Institut für Parallele und Verteilte Systeme

Abteilung verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D – 70569 Stuttgart

Studienarbeit Nr. 2025

Untersuchung von Verfahren zur Partitionserkennung in mobilen ad-hoc Netzen

Gerald Georg Koch

Studiengang:	Informatik (Diplom)
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Rothermel
Betreuer:	Jörg Hähner, Gregor Schiele
begonnen am:	01.06.2005
beendet am:	01.12.2005
CR-Klassifikation:	C.2.2

Inhaltsverzeichnis

Abbildungsverzeichnis....ii
Tabellenverzeichnis....ii

Einleitung: Partitionserkennung in mobilen ad-hoc Netzen....1
Kapitel 1: Über diese Arbeit....3
Abschnitt 1.1 – Rahmen der Arbeit....3
Abschnitt 1.2 – Systemmodell....4
Abschnitt 1.3 – Verwandte Arbeiten....5
Kapitel 2: Designentscheidungen bei Partitionserkennungsalgorithmen....7
Abschnitt 2.1 – Verteilung des Algorithmus....7
Abschnitt 2.2 – Implementierung des Algorithmus....8
Abschnitt 2.3 – Zeitablauf- oder Ereignisbindung....8
Abschnitt 2.4 – Schnittstellen....9
Kapitel 3: Ausgewählte Algorithmen....11
Abschnitt 3.1 – Das k-hop Nachbarschaftsmodell....12
Abschnitt 3.2 – Ausarbeitung des Algorithmus....12
Abschnitt 3.3 – Analyse....15
Abschnitt 3.4 - Optimierung....16
Abschnitt 3.5 – Bewertung....17
Kapitel 4: Das Leuchtturmmodell....19
Abschnitt 4.1 – Beschreibung der Idee....19
Abschnitt 4.2 – Ausarbeitung des Algorithmus....20
Kapitel 5: Implementierung....31
Abschnitt 5.1 – Entscheidungen....31
Abschnitt 5.2 – Detaillierte Beschreibung....33
Abschnitt 5.3 – Analyse....44
Abschnitt 5.4 – Optimierung....46
Abschnitt 5.5 – Bewertung....47
Kapitel 6: Simulation....49
Abschnitt 6.1 – Simulationsumgebung....49
Abschnitt 6.2 – Simulationsergebnisse....50
Kapitel 7: Abschluss und Ausblick....59

Literaturverzeichnis....iii

Abbildungsverzeichnis

Abschnitt 4.2 – Ausarbeitung des Algorithmus
Abbildung 1: Zustandsübergangsdiagramm (die Übergänge nach Z sind ausgelassen).....26
Abschnitt 5.1 – Entscheidungen
Abbildung 2: Struktur.....31
Abschnitt 6.2 – Simulationsergebnisse
Abbildungen 3, 4: Verzögerung der Neuwahl nach Aufeinandertreffen der Partitionen.....51
Abbildungen 5, 6: Dauer bis zur Übereinstimmung mit der berechneten Partition.....52
Abbildungen 7, 8: Wann sich ein ausgeschiedener Knoten zum Leuchtturm erklärt.....53
Abbildungen 9, 10: Dauer bis zur vollständigen Trennung der Partitionen.....54
Abbildungen 11, 12: Mittlere Genauigkeit.....55
Abbildungen 13 - 15: Dauer der Übereinstimmung.....56
Abbildungen 16, 17: Durchschnittlich gesendete Protokollnachrichten.....57
Abbildungen 18, 19: Anzahl der pro Knoten weitergeleiteten Nachrichten.....58

Tabellenverzeichnis

Abschnitt 3.2 – Ausarbeitung des Algorithmus
Tabelle 1: Bezeichnungen im k-hop Nachbarschaftsmodell.....14
Abschnitt 3.3 – Analyse
Tabelle 2: Bezeichnervereinbarungen für Analyse.....15
Abschnitt 4.2 – Ausarbeitung des Algorithmus
Tabelle 3: Bezeichnungen im Leuchtturmmodell.....23 - 24
Tabelle 4: Zustände, Ereignisse und Zustandsübergänge.....25
Tabelle 5: Zustände, Bedingungen, Prozeduren und Übergänge.....27 – 28
Abschnitt 5.2 – Detaillierte Beschreibung
Tabelle 6: Notation.....33
Abschnitt 5.3 – Analyse
Tabelle 7: Bezeichnervereinbarungen für Analyse.....44

Einleitung

Partitionserkennung in mobilen ad-hoc Netzen

Bei mobilen ad-hoc Netzen handelt es sich um spontan verbundene Netze, an denen mobile Geräte beteiligt sind. Ad-hoc Netze besitzen die Eigenschaft, dass sich eine konkrete Ansammlung mobiler und stationärer Geräte weitestgehend selbst konfiguriert und zum Aufbau und Betrieb des Netzes keiner weiteren besonderen Eingriffe durch die Benutzer bedarf. Der Mobilität geschuldet muss jederzeit mit der Neuaufnahme und dem Ausscheiden von Knoten aus dem Netz gerechnet werden.

Die Erfahrungen, die mit dem Betrieb von Protokollen und Anwendungen in festen Netzen gesammelt wurden, lassen sich auf mobile und auf ad-hoc Netze nicht einfach übertragen. Kommunikationsprotokolle müssen für eine zuverlässige Nachrichtenübertragung mit geringen Latenzzeiten sorgen, nun aber in einer Umgebung, in der Verbindungen zwischen konkreten Geräten nur zeitweise bestehen und unvorhergesehen beginnen oder enden. Nutzen mobile Netze die Luft als Medium der Datenübertragung, ist die Datenübertragung um Größenordnungen störanfälliger als über Kabel, und eine naive Behandlung und Vorbeugung fehlerhafter oder fehlschlagender Übermittlungen zieht ein erhöhtes Nachrichtenaufkommen nach sich, was im Hinblick auf das gemeinsam genutzte Medium zu einem geringeren Datendurchsatz führt. Eine völlig neue Größe ist der beschränkte Energievorrat mobiler Geräte.

Mobile ad-hoc Netze müssen der Mobilität der Teilnehmer in besonderem Maße gerecht werden. Es sind Algorithmen erforderlich, neu erscheinende Knoten oder Mengen von Knoten rasch in ein Teilnetz zu integrieren und bei der Trennung von Netzen die Stabilität der Kommunikation in beiden Restnetzen zu gewährleisten. Bei lokalem Verlust einer Verbindung müssen Anstrengungen unternommen werden, die betreffenden Knoten auf anderem Weg zu erreichen; bei vollständigem Verlust der Kommunikationsverbindung zwischen zwei Teilnetzen – der Bildung einer neuen Partition also – sollen fehlschlagende Kommunikationsversuche dagegen so rasch wie möglich aufgegeben werden.

Anwendungen in mobilen ad-hoc Netzen kann daher nicht dieselbe Dienstgüte zur Verfügung gestellt werden, wie sie in festen Netzen erreicht wird. Bei Partitionierung kann z.B. ein verteilter Algorithmus getrennt werden, und es ist Sache der Anwendung, darauf angemessen zu reagieren. Demgegenüber müssen der Anwendung vom Netz her geeignete Informationen zur Verfügung gestellt werden, die eine solche Reaktion ermöglichen und unterstützen. Ein solcher Netzwerkdienst ist die Partitionserkennung.

Der übrige Teil dieser Arbeit ist wie folgt gegliedert.

In Kapitel 1 wird der Bereich der Partitionserkennung beschrieben, den die Arbeit abdecken soll, und mit einem Systemmodell die Grundannahmen festgelegt. Es folgt eine Abgrenzung zu anderen Arbeiten auf dem Gebiet der Partitionserkennung. In Kapitel 2 wird betrachtet, welche Entscheidungen Einfluss auf das Design des zu entwerfenden Protokolls haben können. Das Kapitel enthält außerdem eine Schnittstellenbeschreibung für die Partitionserkennungsprotokolle, die in dieser Arbeit betrachtet werden.

Kapitel 3 beschreibt das k-hop Nachbarschaftsmodell und Kapitel 4 das Leuchtturmmodell als Beispiele für Partitionserkennungsalgorithmen. Beide Algorithmen werden nach Aufwand analysiert und auf mögliche Optimierungen hin untersucht.

Das Leuchtturmmodell wurde im Rahmen vorliegender Arbeit für den Netzwerk-Simulator ns2 implementiert, deshalb gibt es ein extra Kapitel 5 mit detaillierten Entscheidungen und Kurzbegründungen, die dem Algorithmus zugrunde liegen. Die Ergebnisse der Simulation finden sich in Kapitel 6.

Kapitel 7 enthält die abschließenden Gedanken und einen Ausblick auf mögliche weitere Arbeit am vorgestellten Thema.

Kapitel 1

Über diese Arbeit

Abschnitt 1.1 – Rahmen der Arbeit

Partitionserkennung bedeutet zunächst die Entscheidung, ob eine Menge von Knoten aus mindestens zwei disjunkten Teilmengen besteht[1]. In zweiter Stufe soll sie erkennen, welche Elemente der gesamten Knotenmenge welchen Teilmengen zugeordnet sind. Sie benötigt also eine Sicht auf das gesamte Netz ungeachtet seiner Partitionierung.

Man könnte die Gesamtheit aller mobilen Knoten, die über ein gemeinsames Protokoll miteinander Funkverbindungen aufnehmen können, als „das" mobile Netz betrachten. Es ist wahrscheinlich unmöglich, die globale Sicht auf dieses Netz herzustellen, denn das globale mobile Netz ist in viele Netze unterteilt, die sich in absehbarer Zeit nicht zu einem gesamten Netz verbinden werden, wie das beim Internet der Fall ist. Natürlich gibt es auch viele weitere vom Internet getrennte feste Netze. Doch die Trennungsentscheidung ist meist bewusst getroffen und stabil. In mobilen Netzen dagegen ist durch den Aufbau von städtischen Funknetzen und Satellitenübertragung – und natürlich durch die Mobiltät – eine Konvergenz der Netze zu erwarten. Deshalb greift es zu kurz, sich bei der Partitionserkennung auf eine Teilmenge der mobilen Knoten zu beschränken, um eine globale Sicht herstellen zu können.

1 In der Mengenlehre beschreibt das Wort „Partition" eine mögliche Einteilung der Gesamtmenge in disjunkte Teilmengen so, dass die Vereinigung dieser Teilmengen die Gesamtmenge bildet. Im Folgenden wird das Wort „Partition" aber – wie bei der Beschreibung von Netzen allgemein üblich – für die Teilmengen selbst verwendet.

Dennoch, aufgrund der Mobilität, der Funkverbindung und weiterer Aspekte des Verhaltes mobiler Knoten wird kein völlig und dauerhaft zusammenhängendes mobiles Netz zustande kommen können. Wie groß das globale mobile Netz auch sein mag, es kann immer partitioniert sein, und die Partitionierung kann sich rasch und häufig ändern[2]. Es ist realistischerweise auch nicht zu erwarten, dass mobilen Knoten als Ersatz ein zweiter Kanal zur Verfügung stehen wird, über den sie eine Gesamtsicht auf die Partition erhalten, aber nicht kommunizieren können[3]. Mobile Anwendungen oder Algorithmen, die die Gesamtsicht auf ein partitioniertes Netzwerk benötigen, werden sich daher schwerlich durchsetzen können.

Die vorliegende Arbeit konzentriert sich demzufolge auf Algorithmen, die sich in der Partitionserkennung auf die Teilmenge beschränken, in der sich der Knoten befindet, auf dem der Partitionserkennungsalgorithmus ausgeführt wird. Diese Algorithmen werden eine Schnittstelle zur Beantwortung der wichtigsten Fragen bieten, die eine Anwendung haben kann: Mit wie vielen Knoten kann kommuniziert werden? Mit genau welchen? - Und davon abgeleitet: Gab es eine Veränderung der Partition, die die lokale Teilmenge betrifft? Mit welchen Knoten ist die Kommunikation abgebrochen, mit welchen neu aufgebaut worden? Wie groß ist die Partition im Durchschnitt, in der ich mich aufhalte? Wie oft ändert sich die lokale Teilmenge und wie stark?[4]

Die Arbeit ist außerdem eingeschränkt durch die Aufgabenstellung, zwei bestimmte Verfahren zur Erkennung von Netzpartitionen zu untersuchen.

Abschnitt 1.2 – Systemmodell

Für die Beschreibung und Bewertung der Partitionserkennungsalgorithmen ist es notwendig, einige Grundannahmen zu treffen.

Für das Netz soll die Annahme gelten, dass alle Knoten mobil sein können. Insbesondere soll nicht angenommen werden, dass stationäre Knoten existieren, die aufgrund ihrer Energieversorgung, geographischen Lage oder festen Topologie für die Übernahme besonderer Aufgaben geeignet wären. Bezüglich der Funkverbindung treffen wir die naheliegende Annahme, dass Knoten bidirektional kommunizieren können, wenn es überhaupt zur Kommunikation kommt. Zur Unterscheidung der Kommunikationspartner soll jeder einzelne Knoten eine global eindeutige, unveränderliche und nicht geheime Identifikationseinheit besitzen[5]. Weitere Annahmen darüber, welche konkreten Verbindungen zwischen Teilnetzen bestehen oder nicht bestehen können, stehen nicht zur Verfügung. Daher gibt es natürlich auch keine Aussage über die Dauer, ihre Häufigkeit oder Qualität konkreter Verbindungen zwischen zwei Knoten.

2 Einleuchtendes Beispiel ist die Vernetzung der Automobile, deren Gesamtmenge immer partitioniert sein wird, deren Elemente und Teilmengen aber alle erdenklichen Verbindungen miteinander eingehen können.

3 sonst wären die Partitionen ja nicht mehr getrennt

4 Zu einer Diskussion über die Relevanz dieser Fragen siehe [1].

5 z.B. eine IP-Adresse des IPv6 ([2]).

Als Aufgabe der Partitionserkennungsprotokolle legen wir fest, dass sie die Partition *erkennen* sollen, in der sie sich gerade befinden. Insbesondere versuchen sie nicht die Vorhersage und nicht um die Verhinderung einer weiteren Partitionierung. Für ersteres wären weitere Sensoren am Knoten nötig[6] (zum Beispiel GPS-Empfänger, Geschwindigkeitsmesser und Kompass), und für zweites ein Regelkreis aus Beobachtung, Verarbeitung, Entscheidung, Ausführung und Kontrolle, was über die Aufgabenstellung der bloßen Partitionserkennung weit hinausgeht. Außerdem sind derartige Überwachungen immer einseitig, da ihr Gegenstück – der Zusammenschluss von Teilmengen – ohne übergeordnete Sicht nicht vorhergesehen und verhindert oder herbeigeführt werden kann.

An die Protokolle wird die Anforderung gestellt, kurzzeitige und endgültige Ausfälle von Knoten oder Unterbrechungen von Verbindungen tolerieren zu können. Dagegen müssen sie nicht zwischen Partitionierung und Knotenausfällen unterscheiden, denn der Effekt ist derselbe: die Kommunikation mit Prozessen auf den betroffenen Knoten ist unterbrochen. Natürlich kann sie jederzeit wieder aufgenommen werden, wobei unbekannt ist, ob die Anwendungen sich in der Zwischenzeit neu initialisiert haben, angehalten wurden oder einfach weitergearbeitet haben[7].

Wir verlangen von den Protokollen nicht, dass sie byzantinische Fehler oder sabotierende Teilnehmer erkennen und ausschließen können.

Zuletzt soll noch erwähnt werden, dass die Anwendungen trotz allem zeitweilige oder geringfügige Inkonsistenzen zwischen dem zur Verfügung stehenden Wissen und der tatsächlichen Konfiguration tolerieren müssen.

Abschnitt 1.3 – Verwandte Arbeiten

Eine Reihe von Arbeiten beschäftigen sich mit dem Thema Partitionierung; bei den meisten geht es indes um die Vorhersage von Partitionierungen. In [5] wird die Rolle der Gruppenmobilität bei der Bildung von Partitionen besprochen, es wird aber nichts darüber gesagt, wie die für den vorgestellten Clusteringalgorithmus notwendige Sicht auf die Partition hergestellt wird; das in [6] vorgestellte Verfahren kommt ohne die globale Sicht aus, um vor Partitionierung zu warnen. Die Autoren von [7] beschreiben die Vermeidung von Partitionierung mithilfe eines Tiefensuche-Algorithmus. Dafür muss, wie sie bemerken, auf jedem Knoten die vollständige Verbindungsinformation vorliegen – das heißt, es muss eine Infrastruktur zur Sammlung und Verteilung dieser Information bestehen, und solche zu beschreiben ist die Aufgabe vorliegender Arbeit.

6 Eine Ausnahme bildet das Verfahren in [3].

7 In [4] wird ein Verfahren vorgestellt, bei dem Knoten ihr Ausscheiden anmelden. Dann kann z.B. an einem verteilten Algorithmus trotz zeitweiliger Trennung der Knoten weitergearbeitet werden. Derartige Verfahren fußen jedoch auf weitreichenden Annahmen, wie auch der, dass die Knoten irgendwann wieder in Verbindung treten werden. Diese Annahme können wir – ausgehend von obiger Beschreibung der Netzeigenschaften – nicht treffen. Außerdem ist eine derartige Behandlung von Partitionierung Sache der Anwendung. Sie mag dazu benutzt werden, um die Benachrichtigung des Partitionserkennungsdienstes über die Partitionierung „richtig“ zu interpretieren. Die Partitionserkennung muss gleichwohl die Partitionierung melden, denn ihre Aufgabe ist die Kommunikation über ihre Sicht auf die reale Konfiguration des Teilnetzes.

Die hier zu besprechenden Verfahren haben Ähnlichkeiten mit den in [9] beschriebenen. Dort werden Knoten mit vergleichsweise wenig Verbindungen als sogenannte „aktive Knoten“ oder Randknoten ausgewählt, weil von ihnen angenommen wird, dass sie am Rand der Teilmenge liegen und besonders sensibel auf Partitionsveränderungen reagieren werden. Bricht deren Kommunikation zu einem Server (im zentralisierten Fall) oder zu anderen Randknoten (im verteilten Fall) ab, wird Partitionierung angenommen. Dieses Verfahren ist gegen Knotenausfälle nicht unempfindlich, und so wird jedem aktiven Knoten noch ein Partner („Buddy“) zur Seite gestellt, der die Aktivität des aktiven Knotens überwacht und notfalls dessen Arbeit übernimmt, um eine Falschmeldung über Partitionierung zu unterbinden.

Das verteilte Verfahren, das hier als nächstes vorgestellt wird, ist zwar um einiges einfacher gehalten, aber dafür ist es gegen Knotenausfälle unempfindlich. Das Problem des Knotenausfalls habe ich im zentralisierten Verfahren auf andere Art und Weise gelöst, nämlich ohne dem Nachfolger (Buddy) zusätzliche Aufgaben zu übertragen und ohne zusätzliche Kommunikation zu erzeugen. Überdies hat das Verfahren aus [9] nicht den absoluten Überblick über die in der Partition enthaltenen Knoten. Zum Beispiel werden im Zentrum liegende Knoten mit vielen Verbindungen nicht unbedingt erfasst. Somit kann keine vollständige Sicht auf die reale Konfiguration des Netzes mitgeteilt werden. Dagegen wird seine Veränderung mit wenig Aufwand und rasch, aber basierend auf Annahmen und damit ebenfalls nicht vollständig kommuniziert. Die Unterschiede sind größtenteils der dedizierten Anwendung der Verfahren aus [9] geschuldet, nämlich dem Einsatz in mobilen verteilten Spielen. Die Beschreibungen vorliegender Arbeit sollten dagegen so allgemein wie möglich gehalten werden und universell anwendbar sein.

