ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

MASTER THESIS

Routing and Applications in Mobile Ad-Hoc Networks

@ Stefan Bollmann

Prof. Dr. Roger Wattenhofer
Regina O'Dell

Distributed Computing Group
Dept. of Computer Science
ETH Zurich, Switzerland

1. April - 30. September 2004

Zusammenfassung

Mobile Ad-Hoc Netzwerk sind bisher noch nicht gut verstanden. Das Ziel der Arbeit war es denn auch, in diesem Gebiet auf neue Erkenntnisse zu stossen. So wurden verschiedene Routing-Algorithmen getestet auf deren Effektivität. Dazu wurde eine Simulationsumgebung erstellt, welche die verschiedenen Aspekten von Peer-to-Peer Systemen berücksichtigt und insbesonders sehr flexibel gestaltet werden sollte.

Als zentrale Komponente wurde das abstraktes Zeitmodell eingeführt, welches eine diskrete Zeiteinheit in verschiedene Arbeitsschritte – Routing, Bewegung der Hosts und Updaten von Routing Informationen unterteilt. Durch die Bildung eines Verhältnis zwischen diesen drei Schritten, kann jedes reale System simuliert werden. Um die Bewegungen der Hosts im Netzwerk zu untersuchen, wurden mehrere Mobilitätsmodelle erstellt. Schlussendlich hängt die Effektivität eines Routing-Algorithmus entscheidend von der Netzwerk-Topologie und ihrer Veränderung ab.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Eine Eigenschaft von P2P Systemen ist, dass die einzelnen Rechner unabhängig voneinander arbeiten. Dadurch fehlt auch eine zentrale Kontrollstelle, welche eine Übersicht über die aktuellen Verbindungen besitzt und das Routing übernehmen / koordinieren könnte. In statischen Systemen ist die Erfassung der Netzwerk-Topologie eine Angelegenheit, welche einmalig ausgeführt werden muss. Die Verbindungen bleiben solange bestehen, wie der neuhinzugekommene Peer im Netzwerk bleibt. Durch einfaches Abfragen seiner Nachbarn kann der neue Peer die Netzwerk-Topologie erfassen [und z. B. in Form einer Routing-Tabelle speichern], welche konstant bleibt [ausser wenn ein Peer hinzukommt oder wegfällt].

In *mobilen Ad-Hoc Netzwerken* hingegen ändern sich die Verbindungen nicht nur aufgrund hinzukommender oder wegfallender Peers, sondern auch durch die Bewegung der Peers. Die Veränderung der Netzwerk-Topologie erfordert eine ständige Aktualisierung der gespeicherten Informationen. Dies geschieht über sogenannte Update-Nachrichten, welche eine zusätzliche Netzwerklast bewirken. Die Anzahl solcher Update-Nachrichten hängt vom gewählten Routing-Algorithmus ab. Dabei gilt es folgende 2 Faktoren abzuwägen:

1. Soll eine Netzwerk-Topologie-Veränderung sofort global ins Netzwerk propragiert werden (proaktiv) oder soll die Erfassung erst geschehen, wenn eine Routing-Nachricht vorliegt, für welche kein Pfad zur Destination bekannt ist (reaktiv).

2. Geschieht die Aktualisierung aufgrund lokaler oder globaler Informationen

Um eine Aussage über die Qualität von einem Routing-Algorithmus zu erstellen, wurden verschiedene Mobilitätsmodelle erstellt.

- sind die in der Theorie gebildeten Arten von Mobilitätsmodelle für einen Routing-Algorithmus effektiv/merklich verschieden?
- welcher Routing-Algorithmus eignet sich für welches Mobilitätsmodell am besten?
- welche weiteren Faktoren sind entscheidend?

Für die Erhaltung von Messwerten von verschiedenen Routing-Algorithmen wurde eine Simulationsumgebung entwickelt. Dazu konnte für erste Überlegungen auf eine bereits existierende Simulationsumgebung zurückgegriffen werden. Im Verlauf der Zeit

stellten sich logische wie auch technische Unkorrektheiten heraus, sodass die Architektur stark umgebaut wurde, bevor die Erweiterungen in Angriff genommen werden konnten. Insbesondere war die bisherige Lösung auch in keinem Masse flexibel.

Kapitel 2

Architektur

2.1 Überblick

Die Architektur [Bild 2.1] basiert auf einer 3-Tier-Architektur. Die Simulationsumgebung (`SimMobile` bzw. `SimVizMobile`) bildet als primitive Anwendung die Application Layer. Sie versendet Nachrichten von einer gewählten Source zu einer bestimmten Destination und greift dabei auf das Network Layer zu. Das Network Layer "stellt eine netzwerkweite Verbindung zwischen zwei Protokolleinheiten der Transportebene zur Verfügung" [5]. Somit kann auch ein Multi-Hop auftreten. Der Routing-Algorithmus übernimmt die Wegelenkung der Nachrichten. Der Message-Handler beinhaltet die Message-Queues. Alle Nachrichten werden hier zwischengespeichert und bei freier Bandbreite / Kapazität weitergeleitet. In einer Abstraktion der Realwelt wurden ein Zeitmodell [siehe 2.3 oder Kapitel 3] entwickelt, welches die verschiedenen Phasen von Routing, Bewegung der Hosts und Updaten von Routing Informationen trennt und damit eine Quasi-Parallelität simuliert. Änderungen in der Netzwerk-Topologie werden mittels *notify* nach oben propagiert. Das Netzwerk wird aufgeteilt in ein physisches und logisches Netzwerk. Das physische Netzwerk beinhaltet die einzelnen Hosts, wohingegen das logische Netzwerk zusätzlich auch die einzelnen Verbindungen zwischen zwei benachbarten Hosts beinhaltet. Das letztere wird auch Graph genannt. Der Routing-

