Simulative Evaluation des Shared Log Space (SLS)

Diplomarbeit

Freie Universität Berlin

Fachbereich Mathematik und Informatik

Arbeitsgruppe Datenbanken und Informationssysteme

Petra Fiedler

pfiedler@inf.fu-berlin.de

Betreuer: Prof. Dr. Heinz Schweppe,
Dipl.-Inform. Joos-Hendrik Böse

Datum: 13. Dezember 2007

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit bis auf die dem Aufgabensteller bekannten Hilfen selbständig angefertigt und alle verwendeten Hilfsmittel vollständig und genau angegeben habe.

Die Arbeit wurde bisher in dieser oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 13. Dezember 2007

(Petra Fiedler)

Zusammenfassung

Durch die verbesserte drahtlose Kommunikationstechnologien finden mobile ad hoc Netzwerke (sog. MANETs), bestehend aus Geräten wie Handys und PDAs, immer mehr Verbreitung. Bei der Anpassung von atomaren Transaktionen an die neue Umgebung kommt es aufgrund von MANET-Charakteristiken, wie z.B den beschränkten Ressourcen der Geräte und insbesondere deren Mobilität, häufig zu Fehlersituationen in der Commit-Phase.

Mit dem sog. *Shared Log Space* (kurz SLS) wird im Rahmen dieser Arbeit ein Middleware-Ansatz zur Reduzierung der Unsicherheitszeit ausgefallener Transaktionsteilnehmer durch Simulation mit dem Netzwerksimulator NS2 evaluiert. Der SLS nutzt das gesamte MANET zur Speicherung des Transaktionsausgangs für ausgefallene Teilnehmer. Durch die Verteilung des Transaktionsausgangs soll eine probabilistische Garantie für eine erfolgreiche Recovery durch den SLS gewährleistet werden. Die Ergebnisse der Simulationen zeigen, dass die vereinbarten Garantien eingehalten werden können. Bei Verwendung des SLS kann eine deutliche Reduzierung der Unsicherheitszeit ausgefallener Teilnehmer erzielt werden. Allerdings ist diese Verbesserung mit zusätzlichem Nachrichtenaufwand für den Transaktionskoordinator verbunden.

Der SLS wird außerdem um die Berücksichtigung der Bewegung von mobilen Geräten zwischen Clustern erweitert. Es wird ein host-zentriertes Verfahren zur Schätzung der individuellen Abwanderungswahrscheinlichkeit entwickelt. Dieses benötigt als Vorkonfiguration lediglich eine Karte der MANET-Cluster. Das *Lernen* des Bewegungsverhaltens mit einem Maximum-Likelihood-Schätzer erzeugt keine zusätzliche Belastung des Netzes mit Nachrichten. Zur Bewertung dieser Schätzung wird für die Berechnung der Abwanderungswahrscheinlichkeit die untere Grenze des dazu gehörigen Konfidenzintervalls verwendet. Bei der Evaluation des um diese konservative Abschätzung der Abwanderungwahrscheinlichkeit erweiterten SLS kann gezeigt werden, dass damit die vereinbarten Garantien auch in cluster-basierten Szenarien ohne erhebliche zusätzliche Nachrichtenbelastung des MANETs gewährleistet werden können.

Abstract

With the proliferation of mobile devices such as mobile phones and PDAs, mobile ad hoc networks (abbr. MANETs) become realistic. Adapting atomic transactions to the new environment is challenging due to high failure rates caused by limited energy resources and most important by mobility. Failures during the commit phase of atomic transactions hinder progress and possibly leave transaction participants uncertain about the transaction outcome for long periods.

The *Shared Log Space* (abbr. SLS) is a middleware approach to reduce uncertainty periods of failed transaction participants by disseminating the transaction outcome to remote nodes. The main contribution of this thesis is the evaluation of the SLS by means of simulation with the network simulator NS2. The SLS uses the whole MANET as a distributed storage to provide a probabilistic guarantee to allow the successful recovery for failed participants. Simulation results show that these guarantees are successfully provided by the SLS approach also in volatile scenarios. The usage of the SLS leads to the explicit reduction of uncertainty periods. However, this improvement is connected with an increased number of messages sent by the transaction coordinator.

Furthermore, this thesis extends the concept of the SLS by the consideration of mobility between clusters. A host-centric mobility prediction technique is developed to learn and estimate the individual churn rate of a mobile node. The proposed method does not require any configuration in advance and learning of the mobility pattern does not increase the network load. To assess estimated churn rates, the lower bound of the corresponding confidence interval is applied as a conservative estimation. The evaluation of the extended SLS points out that the agreed guarantees can be provided for cluster-based MANETs at very moderate message costs.

Vorwort

Die vorliegende Diplomarbeit ist bei der Arbeitsgruppe Datenbanken und Informationssysteme des Instituts für Informatik an der Freien Universität Berlin entstanden, welche u.a. Themen im Bereich des Datenmanagements in mobilen Systemen bearbeitet.

Besonderer Dank gebührt hierbei Joos-Hendrik Böse, der mich mit großer Geduld bei der Durchführung dieser Arbeit unterstützt hat. Die zahlreichen fachlichen Diskussionen, die wir gemeinsam geführt haben, waren mir eine große Hilfe.

Ebenso möchte ich Herrn Prof. Dr. Heinz Schweppe für das entgegengebrachte Interesse und die gute Unterstützung seitens der Universität danken. Darüber hinaus gilt mein Dank den Mitarbeitern und Diplomanden der Arbeitsgruppe Datenbanken für die Unterstützung und das gute Arbeitsklima.

Bei meiner Familie und meinen Freunden bedanke ich mich dafür, dass sie mich während der gesamten Studienzeit begleitet und unterstützt haben.

Berlin, im Dezember 2007

Petra Fiedler

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
2PC	2-Phasen-Commit-Protokoll
3PC	3-Phasen-Commit-Protokoll
AGBMM	Area Graph-Based Mobilitätsmodell
AODV	Ad-Hoc On-Demand Routing
ATM	Advanced Transaction Models
DirDissCA	Gerichtete Verteilung für Knoten mit Kontextbewusstsen in dichten MANETs
DirDissCUA	Gerichtete Verteilung für Knoten ohne Kontextbewusstsen in dichten MANETs
DSDV	Destination Sequence Distance Vector Routing
DSR	Dynamic Source Routing
FMM	Freeway Mobilitätsmodell
GPS	
HRPLS	Hybrid Routing Protocol for Large Scale Mobile Ad Hoc Networks with Mobile Backbones
LAR	Location Added Routing
LET	Link Expiration Time
NS2	Netzwerksimulator ns-2
MANET	Mobiles Ad Hoc Netwerk
MMM	Manhattan Mobilitätsmodell
$p_{retrieval}$	Wahrscheinlichkeit für den Erhalt des Transaktionslogs nach einem Fehler
R_{SLS}	Vereinbarte Zuverlässigkeit des SLS (bestehend aus $p_{retrieval}$ und t_m)
RET	Route Expiration Time
RGPM	Reference Point Group Mobility Modell
RWPMM	Random Waypoint Mobilitätsmodell
SLS	Shared Log Space
t_m	Garantierter Recoveryzeitraum
UnDirDiss	Ungerichtete Verteilung für dünne MANETs