Kapitel 2

Designentscheidungen bei Partitionserkennungsalgorithmen

Abschnitt 2.1 – Verteilung des Algorithmus

Für die Partitionserkennung kommen zentrale, isolierte und verteilte Verfahren sowie einige Mischformen in Frage. Bei zentralen Verfahren hat ein Knoten den vollständigen Überblick über die Partition. Das kann gewährleistet werden, wenn sich z.B. alle anderen Knoten in Sendereichweite des zentralen Knotens befinden, oder wenn sich alle anderen Knoten vor ihrer Teilnahme beim zentralen Knoten an-, vor ihrem Ausscheiden wieder abmelden und in der Zwischenzeit unbedingt Mitglieder der Partition bleiben. Weder das eine noch das andere kann für beliebige mobile Netze verlangt oder billig eingehalten werden. Ein zentraler Knoten kann jedoch einen unvollständigen Überblick über die Partition bekommen, indem er mit einigen oder allen Knoten der Partition in Verbindung tritt und aus ihren Teilsichten eine Gesamtsicht konstruiert. Das erfordert, wie erwähnt, Nachrichtenaustausch und verlangt von allen Knoten, ein entsprechendes Protokoll zu fahren und das Verfahren nicht zu sabotieren. Außerdem sind vollständige Genauigkeit und Aktualität nicht gewährleistet.

Isolierte Verfahren kommen ohne besonderen Nachrichtenaustausch aus. Sie beobachten den Datenverkehr in Empfangsreichweite des Knotens und schließen aus mitgelesenen Informationen über die Teilnehmer an der Partition. Es gibt es keinerlei Garantie für Zuverlässigkeit, Aktualität und Vollständigkeit der Information, und die Ergebnisse des Verfahrens sind stark von Annahmen über Zeit und

Mobilität abhängig. Wegen der Unabhängigkeit von anderen Knoten gehört es aber zu den flexibelsten von allen.

Nachrichtenaustausch und ein gemeinsames Protokoll sind bei verteilten Verfahren dagegen unbedingte Voraussetzung. Die Knoten arbeiten in den Zyklen Informationsgewinnung – Informationsverbreitung. Sie tauschen ihr Weltwissen unter Nachbarn aus oder verbreiten ihr lokales Wissen in der ganzen Partition. Auch hier kann jeder Knoten die vollständige Partitionsinformation erhalten – im Vergleich zu zentralen Verfahren aber mit erheblicher Verzögerung und höherem Nachrichtenaufwand. Bei wenigen Veränderungen geht die Genauigkeit der Informationen mit der Zeit gegen 100%, starke Veränderungen lassen das Weltwissen aber sehr schnell veralten.

Abschnitt 2.2 – Implementierung des Algorithmus

Die Entscheidung für eines der Verfahren oder einer Kombination von Verfahren legt auch nahe, auf welcher Schicht ein Protokoll zur Partitionserkennung zu implementieren ist, falls es einen Protokollstapel gibt. Besonders isolierte Verfahren beruhen auf Nachbarschaftswissen und dem Screening von Paketen, was eine Implementierung direkt oberhalb der Sicherungsschicht nahelegt. Ein zentrales Verfahren, das Partitionsinformationen durch die Analyse der Sendeleistungen erstellt, wird bis auf die physikalische Übertragungsschicht durchgreifen müssen. Verteilte Verfahren und Mischformen mit zentralen Verfahren dagegen benutzen die Kommunikationsinfrastruktur und setzen viel Berechnung und Speicherung voraus. Sie sind daher eher als Anwendungen zu betrachten, die oberhalb oder eng verknüpft mit der Vermittlungsschicht implementiert werden.

Abschnitt 2.3 – Zeitablauf- oder Ereignisbindung

Eine weitere Designentscheidung betrifft die Anteile zeitablauf- und und ereignisgebundener Elemente der Algorithmen. Wenn Informationen über Veränderungen sofort verteilt werden, liegen sie bei anderen Knoten eher vor als von ihnen angefragte Information. Insbesondere bei großen Veränderungen in der Anzahl der Verbindungen zu anderen Knoten mag eine Warnung die Genauigkeit und Aktualität des verteilten Wissens erheblich steigern. Die Effizienz dieses Verfahrens hängt allerdings von seiner Kalibrierung ab. Einerseits werden Nachrichten nur dann verschickt, wenn es wirklich etwas zu vermelden gibt. Andererseits mag auch viel unerwünschte Information versendet werden, weil sie nur von marginalem Interesse für andere Knoten ist.

Zeitablaufgebundene Verfahren basieren meist auf regelmäßigen Abfragen, generieren einen relativ kontinuierlichen Datenverkehr und können durch Parameter gut reguliert werden. Die Gefahr von Nachrichtenstürmen[8] aufgrund von Veränderungen ist gering. Je nach Ausprägung der Mobilität innerhalb der Partition kann dieses Verfahren mehr Datenverkehr erzeugen, als notwendig wäre, und bietet im Durchschnitt eine weniger aktuelle Sicht auf das System.

8 Vgl. die Beschreibung des Phänomens in [9]

Zeit- und Ereignisbindung sind in Bezug auf Speicherplatz von Bedeutung. Ereignisgesteuert, also ungefragt, versendete Nachrichten müssen vom Empfänger gespeichert werden, bis er sie verarbeiten kann, während andererseits regelmäßig abgefragte Informationen auf der Senderseite gespeichert werden müssen, bis sie angefordert werden.

Insgesamt dürfte sich ein gemischter Ansatz für mobile ad-hoc Netze eignen, bei dem gewisse Daten lokal regelmäßig gesammelt werden, so dass sie auf Anfrage rasch zur Verfügung stehen oder in besonderen Fällen sofort verteilt werden. Eine unnötige Verteilung nicht erwünschter Informationen unterbleibt jedoch.

Abschnitt 2.4 – Schnittstellen

Jeder Knoten soll über die Schnittstelle des Protokolls folgende Informationen beziehen können:

a) Anzahl der Knoten in der Partition
b) Liste der Knotenadressen in der Partition
c) Liste der neu hinzugekommenen oder abgegangenen Knoten seit einer gewissen Zeit
d) Relative Veränderung der Partition seit einer gewissen Zeit

Dabei bestimmen die inhärenten Möglichkeiten des Protokolls, welche Teile lokal zu implementieren oder welche Daten lokal zu halten sind und welche Daten oder Dienste angefordert werden können

Der Zugriff auf die Daten soll jederzeit möglich sein. Jeder Knoten wird Anfragen von anderen Knoten in angemessener Zeit beantworten, er darf die Antwort nicht verweigern, wenn er sie geben kann. Knoten, die für eine längere Zeit nicht auf Protokollnachrichten reagieren, werden als abwesend von der Partition angesehen; das impliziert aber nicht den tatsächlichen Abbruch des Datenverkehrs zu diesen Knoten.

Wenn Knoten ungefragt Informationen versenden, soll dies als Antwort ohne vorhergehende Anfrage aufgefasst werden. Knoten, die eine Antwort senden, dürfen nicht erwarten, dass andere Knoten die versendete Information verarbeiten werden, in dem Sinne, dass sie ihre Sicht auf das Netzwerk anpassen.

Kapitel 3

Ausgewählte Algorithmen

Für die vorliegende Studienarbeit wurden zwei Verfahren zur näheren Beschreibung ausgewählt, von denen eines implementiert und simuliert wurde.

Das erste Verfahren wird als k-hop Nachbarschaftsverfahren bezeichnet und ist ein verteiltes Verfahren mit reaktiven und proaktiven Bestandteilen. Es verzichtet auf eine vollständige Weltsicht und ist darauf spezialisiert, starke Veränderungen in der unmittelbaren Nachbarschaft zu entdecken.

Das zweite Verfahren heißt Leuchtturmmodell und hat Eigenschaften zentralisierter Verfahren. Wie der Name andeutet, gibt es einen ausgezeichneten Knoten, der die Identität der Partition ausmacht, beim Sammeln der Informationen aber auf die Mitarbeit der teilnehmenden Knoten angewiesen ist. Das Verfahren wird in den Kapiteln 4 bis 6 beschrieben.

Abschnitt 3.1 – Das k-hop Nachbarschaftsmodell

Beim Zugriff auf eine der Protokollschnittstellen muss Nachbarschaftsinformation vorliegen. Sie wird erstellt, indem der Knoten eine REQUEST-Nachricht als Broadcast versendet und deren TTL auf k setzt. Alle Knoten, die maximal k Teilstrecken entfernt liegen, werden die Nachricht erhalten und eine RESPONSE an den Absender verschicken. Der Initiator kann die Absenderadressen, gegebenenfalls auch noch die Laufzeiten der Nachrichten, den Teilstreckenzähler usw. auswerten und sich ein entsprechend genaues Bild der Nachbarschaft erstellen, z.B. als Liste oder als Graph.

Durch Veränderung nur eines Parameters – nämlich k – kann die Sicht auf die gesamte Partition ausgeweitet werden. Wird k stark vergrößert, um das gesamte Netz zu erreichen, verliert das Verfahren an Eleganz, denn es erzeugt ein hohes Nachrichtenaufkommen.

Abschnitt 3.2 – Ausarbeitung des Algorithmus

Anforderungen an die Knoten

Alle Knoten haben eine global eindeutige ***ID***.[9]

Anforderungen an den Algorithmus

Jeder Knoten, der k-hop Nachbarschaftsinformation anfordert, muss eine Speicherstruktur mit den IDs der Nachbarknoten halten. Die Speicherstruktur als Ganzes oder jeder Eintrag im Einzelnen sollte mit einem Zeitstempel versehen sein, um veraltete Einträge löschen zu können. Den Einträgen können optionale Felder hinzugefügt werden, wie z.B. die Entfernung in Hopcounts, die Differenz dieser Entfernung bezüglich der vorherigen Messung oder die Nachrichtenlaufzeit zum Knoten.

Der Einsatz eines Timers ist optional. Steht er zur Verfügung, kann die Speicherstruktur regelmäßig auf veraltete Einträge untersucht werden und eine Aktualisierung angestoßen werden. Wenn nicht, muss diese Prozedur beim Zugriff auf die Schnittstelle durchgeführt werden.
Bei der Analyse des Algorithmus wird vom Einsatz eines Timers ausgegangen werden.

Anforderungen an das Protokoll

Steht auf dem Knoten kein Routingprotokoll zur Verfügung, muss es diese zusätzliche Aufgabe selbst übernehmen.

– Broadcast mit unterdrückten Duplikaten ist möglich, indem jeder Knoten eine weitere Speicherstruktur mit der Absender-ID und letzten Sequenznummer der Knoten unterhält, von denen er eine Broadcast-Nachricht erhalten hat. Damit ist erhöhter Speicheraufwand verbunden, der da-

9 Die eingeführten Bezeichnungen werden im Anschluss in Tabelle 1 zusammengefasst dargestellt.

durch minimiert werden kann, dass Einträge nach kurzer Zeit, wenn kein Echo mehr erwartet wird, wieder gelöscht werden.

– Unicast kann annähernd erreicht werden, indem an Broadcast-Nachrichten jeweils die ID des weiterleitenden Knoten angehängt wird. Die beantwortende Unicast-Nachricht kann diesen Pfad entlang zurückgesendet werden. Das Verfahren wird unempfindlicher gegen hohe Mobilität, wenn die weiterleitenden Knoten aus Nachrichten Pfade lernen und diese zum Unicast verwenden. Das erfordert aber wiederum Speicherplatz.

Nachrichten

Es gibt 2 Nachrichtentypen:

REQUEST(UID, SeqNo, TTL)

- sie steht für eine Aufforderung an andere Knoten, ihre Identität bekanntzumachen
- sie wird mit Broadcast versendet
- jeder Empfänger dekrementiert TTL vor dem Weiterversenden
- eine Nachricht mit TTL=0 wird nicht mehr weitergesendet
- jeder Empfänger beantwortet die Nachricht

RESPONSE(SeqNo)

- sie steht für die Beantwortung einer REQUEST-Nachricht
- sie wird per Unicast an den Initiator versendet

Zustände

Der Algorithmus benötigt keine Zustände; eintreffende RESPONSE-Nachrichten können in das bisherige Wissen laufend einbezogen werden. Allerdings trennt das Versenden einer REQUEST-Nachricht den Zeitablauf des Knoten in Epochen, weil die Sequenznummer erhöht wird und RESPONSE-Nachrichten, die sich auf frühere Sequenznummern beziehen, nicht berücksichtigt werden.

Implementierung der Schnittstellen

a) Anzahl der Knoten in der Partition

Es wird ein Packet REQUEST mit einem hohen Wert für TTL (z.B. 32) und einer Sequenznummer per Broadcast versendet. Danach werden für eine Zeitlang alle eingehenden RESPONSE-Nachrichten gezählt, die sich auf die Sequenznummer des REQUEST beziehen.

b) Liste der Knotenadressen in der Partition

Dies ist eine Erweiterung der Lösung aus a): eingehende RESPONSE-Nachrichten nicht zählen, sondern abspeichern.

c) Liste der neu hinzugekommenen oder abgegangenen Knoten seit einer gewissen Zeit

Es wird eine Datenstruktur mit den über Aufgabe b) gelernten Knoten gespeichert. Zu gegebener Zeit wird Aufgabe b) wieder durchgeführt. Von der Vereinigung der alten und der neuen entstehenden Liste wird deren Schnittmenge abgezogen, um Aufgabe c) zu erfüllen.

d) Relative Veränderung der Partition seit einer gewissen Zeit

Es wird eine Datenstruktur mit den über Aufgabe b) gelernten knoten gespeichert. Zu gegebener Zeit wird Aufgabe b) wieder durchgeführt. Wird von der alten Liste die neue Liste abgezogen, bleibt die Menge der abgegangenen Knoten. Wird von der neuen Liste die alte Liste abgezogen, bleibt die Menge der neuen Knoten. Die relative Veränderung berechnet sich wie folgt:

$$Grad\,der\,Veränderung = \frac{|abgegangene\,Knoten| + |neue\,Knoten|}{|alte\,Liste \cup neue\,Liste| + 1}$$

Die 1 im Nenner ist der Knoten selbst und verhindert überdies die Division durch 0.

Knoteneigenschaften	
ID	global eindeutige ID des Knotens
L	Liste von IDs der Knoten, die in k-hop Nachbarschaft liegen
T	Timer [optional]
Verwendete Sendetechniken	
	Broadcast, Unicast
Nachrichten und ihre Headerfelder	
REQUEST(UID, SeqNo, TTL)	
UID	ID des versendenden Knotens
SeqNo	Nachrichtenzähler des versendenden Knotens
TTL	Anzahl der zugelassenen Teilstrecken für die Nachricht
RESPONSE(SeqNo)	
SeqNo	Der Nachrichtenzähler des versendenden Knotens als Referenz

Tabelle 1: Bezeichnungen im k-hop Nachbarschaftsmodell

Abschnitt 3.3 – Analyse

Sei n die Zahl der Knoten in der Partition. Sei m der Aufwand für einen Speicherzugriff. Sei s der akkumulierte Aufwand für das Senden einer Nachricht und r für das Empfangen. Sei E_i der Aufwand, der für das Netz bei Zugriff auf die Schnittstelle i entsteht (entsteht.

n	Zahl der Knoten in der Partition
m	Aufwand für einen Speicherzugriff
s	Aufwand für Senden
r	Aufwand für Empfangen
E_i	Aufwand für Zugriff auf Schnittstelle, $i \in \{a, b, c, d\}$

Tabelle 2: Bezeichnervereinbarungen für Analyse

Für die Analyse wird zunächst optimales Routing für Broadcast- und Unicast-Nachrichten vorausgesetzt. Dann ergeben sich beim Zugriff auf eine der Schnittstellen im schlimmsten Fall folgende Ergebnisse:

a) Anzahl der Knoten in der Partition

Der Knoten sendet REQUEST mit TTL = ∞, also gibt es ein Sendeereignis, $n-1$ Empfangsereignisse für REQUEST und $n-1$ Sende- und Empfangsereignisse für RESPONSE. Jeder antwortende Knoten kann die gesamte Nachbarschaftsinformation haben, die der Initiator der Anfrage auswertet.

$$s + (n-1)*r + (n-1)*s + (n-1)*r + (n-1)*n*m + (n-1)*n*m = \mathbf{m * (2n^2 - 2n) + r * (2n - 2) + s * n} \qquad = E_a$$

b) Liste der Knotenadressen in der Partition

Wie unter a) $\qquad E_a \qquad = E_b$

c) Liste der neu hinzugekommenen oder abgegangenen Knoten seit einer gewissen Zeit

Das erfordert zusätzlich zum Aufwand von a) den Vergleich einer neuen mit einer alten Liste, die beide maximal die Länge n haben. Der Vergleich ist im Fall sortierter Listen in linearer Zeit möglich, so dass sich der Aufwand an Speicherzugriffen um mindestens $n*log(n) + 2n$ erhöht.

$$E_a + m*n*log(n) + m*2n = \mathbf{m * (2n^2 + n*log(n)) + r * (2n - n) + s * n} \qquad = E_c$$

d) Relative Veränderung der Partition seit einer gewissen Zeit

Dazu muss die entstehende Liste und die Differenzlisten noch gezählt werden, zusätzlich zu c) also $m*n$ Aufwand.

$$E_c + m*n = \mathbf{m * (2n^2 + n + n(log(n)) + r * (2n - n) + s * n} \qquad = E_d$$

Gibt es kein Routing, dann müssen Nachrichten per Broadcast verschickt werden, und Knoten, die die Nachricht bereits gesehen haben, dürfen sie nicht mehr weiterleiten. Somit kommt auf jedes Empfangsereignis einer REQUEST-Nachricht ein weiteres Sendeereignis und maximal $n-1$ weitere Empfangsereignisse. Auf jede gesendete RESPONSE kommen statt einem nun maxmial $n-1$ Empfangsereignisse, $n-1$ Sendeereignisse und maximal $(n-1)^2$ weitere Empfangsereignisse. Somit ergibt sich

$$s + (n-1)*r + (n-1)*s + (n-1)^2*r + (n-1)*s + (n-1)^2*r + (n-1)^2*s + (n-1)^3*r + (n-1)*n*m + (n-1)*n*m$$

$$= \mathbf{m * (2n^2 - 2n) + r * n^2 * (n-1) + s * n^2} \qquad = E_a$$

und entsprechend für die abgeleiteten Formeln E_b, E_c und E_d.

Abschnitt 3.4 - Optimierung

a) Anzahl der Knoten in der Partition

Wie die Analyse gezeigt hat, erzeugt das naive Verfahren sehr hohes Nachrichtenaufkommen. Ein besserer Algorithmus ist in [10] zu finden, mit dem allerdings weitere Informationen wie die Teilstreckenzähler usw., die oben angesprochen wurden, nicht mehr erhoben werden können. Natürlich gehen diese Auswertungen auch weit über die der Schnittstellenbeschreibung hinaus.

b) Liste der Knotenadressen in der Partition

Hier reicht das in a) erwähnte optimierte Verfahren nicht aus, da es um mehr als die Knotenanzahl geht. Zu optimieren wäre lediglich das Broadcast-Verfahren. Dazu kann ein eigenes Verfahren auf Ebene des Partitionserkennunsalgorithmus implementiert werden, oder man nutzt ein Routingprotokoll, das Broadcast optimal unterstützt. Außerdem kann das als Broadcast versendete Paket von Routingprotokollen wie AODV [11] genutzt werden, um den Reverse Path zu aktualisieren.

Eine weiterreichende Verbesserung in Bezug auf die Verbreitung des Broadcast in Verbindung mit der Datensammlung für den Partitionserkennungsalgorithmus ist in [12] zu finden. Dabei wird ein optimaler Baum mit einer minimalen Anzahl von Knoten berechnet, die um Informationen angefragt werden müssen, um eine Gesamtsicht auf die Partition zu erhalten. Dieses Verfahren ist nur bei moderater Knotenmobilität anwendbar.

Ein weniger optimaler Algorithmus wie der folgende kann auch bei Mobilität die Zahl der Broadcasts verringern (fußt auf dem Wahlalgorithmus aus [13]): Ein Knoten besitzt (ggf. durch Zugriff auf tiefere Schichten im Protokollstapel, siehe [14]) Informationen über seine erreichbaren Nachbarn. Erhält er einen neuen REQUEST, vermerkt er sich dessen Sequenznummer und leitet ihn weiter. Weitere REQUESTS mit derselben Sequenznummer, aber von anderen Knoten, werden mit ACK beantwortet. So entsteht ein Baum aus Knoten, die genau von den Knoten kein ACK bekommen haben, von denen sie die Beantwortung des REQUEST erwarten. Blattknoten erhalten daher von allen Nachbarn umgehend ein ACK. Sie geben ihre Information an ihren Vorgänger zurück. Dieser sammelt Daten von allen Knoten ein, von denen REQUESTS ausstehen, und leitet diese Sammlung seinerseits an seinen Vorgänger weiter, bis die Informationen – in wenige Pakete gepackt – beim Sender eintreffen.