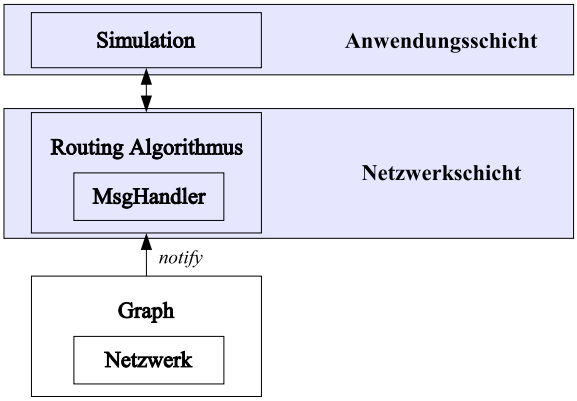

Abbildung 2.1: Architektur

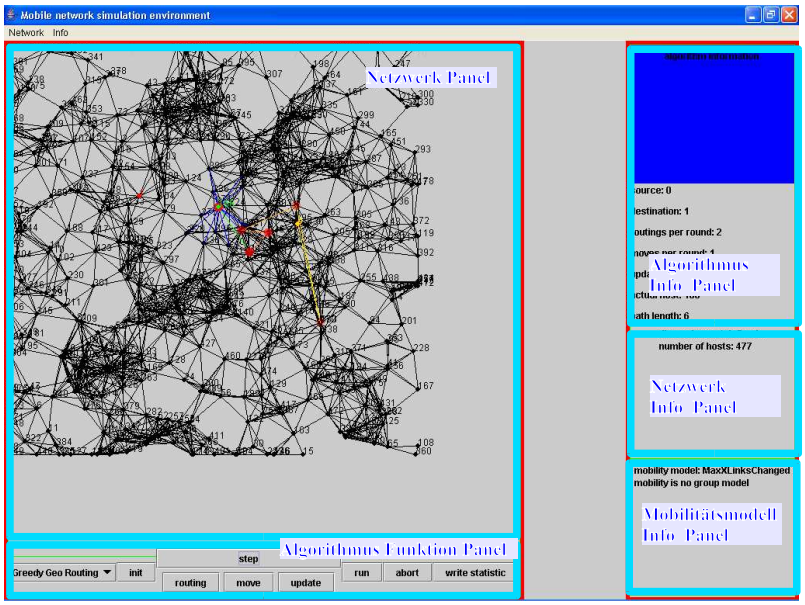

Abbildung 2.2: GUI

Algorithmus arbeitet schlussendlich auch auf dem Graph, welcher Änderungen der Netzwerk-Topologie mittels *notify* dem Routing-Algorithmus meldet.

2.2 Simulationsumgebung

Die Simulationsumgebung ist eine simple Anwendung. Sie sendet eine Nachricht von einer Source zu einer Destination innerhalb eines Netzwerks. Die Simulationsumgebung existiert in einer graphischen und einer nichtgraphischen Variante. Die zweite ist für ausgedehnte Simulationen mit Messungen gedacht, welche auch einen gleichzeitigen Ablauf für mehrere Routing-Algorithmen erlaubt. Die graphische Variante sollte den Ablauf und das Verständnis für einen Routing-Algorithmus verbessern. Ebenso erlaubt es einen Eindruck für die verschiedenen Mobilitätsmodelle zu erhalten. Die Oberfläche des Frames besteht aus verschiedenen Panels, welche modularartig zusammenstellbare Einheiten bilden. Die einzelnen Panels beinhalten Informationen über das Netzwerk, das zugrundeliegende Mobilitätsmodell, der aktuell verwendete Routing-Algorithmus und dessen statistischen Werte wie:

- Source
- Destination
- aktueller Host
- Pfadlänge
- ...

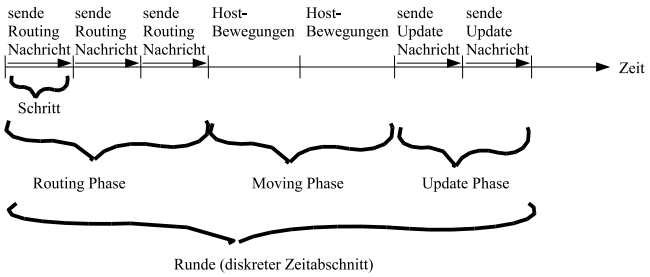

Abbildung 2.3: Zeitmodell

2.3 Zeitmodell

Das Zeitmodell ist die zentrale Komponente der Simulationsumgebung und wird hier kurz umrissen, genaueres folgt dann im Kapitel 3. Um ein möglichst adequates, realtreues Zeitmodell zu erhalten, wurde die Zeit – analog wie bei der Infinitesimalrechnung – in diskrete Zeitabschnitte unterteilt. Ein solcher Zeitabschnitt wird fortan *Runde* genannt. Im Bild 2.3 wird eine Runde dargestellt. Eine Runde wird in drei verschiedene Phasen unterteilt:

Routing In dieser Phase kann in einem Schritt entweder eine neue Routing-Nachricht generiert oder die empfangenenen Routing-Nachrichten verarbeitet und gegebenenfalls an die Destination weitergeleitet werden.

Bewegung der Hosts In dieser Phase werden alle in einem Schritt vorhandenen Bewegungen der Rechner ausgeführt. Dies geschieht transparent im Netzwerk. Veränderungen in der Netzwerk-Topologie werden mittels *notify* an den Routing-Algorithmus gemeldet, und in der darauffolgenden Phase des Updaten von Routing Informationen behandelt. Folgende Ereignisse werden propagiert:

- jede hinzugefügte bzw. entfernte Kanten
- die Bewegung eines Rechner mit Angabe seiner alten und neuen Position
- wenn ein Rechner alle Verbindungenzu seinen Nachbarn neu bestimmt.

Updaten von Routing Informationen In dieser Phase werden vorhandene Update-Nachrichten verarbeitet. Jeder Rechner erneuert aufgrund der erhaltenen Informationen seine Routing Informationen und propagiert Veränderungen an seine Nachbarn mit einer Update-Nachricht weiter.

Eine jeder dieser Phasen wird noch feiner in sogenannte Schritte gegliedert. Die Anzahl der Schritte ist pro Phase einzeln parametrisierbar und kann dadurch variert werden. Grundsätzlich gilt, dass in jedem Schritt des Routing bzw. Updaten von Routing Informationen alle eingehenden Nachrichten empfangen, verarbeitet (im zweiten Fall die Routing Informationen des Empfängers erneuert werden) und neu generierte Nachrichten gegebenenfalls weitergeleitet werden. Der Message-Handler kontrolliert dabei das Medium und garantiert, dass die Bandbreite einer Verbindung zu jedem beliebigen Zeitpunkt (Schritt) nicht überschritten wird.

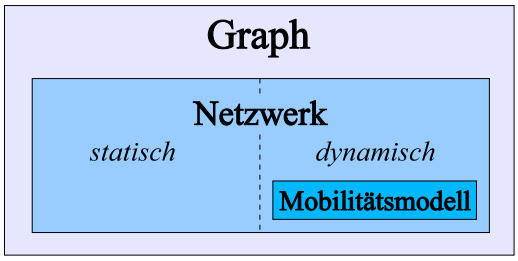

Abbildung 2.4: Netzwerk-Aufbau

2.4 Message-Handler

Der Message-Handler ist für das Übertragen sämtlicher Nachrichten zuständig. Die verschiedenen auftretenden Nachrichten sind Routing-Nachrichten und Update-Nachrichten. Dabei kontrolliert er, dass die verfügbare Bandbreite einer Verbindung nicht überschritten wird. In der vorliegenden Version kann pro Schritt für jede [gerichtete!] Kante eine einzelne Nachricht gesendet werden. Somit kann in einem symmetrisch en Graph in beide Richtungen pro Schritt eine Nachricht übertragen werden. Grundsätzlich wäre aber auch bereits eine andere Einstellung möglich: So könnte die benötigte Bandbreite pro Nachricht – welche momentan für alle Arten von Nachrichten 1 ist – abhängig von der Art der Nachricht und/oder der Grösse der in der Nachricht enthaltenen Daten gewählt werden. Kann eine Nachricht momentan nicht weitergeleitet werden, wird sie in der Output-Queue zurückgehalten, um sie zu einem späteren Zeitpunkt versenden zu können.