Kapitel 1

Einführung

1.1 Motivation

Aufgrund der verbesserten drahtlosen Kommunikationstechnologien können sich mobile Geräte wie Handys, Laptops oder PDAs ad hoc zu Netzwerken (Mobile Ad Hoc Netzwerke, im Folgenden kurz MANETs) zusammenfinden. Diese neuen Kommunikationstechnologien erfordern jedoch auch die Anpassung bekannter Applikationen aus kabelgebundenen Netzen wie z.B. Datenbanksystemen an mobile Geräte. Eine wichtige Anforderung an mobile Anwendungen ist, dass sie dieselbe Zuverlässigkeit wie ihre Gegenstücke in fest verkabelten Netzwerken garantieren können. Im Fall von Datenbanken ist es beispielsweise wünschenswert, dass auch für Transaktionen in mobilen Netzwerken die ACID-Eigenschaften garantiert werden können. Charakteristiken von MANETs wie Verbindungsverlust, Bewegung oder begrenzte Energieressourcen führen jedoch häufig zu Ausfällen während der Interaktion von Geräten. Tritt beispielsweise in der Commit-Phase einer Transaktion aufgrund des Ausfalls eines teilnehmenden Gerätes ein Fehler auf, so müssen ggf. alle Änderungen zurückgenommen werden. Ausfälle müssen daher bei der Entwicklung von Anwendungen für mobile Geräte besonders berücksichtigt werden, so dass z.B. die aus den kabelgebundenen Netzen garantierbaren ACID-Eigenschaften abgeschwächt oder angepasst werden sollten. Insbesondere muss für die Atomarität ein neuer Ansatz entwickelt werden, da erforderlich ist, dass alle Operationen einer Transaktion entweder ganz oder gar nicht ausgeführt werden. Die Gewährleistung dieses Kriteriums ist allerdings ein Problem, wenn Geräte während der Transaktion ausfallen können. Normalerweise verbindet sich ein ausgefallenes Geräte irgendwann wieder mit dem MANET, aber es ist dann unsicher über den Ausgang der Transaktion und kann diese nicht beenden. Es muss daher sichergestellt werden, dass eine Transaktion von allen Teilnehmern erfolgreich beendet werden kann, obwohl unter Umständen ein Teilnehmer oder mehrere von ihnen ausgefallen sind. Ein Middleware-Ansatz für MANETs zur Verkürzung der Unsicherheitszeit eines Transaktionsteilnehmers nach einem Ausfall ist der sog. *Shared Log Space* (im Folgenden kurz *SLS*). Dabei wird das gesamte MANET als ein verteilter Speicher benutzt und der Ausgang einer Transaktion wird innerhalb des ganzen Netzes verteilt, falls einer der Teilnehmer ausfällt. Dadurch soll gewährleistet werden, dass ausgefallene Teilnehmer ohne lange Unsicherheitszeiten den Ausgang der Transaktion erfahren können.

1.2 Zielsetzung

Ziel ist die Evaluation des SLS durch eine umfangreiche Simulation mit dem Netzwerksimulator NS2. Mit dem SLS wird der Ausgang einer Transaktion (das sog. *Transaktionslog*) beim Ausfall eines Knotens durch Datenverteilung im MANET redundant gespeichert. Der Grad der Verteilung eines Logs hängt vorrangig von dem *garantierten Recovery-Zeitraum* t_m sowie einer Wahrscheinlichkeit für das erfolgreiche Auffinden nach einem Ausfall $p_{retrieval_{app}}$ ab. Beide Parameter werden vor der Transaktion zwischen den Teilnehmern vereinbart. Außerdem werden Charakteristiken von MANETs wie z.B. die Netzdichte, technische Ausstattung und beschränkte Energieressourcen

bei der Verteilung eines Logs berücksichtigt. Diese Aspekte werden bei den Strategien für das Wiederauffinden eines Logs durch einen ausgefallenen Transaktionsteilnehmer bedacht.

Zurzeit werden durch den SLS technische Defekte und beschränkte Energieressourcen berücksichtigt. Basierend auf den ersten Simulationsergebnissen, soll der SLS anschließend um die Berücksichtigung der Mobilität von Geräten erweitert werden. Der erweiterte SLS soll ebenfalls simulativ untersucht werden.

1.2.1 Evaluation des Shared Log Space

Zunächst soll in dieser Diplomarbeit der bestehende SLS simulativ evaluiert werden. Dazu werden zunächst der Prozessablauf des SLS sowie ein einfacher Vergleichs-Prozessablaufs ohne die besonderen Charakteristiken des SLS untersucht. Anschließend sollen durch Variation verschiedener Parameter wie z.B. der Netzwerkdichte, Rückschlüsse darüber gewonnen werden, ob und mit welchem Aufwand die vor der Transaktion vereinbarten Garantien erreicht werden können. Insbesondere soll evaluiert werden, ob die Unsicherheitszeiten bei der Recovery eines Teilnehmers nach einem Ausfall durch den SLS verkürzt werden können.

Zur Evaluation soll eine Simulation mit dem Netzwerksimulator NS2 durchgeführt werden. Realistische experimentelle Untersuchungen von MANETs mit echten Geräten sind zum einen teuer und würden zum anderen zeitlich sehr aufwändig werden, insbesondere wenn verschiedene Parameter variiert werden sollen. Für Forschungszwecke bieten sich daher Simulationen an, wie sie beispielsweise mit NS2 möglich sind. Damit die Ergebnisse einer Simulationsstudie jedoch eine gewisse Gültigkeit besitzen, sollte dafür bei der Durchführung auf einige Aspekte geachtet werden (vgl. [32]). Um die Nachvollziehbarkeit der Experimente dieser Arbeit zu gewährleisten, sollten diese *reproduzierbar* sein. Die Simulationen sollten also durch Dokumentation der Implementierung und der Parameter auch durch andere Forscher wiederholbar sein. Ebenso sollten die Ergebnisse *erwartungstreu* sein, d.h. unabhängig von einem speziellen in den Simulationen verwendeten Szenario. Weiterhin sollte das Experiment *zielgerichtet* durchgeführt werden, so dass die Szenarien und Bedingungen in den Simulationen exakt die zu untersuchenden Aspekte widerspiegeln. Zuletzt sollte die Durchführung und Analyse *statistisch einwandfrei* sein, d.h. basierend auf mathematischen Prinzipien. Bei der Umsetzung dieser Zielsetzung soll daher insbesondere auf die Transparenz bei der Vorbereitung und Durchführung der Simulationen mit dem Netzwerksimulator NS2 geachtet werden. Damit soll Nachvollziehbarkeit für mögliche weitere Forschungen gewährleistet werden.