Dieser Algorithmus passt sich auch an Topologieänderungen an. Der Knoten bemerkt – mithilfe von Informationen der Sicherungsschicht - verlorengegangene Links und erwartet von diesen kein Acknowledgements mehr. Ein verlorengegangener Link zum Vorgänger dagegen kann durch geeignete Routingprotokolle und Versenden per Unicast ersetzt werden.

Um das Nachrichtenaufkommen weiter zu senken, kann ein Knoten bereits bei TTL=1 auf ein Rebroadcast verzichten und unverzögert mit einer Nachricht antworten, die die Knoten-IDs seiner 1-hop Nachbarschaft und seine eigene enthält. Dann kommt es bei dem oben beschriebenen supoptimalen

Verfahren zur Informationssammlung allerdings zu Duplikateinträgen von Knoten-IDs, die herausgefiltert werden müssen.

Abschnitt 3.5 – Bewertung

Die Wahl des k beschränkt die Genauigkeit der Sicht – alles, was darüber hinaus geht, bleibt dem Sender verborgen. Eine vollständige Sicht auf das Netz kann also nur dann sichergestellt werden, wenn die maximale Zahl an Teilstrecken zwischen Knoten bekannt ist oder $k = \infty$ gesetzt wird.

Da das Protokoll ausschließlich proaktiv arbeitet, kann es außerdem eine Weile dauern, bis Informationen vom Rand der k-hop Nachbarschaft den Sender erreicht haben und zu diesem Zeitpunkt bereits veraltet sein. Andererseits ist die Wahl der proaktiven Verarbeitung von Anfragen berechtigt:

1. Jeder Knoten hat nur eine k-hop-Sicht auf die Partition, die nicht unbedingt die gesamte Sicht der Partition umfassen muss. Ein innerhalb von k Hop festgestellter Verlust von Knoten berechtigt noch nicht zu der Annahme, die Knoten seien tatsächlich der ganzen Partition verloren gegangen. Eine ungefragt versendete Nachricht über den Verlust der Knoten könnte für Empfänger ein paar Hop weiter einfach falsch sein, wenn die Knoten in deren k-hop Bereich gewechselt sind.

2. Die Empfänger der Nachricht haben einen anderen k-hop-Kreis und sind an Informationen über Veränderungen am Rand des k-hop-Kreises des Senders womöglich gar nicht interessiert.

Die Vorteile des k-hop-Nachbarschaftsalgorithmus liegen auf der Hand:

- der Algorithmus ist unkompliziert
- es werden nur die relevanten Veränderungen in der Nachbarschaft des Senders mitgeteilt
- es ist kein ausgezeichneter Knoten nötig, der mit administrativen Aufgaben belastet wird

Die Nachteile des Verfahrens:

- es ist nahezu unmöglich festzustellen, ob die ganze Partition erreicht wurde
- ein hohes k (als Lösung des vorigen Punktes) steigert das Nachrichtenaufkommen dramatisch, wenn die Knoten Broadcasts nicht verwalten; Verwaltung steigert aber den Aufwand
- viel Information muss selbst gesammelt, gespeichert und berechnet werden

Kapitel 4

Das Leuchtturmmodell

Abschnitt 4.1 – Beschreibung der Idee

Beim Zugriff auf eine der Protokollschnittstellen muss Information über die Teilnehmer an der Partition vorliegen. Sie kann ad hoc oder regelmäßig von einem ausgezeichneten Knoten der Partition, dem „Leuchtturm“, bezogen werden. Die Aufgabe zum Erstellen der Partitionsinformation liegt somit allein beim Leuchtturm.

Er sendet dafür regelmäßig Heartbeat-Nachrichten und zählt alle Knoten zu seiner Partition, die sich darauf melden. Knoten, die kein Heartbeat ihres Leuchtturms mehr erhalten, wählen einen neuen Leuchtturm oder erklären sich selbst dazu. Treffen zwei Partitionen aufeinander, so wählen alle Knoten einen neuen Leuchtturm.

Im Unterschied zum k-hop Nachbarschaftsmodell zeichnen sich logische Partitionen im Leuchtturmmodell durch eine eindeutige Bezeichnung aus, nämlich durch die ID ihres Leuchtturms.

Abschnitt 4.2 – Ausarbeitung des Algorithmus

Anforderungen an die Knoten

Alle Knoten haben eine global eindeutige ***ID*** und eine Uhr, welche eine *lokale Zeit* angibt. Die Uhren der teilnehmenden Knoten brauchen nicht synchronisiert zu sein.[10]

Anforderungen an den Algorithmus

Der Algorithmus speichert Knoteninformationen in einer Struktur ***Info***, die aus ***ID***, ***SeqNo*** und ***Zeit*** besteht.

Knoten, die kein Leuchtturm sind, müssen wenigstens die Information ***LH*** über den Leuchtturm speichern: seine ID, die Sequenznummer seiner letzten Nachricht und die lokale Zeit, wann diese Nachricht erhalten wurde. Sollen Anfragen an den Leuchtturm gestellt werden, müssen die Sequenznummer als ***Kennung*** ihrer Anfrage speichern. Außerdem benötigen sie einen Timer ***Erwarten***, bei dessen Ablauf festgestellt wird, ob der Knoten sich aus einer Partition gelöst hat. Dafür muss eine Zeit ***deltaE*** bekannt sein, nach der weiterer Kontakt zum bisherigen Leuchtturm nicht mehr zu erwarten ist.

Der Leuchtturm muss eine Speicherstruktur ***L*** mit den ***Info***s seiner Partition halten, die jeweils mit einem Marker ***neu*** verbunden sind. Knoten, die zu einer Partition hinzustoßen, werden als *neu* bezeichnet, bis ihr Erscheinen als relative Veränderung an der Partition bekanntgemacht wurde.

Der Leuchtturm speichert noch ***Info***s über zwei spezielle Knoten: seinen ***Nachfolger*** und seinen ***Partner***. Der Nachfolger ist ein Knoten innerhalb der aktuellen Partition, den der Leuchtturm an die Teilnehmer seiner Partition als seinen Stellvertreter meldet. Der Partner ist ein Leuchtturm außerhalb der Partition, mit dessen Partition gerade Kontakt aufgenommen wurde.

Außerdem benötigt der Leuchtturm einen Timer ***Senden***, bei dessen Ablauf eine IDENT-Nachricht gesendet wird. Dafür muss eine Zeit ***deltaS*** bekannt sein, nach der die nächste IDENT-Nachricht gesendet werden muss, um die Partition logisch zusammenzuhalten. Auch der Leuchtturm benötigt eine Zeit ***deltaE***, nach der weiterer Kontakt zu einem bisherigen Teilnehmerknoten nicht mehr zu erwarten ist. Ein solcher Knoten wird aus der Knotenliste ***L*** des Leuchtturms gestrichen. Es wird deutlich, dass ***deltaE*** ein mehrfaches Vielfaches von ***deltaS*** sein sollte; Beispiele für die Kalibrierung und ihre Auswirkung sind in der Simulation weiter unten dargestellt.

10 Die eingeführten Bezeichner werden anschließend in Tabelle 3 nochmals aufgelistet.

Anforderungen an das Protokoll

Steht auf dem Knoten kein Routingprotokoll zur Verfügung, muss es Broadcast- und Unicast-Aufgaben selbst übernehmen; das Vorgehen orientiert sich an den oben beim k-hop Nachbarschaftsmodell skizzierten Lösungen, nachzulesen unter Kapitel 3, Abschnitt 2, „Anforderungen an das Protokoll“ auf Seite 12.

Nachrichten

Es gibt sechs Nachrichtentypen:

IDENT(ID, SeqNo, NachfolgerID, PartnerID)

- steht für die Aufforderung an Teilnehmer der Partition ID, sich beim Leuchtturm zu melden
- wird mit Broadcast versendet
- Angehörige der Partition ID werten SeqNo und NachfolgerID aus und antworten mit ACK
- steht weiterhin für die Information an einen anderen Leuchtturm (einen potentiellen Partner) darüber, dass die beiden Partitionen gerade Kontakt haben
- muss daher auch von Knoten weitergeleitet werden, die nicht zur Partition ID gehören
- der Leuchtturm einer fremden Partition kann den Sender der Nachricht als Partner wählen, indem er sich als dessen Partner bezeichnet und mit IDENT(..., ID) antwortet
- hat sich der Leuchtturm der fremden Partition in der IDENT-Nachricht bereits als Partner vorgefunden, kann er mit PROPOSE zur Neuwahl auffordern

ACK(LeuchtturmID, LeuchtturmSeqNo, ID)

- steht für die Reaktion von Knoten in der Partition auf eine IDENT-Nachricht des Leuchtturms
- wird mit Unicast an den Leuchtturm versendet
- der Leuchtturm aktualisiert daraufhin die Knoteninformation des Senders

QUERY(LeuchtturmID, Kennung, ID)

- steht für die Bitte eines Nicht-Leuchturms an den Leuchtturm der Partition um Partitionsinformation
- wird mit Unicast versendet
- wird vom Leuchtturm mit STATUS beantwortet

STATUS(ID, SeqNo, Kennung, KnotenID)

- steht für die Beantwortung einer QUERY-Nachricht
- wird mit BROADCAST versendet
- kann daher auch von allen anderen Knoten ausgewertet werden

PROPOSE(ID, SeqNo, NachfolgerID, PartnerID)

- steht für die Aufforderung eines Leuchtturms ID an den Leuchtturm der verbundenen Partition PartnerID, die Knoten seiner Partition zur Neuwahl aufzufordern
- wird mit Broadcast versendet
- Angehörige der Partition ID behandeln PROPOSE wie IDENT
- der Leuchtturm PartnerID kann mit ELECT antworten
- der sendende Leuchtturm bindet sich für die Zeit ***deltaV*** an seinen Wahlvorschlag
- hat der Leuchtturm PartnerID bereits selbst die Wahl vorgeschlagen, muss er mit ELECT antworten

ELECT(ID, SeqNo, PartnerID)

- steht für die Aufforderung eines Leuchtturms an die Knoten seiner Partition, sich der Neuwahl des Leuchtturms anzuschließen
- die Nachricht kann gewisse für den Start des Wahlalgorithmus notwendige Informationen enthalten
- die Kontrolle wird an den Wahlalgorithmus übergeben

Zustände

Der Algorithmus benötigt Zustände, um die Wahl des Leuchtturms verwalten zu können.

Ein Knoten ist normalerweise im Zustand **L**, Leuchtturm. Tritt er mit einem anderen Leuchtturm in Verbindung, dann gehen beide nach einer Zeit in Zustand **D** über. Dieser Zustand wird für die Zeit beibehalten, in denen verbindlich abgegebene Wahlvorschläge aufrechterhalten werden. Bei erfolgreich eingeleiteter Wahl gehen Knoten in den Zustand **E** (Election) über, während dem sie nur Nachrichten des Wahlprotokolls verarbeiten. Der gewählte Knoten geht wieder in Zustand **L** über. Unterlegene Knoten warten im Zustand **F**, fertig, auf die Eingliederung in die Partition des neuen Leuchtturms. Erfolgt diese, gehen diese Knoten in Zustand **N**, Nicht-Leuchtturm, über.

Es sei bemerkt, dass man **L** und **N** nicht unbedingt als getrennte Zustände auffassen muss. Jeder Knoten kann durch Prüfung, ob seine ***ID*** der ***ID*** seines Leuchtturms entspricht, feststellen, ob er Leuchtturmaufgaben wahrnehmen muss. Die Trennung der Zustände erleichtert aber die folgenden Darstellungen und ist auch intuitiv leicht erfassbar.

Ereignisse

Um die Arbeitsweise des Algorithmus verallgemeinert darzustellen, wird hier das Konzept des Ereignisses eingeführt.

Nur ein Ereignis kann einen Zustandsübergang eines Knotens herbeiführen. Als Ereignisse werden bezeichnet:

a) der Ablauf eines Timers,
- Timer ***Erwarten, Senden***

b) der Empfang einer Nachricht[11],
- IDENT, ACK, QUERY, STATUS, PROPOSE, ELECT und Wahlprotokollnachrichten

c) der Prozeduraufruf über die öffentliche Schnittstelle, von einer lokalen Anwendung abgesetzt.
- **Start, Stop, Fertig, Fehler, Tabelle**

Die Ereignisse im Leuchtturmmodell sind auch Tabelle 3 zu entnehmen.

Speicherstruktur für Knoteninformation	
Info(ID, SeqNo, Zeit)	
ID	ID des beschriebenen Knotens
SeqNo	Nachrichtenzähler der zuletzt von ihm erhaltenen Nachricht
Zeit	Lokale Zeit des letzten Nachrichteneingangs von ihm
Knoteneigenschaften (allgemein)	
ID	global eindeutige ID des Knotens
Zeit	Lokale Zeit, nicht synchronisiert
Zähler	Nachrichtenzähler
Knoteneigenschaften (Leuchtturm)	
L	{ *Info, neu* }* Liste der Knoten in der Partition
Nachfolger	Informationen über den Stellvertreter des Leuchtturms
Partner	Informationen über den Leuchtturm einer Nachbarpartition
deltaS	Intervall, nach dem IDENT erneut gesendet wird
deltaV	Intervall, nach dem ein Wahlvorschlag zurückgezogen wird
deltaE	Intervall, nach dem ein schweigender Knoten gelöscht wird
Senden	Timer
Vorschlag	Timer
Knoteneigenschaften (Nicht-Leuchtturm)	
Lh	*Info* mit Informationen über den Leuchtturm
Nachfolger	*Info* mit Informationen über den Stellvertreter des Leuchtturms
Kennung	Gespeicherter Nachrichtenzähler als Referenz
deltaE	Intervall, nach dem IDENT erneut erwartet wird
Erwarten	Timer

11 Der Netzwerksimualtor ns2 betrachtet Nachrichten als Ereignisse, siehe [15], Kap. 12, Abschnitt 2.

Nachrichten	
	IDENT(*ID, SeqNo, NachfolgerID, PartnerID****)***
	ACK(*LeuchtturmID, LeuchtturmSeqNo, ID****)***
	QUERY(*LeuchtturmID, Kennung, ID****)***
	STATUS(*ID, SeqNo, Kennung, KnotenID****)***
	PROPOSE(*ID, SeqNo, NachfolgerID, PartnerID****)***
	ELECT(*ID, SeqNo, PartnerID****)***
Zustände	
Z	inaktiv
L	Leuchtturm
N	Nicht-Leuchtturm
D	Leuchtturm, während er an einen Wahlvorschlag gebunden ist
E	Knoten, der den Wahlalgorithmus ausführt
F	Nicht-Leuchtturm, der nach Beendigung des Wahlalgorithmus auf Aufnahme in die Partition des neuen Leuchtturms wartet
Ereignisse	
Nachrichten	
Ident, Ack, Query, Status, Propose, Elect	
	Eingang der entsprechenden Nachricht
Wahl	Eingang einer Nachricht des Wahlalgorithmus
Schnittstellen	
Start	Start des Protokolls
Stop	Anhalten des Protokolls
Fertig	Erfolgreiche Beendigung der Wahl
Fehler	Erfolglose Beendigung der Wahl
Tabelle	Aufforderung, beim Leuchtturm eine Mitgliedertabelle zu holen
Timer	
Timer	Entsprechender Timeout

Tabelle 3: Bezeichnungen im Leuchtturmmodell

Zustandsübergänge

Knoten müssen nicht auf alle Ereignisse reagieren. Reagiert ein Knoten nicht, wird
a) ein abgelaufener Timer nicht neu gestartet
b) eine eingegangene Nachricht verworfen[12].

12 Broadcast als Routingaufgabe wird auch von Knoten durchgeführt, die die Nachricht verwerfen.

In Tabelle 4 ist dargestellt, in welchen Zuständen Knoten auf welche Ereignisse reagieren und welche Zustandsübergänge möglich sind.

Zustand	*Ereignis*	*Folgezustände*
Z – inaktiv	Start – Protokollstart	L – Leuchtturm
L – Leuchtturm	Ack, Query, Timer, Tabelle	L – Leuchtturm
	Ident	L; D – Wahlvorschlag
	Propose	L
	Stop	Z – Stop
D - Wahlvorschlag	Ack, Query	D
	Timer	L
	Ident	L; D
	Propose, Elect	L; E
	Stop	Z
E - Wahl	Wahl	E
	Fertig, Fehler	L, F – Fertig
N – Nicht-Leuchtturm	Ident, Propose, Status, Tabelle	N
	Elect	E
	Timer	L
	Stop	Z
F – Warten nach der Wahl	Ident	N
	Timer	L
	Stop	Z

Tabelle 4: Zustände, Ereignisse und Zustandsübergänge

Zur besseren Übersicht ist in Abbildung 1 noch ein Zustandsübergangsdiagramm abgebildet.

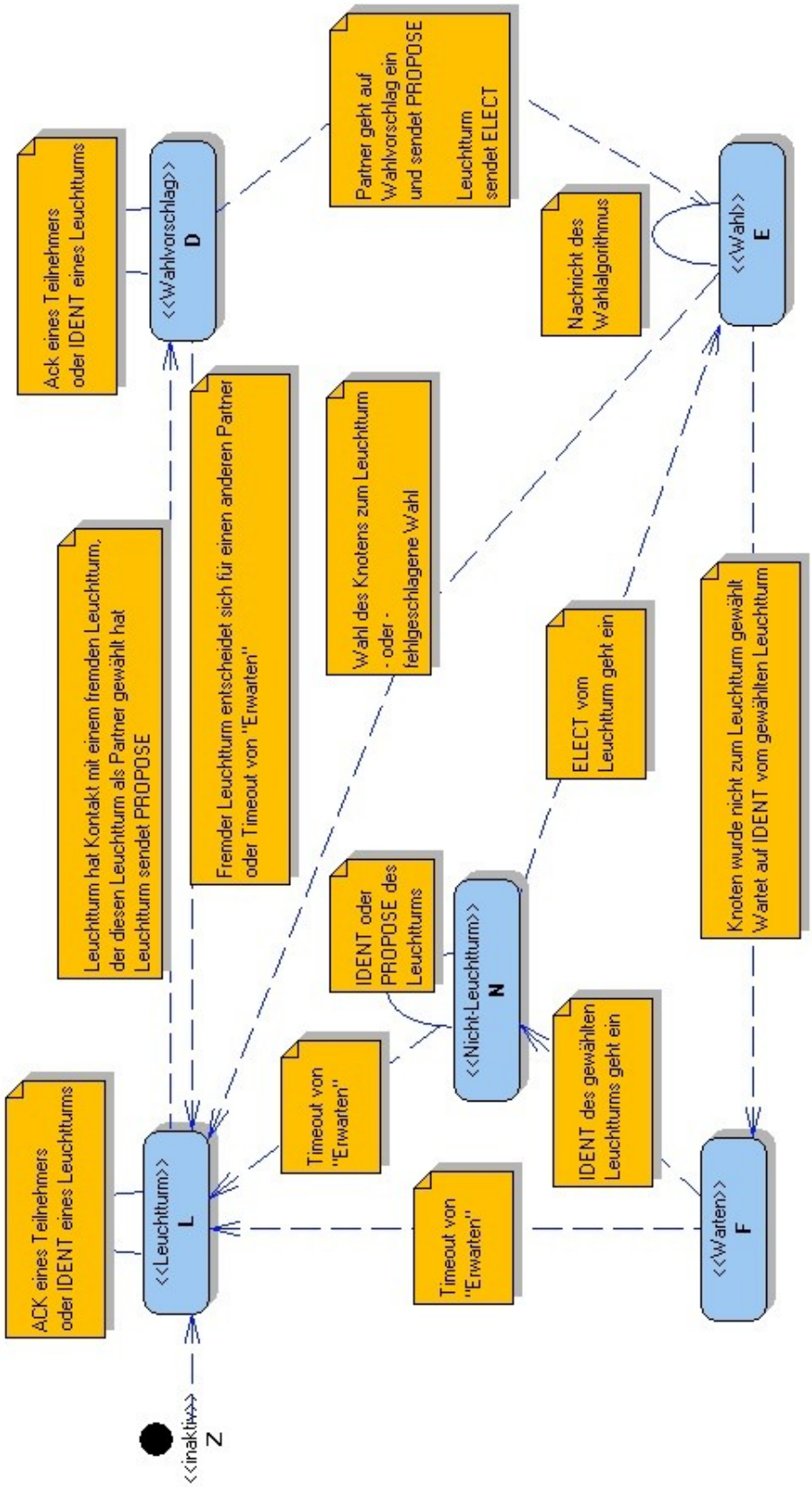

Abbildung 1: Zustandsübergangsdiagramm (die Übergänge nach Z sind ausgelassen)

Die Wahl des Folgezustandes hängt von Bedingungen ab und ist Ergebnis einer Prozedur. Tabelle 5 zeigt die Zusammenhänge von Zuständen, Bedingungen, Prozeduren und Zustandsübergängen für eine nicht optimierte Form des Leuchtturmmodells.