2.5 Netzwerk

Alle Hosts im Netzwerk werden durch eine ID [Integer] eindeutig identifiziert. Das physische Netzwerk [`Network`] umfasst alle Hosts und ihre Positionen im Netzwerk ohne eine Beziehung zwischen den Hosts zu definieren. Diese wird erst durch das darüberliegende logische Netzwerk [`Graph`] bestimmt. Es gibt eine Vielfalt von Kriterien, aufgrund deren eine Verbindung zwischen zwei Hosts u und v existiert bzw. nicht existiert:

- Unit Disk Graph: Bildung aufgrund einer maximalen Distanz von 1 zwischen beiden Hosts
- Gabriel Graph: Definiert einen ungerichteten / symmetrischen Graph. Zwischen u und v existiert eine Kante gdw. innerhalb oder auf dem Rand des Kreis mit Durchmesser (u,v) sich kein anderer Host befindet.
- Delaunay Triangulation: Gegeben drei Hosts u, v und w, welche einen Kreis definieren. Befindet sich innerhalb oder auf dem Rand des Kreises sich kein weiterer Hostbefindet, so existiert zwischen allen drei Hosts jeweils eine Verbindung. Somit wird ein Dreieck zwischen den drei Hosts gebildet.
- Relative Neighborhood Graph: Eine Kanteexistiert gdw. kein dritter Hostw existiert, für den gilt: (u, w) ¡ (v, w) und (v, w) ¡ (u, v)

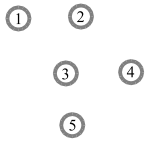

Abbildung 2.5: Netzwerk

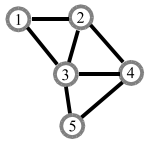

Abbildung 2.6: Unit Disk Graph

- Minimum Spanning Tree: Beinhaltet ein Subset aller Kantenmit minimalen Gewicht, welche zusammen einen Baum bilden.
- ...

Im jetzigen Entwicklungsstand wird nur der Unit Disk Graph [Bild 2.6] zur Verfügung gestellt. Die Software wurde jedoch so gestaltet, dass sie andere Graphen jederzeit unterstützen kann. Wie leicht ersichtlich ist, hat die Wahl des Graphs einen entscheidenden Faktor auf die Effizienz eines Routing-Algorithmus. Bei den folgenden Überlegungen wird jeweils angenommen, dass das physische Netzwerk[`Netzwerk`] identisch ist. Nichtplanare Graphen erlauben durch die höhere Verdrahtung eine kürzere Route. Ebenso können ausgeklügelte Routing-Protokollverschiedene Pfade für ein Ziel verwenden, um z. B. eine gleichmässigere Auslastung aller Verbindungen zu erreichen. Durch Verwendung verschiedener Routen ist auch ein höherer Durchsatz möglich, welcher einen Zeitgewinn verspricht. Ist das Routing-Protokoll jedoch proaktiv und die Mobilität gross, so überwiegt die Anzahl versendeter Update-Nachrichten mit grosser Wahrscheinlichkeit die kürzeren RoutingPfade.

Das Netzwerk resp. der Graph sind vollständig losgelöst vom Routing-Algorithmus. Alle Daten, welche vom Graphen zum Routing-Algorithmus fliessen, durch die Verwendung des Observer Patterns weitergeleitet. Die Daten sind dabei in einer Message [GRAPHUPDATEMESSAGE] gekapselt.

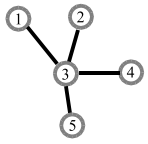

Abbildung 2.7: Minimaler Spannbaum

Kapitel 3

Zeitmodell

Bei dem Zeitmodell handelt es sich um das Kernstück der Arbeit und sollte möglichst flexibel gestaltet werden. Es stellt sich die Frage, wie kann die Koordination von Kommunikation und Mobilität – welche parallel / gleichzeitig (concurrent) ablaufen – sinnvoll gesteuert werden. In der Realwelt könnte dies durch Event-Driven-Programming gelöst. Bei Einhaltung der Ereignisreihenfolge – wobei hier die Frage nach der relativen Wahrnehmung der Zeit offengelassen wird – und der Voraussetzung, dass der Host mehrere Prozesse gleichzeitig verarbeiten kann, kann eine effektive Parallelität erreicht werden. Ein Host reagiert in diesem Fall sofort auf ein Ereignis. Dies tritt insbesondere bei systemkritischen Applikationen ein, welche eine stärkere Form – nämlich Interrupts – zulassen.

In vielen Fällen ist eine echte Parallelität nicht zu erzielen oder keine Voraussetzung. Es genügt gegen aussen eine solche vorzutäuschen. So geht man in der Mathematik das Problem so an, dass man es löst, indem es in kleinste Teile zerteilt wird (Infinitesimalrechung). Einen solchen Ansatz wurde auch hier gewählt um eine Quasi-Parallelität zu simulieren. Eine analoge Vorgehensweise ist auch bei der Daten-Kommunikation bekannt, sogenanntes Zeitmultiplex (Time Division Multiplex, TDM). Somit wird hier auf eine altbewährte Technologie gesetzt.

Diese Methode des Time-Slicing [Bild 3.1] wird in der Regel auch durch Scheduler angewendet. Eine diskrete Zeiteinheit – fortan Runde genannt – wird in Phasen und

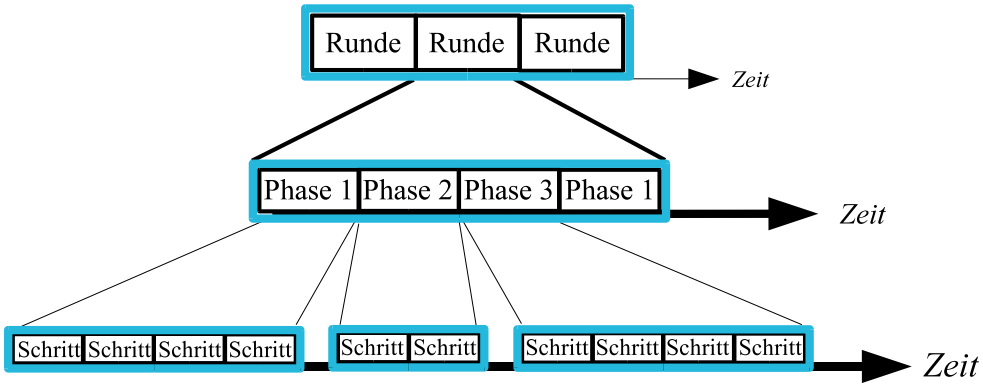

Abbildung 3.1: Zeitmodell: Aufteilung in kleinste Schnitte (Schritte)

diese wiederum in kleinere Zeitschlitze (Time-Slots / Schritte) unterteilt. In den einzelnen Schritten hat im allgemeinen ein beliebiger der anfallenden Prozesse die Möglichkeit die Ressourcen zu benutzen. Das verwendete Zeitmodell schränkt die Flexibilität durch die vorgegebene Reihenfolge für einen bestimmte Art von Prozess durch eine Aufteilung in drei Phasen ein:

1. Routing
2. Bewegung der Hosts
3. Updaten von Routing Informationen

Auf den ersten Blick klingt diese Teilung realitätsfern. Eine praxisnahe Umgebung würde die Schritte einzeln je nach Bedarf zuteilen. Doch stellt sich hier die Frage bei einer Simulation: Welches Muster soll für die Zuteilung gewählt werden?

zufällig Zuteilung Trifft für eine nichtvoraussehbare Lastverteilung zu, kann aber mit geringer Wahrscheinlichkeit für bestimmte Fälle ausarten / anormal verhalten.

synchrone Zuteilung

asynchrone Zuteilung

Zuteilung nach Ressourcenbedarf spezieller Fall der asynchronen Zeitzuteilung aber ressourcen-gesteuert.