1.2.2 Erweiterung des Shared Log Space um Berücksichtigung von Mobilität

In der zweiten Phase der Arbeit soll der SLS dahingehend erweitert werden, dass auch die Mobilität der Geräte bei der Verteilung der Transaktionslogs mit einbezogen wird. Mobilität hat im Wesentlichen einen *kurzfristigen* und einen *längerfristigen* Aspekt. In den Bereich der kurzfristigen Bewegung fallen z.B. die Richtung oder Geschwindigkeit eines Gerätes innerhalb der nächsten Sekunden oder innerhalb weniger Minuten. Längerfristige Mobilität eines Gerätes meint beispielsweise das Verlassen eines Clusters (ein Gebäude oder öffentlicher Platz) für Minuten oder Stunden, bevor das Gerät möglicherweise wiederkommt. Ein Mobilitätsmodell für MANETs, das derartige Netzstrukturen und Bewegungsmuster unterstützt, ist z.B. das Area Graph-based Mobilitätsmodell (vgl. [12]).

Bei der Berücksichtigung der Mobilität soll im Rahmen dieser Arbeit ein Verfahren entwickelt werden, das die längerfristige Mobilität von Geräten zwischen Clustern berücksichtigt. Diese ist von Bedeutung, da mit dem oben eingeführten garantierten Recovery-Zeitraum t_m Knotenfehler aufgrund von Ausfällen durch technischem Defekt oder aufgebrauchte Energieressourcen behan-

delt werden sollen. Diese Zeiträume können Minuten oder Stunden umfassen. Ebenso kann ein Gerät für einen solchen Zeitraum den Cluster verlassen, in dem die Transaktion stattfinden soll. Ein einzelnes Gerät soll daher seine individuellen Bewegungsmuster selbst lernen können, um damit die Wahrscheinlichkeit dafür bestimmen zu können, mit der der aktuelle Cluster verlassen wird. Wichtig dabei ist, dass eine solche Methodik möglichst wenig zusätzliche Paketlast für das MANET verursacht, dennoch aber hinsichtlich eines einzelnen Knotens eine größtmögliche Genauigkeit liefert.

Für die Evaluation dieser Erweiterung soll das Area Graph-based Mobilitätsmodell bei der Simulation mit NS2 verwendet werden, da dies die gewünschten Bewegungsmuster mobiler Knoten unterstützt.

1.3 Aufbau der Arbeit

Im folgenden Kapitel 2 werden zunächst grundlegende Eigenschaften von MANETs beschrieben. Darauf aufbauend werden das in dieser Arbeit verwendete Systemmodell in Abschnitt 2.2 sowie das Fehlermodell in Abschnitt 2.3 eingeführt. Abschließend wird dann auf mobile Transaktionen eingegangen, die zum Transaktionsmodell dieser Arbeit in Abschnitt 2.4.4 führen.

Mit Kapitel 3 wird ein beispielhafter Überblick über Forschungsthemen im Bereich von MANETs gegeben. Dabei wird deutlich, dass insbesondere der Umgang mit Mobilität eine große Herausforderung in diesen Netzen darstellt. Da dies auch ein Kernaspekt der Arbeit ist, wird Abschnitt 3.3 einige Ansätze zur Vorhersage von Mobilität vorstellen. Abschnitt 3.7 zieht dann ein Resumée aus der bisherigen Forschungsarbeit.

Kapitel 4 stellt detailliert den SLS vor. Dabei umfasst ein erster Teil dieses Kapitels mit Abschnitt 4.2 das Zuverlässigkeitsmodell des SLS. Im zweiten Abschnitt 4.3 werden dann die entwickelten Prozessabläufe vorgestellt. Dies beinhaltet zum einen den Prozessablauf des SLS und zum anderen einen Vergleichsprozessablauf für die Evaluation.

Den Kern der Arbeit stellt Kapitel 5 dar. Hier werden der Ablauf der Simulationen des SLS mit NS2 sowie die Ergebnisse dieser Evaluation vorgestellt. Nach der Einführung von NS2 in Abschnitt 5.1 beschreibt Abschnitt 5.2 detailliert die einzelnen Schritte der Untersuchung. Mit Abschnitt 5.3 wird schließlich auf die Ergebnisse eingegangen.

In einem nächsten Schritt soll die Erweiterung des SLS um die Berücksichtigung von Mobilität erfolgen, was in Kapitel 6 beschrieben wird. Zunächst wird in Abschnitt 6.2 das entwickelte Verfahren zur Bestimmung der Abwanderungswahrscheinlichkeit vorgestellt. Darauf aufbauend zeigt Abschnitt 6.3 die notwendigen Änderungen an den Prozessabläufen und der Implementierung für die Erweiterung auf. Zuletzt umfasst Abschnitt 6.4 eine Darstellung der verwendeten Parameter bei den Simulationen sowie die Analyse der Ergebnisse.

Kapitel 7 beschließt diese Arbeit mit einer Zusammenfassung sowie einem Ausblick auf Aspekte, die zukünftig näher untersucht werden sollten.

Kapitel 2

Mobile Ad Hoc Netzwerke und mobile Transaktionen

2.1 Mobile Ad Hoc Netzwerke (MANETs)

Ein MANET ist ein autonomes System von mobilen Knoten, die kabellos miteinander verbunden sind (vgl. z.B. [26, 47, 22]). In einem solchen Netzwerk ist jedes Gerät sowohl Endsystem (oder Endnutzer) wie auch Router, da es nicht nur selbst Nachrichten verarbeitet, sondern auch Pakete für andere Knoten weiterleitet. Die Kommunikation und damit das Weiterleiten von Paketen zwischen zwei Geräten ist möglich, sobald sie sich im gegenseitigen Übertragungsbereich befinden. Ein MANET benötigt daher keine feste Infrastruktur wie beispielsweise Basisstationen, sondern die mobilen Geräte formieren sich selbst ad hoc zu einem Netzwerk. Wesentliche Charakteristiken eines solchen Netzwerkes sind somit (1) die kabellose Verbindung zwischen den einzelnen Geräten, (2) deren Selbst-Organisation (Autonomie), (3) deren Selbst-Konfiguration (Lern- bzw. Anpassungsfähigkeit) sowie (4) die Mobilität, d.h. diese können sich frei bewegen.

Neben den zuvor beschriebenen Eigenschaften eines MANETs sind darauf aufbauend einige weitere Charakteristiken zu nennen:

- Eine wesentliche Eigenschaft eines MANETs ist dessen *dynamische Topologie*. Aufgrund der beliebigen Bewegung der Knoten kann sich dessen Topologie zufällig und häufig verändern. Die Vorhersage von Verbindungsänderungen zwischen Knoten oder der Trennung der Knoten vom Netz ist daher ein komplexes Problem. Es ist zum einen möglich, dass einzelne Knoten dadurch vollständig für das MANET verloren gehen. Zum anderen kann es durch Veränderungen der Netzwerktopologien zum Auftreten von Partitionierungen kommen, so dass die Kommunikation zwischen ganzen Netzteilen unterbrochen wird.

- *Bandbreitenbeschränkungen* sowie unterschiedliche Verbindungskapazitäten sind ebenso charakteristisch für MANETs. Im Gegensatz zu fest verdrahteten Netzwerken verfügen kabellose Verbindungen über geringere Kapazität sowie niedrigeren Durchsatz. Der Grund dafür sind u.a. mehrfacher Zugriff, Abschattung, Störungen bei der Signalausbreitung oder Interferenzen.