Die Tabelle ist optisch nach den Ereignissen gruppiert. Die ausgelösten Prozeduren sind von Zustand und Bedingungen abhängig. Nach rechts eingerückte Bedingungen sind mit darüberstehenden Bedingungen, die weiter links abschließen, logisch zu verunden. In halbfett und kursiv gesetzte Bezeichner sind Variablennamen, die der Knoten verwaltet, auf dem das Ereignis eintritt. Mit Punkt getrennte Bezeichner sind Felder aus dem Header der entsprechenden Nachricht. IDs stehen für die ihnen zugeordneten Subjekte.

Die Prozeduren sind nur skizziert, eine ausführliche Beschreibung folgt. Außerdem sind einige Routineprüfungen weggelassen wie z.B. die, ob die Nachricht für den Empfänger bestimmt war, dass sie nicht vom Empfänger selbst stammt und dass sie kein Duplikat ist.

Ereign.	***Zust.***	***Bedingung***	***Prozedur***	***Folge-zust.***
Start	Z	-	Variablen setzen	L

Ereign.	***Zust.***	***Bedingung***	***Prozedur***	***Folge-zust.***
Stop	*	-	-	Z

Ereign.	***Zust.***	***Bedingung***	***Prozedur***	***Folge-zust.***
Propose	L	IDENT.ID hat ***ID*** als Partner und ***ID*** hat keinen Partner	PROPOSE senden	D
	D	IDENT.ID hat ***ID*** als Partner und ***ID*** hat IDENT.ID als Partner	ELECT senden, Wahl	E
	N	-	Auswerten, ACK senden	N

Ereign.	***Zust.***	***Bedingung***	***Prozedur***	***Folge-zust.***
Elect	D	IDENT.ID hat ***ID*** als Partner und ***ID*** hat IDENT.ID als Partner	ELECT senden, Wahl	E
	N	-	Wahl	E

Ereign.	*Zust.*	*Bedingung*	*Prozedur*	*Folge-zust.*
Fertig	E	Wahl ergab ID als Leuchtturm	Variablen setzen	L
		Wahl ergab nicht ID als Leuchtturm	***Lh*** auf Gewinner setzen	F

Ereign.	*Zust.*	*Bedingung*	*Prozedur*	*Folge-zust.*
Fehler	E	-	Variablen setzen	L

Ereign.	*Zust.*	*Bedingung*	*Prozedur*	*Folge-zust.*
Tabelle	N	-	QUERY senden	N
	L, D	-	Tabelle zurückgeben	L, D

Ereign.	*Zust.*	*Bedingung*	*Prozedur*	*Folge-zust.*
Timer	L	Senden	***L*** aktualisieren, IDENT senden	L
		Erwarten	***Partner*** löschen	L
	D	Senden	***L*** aktualisieren, IDENT senden	D
		Erwarten	***Partner*** löschen	L
	N	Es gibt einen Nachfolger	***Nachfolger*** zum Leuchtturm machen, auf IDENT warten	N
		Es gibt keinen Nachfolger	Leuchtturm werden	L
	F	-	Leuchtturm werden	L

Tabelle 5: Zustände, Bedingungen, Prozeduren und Übergänge

Implementierung der Schnittstelle

a) Anzahl der Knoten in der Partition

Es wird ein Paket QUERY an den Leuchtturm gesendet. Dessen Antwort STATUS enthält die Liste der IDs der Knoten, die seiner Sicht nach logisch zu seiner Partition gehören. Die Anzahl der Knoten in der Liste vermehrt um 1 (den Leuchtturm) ergibt die Anzahl der Knoten in der Partition.

b) Liste der Knotenadressen in der Partition

Diese Aufgabe entspricht dem Vorgehen unter a)

c) Liste der neu hinzugekommenen oder abgegangenen Knoten seit einer gewissen Zeit

Es wird eine Datenstruktur mit den über Aufgabe b) gelernten Knoten gespeichert. Zu gegebener Zeit wird Aufgabe b) wieder durchgeführt. Von der Vereinigung der alten und der neuen entstehenden Liste wird deren Schnittmenge abgezogen, um Aufgabe c) zu erfüllen. Der Leuchtturm unterstützt die Nachfrage nach Änderungen in der Partition, indem er seinen IDENT-Nachrichten eine Prozentzahl hinzufügt, zu der sich die Partition seit dem letzten Versand von IDENT geändert hat. Eine hohe Zahl kann den Knoten anregen, sich für die Änderungen zu interessieren.

d) Relative Veränderung der Partition seit einer gewissen Zeit

Es wird eine Datenstruktur mit den über Aufgabe b) gelernten knoten gespeichert. Zu gegebener Zeit wird Aufgabe b) wieder durchgeführt. Wird von der alten Liste die neue Liste abgezogen, bleibt die Menge der abgegangenen Knoten. Wird von der neuen Liste die alte Liste abgezogen, bleibt die Menge der neuen Knoten. Die relative Veränderung berechnet sich wie folgt:

$$Grad\,der\,Veränderung = \frac{|abgegangene\,Knoten| + |neue\,Knoten|}{|alte\,Liste \cup neue\,Liste| + 1}$$

Der Leuchtturm kann die Erkennung von relativen Veränderungen der Partition unterstützen, indem er mit seinen IDENT-Nachrichten eine Zahl *Grad der Veränderung* mitsendet. Sie wird nach der obigen Vorschrift beim Leuchtturm jeweils zwischen dem Versenden von zwei IDENT-Nachrichten neu berechnet.

Dieser Algorithmus wurde im Rahmen der Studienarbeit in ns2 [16] implementiert. Es folgt eine genauere Beschreibung.

Kapitel 5

Implementierung

Abschnitt 5.1 – Entscheidungen

A. Struktur

Die Implementierung sollte nicht für ein spezifisches Design oder eine bestimmte Gruppe von mobilen Geräten zugeschnitten sein. Auf mobilen Geräten können bereits einige Protokolle installiert sein, während die Ausführung anderer Protokolle an Speicherplatz- oder Energiegrenzen scheitern. Es war daher wichtig, die Implementierung möglichst modular und die Schnittstellen generisch zu halten. So entstand folgende Struktur:

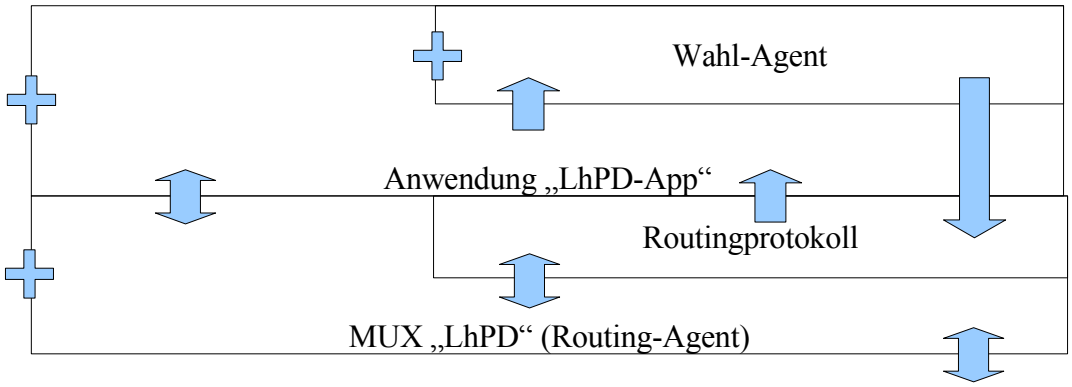

Abbildung 2: Struktur. Die Pfeile stellen mögliche Nachrichtenpfade dar, die Kreuze Schnittstellen.

Dem lagen folgende Überlegungen zugrunde:

– Wenn es auf dem System ein Routingprotokoll gibt, dann kann es leicht eingebunden werden. Nachrichten, die von der Anwendung versendet werden sollen, werden dem Routingprotokoll zur Routenbestimmung übergeben, dann aber vom Multiplexer als dessen Nachrichten versendet. Steht kein Routingprotokoll zur Verfügung, dann können die oben Beschriebenen Ersatzmechanismen greifen, die aber Zugriff auf die Paketstruktur benötigen und daher innerhalb des Protokolls implementiert werden müssen.
– Wenn es auf dem System eine bestimmte Form der Leader Election gibt, kann diese über eine Schnittstelle leicht als Wahl-Agent eingebunden werden[13]. Gibt es keine Wahlprotokolle, können eigene definiert werden. Für die vorliegende Implementierung wurde eine besonders einfache Form der Leadership Election gewählt (der Leuchtturm mit der größten ID gewinnt).

B. Der Multiplexer

Der Multiplexer (auch LhPD-Agent genannt) tritt anderen Knoten gegenüber als Routingprotokoll auf. Er verpackt alle Pakete, die ihn vom Routingprotokoll aus erreichen (auch Kontrollpakete des Routingprotokolls), und versendet sie als LhPD-Pakete. Er packt alle eintreffenden Pakete aus. Kann er sie der Anwendung zuordnen, übergibt er sie direkt an die Anwendung, ansonsten an das Routingprotokoll. Die Schnittstelle des Multiplexers dient zum Ein- und Abschalten des Protokolls, um auf die ID des Knotens zugreifen zu können und um Statusinformationen abzufragen.

C. Das Routingprotokoll

Die Simulationen in Kapitel IV wurden mit dem Routingprotokoll AODV durchgeführt, wie es im Netzwerksimulator ns2[14] implementiert ist. Es wurde so eingestellt, dass es als Ziel stets den Multiplexer hat. Nachrichten vom Multiplexer und vom Wahl-Agenten werden mit Routinginformationen versehen und an den Multiplexer zurückgegeben. Eintreffende Nachrichten, die das Routingprotokoll nicht verarbeiten kann, werden an die Anwendung weitergeleitet. In der vorliegenden Implementierung betrifft das mit Broadcast versendete Pakete.

D. Der Wahl-Agent

Der Wahl-Agent wird durch eine Schnittstelle von der Anwendung angestoßen, den Wahlalgorithmus zu beginnen. Er meldet durch die Schnittstelle der Anwendung zurück, ob die Wahl erfolgreich oder erfolglos verlaufen ist. Die Anwendung kann über die Schnittstelle danach den ermittelten Gewinner erfahren. Der Wahl-Agent kann auf Grundlage einer übermittelten Knotenliste durch eine lokal implementierte Variante des Wahlalgorithmus eine Quasi-Wahl durchführen und den Phantomgewinner ermitteln. Beim implementierten Wahlalgorithmus gewinnt stets die höchste ***ID***.

13 Einige Leader Election Algorithmen sind in [13,17] zu finden.
14 Version 2.28

E. Die Anwendung

Die Anwendung ist der statusbehaftete Teil des Protokolls. Sie reagiert auf Ereignisse. Die Ereignisse sind

a) der Ablauf eines Timers,

b) der Empfang einer Nachricht,

c) der Anstoß einer Prozedur über eine öffentliche Schnittstelle.

Die Schnittstelle zum Wahl-Agenten ist oben beschrieben worden. Eine weitere Schnittstelle dient zum Ein- und Abschalten des Protokolls, um auf die ID des Knotens zugreifen zu können und um Statusinformationen abzufragen. Schließlich wird sie genutzt, damit andere Anwendungen die Mitgliedertabelle abrufen können.

Abschnitt 5.2 – Detaillierte Beschreibung

Es folgt eine detaillierte Angabe der Entscheidungsabläufe bei auftretenden Ereignissen.

Konventionen zur Notation sind in Tabelle 6 angegeben. Vor den beschriebenen Ereignissen stehen, durch Doppelpunkt getrennt, Bedingungen. Eingerückte Bedingungen sind mit weiter links abschließenden Bedingungen darüber mit einem logischen „und“ zu verknüpfen. Bedingungen sind mit anderen Bedingungen, die links bündig mit ihnen abschließen, durch „exklusives Oder“ logisch verknüpft.

[*kursiv in eckigen Klammern*]
Kurzbeschreibungen oder Begründungen
halbfett
Zustands- und Ereignisnamen
Headerfelder empfangener Nachrichten
halbfett kursiv
lokale Variablen oder Strukturen
kursiv
andere Variablen oder Werte
GROSSBUCHSTABEN
Nachrichtennamen

Tabelle 6: Notation

Ereignis **Start** [*Starten des Protokolls*]

Erläuterung:

Ein Gerät wird beim Einschalten grundsätzlich selbst Leuchtturm.

Zustand **Z** [Protokoll gestoppt]

Beschreibung:

- : [*keine Bedingung*]

Variablen initialisieren (***Zähler*** := 1, ***Lh*** := (***ID***, 1, *Zeit*), ***Nachfolger*** = NULL, ***Partner*** = NULL, ***deltaE*** = ?, ***deltaS*** = ?, **L** = <neue Liste>)

Timer starten Senden(***deltaS***)

Nachricht IDENT(***ID***, ***Zähler***, NULL, NULL) versenden

Übergang in Zustand **L**

Aufwand:

9 Variablen initialisieren, 1 Timer starten, 1 Nachricht senden

Ereignis **Ident** [*Eingang einer IDENT-Nachricht*]

Erläuterung:

IDENT informiert über die Erreichbarkeit durch einen Leuchtturm.

Zustand **L** [*Knoten ist Leuchtturm*]

Erläuterung:

Ein Leuchtturm verarbeitet IDENT-Nachrichten eines anderen Leuchtturms, indem er ihn als Partner wählt und Informationen über ihn speichert. Für eine gewisse Zeit wird er dessen ID in eigenen IDENT-Nachrichten als PartnerID mitsenden – ein unverbindliches Signal an den fremden Leuchtturm, eine gemeinsame Partition bilden zu wollen. Sendet der fremde Leuchtturm in IDENT-Nachrichten die ID des Leuchtturms als PartnerID, wird das als erster „Handshake" für eine gemeinsame Partitionsbildung aufgefasst. Der zweite Handshake ist die Beantwortung einer IDENT-Nachricht mit PartnerID durch eine PROPOSE-Nachricht, die die gemeinsame Partitionsbildung für eine gewisse Zeit verbindlich vorschlägt.

Beschreibung:

Falls **IDENT.ID**!=***ID***: [*Eigene Nachricht nicht bearbeiten*]
  Falls ***Partner*** == NULL: [*Fremde Partition ist neuer Kontakt*]
    Falls **IDENT.PartnerID**==NULL: [*Fremder Leuchtturm hat keinen Partner*]
      Partner := (**IDENT.ID**, **IDENT.SeqNo**, *Zeit*)
      Nachricht IDENT(ID, SeqNo, Partner.ID, Nachfolger.ID)[15,16] versenden [*schnelle Reaktion*]
      Übergang in Zustand **L**.
    Falls **IDENT.PartnerID**==***ID***: [*Fremder Leuchtturm hat diesen Leuchtturm als Partner*]
      Partner := (**IDENT.ID**, **IDENT.SeqNo**, *Zeit*)
      Nachricht PROPOSE(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***)[17] versenden [*abgekürztes Warten*]
      Übergang in Zustand **D**. [*Bindung an den Wahlvorschlag*]
    Falls **IDENT.Partner**==**<*ID aus L*>**:
    [*Fremder Leuchtturm hat Nichtleuchtturm dieser Partitions als Partner*]
      Partner := (**IDENT.ID**, **IDENT.SeqNo**, *Zeit*)
      Nachricht IDENT(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden [*schnelle Reaktion*]
      Übergang in Zustand **L**.

  Falls ***Partner*** != NULL: [*Es besteht bereits Kontakt zu einer fremden Partition*]
    Falls ***Partner.ID***==**IDENT.ID** && ***Partner.SeqNo*** < **IDENT.SeqNo**: [*fremde Partition ist bekannt*]
      Falls **IDENT.PartnerID**==NULL && ***Partner.Zeit*** liegt weniger als 3*****deltaE*** zurück:
      [*fremdem Leuchtturm eine Nachrichtenumlaufzeit Zeit geben, um Partner zu wählen*]
        Nachricht IDENT(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
      Falls **IDENT.PartnerID**==***ID***:

15 Das Versenden jeder Nachricht ist mit dem Erhöhen des ***Zähler***s verbunden

16 Das Versenden jeder IDENT-Nachricht ist mit Neustart des Timers ***Senden***(***deltaS***) verbunden

17 Das Versenden jeder PROPOSE-Nachricht ist mit Neustart des Timers ***Senden***(***deltaS***) verbunden

Nachricht PROPOSE(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
Übergang in Zustand **D**. [*Bindung an den Wahlvorschlag*]
Falls **IDENT.Partner==<*ID aus L*>**:
[*Fremder Leuchtturm hat Nichtleuchtturm dieser Partitions als Partner*]
Nachricht IDENT(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
Übergang in Zustand **L**.
Sonst: [*IDENT.PartnerID ist die ID eines anderen Leuchtturms geworden*]
Partner:=NULL
Übergang in Zustand **L**.
Sonst: [*Übergangszeit von altem Partner zu neuem Partner verkürzen*]
Falls **IDENT.PartnerID**!=***Partner.ID*** && ***Partner.Zeit*** liegt mehr als ***deltaE*** zurück:
[*zu bisherigem Partner besteht schon länger kein Kontakt*]
Partner := (**IDENT.ID**, **IDENT.SeqNo**, *Zeit*)
Nachricht IDENT(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
Übergang in Zustand **L**.

Aufwand:
8 Variablen lesen/schreiben, 1 Listendurchlauf, 1 Timer starten, 1 Nachricht senden

Zustand **N** [*Knoten ist nicht Leuchtturm*]

Erläuterung:
Ein Nicht-Leuchtturm beantwortet IDENT-Nachrichten seines Leuchtturms mit einer ACK-Nachricht. (IDENT-Nachrichten fremder Leuchttürme unterliegen dem Broadcast).

Beschreibung:
Falls **IDENT.ID==*Lh.ID*** && **IDENT.SeqNo > *Lh.SeqNo***: [*Neue Nachricht des eigenen Leuchtturms*]
Nachricht ACK(***Lh.ID***, **IDENT.SeqNo**, ***ID***) versenden[18]
Nachfolger:=(**IDENT.NachfolgerID**, 0, *Zeit*)
Übergang in Zustand **N**

Aufwand:
3 Variablen lesen/schreiben, 1 Timer starten, 1 Nachricht senden

Zustand **D** [*Knoten ist Leuchtturm und an einen Wahlvorschlag gebunden*]

Erläuterung:
Der Leuchtturm erwartet den dritten Handshake vom Partner: eine PROPOSE-Nachricht. Eintreffende IDENT-Nachrichten des Partners können die Wartezeit verlängern, wenn der Partner bei seiner Partnerwahl bleibt. Entscheidet sich der Partner für einen anderen Partner, wird die Bildung einer gemeinsamen Partition aufgegeben.