...

Die gewählte asynchrone Zeitzuteilung kann als angebracht betrachtet werden. Durch die Steuerung der Anzahl Schritte in jeder Phase über jeweils einen Parameter, kann jede Runde mit anderen Verhältnissen gearbeitet werden. Wird der Spezialfall mit null Schritten herangezogen, lässt sich jedes andere Muster simulieren. Wir haben somit die gewünschte maximale Flexibiltät erreicht. Im GUI der visuellen Simulationsumgebung kann dies auch vollzogen werden, indem jeder Routing-, Moving- oder Update-Schritt in beliebiger Reihenfolge gewählt werden kann.

In jeder dieser Phasen werden Nachrichten ausgetauscht. Dabei wird in zwei prinzipiell verschiedene Nachrichten unterschieden. Erstere werden üblicherweise auch so bezeichnet, wir nennen sie hier Netzwerk-Nachrichten. Sie dienen der Übertragung von Daten zwischen zwei Hosts – also auf der einen Seite für das Routing selber, andererseits auch für den Austausch von Routing Informationen. Für etwas Verwirrung kann die Benennung der zweiten Art von Nachrichten führen. Die GraphUpdateMessage dient nur innerhalb eines Hosts, um das Eintreten eines Ereignisses im Graph dem Routing-Algorithmus zu melden.

3.1 Phase 1: Routing

In dieser Phase kann ein Host in jedem Schritt eine Nachricht initialisieren, um sie einem anderen Host zu senden. Ebenso ist das Empfangen von Nachrichten und deren Verarbeitung möglich. Ist der Empfänger auch gleichzeitig die Destination, so wird die Nachricht verarbeitet und keine weitere erstellt. Handelt es sich hingegen nicht um die Destination, so wird aufgrund von Kriterien des verwendeten Routing-Algorithmus der beste Pfad zur Destination gesucht und die Nachricht an den nächsten Host auf dem Pfad dorthin forwarded. Die ganze Logik zur Suche des nächsten Hosts auf dem Pfad

zur Destination steckt in der Methode `findNextHost(host)`. Des Weiteren existieren eine obere Schranke für die Anzahl erlaubten Runden bzw. Anzahl Fehlversuche, einen Pfad zu finden. Beim "Uberschreiten der Schranke wird der Routing-Algorithmus abgebrochen. Genaueres siehe Kapitel 5.
[Die entsprechende Methode lautet `doRoutingStep(step)`]

3.2 Phase 2: Bewegung der Hosts

In dieser Phase werden in jedem Schritt die Bewegungen der sich im Netzwerk befindenden Hosts ausgeführt. Dies geschieht vollständig unabhängig vom Routing-Algorithmus. Die Bewegungen werden durch das Mobilitätsmodell festgelegt. Ein Host bestimmt nach der Bewegung seinen neue Position im Netzwerk.

Basierend auf dem Observer-Pattern – Routing-Algorithmus implementiert `GraphObserver` – wird nach jedem Schritt der Routing-Algorithmus von Änderungen in dem Graph durch Senden einer Nachricht (`GraphUpdateMessage`) notifiziert. Eine solche Nachricht enthält jeweils als Daten die Ursache der Meldung. Der Routing-Algorithmus kann darauf basierend seine Routing Informationenerneuern. Falls die Routing Informationen sich geändert haben, wird eine `UpdateMessage` erstellt und in der Output-Queue des `Message-Handler` zum Versand in Phase 3 vorbereitet. Folgende Ereignisse werden an alle `GraphObserver` propagiert:

- **jede hinzugefügte bzw. entfernte Kanten**
- **die Bewegung eines Rechner mit Angabe seiner alten und neuen Position.** Es gilt dabei zu beachten, dass der Host sich zwar bewegt hat, die Verbindungen zu seinem Nachbarn aber zu diesem Zeitpunkt **noch nicht** neu erfasst wurden! Es wird also nur angezeigt, dass sich ein Host bewegt hat und von welcher Position zu welcher neuen.
- **dass ein Rechner alle Verbindungenzu seinen Nachbarn neu erfasst hat** Mit den Methoden mboxgetRemovedNeighbors`(host)` bzw. mboxgetAddedNeighbors`(host)` köennen die ehemaligen bzw. neu hinzugekommene bestimmt werden. Aufgrund dieser kann entsprechend in

[Die entsprechende Methode lautet `performeMove(round, step)`]

3.3 Phase 3: Updaten von Routing Informationen

In dieser Phase wird in jedem Schritt aus der erhaltenen Update-Nachricht die neuen Informationen gelesen um die eigenen Routing Informationen anzugleichen. Bei Änderungen der eigenen Routing Informationen werden diese mittels einer neueerzeugten Update-Nachricht weitergeleitet. [Die entsprechende Methode lautet `doUpdateStep()`]

Kapitel 4

Mobilitätsmodelle

Eine der wichtigsten Methoden zur Evaluierung der Charaktistik von mobile Ad-Hoc Netzwerk-Protokollen ist die Simulation [3]. Die Effektivität eines Routing-Algorithmus hängt von dem sich stetig ändernden Netzwerk ab. Die Netzwerk-Topologie und ihre stetige Änderung durch die Bewegung von Hosts, wird dabei durch das Mobilitätsmodell festgelegt. Sind die Hosts einmal verteilt initialisiert mit ihren Positionen, diktiert das Mobilitätsmodell deren Bewegung und somit den aktuellen Standort. Die Wahl des Mobilitätsmodell hat deshalb einen grossen Einfluss auf die Performance eines Routing-Algorithmus, z. B. die Reaktion auf die Teilung des Netzwerks in disjunkte Netzwerke oder deren Zusammenführung.

Beispiel: So ist die Reaktion von Distance Vector Routing bei einer Splittung abnormal. Der Part des Netzwerks, der nicht mehr mit der Destination verbunden ist, führt folgende Reaktion aus [siehe Bild 4.1]:

Im Zustand vor der Splittung soll u der nächste Host auf dem einzigen Pfad von t (und somit auch von s) nach d sein. Nach der Trennung der Verbindung löscht t den Eintrag für u. Nun besitzt er hingegen immer noch Einträge für Pfade zur Destination fr die übriggebliebenen Nachbarn a und c. Diese scheinen einen Pfad zu kennen. Dass dieser wiederum über den Host t selber führt, ist aus dem Eintrag in der Routing-Tabelle nicht ersichtlich. Es entsteht ein endloser Zyklus (Count-To-Infinity Problem), in dem sich die beiden Hosts a und t jeweils die Nachricht weitersenden in der Meinung, den besten Weg gefunden zu haben. Eine Lösungsmöglichkeit zu detektieren ist, dass im Routing-Tabelle-Eintrag der ganze Pfad gespeichert wird wie beim Path Vector Routing.