- *Energiebeschränkungen* der mobilen Geräte sind ein weiteres Merkmal von MANETs. Die Geräte verfügen beispielsweise über Akkus oder andere erschöpfbare Energieressourcen. Diese müssen in regelmäßigen Abständen wieder aufgeladen werden, so dass die Geräte während dieses Zeitraums möglicherweise nicht mit dem MANET verbunden sein können.

- Zuletzt ist noch die begrenzte *physikalische Sicherheit* zu nennen. Aufgrund der Selbst-Organisation eines MANETs ist die Gefahr für Angriffe wie z.B. Lauschangriffe, Manipulationen oder Denial-of-Service-Angriffe größer als in einem festen Netzwerk.

2.2 Systemmodell

Mit dem hier vorgestellten Systemmodell wird auf die im Rahmen dieser Arbeit verwendeten Annahmen über MANETs und die mobilen Geräte darin eingegangen. Die mobilen Geräte in einem MANET, wie z.B. Handys, PDAs und Laptops, werden im weiteren Verlauf der Arbeit auch allgemein als *mobile Knoten* bezeichnet.

Die Knoten eines MANETs können direkt miteinander kommunizieren mittels drahtloser Verbindung. In einem MANET gibt es weder Basisstationen noch fest verkabelte Server, die zentral Dienste anbieten oder Informationen liefern. Der Bewegung von Knoten liegt ein vorgegebenes Bewegungsmodell zugrunde, so können sich z.B. Personen entlang von Straßen bewegen, die durch Häuser u.ä. vorgegeben sind (einige Beispiele für mögliche Bewegungsmodelle werden im folgenden Kapitel 3 vorgestellt).

Damit zusammenhängend wird angenommen, dass Partitionen in einem MANET nur temporär auftreten und sich durch die Mobilität der Knoten wieder aufheben oder neu bilden. Somit verändert sich kontinuierlich die Menge von Knoten, mit denen ein Knoten kommunizieren kann. Zudem wird davon ausgegangen, dass sich die Knotendichte im Verlauf der Zeit nicht signifikant verändert aufgrund von Ausfällen (vgl. dazu Fehlermodell in Abschnitt 2.3).

Ausfälle in einem MANET resultieren bei den Knoten z.B. aus den beschränkten Energieressourcen und der Möglichkeit für den Auftritt technischer Defekte. Betrachtet man diese Faktoren zusammen, so resultiert daraus, dass die Kommunikation zwischen den Knoten unzuverlässig ist. Das Fehlermodell im folgenden Abschnitt 2.3 wird darauf detaillierter eingehen.

Jeder Knoten in einem MANET ist zunächst unabhängig von seiner technischen Ausstattung in der Lage, die Anzahl k seiner Nachbarn zu schätzen. Weiterhin wird jedoch die Heterogenität des MANETs angenommen. Jeder Knoten verfügt über unterschiedliche Energieressourcen und technische Ausstattung und weist ein individuelles Bewegungsverhalten auf. Die unterschiedliche technische Ausstattung führt dazu, dass Knoten ihren eigenen Kontext mehr oder weniger gut bestimmen können. Der Begriff *Kontext* meint in diesem Zusammenhang, dass diese Knoten über mehr Informationen über ihre Umgebung verfügen und auch ihren eigenen Zustand besser einschätzen können. Sie können beispielsweise Informationen über ihre verbleibende Energie und ihre Bewegung verarbeiten sowie ihre jeweilige Position bestimmen. Zudem sind sie in der Lage, ihr zukünftiges Verhalten bzw. ihren zukünftigen Zustand einzuschätzen, und können aufwändigere Berechnungen durchführen. Es ist diesen Knoten daher auch möglich, Annahmen und Schätzungen bezüglich des ganzen Netzes anzustellen und basierend darauf Entscheidungen zu treffen. Entsprechend werden derartige Geräte als *Knoten mit Kontextbewusstsein* bezeichnet. Geräte ohne diese Fähigkeiten werden *Knoten ohne Kontextbewusstsein* genannt.

Aus den obigen Annahmen zur Berechnung von Ausfallwahrscheinlichkeiten folgt, dass die Knoten im Hinblick auf ihre zukünftige Verfügbarkeit im MANET – ihre sog. Zuverlässigkeit – ebenso heterogen sind. Einige Knoten werden aufgrund ihrer Ressourcen länger zur Verfügung stehen und sind demzufolge zuverlässiger als Knoten, deren Energie fast erschöpft ist. Dies bedeutet auch, dass in der Umgebung eines Knotens die Nachbarn unterschiedlich zuverlässig sein können.

Alle diese Aspekte ermöglichen es, für die jeweiligen Netze verschiedene Strategien zu entwickeln, um die Wahrscheinlichkeit für einen erfolgreichen Abschluss von Transaktionen in MANETs zu erhöhen. Auf das hier verwendete Modell für Transaktionen wird im Folgenden näher eingegangen.

2.3 Fehler in MANETs

Die Charakteristiken von MANETs führen dazu, dass mehr *Fehler* auftreten als in fest verkabelten Netzen. Aufgrund der in Abschnitt 2.1 beschriebenen Charakteristiken von MANETs kann es häufiger zu Fehlersituationen kommen. Geräte sind z.B. nicht mehr erreichbar oder fallen aufgrund von erschöpften Energieressourcen aus und sind nicht mehr mit dem Netz verbunden. Als Folge davon kann es zur Unterbrechung oder zum Abbruch von Anwendungen kommen.

Zunächst wird das im Rahmen dieser Arbeit verwendete Fehlermodell vorgestellt. Daran anschließend wird in Abschnitt 2.3.2 näher auf Auslöser für Fehler eingegangen.

2.3.1 Fehlermodell

Ein MANET besteht (ebenso wie ein verteiltes System) aus zwei Komponenten: *Knoten* (hier mobile Geräte), die Informationen verarbeiten, und *Verbindungen* zwischen den Knoten (vgl. [9] S. 218). Zu den bereits aus fest verkabelten Netzen bekannten Möglichkeiten für Fehler kommen neue Faktoren hinzu. Diese beruhen auf den in Abschnitt 2.1 beschriebenen Charakteristiken von MANETs wie beschränkte Energieressourcen oder Mobilität. Dadurch besteht eine höhere Wahrscheinlichkeit sowohl für Knotenfehler wie für Verbindungsabbrüche. Als Folge von Fehlern kann es zur Unterbrechung oder zum Abbruch von Anwendungen – im Rahmen dieser Arbeit von Transaktionen – kommen. Im Folgenden werden die beiden möglichen Fehlerarten in MANETs, *Knotenfehler* und *Kommunikationsfehler*, näher erläutert. Formal bedeutet ein *Ausfall* bzw. ein *Fehler* die *Beendigung der Fähigkeit eines Gerätes oder Systems, eine bestimmte Funktion zu leisten* (vgl. [25] S. 10).