18 Das Versenden jeder ACK-Nachricht ist mit Neustart des Timers ***Erwarten***(***deltaE***) verbunden

Beschreibung:

Falls **IDENT.ID**!=***ID***: [*Eigene Nachricht nicht bearbeiten*]
  Falls ***Partner.ID***==**IDENT.ID** && ***Partner.SeqNo*** < **IDENT.SeqNo**: [*fremde Partition ist bekannt*]
    Falls **IDENT.PartnerID**==***ID***:
      Nachricht PROPOSE(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
      Übergang in Zustand **D**. [*weitere Bindung an den Wahlvorschlag*]
    Sonst: [*Fremder Leuchtturm anderen Partner oder keinen mehr*]
      Partner:=NULL
      Nachricht IDENT(***ID***, ***SeqNo***, NULL, ***Nachfolger.ID***) versenden
      Übergang in Zustand **L**.
  Sonst: [*Übergangszeit von altem Partner zu neuem Partner verkürzen*]
    Falls **IDENT.PartnerID**!=***Partner.ID*** && ***Partner.Zeit*** liegt mehr als ***deltaE*** zurück:
    [*zu bisherigem Partner besteht schon länger kein Kontakt*]
      Partner := (**IDENT.ID**, **IDENT.SeqNo**, *Zeit*)
      Nachricht IDENT(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
      Übergang in Zustand **L**.

Aufwand:

7 Variablen lesen/schreiben, 1 Listendurchlauf, 1 Timer starten, 1 Nachricht senden

Zustand **F** [*Knoten ist nicht Leuchtturm und wartet nach Wahl auf Anschluss an seine Partition*]

Erläuterung:

Ein Nicht-Leuchtturm wartet nach erfolgter Wahl auf IDENT-Nachrichten seines Leuchtturms, die nach Beendigung der Wahl versendet wurden.

Beschreibung:

Falls **IDENT.ID**==***Lh.ID*** && **IDENT.SeqNo** > ***Lh.SeqNo***: [*Neue Nachricht des eigenen Leuchtturms*]
  Nachricht ACK(***Lh.ID***, **IDENT.SeqNo**, ***ID***) versenden
  Nachfolger:=(**IDENT.NachfolgerID**, 0, *Zeit*)
  Übergang in Zustand **N**

Aufwand:

3 Variablen lesen/schreiben, 1 Timer starten, 1 Nachricht senden

Ereignis **Ack** [*Erhalt einer ACK-Nachricht*]

Erläuterung:

ACK-Nachrichten werden nur von dem Leuchtturm bearbeitet, an den sie gerichtet sind. Er setzt den Zeit-Eintrag seiner Knotenliste ***L*** auf die lokale *Zeit* der Ankunft der ACK-Nachricht.

Zustände **L**, **D** [*Leuchtturm*]

Beschreibung:

Falls **ACK.ID==*ID***: [*ACK war für Leuchtturm bestimmt*]

Falls **ACK.ID** nicht in ***L***: [*neuer Knoten in Partition*]

((**ACK.ID**, **ACK.SeqNo**, *Zeit*), ***neu***=true) in L einfügen

Übergang in Zustand **L** bzw. **D**.

Sonst:

L.Info*[ACK.ID].*Zeit:= *Zeit*

Übergang in Zustand **L** bzw. **D.**

Aufwand:

6 Variablen lesen/schreiben, 1 Listendurchlauf

Ereignis **Query** [*Erhalt einer QUERY-Nachricht*]

Erläuterung:

QUERY-Nachrichten werden nur von dem Leuchtturm bearbeitet, an den sie gerichtet sind. Er versendet die Knotenliste L im Nachrichtenbody der STATUS-Nachricht.

Zustände **L**, **D** [*Leuchtturm*]

Beschreibung:

Falls **QUERY.ID==*ID***: [*QUERY war für Leuchtturm bestimmt*]

Nachricht STATUS(***ID***, ***SeqNo***, ***Nachfolger.ID***, ***Partner.ID***) senden

Aufwand:

4 Variablen lesen/schreiben, 1 Listendurchlauf, 1 Nachricht senden

Ereignis **Status** [*Erhalt einer STATUS-Nachricht*]

Erläuterung:

STATUS-Nachrichten werden nur von Knoten bearbeitet, die in der Partition des Leuchtturms liegen. Daher wird mit in Verbindung mit ihnen kein Sende- und Erwartungstimer neu gestellt.

Zustand **N** [Nicht-*Leuchtturm*]

Beschreibung:

Falls **STATUS.ID==*Lh.ID***: [*STATUS kam vom Leuchtturm der Partition*]

Partner:=(**STATUS.PartnerID**, 0, *Zeit*)

Falls **STATUS.KnotenID==*ID*** && **STATUS.Kennun**g==***Kennung***[19]

Nachrichtenbody verarbeiten

Aufwand:

5 Variablen lesen/schreiben

19 Bei der weiter unten vorgestellten Optimierung muss diese Bedingung entfallen.

Ereignis **Propose** [*Eingang einer PROPOSE-Nachricht*]

Erläuterung:

PROPOSE bindet einen Leuchtturm für eine Zeitlang an den Vorschlag zur Vereinigung mit einer anderen Partition.

Zustand **L** [*Knoten ist Leuchtturm*]

Erläuterung:

Ein Leuchtturm betrachtet PROPOSE-Nachrichten eines anderen Leuchtturms als zweiten Handshake zur Bildung einer gemeinsamen Partition. Da er sich selbst nicht an diesen Partner gebunden hat, kann er die PROPOSE-Nachricht unbeantwortet lassen, und nach einiger Zeit wird der Partner aufgeben. Andernfalls kann er mit sofortiger Einleitung der Wahl reagieren. Erhält der Leuchtturm PROPOSE eines unbekannten Leuchtturms, nimmt er ihn als Partner auf, wartet aber weitere Nachrichten ab, da die Kommunikation in eine Richtung offensichtlich behindert ist.

Beschreibung:

Falls **PROPOSE.ID**!=***ID***: [*Eigene Nachricht nicht bearbeiten*]
  Falls **PROPOSE.PartnerID**==***ID***: [*PROPOSE an diesen Leuchtturm*]
    Falls ***Partner***==NULL: [*PROPOSE von unbekanntem Partner*]
      Partner := (**IDENT.ID**, **IDENT.SeqNo**, *Zeit*)
      Nachricht PROPOSE(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
      Übergang in Zustand **D**. [*Bindung an den Wahlvorschlag*]
    Falls ***Partner.ID***==**PROPOSE.ID** && **PROPOSE.SeqNo** > ***Partner.SeqNo***
      Nachricht ELECT(***ID***, ***SeqNo***, ***Partner.ID***) versenden
      Übergang in Zustand **E**. [*Wahl*]

Aufwand:

7 Variablen lesen/schreiben, 1 Timer starten, 1 Nachricht senden

Zustand **N** [*Knoten ist nicht Leuchtturm*]

Erläuterung:

Ein Nicht-Leuchtturm behandelt PROPOSE-Nachrichten seines Leuchtturms wie IDENT.

Beschreibung:

Falls **PROPOSE.ID**==***Lh.ID*** && **PROPOSE.SeqNo** > ***Lh.SeqNo***:
  Nachricht ACK(***Lh.ID***, **PROPOSE.SeqNo**, ***ID***) versenden
  Nachfolger:=(**IDENT.NachfolgerID**, 0, *Zeit*)
  Übergang in Zustand **N**

Aufwand:

3 Variablen lesen/schreiben, 1 Timer starten, 1 Nachricht senden

Zustand **D** [*Knoten ist Leuchtturm und an einen Wahlvorschlag gebunden*]

Erläuterung:

Die eintreffende PROPOSE-Nachricht ist der dritte Handshake, und der Leuchtturm kann seine Partition zur Wahl aufrufen.

Beschreibung:

Falls **PROPOSE.ID**!=***ID***: [*Eigene Nachricht nicht bearbeiten*]
  Falls **PROPOSE.PartnerID**==***ID***:
    Falls ***Partner.ID***==**PROPOSE .ID** && ***Partner.SeqNo*** < **PROPOSE.SeqNo**:
      Nachricht ELECT(***ID***, ***SeqNo***, ***Partner.ID***) versenden
      Übergang in Zustand **E**. [*Wahl*]

Aufwand:

4 Variablen lesen/schreiben, 1 Nachricht senden

Ereignis **Elect** [*Eingang einer ELECT-Nachricht*]

Erläuterung:

ELECT fordert den Empfänger zur Teilnahme an der Leuchtturmwahl auf.

Zustand **N** [*Knoten ist nicht Leuchtturm*]

Beschreibung:

Falls **ELECT.ID**==***Lh.ID*** && **PROPOSE.SeqNo** > ***Lh.SeqNo***:
  Übergang in Zustand **E**

Aufwand:

2 Variablen lesen/schreiben

Zustand **D** [*Knoten ist Leuchtturm und an einen Wahlvorschlag gebunden*]

Erläuterung:

Die eintreffende ELECT-Nachricht ersetzt eine verlorengegangene PROPOSE-Nachricht als dritten Handshake.

Beschreibung:

Falls **ELECT.ID**!=***ID***: [*Eigene Nachricht nicht bearbeiten*]
  Falls **ELECT.PartnerID**==***ID***:
    Falls ***Partner.ID***==**ELECT.ID** && ***Partner.SeqNo*** < **ELECT.SeqNo**:
      Nachricht ELECT(***ID***, ***SeqNo***, ***Partner.ID***) versenden
      Übergang in Zustand **E**. [*Wahl*]

Aufwand:

4 Variablen lesen/schreiben, 1 Nachricht senden

Ereignis **Fertig** [*Wahl-Agent meldet über die Schnittstelle den erfolgreichen Abschluss der Wahl*]

Erläuterung:

Die Anwendung fordert daraufhin die ID des Gewinners über die Schnittstelle an und erkennt, ob sie Leuchtturm geworden ist. Falls nicht, wartet er auf IDENT-Nachrichten des Leuchtturms.

Zustand **E** [*Wahl*]

Beschreibung:

Falls ***ID==Lh.ID***: [*Leuchtturm*]

Variablen initialisieren (***Nachfolger*** = NULL, ***Partner*** = NULL, ***deltaE*** = ?, ***deltaS*** = ?, ***L*** = <neue Liste>)

Timer starten ***Senden***(***deltaS***)

Nachricht IDENT(***ID***, ***Zähler***, NULL, NULL) versenden

Übergang in Zustand **L** .

Sonst:

Timer starten ***Erwarten***(3****deltaE***)

Übergang in Zustand **F**.

Aufwand:

5 Variablen initialisieren, 1 Timer starten, 1 Nachricht senden

Ereignis **Fehler** [*Wahl-Agent meldet über die Schnittstelle den erfolglosen Abschluss der Wahl*]

Erläuterung:

Wegen der offensichtlichen Probleme erklärt sich jeder selbst zum Leuchtturm

Zustand **E** [*Wahl*]

Beschreibung:

- : [*keine Bedingung*]

Variablen initialisieren (***Zähler*** := 1, ***Lh*** := (***ID***, 1, *Zeit*), ***Nachfolger*** = NULL, ***Partner*** = NULL, ***deltaE*** = ?, ***deltaS*** = ?, ***L*** = <neue Liste>)

Timer starten Senden(***deltaS***)

Nachricht IDENT(***ID***, ***Zähler***, NULL, NULL) versenden

Übergang in Zustand **L**

Aufwand:

9 Variablen initialisieren, 1 Timer starten, 1 Nachricht senden

Ereignis **Tabelle** [*Eine Anwendung fordert über die Schnittstelle die Partitionstabelle an*]

Erläuterung:

Das ist der eigentliche Zweck des Protokolls.

Zustände **L,D** [*Knoten ist Leuchtturm*]

Beschreibung:
- : [*keine Bedingung*]
  Tabelle zurückgeben

Aufwand:
1 Listendurchlauf

Zustand **N** [*Knoten ist nicht Leuchtturm*]

Beschreibung:
- : [*keine Bedingung*]
  Kennung := Zähler+1;
  Timer ***Erwarten***(***deltaE***) starten
  Nachricht QUERY(***Lh.ID***, ***UID***, ***Zähler***) versenden
  Übergang in Zustand **N**

Aufwand:
5 Variablen lesen/schreiben, 1 Timer starten, 1 Nachricht senden

Ereignis **Timer** [*Timeout eines Timers*]

Erläuterung:
Timeouts stoßen stets Prozeduren an.

Zustand **L** [*Knoten ist Leuchtturm*]

Erläuterung:
Leuchttürme haben mindestens einen Timer für das Versenden von IDENT-Nachrichten. Sie können einen zweiten Timer haben, der ***Partner***-Informationen ungültig macht, wenn sie zu alt werden.

Beschreibung:
Falls Timer ***Senden***:
  L durchlaufen, dabei alle Markierungen ***neu***:=false setzen und zählen
  alle Einträge löschen, deren ***L.Info[KnotenID].Zeit*** länger als ***deltaE*** zurückliegt und zählen
  Grad der Veränderung berechnen
  Auf Grundlage der neuen Liste einen Phantomnachfolger wählen
  Nachfolger:=(*PhantomID*, 0, *Zeit*) setzen
  Nachricht IDENT(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
Falls Timer ***Erwarten***:
  Partner:=NULL

Aufwand:

6 Variablen lesen/schreiben, 2 Listendurchläufe, 1 Quasi-Wahlalgorithmus, 1 Nachricht senden, 1 Timer starten

Zustand **N** [*Knoten ist nicht Leuchtturm*]

Erläuterung:

Ein Nicht-Leuchtturm erwartet beim Timeout zunächst IDENT-Nachrichten des Nachfolgers. Bleiben sie aus, wird er Leuchtturm.

Beschreibung:

Falls ***Nachfolger***!=NULL: [*erste Runde*]
  Lh:=***Nachfolger***; ***Nachfolger***:=NULL
  Timer ***Erwarten***(***deltaE***) starten
  Übergang in Zustand **N**
Sonst: [*zweite Runde*]
  Variablen initialisieren (***Lh*** := (***ID***, ***Zähler***, *Zeit*), ***Nachfolger*** = NULL, ***Partner*** = NULL, ***deltaE*** = ?, ***deltaS*** = ?, ***L*** = <neue Liste>)
  Timer starten Senden(***deltaS***)
  Nachricht IDENT(***ID***, ***Zähler***, NULL, NULL) versenden
  Übergang in Zustand **L**

Aufwand:

9 Variablen lesen/schreiben, 1 Timer starten, 1 Nachricht senden

Zustand **D** [*Knoten ist Leuchtturm und an einen Wahlvorschlag gebunden*]

Erläuterung:

Der Leuchtturm sendet statt IDENT-Nachrichten PROPOSE-Nachrichten.

Beschreibung:

Falls Timer ***Senden***:
  L durchlaufen, dabei alle Markierungen ***neu***:=false setzen und zählen
  alle Einträge löschen, deren ***L.Info[KnotenID].Zeit*** länger als ***deltaE*** zurückliegt und zählen
  Grad der Veränderung berechnen
  Auf Grundlage der neuen Liste einen Phantomnachfolger wählen
  Nachfolger:=(*PhantomID*, 0, *Zeit*) setzen
  Nachricht PROPOSE(***ID***, ***SeqNo***, ***Partner.ID***, ***Nachfolger.ID***) versenden
Falls Timer ***Erwarten***:
  Partner:=NULL

Aufwand:

6 Variablen lesen/schreiben, 2 Listendurchläufe, 1 Quasi-Wahlalgorithmus

Zustand **F** [*Knoten ist nicht Leuchtturm und wartet nach Wahl auf Anschluss an seine Partition*]

Erläuterung:

Ein Nicht-Leuchtturm wartet nach der Wahl zu lange auf IDENT-Nachrichten des Leuchtturms und erklärt sich selbst zu einem solchen.

Beschreibung:

- : [keine Bedingung]

Variablen initialisieren (***Lh*** := (***ID***, ***Zähler***, *Zeit*), ***Nachfolger*** = NULL, ***Partner*** = NULL, ***deltaE*** = ?, ***deltaS*** = ?, ***L*** = <neue Liste>)

Timer starten Senden(***deltaS***)

Nachricht IDENT(***ID***, ***Zähler***, NULL, NULL) versenden

Übergang in Zustand **L**

Aufwand:

9 Variablen lesen/schreiben, 1 Timer starten, 1 Nachricht senden

Abschnitt 5.3 – Analyse

Sei n die Zahl der Knoten in der Partition. Folgende Abkürzungen stehen für Ereignisse: **S** für Start, **Z** für Stop, **I** für Ident, **A** für Ack, **Q** für Query, **U** für Status, **P** für Propose, **E** für Elect, **W** für eine Wahlnachricht, **T** für ein Timeout, **F** für Fertig, **B** für Tabelle, **X** für Fehler. Subskripte zeigen den Zustand an, in dem sie beim Knoten eintreten.

Sei m der Aufwand für einen Speicherzugriff. Sei s der Aufwand für das Senden einer Nachricht und r für das Empfangen.

Sei E_x der Aufwand für das Netz, mit $x \in \{Einschalten, Betrieb, Verbinden, Anfrage\}$, wobei *Einschalten* und *Betrieb* für eine Partition, *Verbinden* für zwei Partitionen und *Anfrage* für einen Knoten und seine Partition betrachtet werden.

Für die Analyse wird optimales Routing für Broadcast- und Unicast-Nachrichten vorausgesetzt.

Variablen	***n***	Zahl der Knoten
Aufwand	***m***	Speicherzugriff
	s	Senden einer Nachricht
	r	Empfangen einer Nachricht
	t	eine Zeiteinheit Timer
	w	Wahlalgorithmus (komplett)
	E_x	Aufwand für x (siehe Text)
Intervalle	***dS***	Sendeintervall
	dE	Erwartungsintervall
	dV	Bindung an Wahlvorschlag
	C	$= \frac{dE + dV}{dS}$
Ereignisse	***S, Z, I, A, Q, U, P, E, W, T, F, B, X***	Abkürzungen für Ereignisse
	I_L	Indizes: Zustand des Knotens

Tabelle 7: Bezeichnervereinbarungen für Analyse

Einschalten und Bilden einer Partition

Beim Start gibt es n Startereignisse. Danach können sich nur je zwei Partitionen in einem Wahlgang zusammenschließen; maximal gibt es n-1 Wahlvorgänge. Ein Leuchtturm geht längstens nach der Dauer dE bei Ankunft der zweiten Nachricht des Partners in Zustand **D**, wo er maximal für die Dauer von dV bleibt. Während der ganzen Zeit gibt es Sendetimerereignisse, die bei den n-2 .. 0 anderen Leuchttürmen (plus dem Partner) Ident-Ereignisse auslösen (und zwar von beiden Leuchttürmen)

$$(E_1) \qquad n*S+(n-1)*2*C*T_L+(n-1)*n*C*I_L \;=\; n*S \;+\; C*(n-1)*(T_L*2+I_L*n)$$

Außer diesen Leuchttürmen gibt es noch n-2 ... 1 Leuchttürme, die während der ganzen Zeit Sendetimerereignisse haben; diese erzeugen bei den anderen n-3 ... 0 Leuchttürmen Ident-Ereignisse.

$$\frac{(n-1)(n-2)}{2}*C*T_L \;+\; \frac{n^3+6n^2+11n-6}{3}*C*I_L$$

$$(E_2) \qquad = \; C*(T_L*\frac{(n-1)(n-2)}{2} \;+\; I_L*\frac{n^3+6n^2+11n-6}{3})$$

Desgleichen existieren *0 .. n-2* Nichtleuchttürme, die in *dS*-Abständen Ident-Ereignisse haben und während der ganzen Zeit einen Timer unterhalten. Das führt bei ihrem Leuchtturm zu **Ack**-Ereignissen.

$$C*(I_N+t+A_L)*\frac{(n-1)(n-2)}{2} \qquad (E_3)$$

Die n-1 Wahlen erzeugen jeweils bei allen 0 ... n-2 Nichtleuchttürmen Elect-Ereignisse, die bei den beiden Leuchttürmen gerade eingetreten sind; Dann wählen 2 ... n Knoten, von denen alle ein Fertig-Ereignis haben, aber nur einer der Leuchtturm wird.

$$E_N*\frac{(n-1)(n-2)}{2}+E_L*2*(n-1) \ + \ w*(\frac{(n+1)*n}{2}+1) \ + \ F_N*(\frac{(n+1)*n}{2}) \ + \ F_L \qquad (E_3)$$

$E_{Einschalten}$= $E_1+E_2+E_3+E_4$

Betrieb

Der Betrieb wird durch Timer-Ereignisse beim Leuchtturm getaktet. Jedes Timer-Ereignis erzeugt Ident-Ereignisse bei allen Nicht-Leuchttürmen und jedes Ident-Ereignis wiederum ein Ack-Ereignis beim Leuchtturm. Sei ***d*** die Dauer des Betriebes ohne Veränderungen an der Partition.