Die Wahl des Routing-Algorithmuss muss wohlüberlegt geschehen. Nichtrealistische Mobilitätsmodell erlauben es nicht, die wirkliche Performance eines Routing-Protokoll aufzuzeigen[2]. Dieses Problem wurde versucht zu umgehen, indem die einzelnen Eigenschaften eines Mobilitätsmodell orthogonalisiert und separiert wurden. Die nachfolgend erwähnten Eigenschaften können beliebig kombiniert werden. Es wird dabei dem Anwender die Aufgabe übertragen, einzuschätzen wie realistisch seine gebildete Kombination ist. Eventuell sind aber auch unrealistische Varianten gewünscht

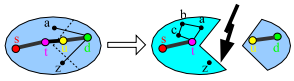

Abbildung 4.1: Count To Infinity Problem

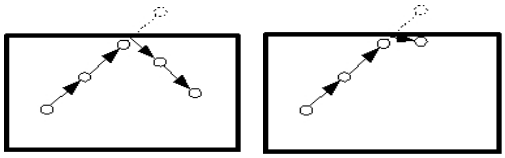

Abbildung 4.2: Abprallverhalten: links für Velocity Model, rechts für alle anderen

wie für Worst-Case Analysen. Vielleicht ist dies eine Erklärung dafür, dass gemäss [2] die Mehrheit der existierenden Mobilitätsmodelle keine realistischen Bewegungs-Szenarien repräsentieren. In der Regel sind sie beschränkt auf Random Walks ohne Betrachtung von Hindernissen [2]. Folgende Eigenschaften bilden für die unterstützten Mobilitätsmodelle eine Grobunterteilung:

- Methode um den Geschwindigkeitsvektor zu berechen
- Gruppenbildung

Diese beiden Eigenschaften werden noch feiner unterteilt. Die erste beschäftigt sich mit der Art – eine mathematische Formel oder eine andere Regel – der Bestimmung des Geschwindigkeitsvektors. Der Geschwindigkeitsvektor ist derjenige Vektor, um welchen sich ein Host in einer Zeiteinheit verschieben kann. Hier sind folgende orthogonale Merkmale zu nennen:

- Normalisierung des Geschwindigkeitsvektors
- Abprallverhalten an den Grenzen des Netzwerks

Das Verhalten am Netzwerk-Rand kann zwei Ausprägungen haben. Entweder wird der Geschwindigkeitsvektor so angepasst, dass der Aufprallswinkel gleich dem Ausfallswinkel ist (Prinzip Billardkugel). Dies wird im Falle von `Velocity Model` ist in Bild 4.2 links ersichtlich. Dabei wird der Geschwindigkeitsvektor neu bestimmt und gilt bis zum nächsten Aufprall an einer Netzwerk-Grenze. Alle anderen Mobilitätsmodelle verwenden das Entlanglaufen an der Grenze. Die zweite Variante [Bild 4.2] ist für Velocity Model nicht anwendbar. Ein Host würde irgendwann an eine Grenze stossen und dann entlang dieser in einer Ecke eintreffen. Nach kurzer Zeit würden sich alle Hosts in den Ecken des Netzwerks befinden. Alle anderen Mobilitätsmodelle hingegen verwenden keinen konstanten Geschwindigkeitsvektor, sondern er wird jedesmal wieder zufällig gewählt. Somit ist die Wahrscheinlichkeit gross, dass sie sich nach einiger Zeit wieder von der Wand wegbewegen.

Eine Gruppe kann auf folgende Art geschehen:

- keine Gruppenbildung
- zufällige Einteilung in Gruppen
  - beliebig viele Gruppen
  - Begrenzung der maximalen Anzahl Gruppen
- Einteilung aufgrund der maximalen Distanz
  - zwischen zwei Hosts paarweise
  - relative zu einem der anderen Gruppenmitglieder (hier kann je nach Aufbau schlussendlich nur eine Gruppe vorhanden sein)

Geschwindigkeitsvektor Bestimmung	**Gruppierung**	**Steuerungs-Parameter**	**Beispiel**
MaxXLinksChanged Änderung von max. X Links erlaubt	keine	X	kein Beispiel
	zufällig unbegrenzt	X	kein Beispiel
	zufällig begrenzt	X Anzahl Gruppen	kein Beispiel
	maximale Distanz paarweise	X maximale Distanz	kein Beispiel
	maximale Distanz paarweise	X maximale Distanz	kein Beispiel
MaxXNodesMoves Bewegung von max. X Hosts erlaubt	keine	X	kein Beispiel
	zufällig unbegrenzt	X	kein Beispiel
	zufällig begrenzt	X Anzahl Gruppen	kein Beispiel
	maximale Distanz paarweise	X maximale Distanz	kein Beispiel
	maximale Distanz paarweise	X maximale Distanz	kein Beispiel
Random jeder Host bewegt sich um einen zufällig bestimmten Geschwindigkeitsvektor	keine	keine	kein Beispiel
	zufällig unbegrenzt	keine	kein Beispiel
	zufällig begrenzt	Anzahl Gruppen	kein Beispiel
	max. Distanz paarweise	maximale Distanz	kein Beispiel
	max. Distanz paarweise	maximale Distanz	kein Beispiel
Random Way Point	keine	keine	kein Beispiel
	zufällig unbegrenzt	keine	kein Beispiel
	zufällig begrenzt	Anzahl Gruppen	kein Beispiel
	maximale Distanz paarweise	maximale Distanz	kein Beispiel
	maximale Distanz paarweise	maximale Distanz	kein Beispiel

Geschwindigkeitsvektor Bestimmung	**Gruppierung**	**Steuerungs-Parameter**	**Beispiel**
RMSM (Revised Minimum Standard Model)	keine	Wahrscheinlichkeit keiner Bewegung Wahrscheinlichkeit keiner Bewegung Epsilon	kein Beispiel
	zufällig unbegrenzt	Wahrscheinlichkeit keiner Bewegung Wahrscheinlichkeit keiner Bewegung Epsilon	kein Beispiel
	zufällig begrenzt	Wahrscheinlichkeit keiner Bewegung Wahrscheinlichkeit keiner Bewegung Epsilon Anzahl Gruppen	kein Beispiel
	maximale Distanz paarweise	Wahrscheinlichkeit keiner Bewegung Wahrscheinlichkeit keiner Bewegung Epsilon maximale Distanz	kein Beispiel
	maximale Distanz paarweise	Wahrscheinlichkeit keiner Bewegung Wahrscheinlichkeit keiner Bewegung Epsilon maximale Distanz	kein Beispiel
Velocity jeder Host / Gruppe erhält einen zufällig initierten, fixen Geschwindigkeitsvektor zugeordnet. Dieser kann sich nur ändern, wenn der Host auf den Rand des Netzwerks auftritt.	keine	keine	kein Beispiel
	zufällig unbegrenzt	keine	kein Beispiel
	zufällig begrenzt	Anzahl Gruppen	kein Beispiel
	maximale Distanz paarweise	maximale Distanz	kein Beispiel
	maximale Distanz paarweise	maximale Distanz	kein Beispiel

