler in der Implementation.

Mit dem Heranreifen der MANET Technologie auf der einen Seite und der Weiterentwicklung der WLAN Hardware sind neben den in Kapitel 3.2.1 vorgestellten Szenarien diverse andere Einsatzgebiete vorstellbar. Denkbar wären beispielsweise *VoiceOverIP* basierte Telefone die durch MANETs unabhängig von jeglicher Provider-Infrastruktur wären oder die nahezu flächendeckende Vernetzung ganzer Städte und Ballungsräume. MANETs könnten schließlich eine weitere Stufe in der Entwicklung des Internets darstellen.

Literaturverzeichnis

[CM99] CORSON, S. und J. MACKER: *RFC2501: Mobile Ad hoc Networking MANET - Routing Protocol Performance Issues and Evaluation Considerations*, 1999. URL: `http://www.ietf.org/rfc/rfc2501.txt`.

[Dav03] DAVIES, JOSEPH: *Understanding IP Version 6*. Microsoft Press, 2003. ISBN: 0-7356-1245-5.

[DH98] DEERING, S. und R. HINDEN: *RFC2460: Internet Protocol, Version 6 (IPv6) Specification*, 1998. URL: `http://www.ietf.org/rfc/rfc2460.txt`.

[GPVD99] GUTTMAN, E., C. PERKINS, J. VEIZADES und M. DAY: *RFC2608: Service Location Protocol, Version 2*, 1999. URL: `http://www.ietf.org/rfc/rfc2501.txt`.

[HD98] HINDEN, R. und S. DEERING: *RFC2373: IP Version 6 Addressing Architecture*, 1998. URL: `http://www.ietf.org/rfc/rfc2373.txt`.

[IEE99a] IEEE: *802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999. URL: `http://standards.ieee.org/getieee802/download/802.11-1999.pdf`.

[IEE99b] IEEE: *802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications – Amendment 1: High-speed Physical Layer in the 5 GHz band*, 1999. URL: `http://standards.ieee.org/getieee802/download/802.11a-1999.pdf`.

[IEE99c] IEEE: *802.11b: Wireless LAN MAC and PHY specifications: Higher speed Physical Layer (PHY) extension in the 2.4 GHz band*, 1999. URL: `http://standards.ieee.org/getieee802/download/802.11b-1999.pdf`.

[IEE01] IEEE: *802.11d: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Specification for Operation in Additional Regulartory Domains*, 2001. URL: `http://standards.ieee.org/getieee802/download/802.11d-2001.pdf`.

[JM96] JOHNSON, DAVID B. und DAVID A. MALTZ: *Dynamic Source Routing in Ad Hoc Wireless Networks*. In: IMIELINSKI, THOMASZ und HENRY F. KORTH (Herausgeber): *Mobile Computing*, Band 353. Kluwer Academic Publishers, 1996. ISBN: 0-7923-9697-9.

[JMB01] JOHNSON, D., D. MALTZ und J. BROCH: *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*. In: PERKINS, CHARLES E. (Herausgeber): *Ad Hoc Networking*. Addison-Wesley, 2001. ISBN: 0-201-30976-9.

[NNS98] NARTEN, T., E. NORDMARK und W. SIMPSON: *RFC2461: Neighbor Discovery for IP Version 6 (IPv6)*, 1998. URL: `http://www.ietf.org/rfc/rfc2461.txt`.

[PB96] PERKINS, CHARLES E. und PRAVIN BHAGWAT: *Routing over Multi-Hop Wireless Network of Mobile Computers*. In: IMIELINSKI, THOMASZ und HENRY F. KORTH (Herausgeber): *Mobile Computing*, Band 353. Kluwer Academic Publishers, 1996. URL: `http://www.ce.chalmers.se/undergraduate/IMP/EDA435/lastyear/pictures/dsdv.pdf`. ISBN: 0-7923-9697-9.

[PBRD03] PERKINS, CHARLES E., ELIZABETH M. BELDING-ROYER und SAMIR R. DAS: *RFC3561: Ad hoc On-Deman Distance Vector (AODV) Routing*, 2003. URL: `http://www.ietf.org/rfc/rfc3561.txt`.

[PR01] PERKINS, CHARLES E. und ELIZABETH M. ROYER: *The Ad Hoc On-Demand Distance Vector Protocol*. In: PERKINS, CHARLES E. (Herausgeber): *Ad Hoc Networking*. Addison-Wesley, 2001. ISBN: 0-201-30976-9.

[Rot02] ROTH, JÖRG: *Mobile Computing: Grundlagen, Technik, Konzepte*. dpunkt. Verlag, 1. Auflage, 2002. URL: `http://www.dpunkt.de/mobile/`. ISBN: 3-89864-165-1.

[Tan03] TANENBAUM, ANDREW S.: *Computer Networks*. Prentice Hall, 4. Auflage, 2003. ISBN: 0-13-066102-3.

[TN98] THOMSON, S. und T. NARTEM: *RFC2462: IPv6 Stateless Address Autoconfiguration*, 1998. URL: `http://www.ietf.org/rfc/rfc2462.txt`.