Knotenfehler: Ein Knotenfehler liegt vor, wenn ein mobiles Gerät aufgrund eines Ausfalls im Netz nicht mehr zur Verfügung steht. Tritt bei einem Knoten im MANET eine Fehlersituation auf, so werden alle laufenden Aktionen abgebrochen und er ist weder für das Routing noch für andere Netzwerkanwendungen verfügbar. Hat sich ein Knoten nach einem Ausfall erholt (z.B. durch Aufladen des Akkus oder einer Reparatur), so kann er sich wieder mit Netz verbinden und eine sog. *Recovery* starten. Dabei versucht er, die durch seinen Ausfall verloren gegangenen Informationen zu beschaffen, um einen konsistenten Zustand zu erreichen. Es wird bei diesem Fehlermodell angenommen, dass ein Knoten entweder funktionstüchtig oder ausgefallen ist. Ursachen für Knotenausfälle sind Faktoren, die direkt mit dem Knoten selbst zusammenhängen. Diese können z.B. technische Fehler oder technisches Versagen (Produktionsfehler, Transport usw.), erschöpfte Energie, falsche Bedienung oder Benutzung im falschen Umfeld durch den Anwender sein. Auf Auslöser wird detaillierter im folgenden Abschnitt 2.3.2 eingegangen.

Kommunikationsfehler: Ein Kommunikationsfehler liegt vor, wenn zwei Knoten A und B nicht mehr miteinander kommunizieren können, obwohl bei keinem von beiden ein Knotenfehler aufgetreten ist. Es gibt verschiedene Ursachen für derartige Fehler, z.B. Störungen der Verbindung durch Mobilität oder Fehlfunktion, Kommunikationsreichweite der Knoten oder physikalische Faktoren. Auch hier sei auf Abschnitt 2.3.2 für Details verwiesen.

Durch Kommunikationsfehler allein oder eine Kombination aus Knoten- und Kommunikationsfehlern kann, wie bereits zuvor festgestellt, die Kommunikation zwischen Knoten zusammenbrechen. Existiert kein Pfad mehr zwischen zwei Knoten A und B, so wird dies als *Netzwerkpartitionierung* bezeichnet. Eine solche Partitionierung unterteilt das Netz in zwei oder mehr Komponenten. Innerhalb einer Komponente können die Knoten miteinander kommunizieren, aber Knoten aus unterschiedlichen Komponenten können dies nicht. Es wird hier davon ausgegangen,

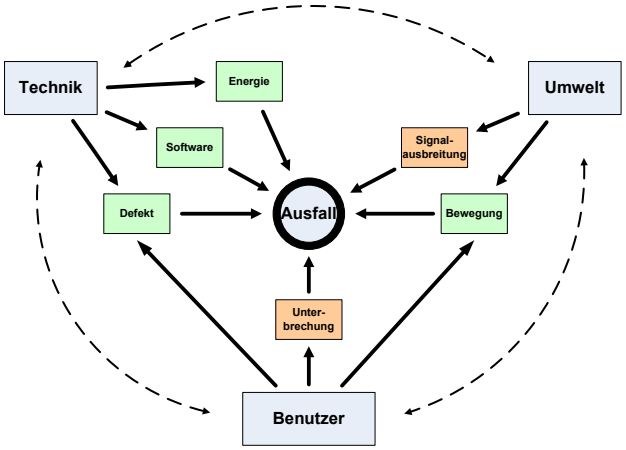

Abbildung 2.1: Klassifikation von Ausfallauslösern in MANETs

dass Netzwerkpartitionierungen in einem MANET u.a. aufgrund der Mobilität der Knoten sowie der Aufladbarkeit von Energieressourcen temporär auftreten.

Effekte von Fehlern

Mit *Ausfalleffekt* bezeichnet man einen *Zustand oder eine Folge von Ereignissen, ausgelöst durch einen Ausfall.* Als Folge der beschriebenen Fehlersituationen kommt es dazu, dass Nachrichten nicht zustellbar sind. Eine Möglichkeit ist, dass dadurch eine erwartete Nachricht gar nicht erst verschickt wird, weil der Sender ausgefallen ist. Es kann allerdings auch passieren, dass der Empfänger ausgefallen ist und die Nachricht nicht erhalten kann. Zuletzt besteht die Möglichkeit, dass sowohl Sender wie Empfänger funktionstüchtig sind, sich allerdings in unterschiedlichen Partitionen befinden. Auch dann können Nachrichten das Ziel nicht erreichen.

2.3.2 Auslöser von Fehlern

Der vorhergehende Abschnitt zeigt bereits einige Faktoren für Ausfälle auf. Formal ist ein *Ausfall¨auslöser* ein *Prozess, der das Gerät beeinflusst und zu seinem Ausfall führt* (vgl. [25] S. 12). Hier sollen diese Ausfälle näher betrachtet und strukturiert werden. Sie lassen sich in drei grundlegende Gruppen aufteilen. Abbildung 2.1 stellt die im Folgenden beschriebenen Zusammenhänge grafisch dar.

Eine Unterteilung der Auslöser ist prinzipiell in drei Gruppen möglich, wobei es teilweise keine klare Trennung gibt, sondern die Grenzen überschneiden sich:

1. *Technische Ursachen*: Unter technischen Ursachen versteht man in der Regel den gewöhnlichen Alterungs- bzw. Abnutzungsprozess technischer Geräte. Dies beinhaltet zum einen die sog. echten *Defekte*, d.h. dass einzelne Teile eines Gerätes oder das Gerät selbst aufgrund eines Bauteils, der Produktion, des Transports, seines Alters oder seiner Benutzung seine Funktionsfähigkeit temporär oder vollständig verliert (vgl. für umfangreiche Informationen zu technischen Defekten [38]). Zu den technischen Ursachen kann man auch den Ausfall oder das Fehlverhalten der *Software* eines Gerätes zählen. Auf der anderen Seite umfasst

diese Ursachengruppe auch den Aspekt, dass Geräte nur über *beschränkte Energieressourcen* verfügen und diese regelmäßig wieder aufgeladen werden müssen. Prinzipiell sind die technischen Ursachen selbst in weitere detailliertere Klassen einzuteilen. Eine Analyse der technischen Zuverlässigkeit und Ausfälle findet man z.B. in [38]. Hier wird neben der Unterteilung nach Umwelteinflüssen noch differenzierter in Ausfälle aufgrund des Transports, chargenbedingte Ausfälle, in Ausfälle aufgrund von Verpackungen und Verschleiß klassifiziert, um nur eine grobe Übersicht zu geben. Für diese Arbeit sind diese jedoch nicht relevant, so dass die Annahme ausreichend ist, dass der Ausfall bedingt durch Defekt (Hard- oder Software) oder aufgrund erschöpfter Energieressourcen aufgetreten ist.