Kapitel 5

Routing-Algorithmus

Die Aufgabe eines Routing-Protokoll s ist die Ermöglichung der Kommunikation zwischen zwei beliebigen Hosts im Netzwerk. Routing-Algorithmen bestimmen über welchen nächsten Hosteine Nachricht an die Destination weitergeleitet wird. Der Routing-Algorithmus berücksichtigt dabei den nach seinem aktuellen Wissen besten (in der Regel kürzesten) Pfad. Dazu existiert eine Vielzahl an Routing-Algorithmen – mehr als 50 gemäss [4]. Das Routing-Protokoll definiert folgende Aufgaben:

Bestimmung von Kommunikationswegen (z.B. Aufbau einer Distanztabelle für kürzeste-Wege-Routing)

Datentransport entlang dieser Wege (z.B. First-In-First-Out, Longest-In-System, Last- In-Last-Out, Oldest-Packet-First). Dieser Part ist nicht Teil der Betrachtung der Arbeit.

Die Klassifizierung von Routing-Protokollen wird aufgrund der Art der Informationsbeschaffung vorgenommen. Die dabei verwendeten Routing Informationen knnen mittels zwei grundlegenden Techniken oder einer Mischform davon ermittelt werden.

- **proaktive (precomputed / table driven) Routingprotokolle** Sie zeichnen sich durch eine kontinuierliche Informationsgewinnung durch automatische Aktualisierungsverfahren [[6]] erfolgt. Die Routing Informationen werden im voraus gesammelt und somit ist jederzeit ein Pfad zur Verfügung - es entsteht in statischen Netzwerke keine Latency und in dynamischen ein Stretch. Dies ist aber mit einem kostspieligen Aufwand verbunden. Auf der einen Seite müssen die Routing Informationen gespeichert werden und auf der anderen Seite verbrauchen die Update-Nachrichten auch viel Bandbreite. In diesem Zusammenhang wird beim Senden v. a. in Wireless Umgebungen auch viel Energie benötigt. Proaktive Protokolle sind für stabile Netzwerke effizienter und werden oft in Festnetzen verwendet[]. Hingegen fallen durch ständige Updates in Netzwerke mit starker Mobilität viel Overhead an, insbesondere auch für Pfade, die gar nicht verwendet werden.
- **reaktive (on demand) Routingprotokolle** In diesem Fall existiert in der Regel kein Pfad– wenn doch, dann wird dieser genommen. Dieser muss sonst zuerst bestimmt werden (Route Discovery). Fällt während der Übertragung ein Link aus, so wird dies entdeckt und behoben. Dies wird auch Route Maintenance genannt. Dieses Verfahren benötigt wenig Overhead und dieser ist proportional zu

den Anzahl verwendeter Pfade. Insbesondere für Netzwerke mit hoher Mobilität geeignet, weil hier das Halten von meist veralteter Routing Informationen sich nicht lohnt. Der grössere Aufwand von proaktiven Protokollen liegt darin, dass sie die Routing Informationen fr das gesamte Netz (Global State Information) speichern, wohingegen reaktive Verfahren meist nur die der Nachbarn (Local State Information) speichern.

- **hybride Routingprotokolle** Hybride Protokolle versuchen die guten Eigenschaften von proaktiven und reaktiven Protokollen zu vereinen in einer Lösung.
- **hierarchische Routingprotokolle**

Als eine weitere, wenn auch relativ primitive Form nennt [1] auch Flooding. Zou [6] führt in seinem Paper eine noch detailliertere Unterteilung ein.

- precomputed vs. on demand Routingprotokolle
- periodische Updates vs. Event-driven Updates
- flach oder hierarchisch strukturierte Routingprotokolle
- source routing vs. hop-by-hop Routingprotokolle
- single Pfad vs. multiple Pfade
- hybride Routingprotokolle
- hierarchische Routingprotokolle

Als Metrik für die Güte eines Routing-Algorithmus können verschiedene Faktoren aufgezäehlt werden, welche sich teilweise gegenseitig widersprechen (Trade-Off):

- Minimale Anzahl Hops
- Minimaler Delay
- Minimale Paketverluste
- Minimale Stausituationen (Load Balancing)
- Minimale Interferenzen
- Maximale Signalstabilitt und zeitlich stabile Route
- Maximale Batterielaufzeit eines mobilen Gerätes
- Maximale Lebenszeit des gesamten Netzes
- Partitionierung durch Batterielaufzeit-bedingten Ausfall von Knoten
- …

Folgende zwei Beispiele sollen den Trade-Off verdeutlichen:

- Um die Destination schnellstmöglichst zu erreichen, wird die Weglänge minimiert. Benutzen die anderen Hosts ebenfalls dieselben Routing-Kriterien, werden die Nachrichten auf wenige Kanle konzentriert. Dies widerspricht der Vermeidung bzw. Minimierung von Stausituationen durch Load Balancing)

Protokoll	**Speicherbedarf**	**Zeit**	**Kontrollpaketgrösse**	**Kommunikationsaufwand**
LSR	O(N x A)	O(D)	O(A)	O(N)
DVR	O(N)	O(D)	O(N)	O(N)
DSDV	O(N)	O(D)	O(N)	O(N)
GSR	O(N x A)	O(D)	O(N)	O(N)
CGSR	O(2N)	O(D)	O(N)	O(N)
WRP	O(N x A)	O(D)	O(N + A)	O(N)
DSR	O(D)	O(2D)	O(D)	O(2N)
AODV	O(D_d)	O(2D)	O(D_d)	O(2N)
TORA	O($D_d x A$)*	O(2D)	O(1)	O(2N)
DST	O(D_d x A)	O(2D)	O(D_d)	O(2N)
ABR	O(D + A)	O(D + Z)	O(D)	O(N + Y)
SSA	O(D + A)	O(D + Z)	O(1)	O(N + Y)
HSR	O(M x $log_M N$)	O(D)	O(M)	O(N)

Tabelle 5.1: Komplexitäet von typischen Routing-Algorithmen

- Um den Energiebedarf minim zu halten, soll ein Host sich selber kurz abschalten können, wie es das ALOHA Prinzip vorsieht. Zu gewissen Zeitpunkten ist ein Host aktiv und kann senden und empfangen, andernfalls führt dies zu Paketverlusten.