[VGPK97] VEIZADES, J., E. GUTTMAN, C. PERKINS und S. KAPLAN: *RFC2165: Service Location Protocol*, 1997. URL: `http://www.ietf.org/rfc/rfc2501.txt`.

[WZ02] WENIGER, KILIAN und MARTINA ZITTERBART: *IPv6 Autoconfiguration in Large Scale Mobile Ad-Hoc Networks*. In: *European Wireless 2002*, 2002. URL: `http://www.ing.unipi.it/ew2002/proceedings/111.pdf`.

WWW-Verzeichnis

[1] *6tunnel*. URL: `http://toxygen.net/6tunnel/` [Letzter Zugriff am 07.03.2004].

[2] *Ad hoc protocol list – Wikipedia*. URL: `http://en.wikipedia.org/wiki/Ad_hoc_protocol_list` [Letzter Zugriff am 14.01.2004].

[3] *AODV Userspace Implementation*. URL: `http://user.it.uu.se/~henrikl/aodv/` [Letzter Zugriff am 05.11.2003].

[4] *Description of Automatic Private IP Addressing in Windows Millennium Edition*. URL: `http://support.microsoft.com:80/support/kb/articles/Q307/2/87.ASP` [Letzter Zugriff am 08.03.2004].

[5] *Gnutella Protocol Development*. URL: `http://rfc-gnutella.sourceforge.net` [Letzter Zugriff am 23.04.2004].

[6] *Intel PRO/Wireless Network Connection*. URL: `http://www.intel.com/products/mobiletechnology/prowireless.htm` [Letzter Zugriff am 08.03.2004].

[7] *Jini Network Technology*. URL: `http://wwws.sun.com/software/jini/` [Letzter Zugriff am 23.02.2004].

[8] *Libnet*. URL: `http://www.packetfactory.net/projects/libnet/` [Letzter Zugriff am 26.01.2004].

[9] *Libpcap*. URL: `http://www.tcpdump.org/` [Letzter Zugriff am 26.01.2004].

[10] *Linux WLAN Project*. URL: `http://www.linux-wlan.org` [Letzter Zugriff am 13.03.2004].

[11] *NdisWrapper*. URL: `http://ndiswrapper.sourceforge.net/` [Letzter Zugriff am 08.03.2004].

[12] *The Network Simulator ns2*. URL: `http://www.isi.edu/nsnam/ns/` [Letzter Zugriff am 16.11.2003].

[13] *ProteanForge: OLSR*. URL: `http://pf.itd.nrl.navy.mil/projects/olsr/` [Letzter Zugriff am 5.1.2004].

[14] *Rendezvous*. URL: http://developer.apple.com/macosx/rendezvous/ [Letzter Zugriff am 08.03.2004].

[15] *Unix and Linux Zeroconf Networking*. URL: http://zeroconf.sourceforge.net/ [Letzter Zugriff am 12.12.2003].

[16] *Wireless Tools for Linux*. URL: http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html [Letzter Zugriff am 13.11.2003].

[17] *Zero Configuration Network Taskforce*. URL: http://www.ietf.org/html.charters/zeroconf-charter.html [Letzter Zugriff am 13.03.2004].

[18] *MANET Working Group Charter*, 2003. URL: http://www.ietf.org/html.charters/manet-charter.html [Letzter Zugriff am 23.10.2003].

[19] *Pressemeldung: Popular Wireless Local Area Networks Gain Large Boost in Speed*, 2003. URL: http://standards.ieee.org/announcements/80211gfinal.html [Letzter Zugriff am 27.10.2003].

[20] *wLAN - Aktuelle Entwicklungen*, 2003. URL: http://www.ecin.de/state-of-the-art/wlanmarkt/ [Letzter Zugriff am 26.10.2003].

[21] BENK, TIMO: *Timo's Rescue CD Set*. URL: http://rescuecd.sourceforge.net/ [Letzter Zugriff am 22.02.2204].

[22] BHARGAVA, SONALI und DHARMA P. AGRAWAL: *Security Enhancements in AODV protocol for Wireless Ad Hoc Networks*, 2001. URL: http://www.mcl.hu/adhoc/literature/QoS%20and%20Security/Security%20Enhancements%20in%20AODV%20protocol%20for%20Wireless%20Ad%20Hoc%20Networks.pdf [Letzter Zugriff am 22.02.2004].

[23] BIERINGER, PETER: *Linux IPv6 HOWTO*, 2003. URL: http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/Linux+IPv6-HOWTO.html [Letzter Zugriff am 07.08.2003].

[24] CHESHIRE, STUART: *IPv4 Address Conflict Detection*, 2003. URL: http://files.stuartcheshire.org/draft-cheshire-ipv4-acd.txt [Letzter Zugriff am 12.12.2003].

[25] CHESHIRE, STUART, BERNARD ABOBA und ERIK GUTTMAN: *Dynamic Configuration of Link-Local IPv4 Addresses*, 2003. URL: http://files.zeroconf.org/draft-ietf-zeroconf-ipv4-linklocal.txt [Letzter Zugriff am 12.12.2003]. work in progress.