$E_{Betrieb}$ = $(T_L+(n-1)*I_N+(n-1)*A_L)*\frac{d}{dS}$

Verbindung von Partitionen

Die Verbindung von Partitionen setzt sich aus dem Betrieb wie bisher zusammen und aus neuen Komponenten. Sei k die Knotenanzahl der neuen Partition. Die beiden Partitionen werden wie bisher betrieben, außerdem hören auch alle Knoten in der anderen Partition die IDENT-Nachichten, insbesondere auch die Leuchttürme.

$$2*\ (T_L \ + \ (k+n-2)*(I_N+A_L) \ + \ I_L)*\frac{dE}{dS} \qquad (E_5)$$

Nach dem Wahlvorschlag treten Propose-Ereignisse auf.

$$2*\ (T_L \ + \ (k+n-2)*(P_N+A_L) \ + \ P_L)*\frac{dV}{dS} \qquad (E_6)$$

Bei jedem Knoten kann nur einmal das Wahl- und das Fertig-Ereignis eintreten. Überdies bleibt ein Leuchtturm übrig.

$$2*(E_N*(n+k-1) \ + \ E_L) \ + \ w*(n+k) \ + \ F_N*(n+k-1) \ + \ F_L \qquad (E_7)$$

$E_{Verbinden}$ = $E_5+E_6+E_7$

Trennen von Partitionen

Das Trennen von Partitionen erfordert nichts von der abgebenden Partition. Für die ausscheidende Partition gelten die Regeln zum Einschalten.

Anfrage und Beantwortung

Anfragen werden neben dem normalen Betrieb beantwortet. Zuerst gibt es ein Tabelle-Ereignis, und falls der Knoten kein Leuchtturm war, ein Query-Ereignis beim Leuchtturm, und ein Status-Ereignis bei allen Knoten der Partition.

$\mathbf{E_{Anfrage}} =$ $$B_N + A_L + U*(n-1)$$

Abschnitt 5.4 – Optimierung

Die in diesem Kapitel vorgestellte Implementierung enthält bereits einige Optimierungen:

- verkürzte Wartezeit auf potentielle Partner in speziellen Fällen
- beim Ausfall des Leuchtturms wird eine Neuwahl vermieden, wenn sich die Knoten auf den Stellvertreter als neuem Leuchtturm einigen.

Das Protokollverhalten an sich kann optimiert werden, indem man die Intervalle verändert, in denen IDENT-Nachrichten gesendet und erwartet werden.

IDENT-Nachrichten öfter zu senden führt zu einer schnelleren Informationsausbreitung und damit aktuelleren und genaueren Sichten auf die Partition. Andererseits steigt das Nachrichtenaufkommen drastisch, denn IDENT-Nachrichten werden per Broadcast versendet. Auftretende Kollisionen können das erfolgreiche Bilden einer Partition wieder verzögern. IDENT-Nachrichten seltener zu erwarten führt zu einem nachsichtigeren Verhalten des Protokolls: zeitweilig verlorengegangene Verbindungen werden übersehen und führen zu einer weniger genauen, dafür stabileren Sicht auf die Partition. Es darf nicht übersehen werden, dass der Verlust der Verbindung lediglich durch das Ausbleiben der ACK-Nachricht festgestellt werden kann, was auch auf Verzögerungen beim eingesetzten Routingprotokoll zurückgeführt werden kann.

Die Intervalle können automatisiert angepasst werden, indem man sie an Häufigkeit und Ausmaß bisheriger Partitionsveränderungen koppelt, wenn die Standardabweichung dieser Größen über die Zeit gesehen nicht zu groß ist.

Eine denkbare Optimierung betrifft die Beantwortung von QUERY-Nachrichten. In der vorgestellten Implementierung werden STATUS-Nachrichten als Broadcast versendet, da ihre Erstellung beim Leuchtturm einigen Aufwand verursacht und angenommen wird, dass viele Teilnehmer diese Information „im Vorübergehen“ auswerten werden, auch wenn kein dringender Bedarf besteht. Erweist sich diese Annahme als unzutreffend, so kann STATUS als Unicast versendet werden.

Falls mehrere Knoten gleichzeitig eine STATUS-Nachricht anfordern, wären mehrere Broadcasts STATUS-Nachrichten in kurzen Abständen die Folge bzw. mehrere Unicast-Nachrichten, für die jeweils von neuem die Liste teilnehmender Knoten bearbeitet werden muss. Hier bietet sich folgende Optimierung an: Der Leuchtturm sammelt eingehende QUERY-Nachrichten, bis ein Intervall abgelaufen ist oder ein gewisser Anteil aller Knoten Interesse bekundet hat – je nachdem, was früher eintritt. Danach wird die Liste generiert und per Broadcast an alle oder per multiplem Unicast an wenige Interessenten versendet. Das erfordert zusätzlichen Speicherplatz beim Leuchtturm und setzt voraus, dass Knoten auf ihre Sicht auf die Partition eine Zeit warten können, die weit über die normale Nachrichtenumlaufzeit hinausgeht.

Abschnitt 5.5 – Bewertung

Der eindeutige Vorteil des Verfahrens – insbesondere gegenüber dem vorher vorgestellten k-hop Nachbarschaftsverfahren – liegt darin, dass eine vollständige Sicht auf die Partition möglich ist.

Weitere Vorteile sind die folgenden:

- die Tabelle wird zentral verwaltet; Kommunikation marginaler Veränderung kann unterdrückt werden
- die Partition besitzt eine Identität durch den Leuchtturm

Einige Nachteile des Verfahrens:

- ein komplizierterer Algorithmus, besonders durch die Verwaltung von Stati
- Belastung eines Knotens mit Verwaltungsaufgaben
- Sicht des Knotens auf seine Nachbarschaft kann von der Sicht des Leuchtturms abweichen
- lange Rüstzeiten bis zur Partitionsbildung bei der Initialisierung des Verfahrens, die von der Anzahl der beteiligten Knoten abhängt
- Rechenbelastung der Knoten und Nachrichtenaufkommen für das Wahlverfahren

Die Entscheidung der Implementierung ist dennoch für das Leuchtturmverfahren gefallen, und zwar aus folgenden Gründen:

1. Der Algorithmus ist mit Parametern abstimmbar und besitzt kaum reaktive Elemente, wodurch das erwartete Nachrichtenaufkommen relativ konstant ist. Die Vermeidung von Broadcast-Stürmen ist ein Qualitätsmerkmal von Protokollen.
2. Der Algorithmus ermöglicht eine Sicht auf die komplette Partition.
3. Ansatz, Implementierung und Optimierungen des Verfahrens sind auf den ersten Blick nicht offensichtlich, so dass ein höherer Beitrag zum Verständnis der Thematik zu erwarten war.
4. Die Ergebnisse der Simulation sind interessanter und waren weniger vorhersagbar.

Kapitel 6

Simulation

Abschnitt 6.1 – Simulationsumgebung

Anhand der Simulation sollte festgestellt werden, ob sich das beschriebene Verfahren zur Partitionserkennung eignet und wo seine Stärken und Schwächen liegen. Das Augenmerk lag dabei auf der Genauigkeit des Verfahrens, der Geschwindigkeit der Datenausbreitung und natürlich auf dem Nachrichtenaufwand.

Die Simulation wurde für quadratische Ebenen mit den Seitenlängen 500m, 1000m und 1500m durchgeführt, und zwar einmal mit einer Knotendichte von 25 Knoten pro km² und ein weiteres Mal mit einer Knotendichte von 50 Knoten pro km². Für jede dieser Simualtionen liegen Messergebnisse bei einer maximalen Knotengeschwindigkeit von 1,4m pro Sekunde (das entspricht der Geschwindigkeit eines Fußgängers) sowie 27,7m pro Sekunde (Geschwindigkeit eines Autos im Überlandverkehr) vor. Die Knoten haben eine Sendereichweite von 250m. Die Dauer der Simulation betrug einheitlich 240 Sekunden.

Die Parameter des Protokolls wurden so eingestellt, dass Leuchttürme alle 0,2 Sekunden eine IDENT-Nachricht senden und spätestens nach einer Sekunde eine ACK-Nachricht erwarten. Auch die Nicht-Leuchttürme erwarten eine IDENT-Nachricht wenigstens jede Sekunde.

Für jede Simulation wurden mit dem Werkzeug `setdest`[20] zehn zufällige Bewegungsmuster generiert. Die Bewegungsmuster wurden mit dem Werkzeug `calcpartition`[21] auf Partitionsbildung untersucht und annotiert. `calcpartition` gehört zu einem Paket von ns2-Erweiterungen, mit denen auch das implementierte Protokoll instrumentiert wurde. So kann das Protokoll erfahren, wann es eine logische Änderung in den Partitionsdaten gegeben hat, und kann seine eigene Sicht auf das Netz damit vergleichen.

Abschnitt 6.2 – Simulationsergebnisse

Notation

Auf den Diagrammen ist mit einem Punkt der Durchschnitt gekennzeichnet. Eine dünne Linie bezeichnet die *Spannweite* vom minimalen bis zum maximalen gemessenen Wert. Die dickere Linie zeigt die errechnete *Standardabweichung* an. Der Parameter *n* ist die *Anzahl* der für die Berechnung herangezogenen Werte .

20 Erhältlich mit ns2.

21 Siehe [18].

Geschwindigkeit der Datenausbreitung

Zwischen dem berechneten Aufeinandertreffen zweier Partitionen und dem Auslösen der Leuchtturmwahl lag bei geringer wie hoher Knotendichte und bei allen Netzdurchmessern weniger als eine Sekunde. Die Partitionen waren über den Kontakt also rechtzeitig informiert.

Die untere Grenze bildet die Nachrichtenlaufzeit der vier benötigten Nachrichten (zwei IDENT, zwei PROPOSE-Nachrichte). Eine starke Verzögerung der Neuwahl ist selten und tritt dann auf, wenn der Wahlvorschlag des Leuchtturms durch IDENT-Nachrichten des Partners aufrechterhalten wird, der Partner den Wahlvorschlag aber lange nicht mit PROPOSE beantwortet.

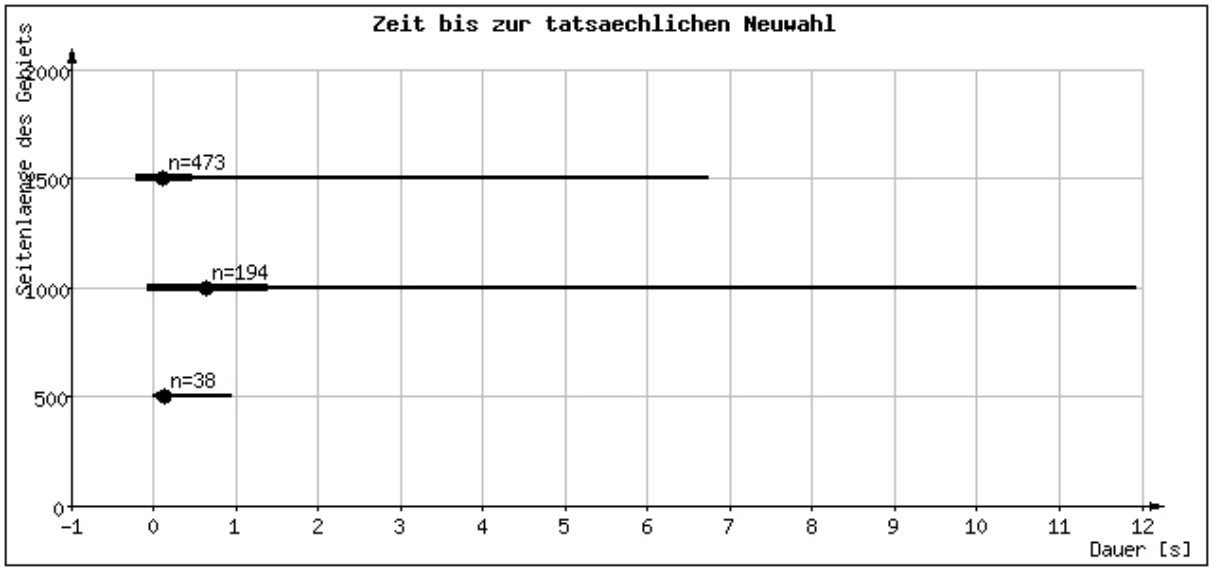

Abbildung 3: Verzögerung der Neuwahl nach Aufeinandertreffen der Partitionen, 25 Knoten/km², 100km/h

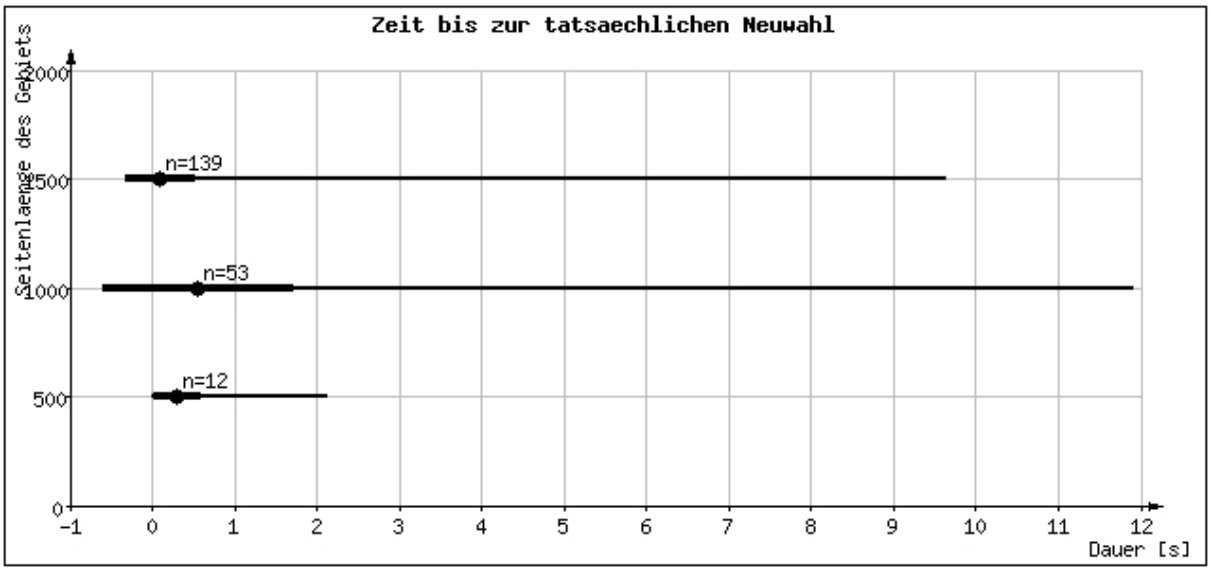

Abbildung 4: Verzögerung der Neuwahl nach Kontakt der Partitionen, 50 Knoten/km², 100km/h

Die tatsächliche Eingliederung *aller* Knoten in die neu gebildete Partition konnte jedoch lange Zeit in Anspruch nehmen – bis fast zur vollständigen Dauer der Simulation.

Abgebildet sind nur die Partitionen, die schließlich vollständig mit der berechneten Partition übereinstimmten. Bei größeren Gesamtzahlen von Knoten konnten nur wenige Partitionen vollständig beobachtet werden; bei Knotenzahlen von 50 bis 112 Knoten und einer maximalen Knotengeschwindigkeit von 100 km/h wurde gar keine Partitionsverbindung mehr vollständig beobachtet. Im Gegenzug nahm es wenig Zeit in Anspruch, eine zu 100% mit der berechneten Partition übereinstimmende Sicht auf die Partition zu erreichen, wenn dies überhaupt möglich war.

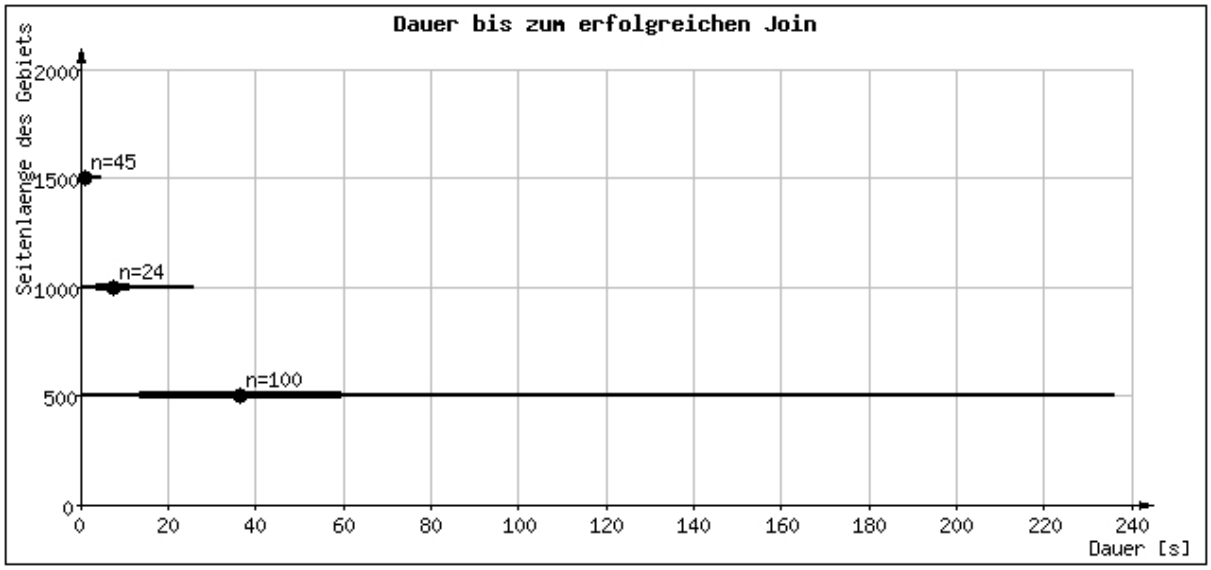

Abbildung 5: Dauer bis zur Übereinstimmung mit der berechneten Partition, 25 Knoten/km², 100 km/h

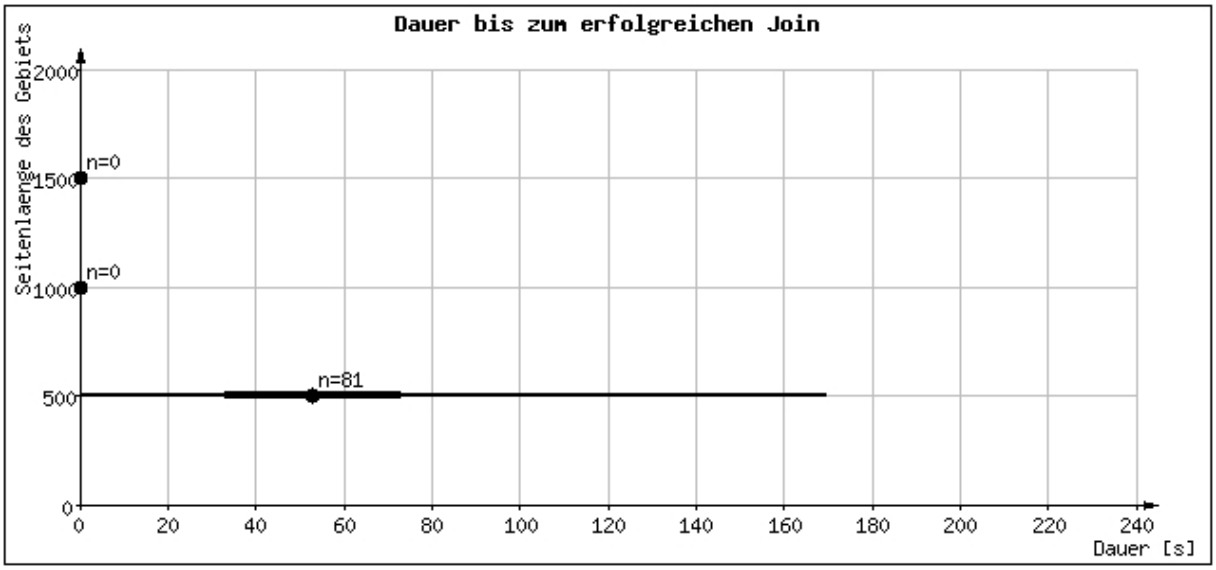

Abbildung 6: Dauer bis zur Übereinstimmung mit der berechneten Partition, 50 Knoten/km², 100km/h