Protokoll	**Route berechnen**	**Struktur**	**# Routen**	**Source Routing**	**RRM**	**BR**
LSR	proaktiv / selbst	flach	single oder multiple	nein, ja möglich	N/A	nein
DVR	proaktiv / verteilt	flach	single	nein	N/A	nein
DSDV	proaktiv / verteilt	flach	single	nein	N/A	nein
GSR	proaktiv / verteilt	flach	single oder multiple	nein, ja möglich	N/A	nein
CGSR	proaktiv / verteilt	hierarchisch	single	nein	N/A	nein
WRP	proaktiv / verteilt	flach	single		N/A	ja
DSR	reaktiv / broadcast Query	flach	multiple	ja	löscht Route, notify Source	nein
AODV	reaktiv / broadcast Query	flach	multiple	nein	löscht Route, notify Source	ja
TORA	reaktiv / broadcast Query	flach	multiple (DAG)	nein	Link Reversal, Route repair	nein
DST	reaktiv / broadcast Query	flach	single, multiple möglich	nein, ja möglich	Route repair	nein
ABR	reaktiv / broadcast Query	flach	single	ja	lokale Broadcast Query	ja
SSA	reaktiv / broadcast Query	flach	single	nein	löscht Route, notify Source	ja
HSR	proaktiv / reaktiv (hier. addr.)	hierarchisch	single	nein	N/A	nein

Tabelle 5.2: Vergleich typischer Routing Protokolle (1)

Protokoll	**gespeicherte Informationen**	**Update Periode**	**Update Informationen**	**Update Destination**	**Methode**
LSR	ganze Netzwerk-Topologie	hybrid	Link-Status des Nachbarn	alle Hosts	Flooding
DVR	Distanzvektor	periodisch	Distanzvektor	Nachbarn	Broadcast
DSDV	Distanzvektor	hybrid	Distanzvektor	Nachbarn	Broadcast
GSR	vollständige Netzwerk-Topologie	periodisch	Link-Status aller Hosts	Nachbarn	Broadcast
CGSR	Cluster Mem.Table, Dist. Vect.	periodisch	Cluster Mem.Table, Dist. Vect.	Nachbarn & Cluster-Head	Broadcast
WRP	Dist. / routing /link-cost Table, MRL	hybrid	Dist. Vect., List of Resp.	Nachbarn	Broadcast
DSR	Routen zu gewünschter Destination	Event-driven RM	ROUTE_ERROR	Source	Unicast
AODV	nächster Hop für gewünschte Destination	Event-driven RM	ROUTE_ERROR	Source	Unicast
TORA	Höhen der Nachbarn	Event-driven	Host-Höhe	Nachbarn	Broadcast
DST	Distance / routing / query Table	Event-driven	Routing-Tabelle	Nachbarn	Broadcast
ABR	?	periodisch / Event-driven RRC	?	Nachbarn / Source	Broadcast / Unicast
SSA	Signal-Stabilität / Routing-Tabelle	periodisch / Event-driven RRC	ROUTE_ERROR	Source	Unicast
HSR	hierarchische Topologie	periodisch / Event-driven	virtueller Link-Status/.	Hosts in Cluster	Broadcast

Tabelle 5.3: Vergleich typischer Routing Protokolle (2)

Kapitel 6

Messungen

6.1 Einleitung

Um die verschiedenen Routing-Algorithmen für ein bestimmtes Netzwerk miteinander vergleichen zu können, wurden folgende verschiedene Parameters gemessen:

- Berechnungszeit (in Sekunden)
- Anzahl Nachrichten, die Destination erreicht haben
- # benötigter Runden
- # gesendeter Nachrichten pro Runde und insgesamt
- # gesendetes Update-Nachrichten pro Runde und insgesamt
- Pfad-Länge
- Destination erreicht
- # hinzukommender, wegfallender Links pro Runde und insgesamt
- # der Bewegungen von Hosts pro Runde und insgesamt
- ...

Die Routing-Algorithmen wurden auf verschiedenen Netzwerke simuliert. Ein Netzwerk kann durch folgende Parameter generiert und variert werden:

- statisch oder mobiles Netzwerk
- Dichte
- Dimension
- Mobilitätsmodell

Besonders entscheidend ist dabei das verwendete Mobilitätsmodell, welches durch unterschiedliche Parameter gesteuert werden kann. Diese wurden genauer im vorhergehenden Kapitel 4 erläutert.

Falls nicht anders erwähnt wird von der folgenden Anfangskonfiguration ausgegangen:

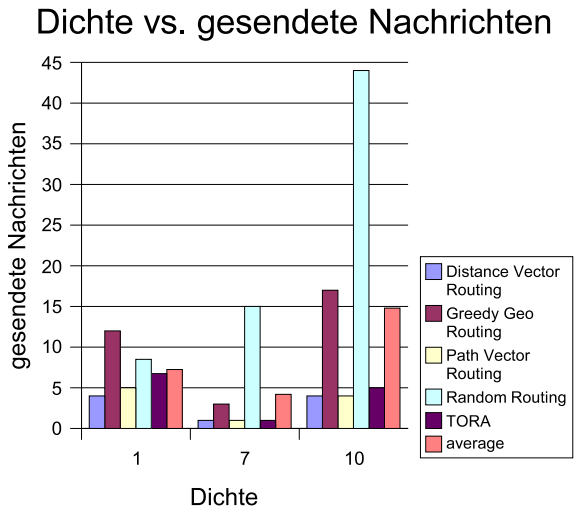

Abbildung 6.1: durchschnittliche Anzahl Nachrichten bei gegebener Dichte

- Dimension des Netzwerks: 10 x 10
- Host-Dichte: 10
- Mobilitätsmodell mit normiertem Bewegungsvektor = 1
- maximale # Runden, bevor Suche abgebrochen wird: n
- # Routings pro Runde: 1
- # Movings pro Runde: 1
- # Updates pro Runde: 1
- # maximaler Fehlversuche von Nachrichten senden, bevor abgebrochen wird: 20
- # gesendeter Routing-Nachrichten(Requests): 30

Mittels `Simulation` kann jeweils ein Parameter der Eingabe variert werden. So wurden auch die folgenden Resultate erhalten. Die ganze Simulation wurde jeweils für 10 Referenz-Netzwerke erstellt. Aus Zeitgründen wurde eine kleine Zahl gewählt, wodurch die Aussagekraft der Ergebnisse geschmälert wird.

6.2 Variation der Dichte

Hier wurde von Grundkonfiguration ausgegangen, wobei die Dichte folgende Werte einnahm: (1, 7, 10). Folgende Grafiken demonstrieren die Ergebnisse:

In Graphik 6.1 ist erstaunlicherweise ersichtlich, dass die Anzahl gesendeter Nachrichten bei kleiner Dichte bei allen Routing-Algorithmus ähnlich gross ist. Mit steigender Dichte nimmt die Anzahl der Nachrichten ab. Während alle Routing-Algorithmus eigentlich also besser werden, verschlechtert sich Random Routing. Doch dies lässt sich mit der Wahrscheinlichkeitsrechnung leicht nachvollziehen. Sei ein beliebiger

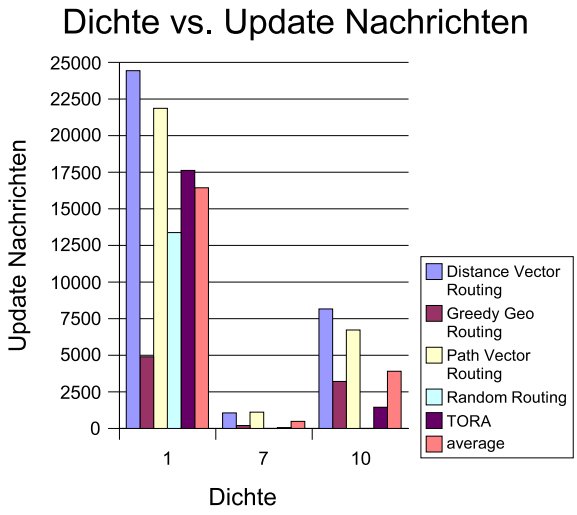

Abbildung 6.2: durchschnittliche Anzahl Update-Nachrichten bei gegebener Dichte

Graph gegeben. Dann nimmt Random Routing mit Wahrscheinlichkeit 1/2 die falsche Richtung und muss diese Entscheidung korrigieren, indem er mindestens den gleichen Weg wieder zurücknehmen muss. Dies tritt bei Greedy Geo Routing in kleinerem Rahmen auch auf.