2. *Einflüsse der Umwelt*: Diese Gruppe von Auslösern umfasst beispielsweise Einflüsse durch thermale, mechanische, elektrische, chemische oder magnetische Quellen (vgl. [25] S. 12), die dazu führen, dass Geräte dem MANET temporär oder dauerhaft nicht mehr zur Verfügung stehen. Zum einen wirken sie auf die Signalausbreitung der Geräte und zum anderen auch auf deren technische Funktionsfähigkeit. Im Hinblick auf Signalausbreitung können Einflüsse die Kommunikationsfähigkeit stören. Befinden sich in der Umgebung der Geräte z.B. Hindernisse wie Gebäude, so kann dadurch die Signalausbreitung eingeschränkt bzw. verhindert werden. Abschattung kann Geräte völlig von anderen abschneiden und Effekte wie Reflexion können Signale abschwächen. Ebenso führen auch Streuung und Beugung dazu, dass die Signalausbreitung vermindert wird (vgl. dazu ausführlicher [47] S. 55 ff.). Einflüsse wie Feuchtigkeit oder Wärme können beispielsweise die technische Funktionsfähigkeit der Geräte einschränken. Weiterhin kann insbesondere auch *Mobilität* zum temporären oder vollständigen Verlust der Verbindung zwischen mobilen Geräten führen. Gerade dieser Aspekt, der eng mit dem im Folgenden vorgestellten Benutzerverhalten zusammenspielt, ist in MANETs von besonderer Bedeutung. Hier kann man – wie in der Einführung in Abschnitt 1.2.2 vorgestellt – zwischen *kurzfristiger* und *langfristiger* Mobilität unterscheiden. Der erste Aspekt bezieht sich in der Regel auf die Bewegung in den nächsten Sekunden oder Minuten innerhalb eines Clusters. Diese führt häufig zu Verbindungsabbrüchen und resultiert in Kommunikationsfehlern. Langfristige Mobilität bezieht sich hingegen auf das Verlassen eines bestimmten Clusters, konkret einer Partition (z.B. ein öffentlicher Platz oder ein Gebäude). Dies führt nicht nur zu Kommunikationsfehlern durch die Bewegung, sondern auch zu Knotenfehlern, da der Knoten durch die sog. *Abwanderung* nicht mehr zur Verfügung steht.

3. *Benutzerverhalten*: Die dritte Gruppe von Auslösern bezieht sich auf den Umgang des Benutzers mit dem Gerät. Dabei sind die Grenzen dieser Gruppe zu den beiden vorher vorgestellten nicht klar trennbar. Eindeutig Auslöser von Ausfällen, die rein benutzerabhängig sind, sind z.B. das Ausstellen eines Gerätes oder auch die Unterbrechung von Anwendungen. Dabei kann dies jeweils bewusst geschehen oder aufgrund eines Bedienungsfehlers. Gleichzeitig kann das Verhalten des Benutzers jedoch auch zusammen mit Umwelteinflüssen zum Kommunikationsverlust führen, z.B. Bedienung in der falschen Umgebung. Das ist beispielsweise auch bei Mobilität des Benutzers der Fall oder wenn der Benutzer sich in einer Umgebung bewegt, wo die Signalausbreitung der Geräte eingeschränkt ist. Ähnliche Ereignisse können auch im Zusammenspiel von Benutzer und den anderen Umwelteinflüssen auftreten. Im Hinblick auf technische Defekte kann auch der Benutzer als Auslöser beteiligt sein, z.B. wieder durch falsche Bedienung oder Fahrlässigkeit (Fallen lassen eines Gerätes usw.)

2.4 Mobile Transaktionen

Durch zunehmende Verbreitung mobiler Geräte, besteht auch der Bedarf bekannte Anwendungen aus verkabelten Netzen auf Handys, PDAs oder Laptops zur Verfügung zu stellen. Diese sollen dieselbe Sicherheit und Zuverlässigkeit bieten wie ihre Gegenstücke in verkabelten Netzwerken.

Anwendungsbereiche sind beispielsweise mobile und drahtlose Informationssysteme, die gemeinsame Daten und Dienste nutzen. Ebenso wie in klassischen Datenbanksystemen sollen diese korrekt und fehlertolerant sein sowie den Zugriff mehrerer Benutzer gleichzeitig unterstützen. Hierbei ist insbesondere die Transaktionsverwaltung von großer Bedeutung. Diese soll die anwendungsunabhängige und für Anwendungen transparente Synchronisation nebenläufiger Zugriffe auf die Datenbank übernehmen. Außerdem muss gewährleistet werden, dass Anwendungen nicht anfällig gegen Anwendungs- und Systemfehler sind. Aufgrund der in Abschnitt 2.3 beschriebenen möglichen Fehlersituationen in MANETs z.B. durch Energiebeschränkungen und Mobilität müssen existierende Transaktionsmodelle und Ausführungsprotokolle entsprechend angepasst werden.

Im Folgenden wird zunächst auf Transaktionen allgemein und die ACID-Eigenschaften eingegangen sowie insbesondere auf mobile Transaktionen. Darauf aufbauend werden Problematiken bei der Umsetzung von Transaktionen in MANETs aufgezeigt und Anforderungen an mobile Transaktionen formuliert.

2.4.1 Definition von Transaktionen und ACID-Eigenschaften

Eine *Transaktion* ist eine Menge von logisch zusammengehörenden Operationen, die gekapselt und unter bestimmten Garantien ausgeführt werden (vgl. [24] S. 141). In klassischen Datenbanken setzen sich Transaktionen aus Operationen zusammen, die auf demselben Datenbanksystem ausgeführt werden. Im Gegensatz dazu laufen die Operationen bei verteilten Transaktionen in mehreren Datenbanksystemen ab. Handelt es sich um eine mobile Transaktion, so sind eine oder mehrere mobile Datenbanken, d.h. mindestens ein mobiler Rechner, beteiligt. Die Ausführungsgarantien sollen dabei – wie z.B. bei klassischen Transaktionen – transparent durch das Datenbanksystem umgesetzt werden. Dies bedeutet, dass u.a. die im Folgenden erläuterten ACID-Eigenschaften (vgl. [29, 24]) garantiert werden:

Atomarität: Eine Transaktion ist eine atomare Ausführungseinheit, d.h. sie wird als nicht weiter zerlegbar betrachtet. Die Anforderung ist, dass entweder alle durch diese Transaktionen vorgenommenen Änderungen in der Datenbasis vorgenommen werden oder keine davon.

Konsistenz: Eine Transaktion muss einen konsistenten Datenbankzustand hinterlassen. Es sind zwar inkonsistente Zwischenzustände erlaubt, aber sowohl vor wie nach der Transaktion muss ein konsistenter Zustand sichergestellt werden.

Isolation: Nebenläufige (parallele, gleichzeitige) Transaktionen auf einer Datenbasis dürfen keinen gegenseitigen Einfluss aufeinander haben, d.h. weder die Transaktion noch deren Effekte dürfen für die anderen sichtbar sein. Jede Transaktion und ihr Effekt muss so aussehen, als wäre es die einzige ausgeführte Transaktion.

Dauerhaftigkeit: Am Ende einer Transaktion muss der daraus resultierende Effekt dauerhaft erhalten bleiben. Insbesondere muss garantiert werden, dass der Effekt auch spätere Systemfehler überlebt.

Im Rahmen dieser Arbeit wird der Fokus auf die Atomaritätseigenschaft von Transaktionen gerichtet sein. Dieses Kriterium ist bei Commit-Protokollen ein Problem, da hier Unsicherheitszustände entstehen, wie im Folgenden gezeigt wird.