Trennten sich Knoten von der Partition, nahm es erwartungsgemäß nur wenig mehr als die Timeoutzeit, bis die Trennung von der ursprünglichen Partition bemerkt wurde:

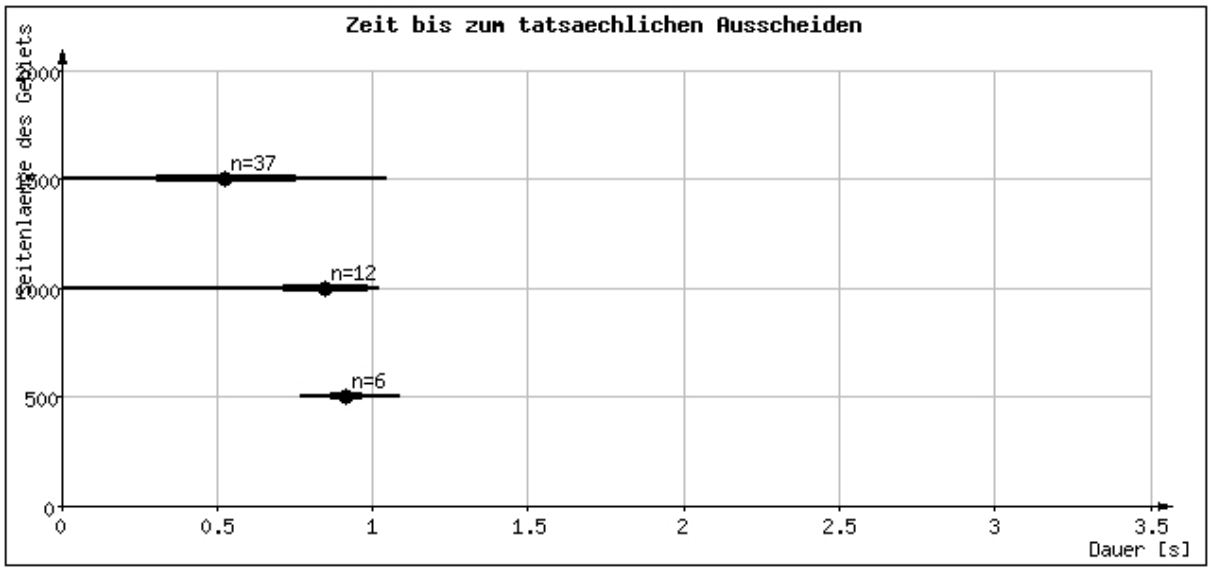

Abbildung 7: Wann sich ein ausgeschiedener Knoten zum Leuchtturm erklärt, 25 Knoten/km², 5km/h

Bei höherer Mobilität ging die Trennung im Durchschnitt schneller vonstatten. Der Grund ist, dass wegen der hohen Mobilität zum Zeitpunkt der errechneten Trennung von der Partition tatsächlich schon eine Trennung vom Leuchtturm erfolgt war und die Zeit bis zur Trennung mit 0 veranschlagt wurde.

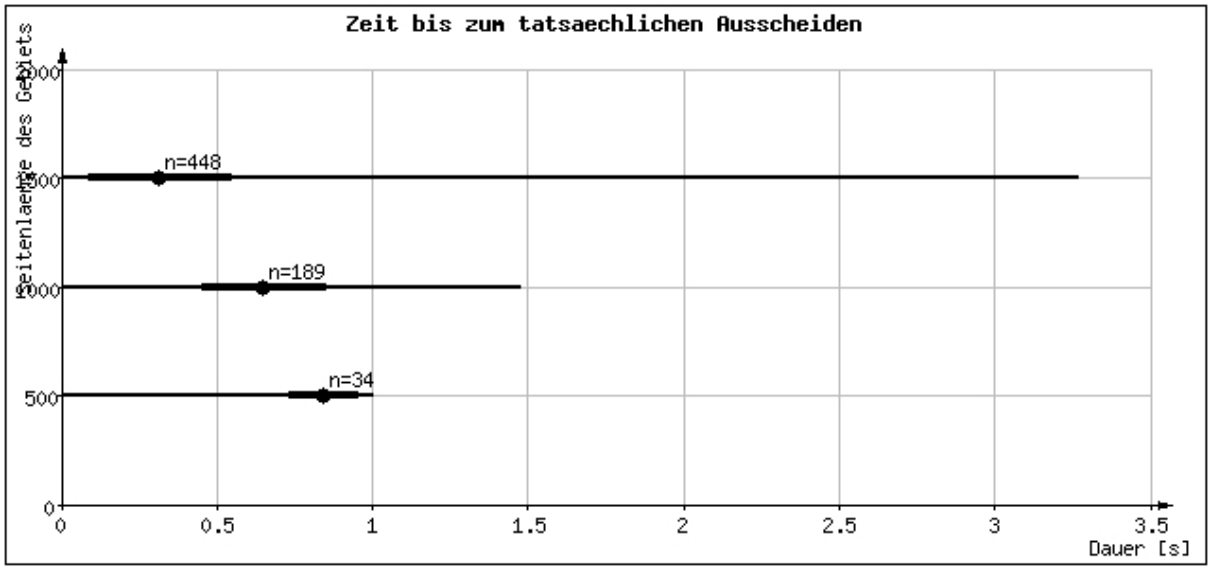

Abbildung 8: Wann sich ein ausgeschiedener Knoten zum Leuchtturm erklärt, 25 Knoten/km², 100 km/h

Es scheint nicht viel länger zu dauern, bis die beiden neu gebildeten Partitionen eine jeweils zu 100 Prozent mit den errechneten Mengen übereinstimmende Sicht aufgebaut hatten. Das liegt daran, dass sich oft nur einzelne Knoten oder sehr kleine Knotenmengen abspalten, so dass eine vollständige Beobachtung der Partition bei der ausgeschiedenen wie bei der verlassenen Partition sehr rasch wieder möglich war.[22]

Allerdings gelang die vollständige Beobachtung der neuen Partitionen nur in einem kleinen Teil der Fälle (vgl. mit den Zahlen ausgeschiedener Knoten, die neue Leuchttürme wurden, auf der vorigen Seite).

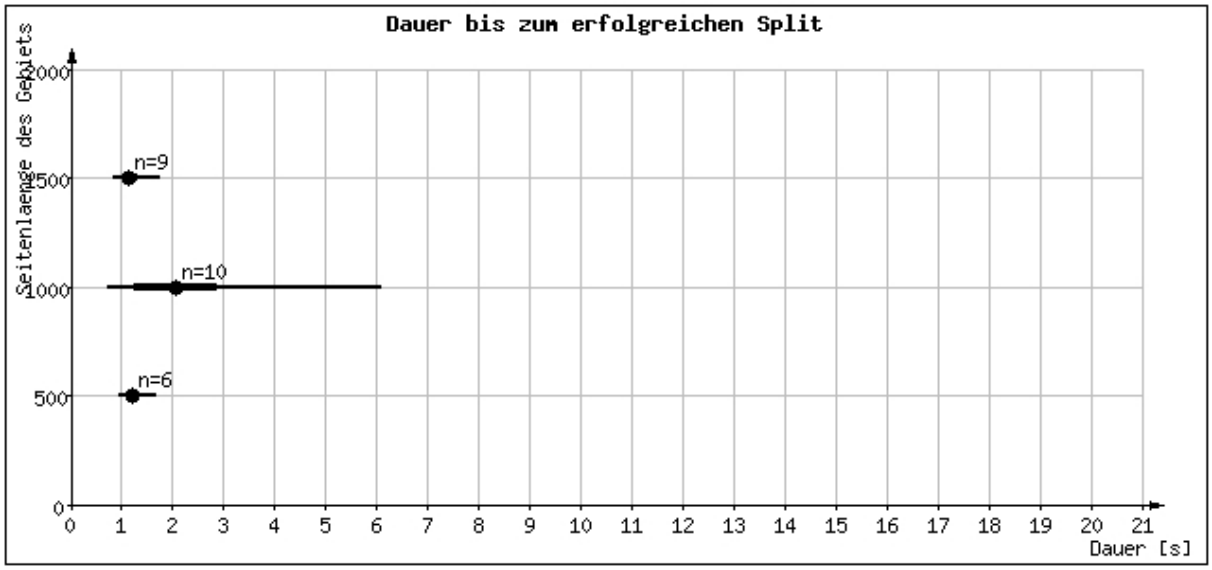

Abbildung 9: Dauer bis zur vollständigen Trennung der Partitionen, 25 Knoten/km², 5 km/h

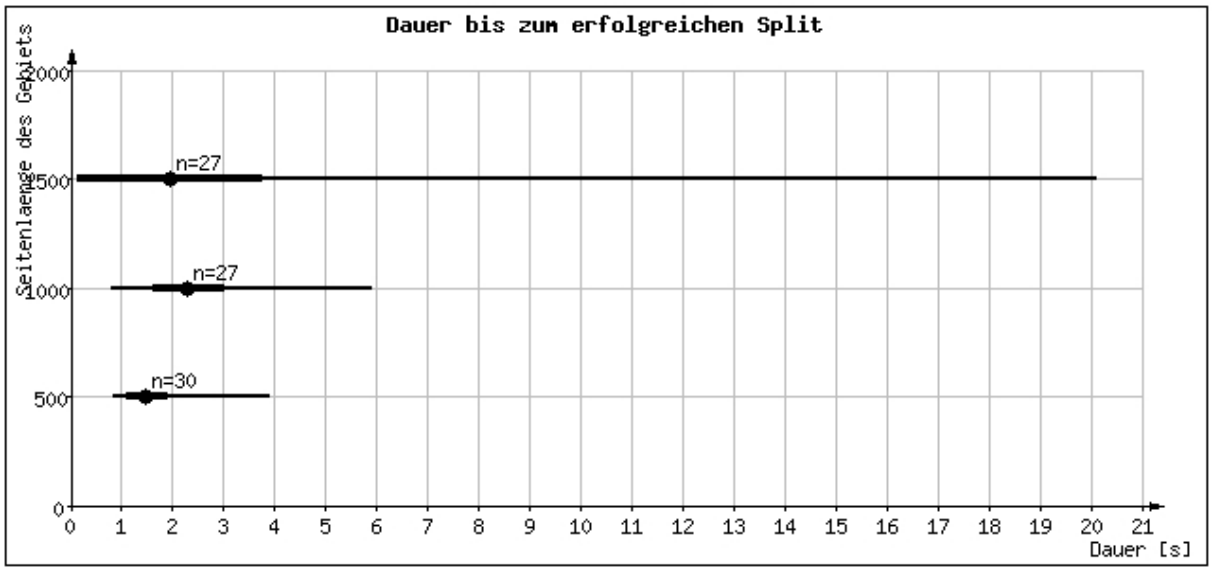

Abbildung 10: Dauer bis zur vollständigen Trennung der Partitionen, 25 Knoten/km², 100 km/h

22 Das stimmt mit der Beobachtung in [1] überein.

Genauigkeit der ermittelten Partitionsdaten

Die mittlere Übereinstimmung der vom Protokoll ermittelten Informationen mit den berechneten Partitionsdaten nimmt erwartungsgemäß mit der Zahl der teilnehmenden Knoten, wegen des Nachrichtenaufkommens auch mit zunehmender Knotendichte und natürlich mit zunehmender Mobilität ab. Unter günstigen Bedingungen sind 94% Übereinstimmung zu erreichen – gerechnet über die Lebensdauer der Partition – bei hoher, ungerichteter Mobilität kommt man nur auf knapp 42%. Da ACK-Nachrichten mit AODV geroutet zum Leuchtturm gesendet werden, ist dabei die Zeit zu berücksichtigen, die AODV benötigt, um eine verlorengegangene Route zu bemerken und eine neue ausfindig zu machen.

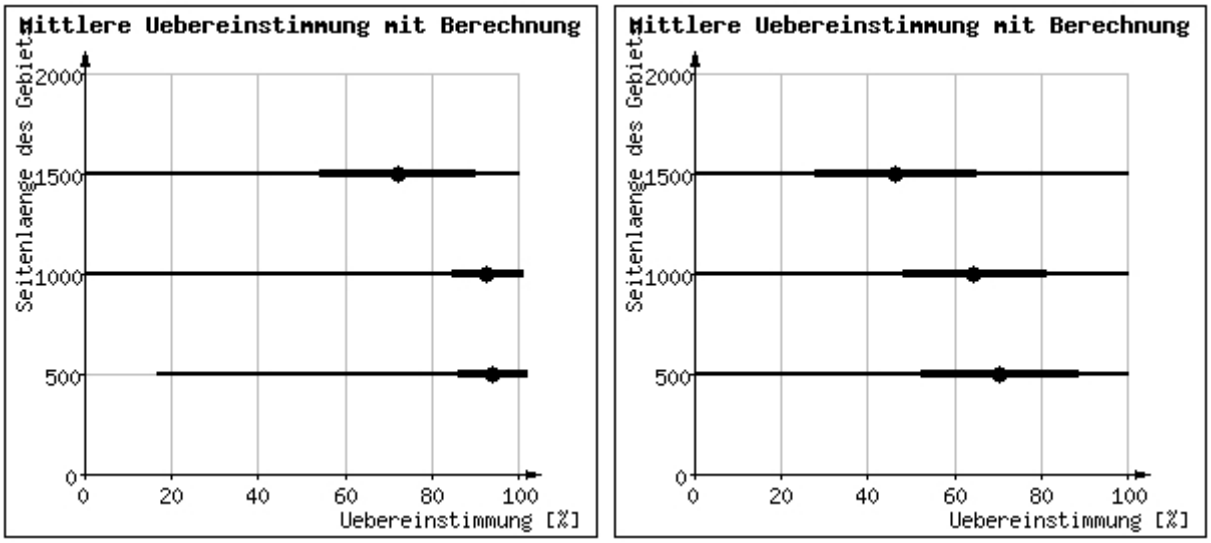

Abbildung 11: Mittlere Genauigkeit, 25 Knoten/km², links bei 5 km/h, rechts bei 100 km/h

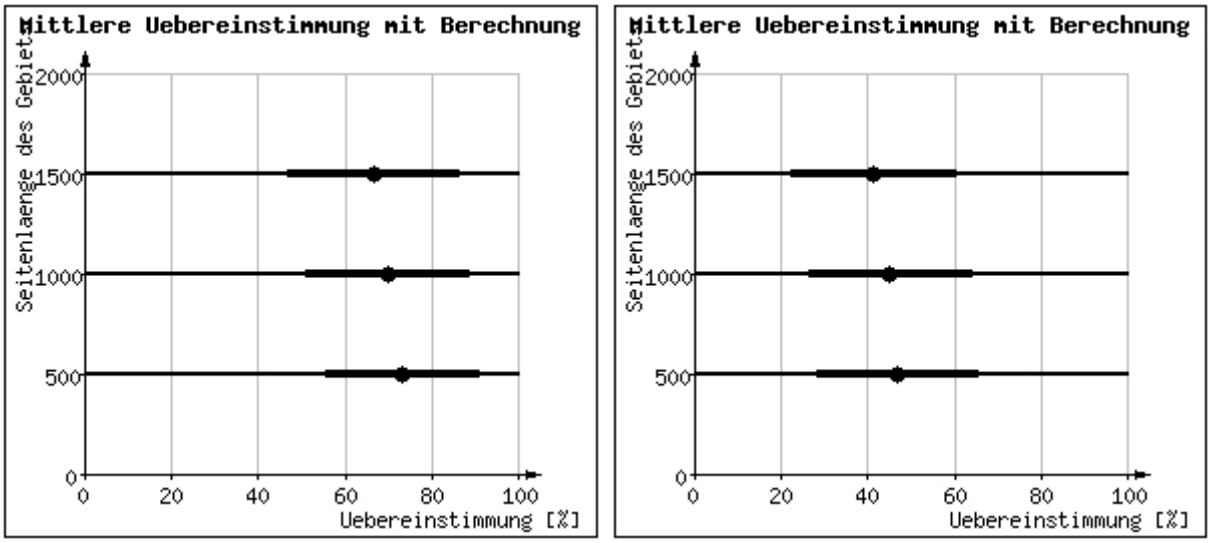

Abbildung 12: Mittlere Genauigkeit, 50 Knoten/km², links bei 5 km/h, rechts bei 100 km/h

Interessant ist zu bestimmen, wie lange eine bestimmten prozentuale Übereinstimmung der tatsächlichen Sicht mit der berechneten Konfiguration erreicht wurde. Die folgenden Diagramme zeigen die Durchschnitte über alle Partitionen. Auf der y-Achse steht der Grad der Übereinstimmung, auf der x-Achse die durchschnittliche Lebensdauer von Sichten, die zu diesem Grad mit der berechneten Konfiguration übereinstimmten.

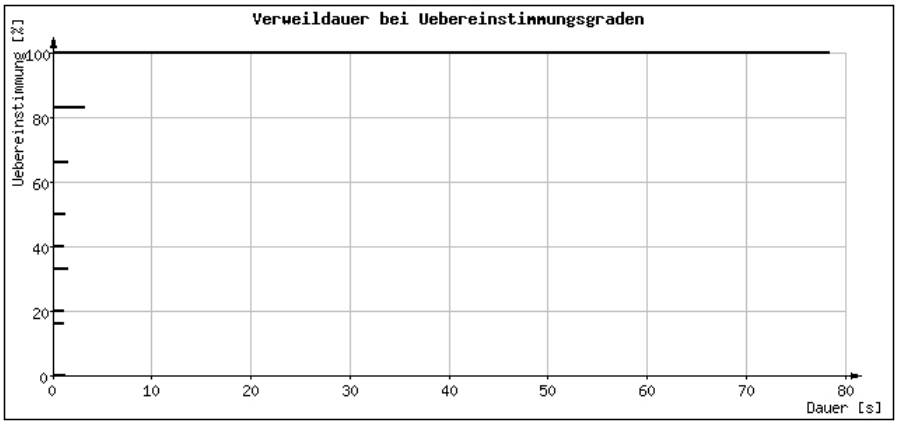

Abbildung 13: Dauer der Übereinstimmung, 6 Knoten, 25 Knoten/km², 5 km/h

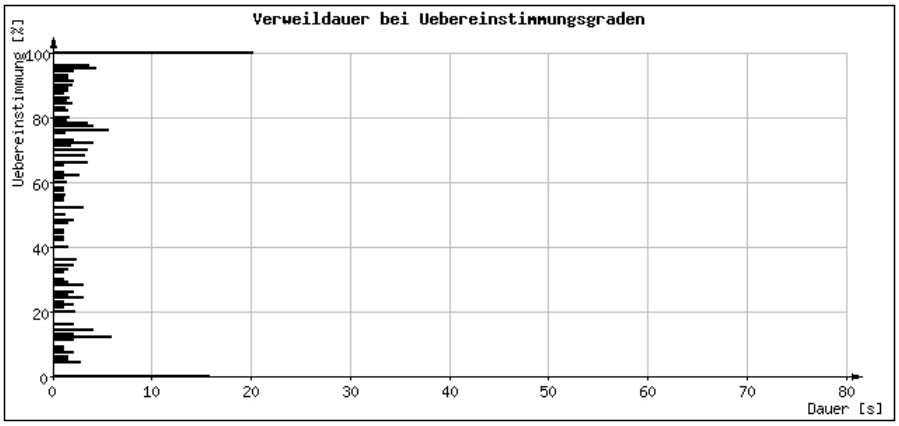

Abbildung 14: Dauer der Übereinstimmung, 25 Knoten, 25 Knoten/km², 5 km/h

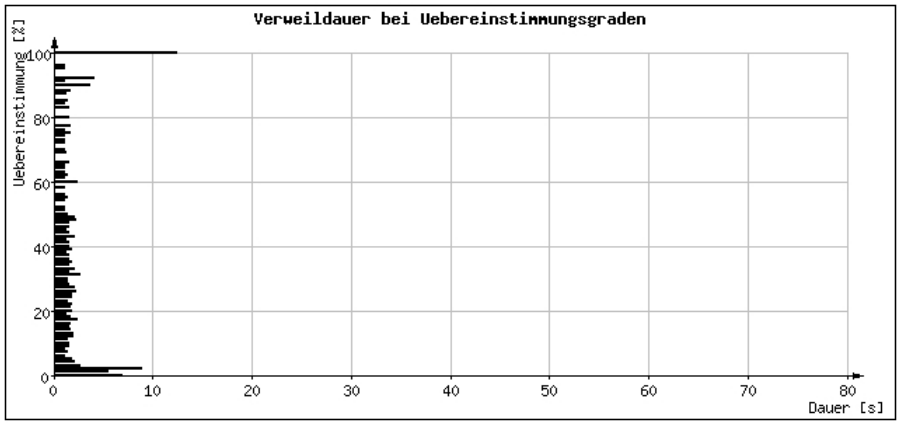

Abbildung 15: Dauer der Übereinstimmung, 56 Knoten, 25 Knoten/km², 5 km/h

Die Ergebnisse sind akzeptabel bei geringer Knotendichte und geringer Mobilität, aber nicht verwertbar bei hoher Dichte und hoher Mobilität.