Die Grafik 6.2 ist etwas ungeschickt gebildet und mag den Anschein erwecken, dass bei kleiner Dichte viele Update-Nachrichten versendet werden. Dies ist mit Sicherheit nicht der Fall, da entsprechend weniger Verbindungen vorhanden sind. Hingegen ist die Zeit, bis die Nachricht die Destination erreicht bei kleiner Dichte gross, sodass für das Versenden der Nachricht lange gebraucht wird, weil unter Umständen keine Verbindung existiert.

In Grafik 6.3 wird ersichtlich, dass bei geringer Dichte in der Regel ähnlich gross oder kleiner ist. Dies ist für mich nicht offensichtlich klar, schliesslich kann bei grosser Dichte aus mehr Hosts ausgesucht werden, von denen sicher einer sehr gut approximiert. Vielleicht ist die Pfad-Länge kürzer, weil bei kleiner Dichte gar keine Sendemöglichkeit besteht und deshalb gewartet werden muss.

In Grafik 6.4 wird wiederum ersichtlich, weshalb Random Routing schlecht abschneidet. Er kommt meist gar nicht am Ziel an. Distance & Path Vector Routing funktionieren sehr zuverlässig. Sie sind aber abhängig von der aktuellen Routing-Tabelle. Dies ist wohl auch der Grund, weshalb Greedy Geo Routing am besten abschneidet. Der Host kann nämlich die Qualität eines Pfades selbst bestimmen, vorausgesetzt er weiss die Position der Destination ziemlich genau. Ebenso kommt der reaktive Routing-Algorithmus TORA an die Qualität von sehr gut an.

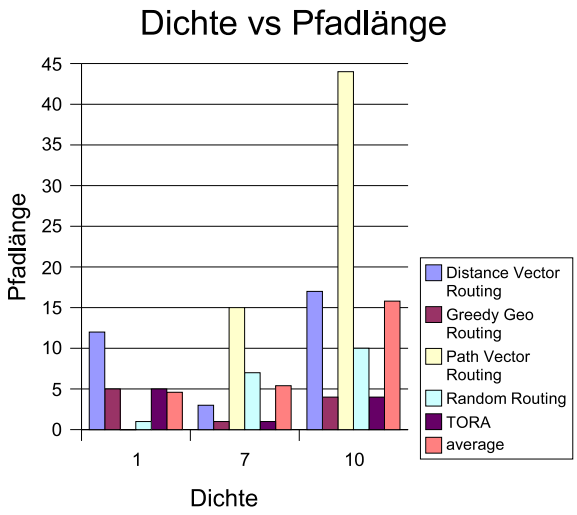

Abbildung 6.3: durchschnittliche Pfad-Länge bei gegebener Dichte

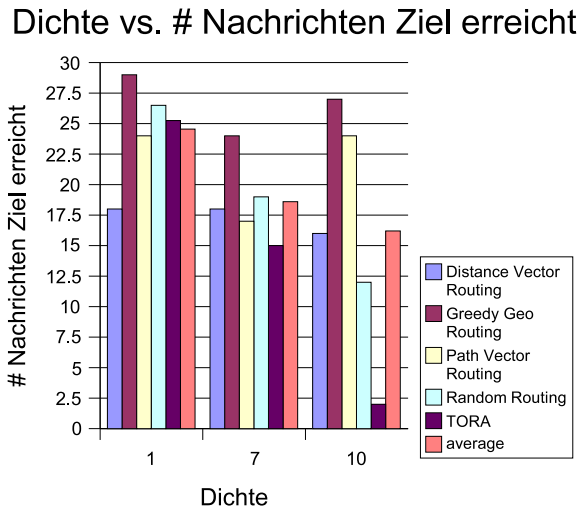

Abbildung 6.4: durchschnittliche Anzahl Nachrichten, die Ziel erreichen bei gegebener Dichte

6.3 Fazit

Mit den Messungen und den bisher ausgewerteten Daten kann keine Aussage gemacht werden, ob ein bestimmter Routing-Algorithmus auf einem Netzwerk mit einem bestimmten Mobilitätsmodell besser geeignet ist. Greedy Geo Routing ist bei fester oder nur sich leicht verändernder Position der Destination am besten.

Kapitel 7

Fazit

Aufgrund falscher Einschätzung der benötigten Zeit für eine ausführliche Analyse der Messungen, konnten leider nicht viele Resultate aufgezeigt werden. Die Arbeit hat sich deshalb vor allem um die theoretische Aufarbeitung und die Bildung von Modellen auseinandergesetzt. Diese konnten erfolgreich umgesetzt werden, jedoch nicht in aller Ausführlichkeit getestet werden.

Dennoch lassen sich durch die visuelle Darstellung mit den zahlreichen Zusatzinformationen erste Eindrücke und Ideen sammeln, welche Messungen interessant sein können. Dazu stehen viele varierbare Parameter zur Verfügung. Auch eine Erweiterung um zusätzliche Graphen ist ermöglicht worden.

Ich konnte mit der Arbeit einen tieferen Einblick erhalten, in eine der grundlegendsten Technologien, die in der Informatik gebraucht werden, um erst das Versenden von Daten zwischen verschiedenen Hosts zu ermöglichen. Ich möchte mich hier bedanken bei Prof. Wattenhofer, unter deren Leitung die ganze Arbeit gestanden hat. Genauso aber möchte ich mich bei Frau O'Dell bedanken, welche sich in einer "Ad-Hoc" Übung einverstanden erklärt hat, mir ein offene formulierte Aufgabe zu offerieren. Vielen Dank für die Unterstützung.

Literaturverzeichnis

[1] Mobile Ad-hoc-Netze. 2003.

[2] K. C. A. S. S. Amit Jardosh, Elizabeth M. BeldingRoyer. The Obstacle Mobility Model Project. September 2003.

[3] P. J. A. N. G. R. Cdric Adjih, Graud Allard. Network traffic and architecture models.

[4] C. S. und Klaus Volbert. Algorithmische Probleme in Funknetzwerken. 2002/03.

[5] M. Weber. *Verteilte Systeme*. Spektrum Akademischer Verlag.

[6] B. R. u. S. M. Xukai Zou. Routing Techniques in Wireless Ad Hoc Networks – Classification and Comparison. Juli 2002.