[26] CORSON, S., S. PAPADEMETRIOU, P. PAPADOPULUS, V. PARK und A. QUAYYUM: *An Internet MANET Encapsulation Protocol (IMEP) Specification*, 1999. URL: http://www.potaroo.net/ietf/old-ids/draft-ietf-manet-imep-spec-01.txt [Letzter Zugriff am 08.02.2004]. work in progress.

[27] GREIS, MARC: *Tutorial for the Network Simulator "ns"*. URL: `http://www.isi.edu/nsnam/ns/tutorial/index.html` [Letzter Zugriff am 16.11.2003].

[28] JEONG, JAEHOON PAUL, JUANGSOO PARK, HYOUNGJUN KIM und DONGKYUN KIM: *Ad Hoc IP Address Autoconfiguration*, 2003. URL: `http://www.adhoc.6ants.net/~paul/publications/ietf-internet-draft/draft-jeong-adhoc-ip-addr-autoconf-01.txt` [Letzter Zugriff am 16.01.2004]. work in progress.

[29] KLEIN-BERNDT, LUKE: *Quickguide to AODV*. URL: `http://w3.antd.nist.gov/wctg/aodv_kernel/aodv_guide.pdf` [Letzter Zugriff am 10.01.2004].

[30] LEE, PETER: *AODV For IPv6 Network*. URL: `http://members.shaw.ca/aodv6-sfu/` [Letzter Zugriff am 10.11.2003].

[31] NING, PENG und KUN SUN, 2003. URL: `http://www.mts.jhu.edu/~marchette/ID04/Papers/TRMisuseAODV.pdf` [Letzter Zugriff am 22.02.2004].

[32] PARK, JUNG-SOO, YONG-JIN KIM und SUNG-WOO PARK: *Stateless address autoconfiguration in Mobile Ad Hoc Networks using site-local address*, 2001. URL: `http://www.potaroo.net/ietf/old-ids/draft-park-zeroconf-manet-ipv6-00.txt` [Letzter Zugriff am 07.03.2004]. expired.

[33] PARK, V. und S. CORSON: *Temporally-Ordered Routing Algorithm (TORA) Version 1 – Functional Specification*, 2001. URL: `http://www.potaroo.net/ietf/old-ids/draft-ietf-manet-tora-spec-04.txt` [Letzter Zugriff am 14.01.2004]. work in progress.

[34] PERKINS, CHARLES E., ELIZABETH M. BELDING-ROYER und SAMIR R. DAS: *Ad hoc On-Demand Distance Vector (AODV) Routing [Draft 10]*, 2002. URL: `http://members.shaw.ca/aodv6-sfu/aodv-ietf-10.txt` [Letzter Zugriff am 10.11.2003].

[35] PERKINS, CHARLES E., ELIZABETH M. BELDING-ROYER und SAMIR R. DAS: *Ad hoc On-Demand Distance Vector (AODV) Routing [Draft 13]*, 2003. URL: `http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-13.txt` [Letzter Zugriff am 10.11.2003].

[36] PERKINS, CHARLES E., JARI T. MALINEN, RYUJI WAKIKAWA, ELIZABETH M. BELDING ROYER und YUAN SUN: *IP Address Autoconfiguration for Ad Hoc Networks*, 2001. URL: `http://people.nokia.net/~charliep/txt/aodvid/autoconf.txt` [Letzter Zugriff am 16.01.2004]. work in progress.

[37] TOURRILHES, JEAN: *Wireless Extensions for Linux*, 1997. URL: `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html` [Letzter Zugriff am 07.03.2004].

[38] TOURRILHES, JEAN: *The Linux Wireless LAN HOWTO*, 2003. URL: http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Wireless.html [Letzter Zugriff am 07.08.2003].

[39] ZHOU, HONGBO: *A Survey on Autoconfiguration for MANET*, 2003. URL: http://www.cse.msu.edu/~zhouhon1/doc/A%20Survey%20on%20Autoconfiguration%20for%20MANET.pdf [Letzter Zugriff am 10.08.2003].

[40] ZHOU, HONGBO, LIONEL M. NI und MATT W. MUTKA: *Prophet Adress Allocation for Large Scale MANETs*, 2003. URL: http://www.cse.msu.edu/~zhouhon1/doc/ProphetAddressAlloc.PDF [Letzter Zugriff am 19.01.2004].

[41] ZHOU, HONGBO, LIONEL M. NI und W. MUTKA: *IP Address Handoff in the MANET*, 2003. URL: http://www.cse.msu.edu/~zhouhon1/doc/IP%20address%20handoff.pdf [Letzter Zugriff am 22.02.2004].

[42] ZHU, FENG: *Security for Ad hoc Networks*. URL: http://www.ccs.neu.edu/home/zhufeng/security_manet.html [Letzter Zugriff am 22.02.2004].

Abbildungsverzeichnis

CD

Diese bootfähige CD-ROM enthält neben einem Linux Live System auch alle in der Arbeit erwähnten Quelltexte. Die Daten auf dieser CD stehen alternativ auch unter `http://www.splitbrain.org/go/diplom` zum Download zur Verfügung.

Eigenständigkeitserklärung

Ich erkläre, daß ich die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.