Nachrichtenaufkommen

Wie zu erwarten steigt die Anzahl der gesendeten Nachrichten mit steigender Knotendichte, wachsender Knotenzahl und zunehmender Mobilität leicht an. Da das Verfahren vorrangig auf Timern basiert, fällt der Anstieg jedoch sehr moderat aus. Nicht-Leuchttürme und Leuchttürme senden fast dieselbe Anzahl Nachrichten – entweder IDENT- oder ACK-Nachrichten.

Die Simulation wurde 240 Sekunden lang durchgeführt, die Dauer zwischen zwei IDENT-Nachrichten betrug 0,2 Sekunden, es waren daher 1200 IDENT- Nachrichten (bei Leuchttürmen) und ebensoviele ACK-Nachrichten (bei Nicht-Leuchttürmen) zu erwarten.

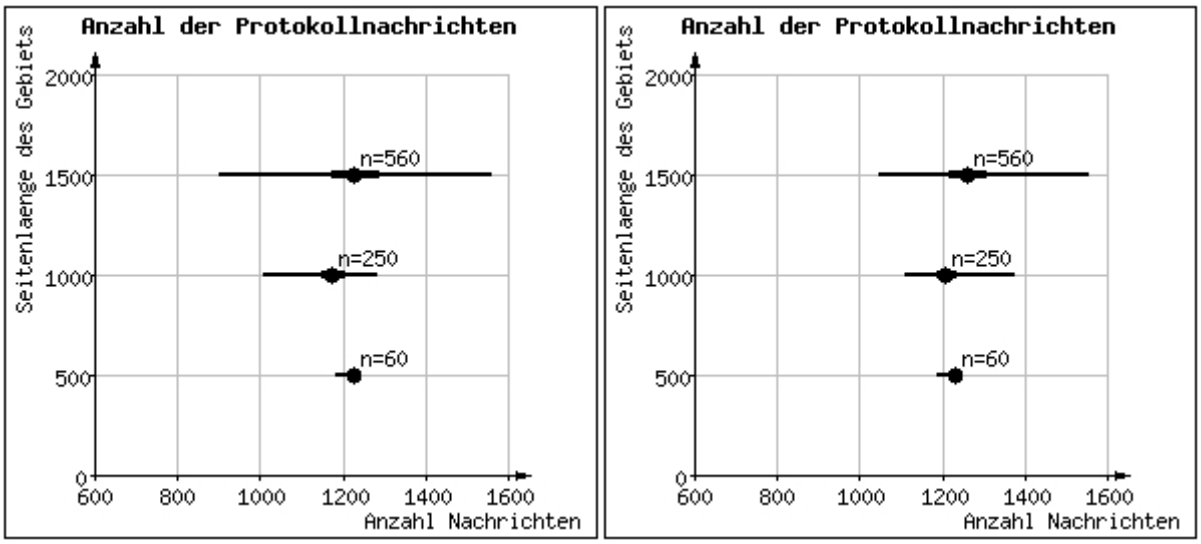

Abbildung 16: Durchschnittlich gesendete Protokollnachrichten, 25 Knoten/km², links 5km/h, rechts 100km/h

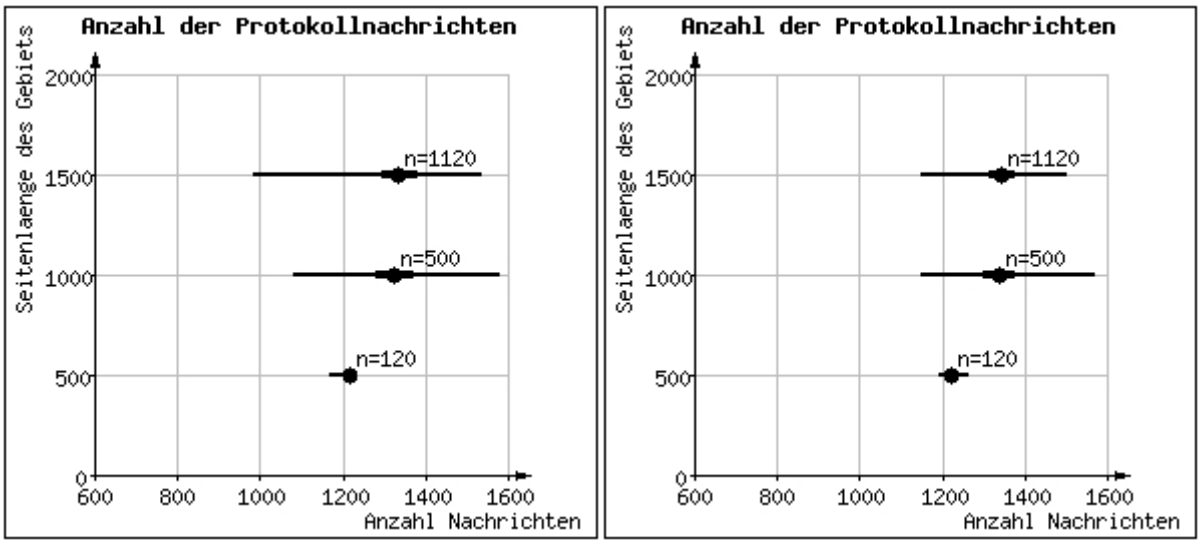

Abbildung 17: Durchschnittlich gesendete Protokollnachrichten, 50 Knoten/km², links 5km/h, rechts 100km/h

Während der Leuchtturm viele ACK-Nachrichten empfangen muss, werden die anderen Knoten mit Weiterleitungsaufgaben belastet. Hier ist auch die Zunahme des Verkehrs durch die höhere Knotenanzahl viel deutlicher zu erkennen, obwohl dieselben Parameter verwendet wurden wie auf der Seite zuvor.

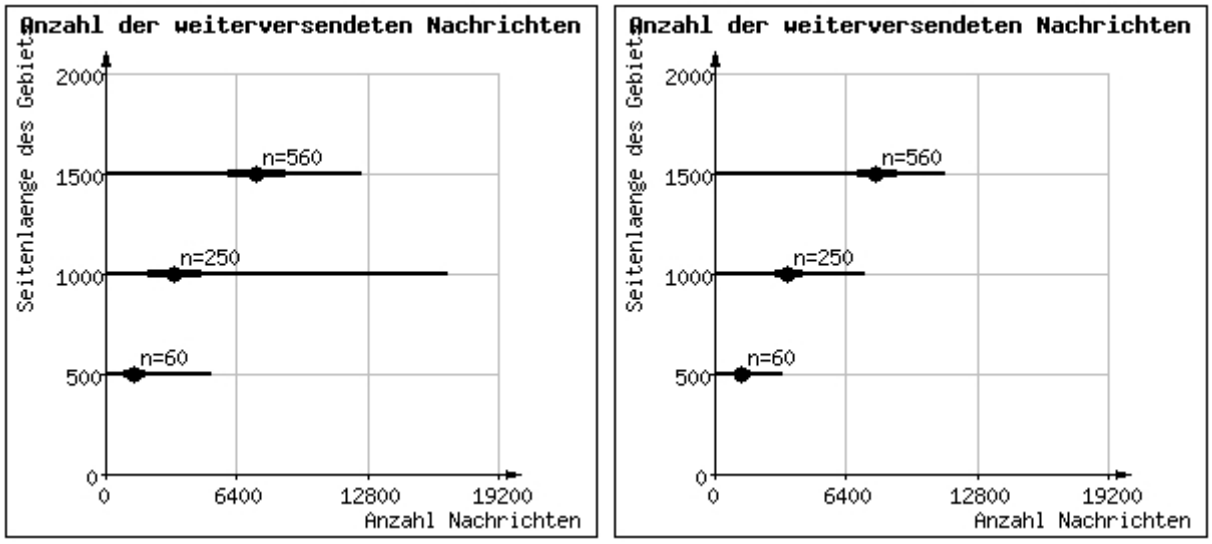

Abbildung 18: Anzahl der pro Knoten weitergeleiteten Nachrichten, 25 Knoten/km², links 5 km/h, rechts 100 km/h

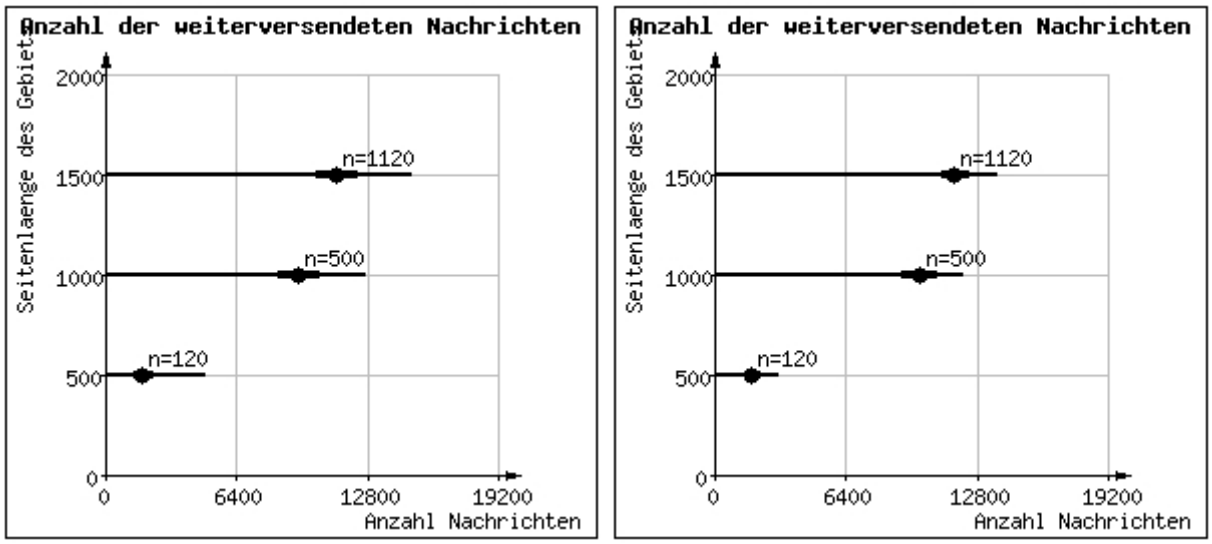

Abbildung 19: Anzahl der pro Knoten weitergeleiteten Nachrichten, 50 Knoten/km², 100 km/h

Diese Ergebnisse legen nahe, die beschriebenen Optimierungsmöglichkeiten bei mit Broadcast versendeten Nachrichten – das sind IDENT, PROPOSE und ELECT – auszunutzen.

Kapitel 7

Abschluss und Ausblick

In dieser Arbeit wurde zuerst beleuchtet, dass bei der generellen Betrachtung mobiler Netze nur eine sehr eingeschränkte Form von Partitionserkennung ermöglicht werden kann. Es wurde darauf eingegangen, welche Designentscheidungen getroffen werden können, wenn ein Verfahren implementiert werden soll. Zwei sehr gegensätzliche Verfahren zur Erhebung von Informationen über die eigene Partition wurden vorgestellt: k-hop Nachbarschaft, ein verteiltes, zustandsloses Verfahren, das noch umfangreich optimiert werden kann; andererseits das Leuchtturmmodell, ein zentralisiertes Verfahren mit Zuständen, verschiedenen Nachrichtentypen und Parametern. Detailangaben zur Implementierung des Leuchtturmmodells folgten.

In der Simulation wurde deutlich, dass das Leuchtturmmodell nur bei geringen Knotendichten, geringen Knotenzahlen und geringer Mobilität zu akzeptablen Ergebnissen kam. Das kann eine Reihe von Gründen haben:

– Möglicherweise waren die Sende- und Empfangsintervalle mit 0,2 und 1,0 Sekunden nicht optimal eingestellt. Es wäre eine weitere Aufgabe herauszufinden, wo das Optimum der zwei Größen, auch im Verhältnis zueinander, liegt. Lange Intervalle überdecken kurzfristige Veränderungen, führen aber zu einer nicht zu vernachlässigenden kumulativen Verzögerung. So mag es im allgemeinen günstig sein, eine Neuwahl nicht sofort bei dem Kontakt mit einer anderen Partition zu veranlassen, um kurzfristige Kontakte zu übergehen. Bei Initialisierungsvorgängen,

bei denen immer nur je 2 Partitionen miteinander fusionieren können, sollte dann jedoch die Wartedauer dynamisch reduziert werden.

- Das Protokoll ist abhängig vom Routingprotokoll für Unicast-Nachrichten. Die Kulanzzeit des Knoten gegenüber seinem Leuchtturm (und umgekehrt) addiert sich zu den Fristen des Routingprotokolles, die vergehen, bis eine nicht mehr bestehende Route aufgegeben und eine neue entdeckt wird. Die Folge ist ein Timeout auf einer oder beiden Seiten und ein erneutes Wahlverfahren.
- Es scheint sich herauszustellen, dass das Konzept nicht tragbar ist, sobald der Leuchtturm zu viele Knoten verwalten muss. Das mag an der Menge zu empfangender Nachrichten liegen oder an der aufwändigen Listenverwaltung liegen. Hier gibt es Optimierungsmöglichkeiten, wie die oben in Abschnitt 3.4 auf Seite 16 beschriebenen Broadcast- und Datensammlungsverfahren.

Aufschlussreich wäre bei einer Weiterentwicklung des Verfahrens, ob durch geschickte Verhandlung oder intelligentes Verhalten der Knoten eine Reduzierung der Wahlgänge möglich wäre, so dass sich z.B. eine kleine Partition ohne Wahl in eine große einordnet und deren Leuchtturm übernimmt.

Auch könnte man die in Kapitel 2 erwähnten verwandten Verfahren heranziehen, um durch Partitionsvorhersage besonderes Augenmerk auf gewisse Regionen zu richten, in denen gerade Veränderungen zu erwarten sind, und weitgehend stabile Bereiche nicht unnötig mit Kontrollnachrichten zu belasten.

Eine interessante Aufgabe wäre auch die Reorganisation der beschriebenen Verfahren unter dem Gesichtspunkt, byzantinische Fehler aufzuspüren und auszuschließen, bzw. sich sonstwie fehlverhaltende Knoten zumindest aus dem Partitionserkennungsprozess herauszuhalten.

Literaturverzeichnis

[1] Hähner, J., Siedkows, D., Marrón, P. J., Rothermel, K.: A quantitative analysis of partitioning in mobile ad hoc networks. In *Proceedings of the Joint international Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, June 10 - 14, 2004). SIGMETRICS 2004/PERFORMANCE 2004. ACM Press, New York, NY, 400-401.

[2] Internet Protocol Version 6 (IPv6) Specification. http://www.ietf.org/rfc/rfc2460.txt

[3] Hauspie, M.; Carle, J.; Simplot, D.: Partition detection in mobile ad-hoc networks using multiple disjoint path set. In *Proc. 1st International Workshop on Objects models and Multimedia technologies (OMMT)* (Geneva, Switzerland, 2003). http://www.lifl.fr/RD2P/download/hcs2003.pdf

[4] Conan, D.; Bhatti, M.U: Fault-tolerance in Mobile Environments: A Partition Detection System. *Proceedings of 2nd International Workshop on Frontiers of Information Technology*, Islamabad, Pakistan, December 2004

[5] Wang, K.H.; Li, B.: Group Mobility and Partition Prediction in Wireless Ad-Hoc Networks. IEEE International Conference on Communications (ICC 2002). In *Proceedings of IEEE International Conference on Communications (ICC)* (New York, NY, April 2002), Bd. 2, S. 1017-1021. http://toronto.edu/~bli/papers/icc02.pdf

[6] Milic, B.; Milanovic, N.; Malek, M.: Prediction of Partitioning in Location-Aware Mobile Ad Hoc Networks. In *Proceedings of the Proceedings of the 38th Annual Hawaii international Conference on System Sciences (Hicss'05) - Track 9 - Volume 09* (January 03 - 06, 2005). HICSS. IEEE Computer Society, Washington, DC, S. 306.3

[7] Goyal, D.; Caffery, J.: Partitioning Avoidance in Mobile Ad Hoc Networks Using Network Survivability Concepts. In *Proceedings of the Seventh international Symposium on Computers and Communications (Iscc'02)* (July 01 - 04, 2002). ISCC. IEEE Computer Society, Washington, DC, S. 553

[8] Ritter, H.; Winter, R.; Schiller, J.; Zippan, T: A partition detection system for distributed mobile games. In *Proceedings of 3rd ACM SIGCOMM Workshop: Network and System Support For Games* (Portland, Oregon, USA, August 30 - 30, 2004). NetGames '04. ACM Press, New York, NY, S. 169-169.

[9] Ni, S.Y.; Tseng, Y.C.; Chen, Y.S.; Sheu, J.P.: The broadcast storm probloem in a mobile ad hoc network. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile computing and Networking*, Seattle, Washington, USA, August 1999, S. 151-162.

[10] Hatzis, K.P.; Pentaris, G.P.; Spirakis, P.G.; Tampakas, V.T.; and Tan, R.B.: Fundamental control algorithms in mobile networks. In *Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures* (Saint Malo, France, June 27 - 30, 1999). SPAA '99. ACM Press, New York, NY, S. 251-260

[11] Perkins, C.; Royer, E.; Das, S.: Ad hoc On-demand Distance Vector (AODV) Routing, RFC 3561. www.ietf.org/rfc/rfc3561.txt

[12] Lou, W.; Wu, J.: A k-hop Zone-Based Broadcast Protocol. In *Mobile Ad Hoc Networks*. IEEE Globecom 2004, S. 1665-1669

[13] Vasudevan, S.; Kurose, J.; Towsley, D.: Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ECNP'04)*, October 05-08, 2004, S. 350 – 360

[14] Goldsmith, A.J.; Wicker, S.B.: Design Challenges for Energy-Constrained Ad Hoc Wireless Networks, IEEE Wireless Communications., Bd. 9, Nr. 4, August 2002, S. 8-27

[15] The ns Manual, Hrsg: Kevin Fall. http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

[16] The network simulator ns2. http://www.isi.edu/nsnam/ns/.

[17] Malpani, N., Welch, J. L., and Vaidya, N.: Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th international Workshop on Discrete Algorithms and Methods For Mobile Computing and Communications* (Boston, Massachusetts, United States, August 11 - 11, 2000). DIALM '00. ACM Press, New York, NY, S. 96-103.

[18] Khelil, A.; Marrón, P.J.; Dietrich, R., Rothermel, K.: Evaluation of Partition-Aware MANET Protocols and Applications with ns-2, http://canu.informatik.uni-stuttgart.de/calcpartition/ publications/khelil_partitioning_spects05.pdf