2.4.2 Modell der verteilten Transaktionen

In verteilten Systemen können an einer Transaktion mehrere Rechner oder Prozesse beteiligt sein, die auf gemeinsamen Daten arbeiten, d.h. diese abhängig voneinander verändern. Bei sog. verteilten Transaktionen ist das Hauptproblem, dass die Aktionen der beteiligten Rechner bzw. Prozesse Daten auf unterschiedlichen Knoten ändern. Es muss dann eine Einigung über das globale Commit oder Abort der Transaktion hergestellt werden. Während der gesamten Transaktionsdauer kann es jedoch zu Fehlersituationen kommen, wie dies in Abschnitt 2.3 beschrieben wird.

2.4.2.1 Transaktionen in verteilten Systemen

Das Problem in verteilten Systemen ist u.a. die sog. EOT-Behandlung (EOT ist engl. *End of Transaction*) von globalen Transaktionen. Diese müssen atomar beendet werden, d.h. entweder müssen alle beteiligten Rechner bzw. Prozesse den gleichen Zustand festschreiben oder keiner davon. Genau diese Garantie ist jedoch problematisch in verteilten Systemen, da Rechner unabhängig voneinander ausfallen können, d.h. im Netzwerk nicht mehr zur Verfügung stehen.

Um dennoch Atomarität bei der EOT-Behandlung in verteilten Systemen gewährleisten zu können, werden verteilte Commit-Protokolle eingesetzt. Diese sollen eine globale Einigung auf Commit oder Abort sicherstellen, wenn eine Transaktion auf mehreren Rechnern ausgeführt wird. Das Commit-Protokoll regelt dabei u.a. die Protokollierung von Informationen und das Festschreiben der Daten. Ein sehr bekanntes und häufig verwendetes Verfahren ist das sog. Zwei-Phasen-Commit-Protokoll (kurz 2PC-Protokoll), auf das im Folgenden kurz eingegangen werden soll. Andere Beispiele sind das Ein-Phasen-Commit-Protokoll und das Drei-Phasen-Commit-Protokoll (vgl. [29, 9]). Auf Letzteres wird nur kurz am Ende von Abschnitt 2.4.2.2 eingegangen.

2.4.2.2 2-Phasen-Commit-Protokoll

Bei dem 2PC-Protokoll (vgl. dazu z.B. [29, 24, 50, 51]) gibt es einen Rechner bzw. Prozess, der als sog. Koordinator fungiert. Dieser überwacht das gesamte Transaktionsprotokoll und soll garantieren, dass die Teilnehmer an einer Transaktionen entweder alle die Änderungen festschreiben oder diese zurücknehmen. Als Teilnehmer werden die an einer Transaktion beteiligten Rechner oder Prozesse bezeichnet. Das Protokoll ist in zwei Phasen unterteilt. In der ersten sog. Abstimmungsphase (auch Entscheidungsfindung genannt, vgl. [29]) sammelt der Koordinator die Zustimmung oder die Ablehnung (Votum für Commit mit *Ready* bzw. Abort mit *Failed*) zur Festschreibung der Änderungen von den Teilnehmern ein. Haben sich alle Teilnehmer für Commit ausgesprochen, wird der Koordinator global auf Commit entscheiden, anderenfalls bei Nicht-Einstimmigkeit zu Abort.

Ist die Entscheidung über den Transaktionsausgang gefallen, teilt der Koordinator den Teilnehmern diese in der zweiten Phase des Protokolls mit. Die Teilnehmer werden dann entweder entsprechend der globalen Entscheidung die Änderungen festschreiben oder die gesamte Transaktion rückgängig machen. Anschließend können die Teilnehmer alle für die Transaktion gehaltenen Sperren wieder freigeben. Außerdem wird der Koordinator mittels einer Bestätigung über die Durchführung ihrer Änderungen informiert.

Durch das 2PC-Protokoll werden verschiedene Fehlersituationen in verteilten Systemen berücksichtigt. Dazu müssen sowohl der Koordinator wie die Teilnehmer während des Transaktionsablaufs Einträge in eine ausfallsichere Log-Datei schreiben. Dieses Verfahren soll sicherstellen, dass die beteiligten Rechner bzw. Prozesse nach einem Ausfall rekonstruieren können, an welcher Stelle des Protokolls die Unterbrechung aufgetreten ist.

Ein möglicher Fehlerfall ist der Ausfall des Koordinators. Stürzt der Koordinator vor dem Versenden der Commit-Nachricht ab, kann er die Änderungen der Transaktion durch das Versenden eines Aborts wieder zurücksetzen bei der Recovery. Wenn jedoch der Ausfall auftritt, nachdem einige Teilnehmer ihr Ready geschickt haben, können diese die Transaktion nicht mehr abbrechen. Diese Teilnehmer können weder ein Commit noch ein Abort durchführen, da sie die Entscheidung des Koordinators nicht kennen. Sie müssen daher in ihrem Unsicherheitszustand verbleiben. Die Kontrolle über die Transaktion liegt beim Koordinator. Lediglich die Teilnehmer, die vor dem Absturz des Koordinators noch kein Votum geschickt haben, können die Transaktion selbstständig abbrechen.

Der Absturz eines Teilnehmers ist eine weitere Problematik. Hier muss der Teilnehmer bei seiner Recovery nach einem Absturz die Log-Datei überprüfen. Findet er keinen Eintrag, der auf ein gesendetes Votum verweist, kann er die Transaktion abbrechen. Findet der Teilnehmer jedoch die Information, dass bereits ein Ready an den Koordinator geschickt wurde, muss bei diesem nach dem Transaktionsausgang gefragt werden. Sollte der Koordinator mit einem Commit antworten, muss der Teilnehmer das Redo der Transaktion durchführen, bei einem Abort entsprechend das Undo. Ein Redo der Transaktion muss auch dann vorgenommen werden, wenn der Teilnehmer in der Log-Datei einen Commit-Eintrag findet.

Eine weitere Fehlerquelle in verteilten Systemen sind verloren gegangene Nachrichten. Dies bezieht auch den zuvor genannten Aspekt mit ein. Erhält der Koordinator von einem Teilnehmer kein Ready oder Failed innerhalb eines Timeout-Intervalls, so nimmt er dessen Ausfall an. Er kann sich dann für den Abbruch der Transaktion entscheiden. Dabei ist irrelevant, ob der Teilnehmer wirklich ausgefallen ist oder ob er eine Nachricht gesendet hat, die den Koordinator nicht erreicht hat. Ein größeres Problem stellt die Situation des Teilnehmers dar. Wenn dieser ein Votum an den Koordinator geschickt hat, aber keine Nachricht über die Entscheidung erhält, verbleibt er in einem Unsicherheitszustand. Hier ist keine Entscheidung seitens des Teilnehmers möglich und die von ihm gehaltenen Sperren bleiben aufrecht erhalten.

Das hier kurz vorgestellte 2PC-Protokoll ist lediglich ein Beispiel für ein Commit-Protokoll in verteilten Systemen. Das Problem der Blockierung von Teilnehmern bleibt bestehen. Mit dem sog. Drei-Phasen-Commit-Protokoll sollen derartige Blockierungen weiter reduziert werden. Dieses Protokoll basiert auf der sog. *Single Failure Assumption.* Es toleriert Knotenfehler, jedoch keine Kommunikationsfehler (vgl. [9]). Dazu wird ein zusätzlicher sog. Precommit-Zustand eingeführt. In diesem Zustand ist mindestens k Teilnehmern bereits die globale Entscheidung bekannt. Dadurch können ein neuer Koordinator gewählt und das Protokoll beendet werden. Das Risiko für eine Blockierung wird dadurch reduziert, es sind jedoch aufgrund des zusätzlichen Zustandes mehr Nachrichten notwendig. Aber es bleibt auch hier die Möglichkeit, dass Unsicherheitszustände auftreten und inkonsistente Entscheidungen getroffen werden, z.B. aufgrund von Netzpartitionen.

Prinzipiell bleibt bei allen Commit-Protokollen immer die Möglichkeit für Unsicherheitszustände, in denen die Teilnehmer unsicher sind im Hinblick auf den Ausgang der Transaktion. Deren Vermeidung würde erfordern, dass ein Prozess gleichzeitig sowohl sein Votum abgibt wie über die Voten aller anderen Prozesse informiert wird. (vgl. [9] S. 226). Durch die Charakteristiken von MANETs wird die Wahrscheinlichkeit für Ausfälle während der gesamten Transaktion und somit für Unsicherheitszustände im Commit-Protokoll noch erhöht. Auf die Problematiken

und Herausforderungen bei der Umsetzung von Commit-Protokollen in MANETs wird daher im Folgenden näher eingegangen.

2.4.3 Problematiken bei Transaktionen in MANETs

Durch die in Abschnitt 2.3 vorgestellten besonderen Charakteristiken von MANETs werden die bei verteilten Transaktionen bereits bekannten Problematiken noch verstärkt. Es treten häufiger Fehlersituationen während der Transaktion auf, wodurch auch die Wahrscheinlichkeit für Unsicherheitszustände in der Commit-Phase höher ist.

Traditionelle Transaktionsmodelle definieren ihre Korrektheitskriterien bezüglich der ACID-Eigenschaften. Für verteilte Systeme sind inzwischen auch sog. *Advanced Transaktionsmodelle* (Abk. ATMs) entwickelt worden. Diese definieren Korrektheit durch Abschwächung der ACID-Eigenschaften und Hinzufügen neuer Kriterien, die in Bezug zur Korrektheit in neuen Anwendungsgebieten stehen (vgl. z.B. [20]). So sollen damit z.B. langlebige Transaktionen unterstützt werden, die auf viele Daten zugreifen und über lange Zeit im Netz bleiben. Dadurch kann es leicht zu Fehlersituationen kommen. So muss dann beispielsweise wegen der Atomaritätsanforderung sofort ein Rollback durchgeführt werden, oder wegen der Isolationsanforderung müssen Sperren bis zum Ende der Transaktion gehalten werden. Dies kann zu Deadlocks führen. Außerdem kann die *Einschränkung von Parallelität und Kooperation* zwischen Transaktionen die Folge sein. ATMs definieren deshalb die Korrektheit durch Abschwächung der ACID-Eigenschaften und das Hinzufügen neuer Kriterien, die in Bezug zu den Problematiken in verteilten Systemen stehen (vgl. dazu z.B. [20]).

Die Anwendung von Verfahren wie ATMs bietet sich auch für MANETs an, um mit Fehlersituationen im Verlauf der Transaktionsdurchführung umzugehen. Dadurch kann Abbrüchen vor der Commit-Phase vorgebeugt werden. Allerdings bleibt die Wahrscheinlichkeit für Unsicherheitszustände in der Commit-Phase bestehen. Deshalb ist die Verringerung der Wahrscheinlichkeit für Unsicherheitszustände durch Fehler in dieser Phase eine wesentliche Herausforderung. Im Rahmen dieser Arbeit soll deshalb insbesondere die Commit-Phase mit deren Unsicherheitsfenster untersucht werden. Dazu wird im Folgenden zunächst das verwendete Transaktionsmodell vorgestellt. Zur Untersuchung der Unsicherheitszustände von Knoten wird im Rahmen der Arbeit das traditionelle Transaktionsmodell verwendet.

2.4.4 Verwendetes Transaktionsmodell

Der Ablauf von Transaktionen wird im Rahmen dieser Arbeit vereinfacht betrachtet. Das bedeutet, dass hier insbesondere das in den Abschnitten 2.4.2.2 und 2.4.3 aufgezeigte Unsicherheitsfenster bei allen Commit-Protokollen betrachtet werden soll. Es wird dabei von dem traditionellen Transaktionsmodell ausgegangen, das u.a. strikte Atomarität fordert. Transaktionsabläufe, wie sie im Verlauf dieser Arbeit verwendet werden, sind daher keinem der bekannten Protokolle eindeutig zuzuordnen. Sie sollen nur das Unsicherheitsfenster der Commit-Protokolle repräsentieren. Die Einschränkung auf die Unsicherheitszeiten soll ermöglichen, die Effektivität von Strategien zur Reduzierung dieser Unsicherheitszustände besser untersuchen zu können. Die Prozessabläufe werden entsprechend dem Untersuchungsziel angepasst, so dass hier zunächst die grundlegenden Annahmen zu Transaktionen vorgestellt werden.

Transaktionen können zwischen zwei oder mehreren Knoten, die über mehrere Hops voneinander entfernt sind, durchgeführt werden.

Einer der Knoten übernimmt dabei die Koordinierung der Transaktion und entscheidet über den Abbruch, falls sich kein Teilnehmer meldet, oder über den Ausgang der Transaktion (sog. *Commit*). Dieser Knoten wird als *Koordinator* bezeichnet. Koordinatoren verfügen über eine

derart ausgereifte technische Ausstattung (vgl. Systemmodell in Abschnitt 2.2), dass sie aufgrund ihrer Kontextinformationen global für alle Knoten im MANET Annahmen treffen können. Die anderen Knoten, die an einer Transaktion teilnehmen, werden als *Teilnehmer* oder *Klienten* bezeichnet.

Im Rahmen dieser Arbeit wird davon ausgegangen, dass bei Beginn der Transaktion dem Koordinator die Teilnehmer bekannt sind. Ebenso kennen die Teilnehmer bereits den Koordinator. Um den Vorlauf der Transaktion bis zur Commit-Phase abzubilden (die eigentliche Abstimmungsphase), in dem diese Aushandlungen über die Teilnehmer, den Koordinator sowie den Zeitpunkt des Beginns der Commit-Phase stattfinden, wird hier eine sog. *Prozess-Phase* vorgeschaltet. Am Ende dieser Phase wird der Teil des Commit-Protokolls gestartet, in dem das zu untersuchende Unsicherheitsfenster auftritt.

Der grundlegende weitere Verlauf der Transaktion ist so gestaltet, dass die Teilnehmer dem Koordinator ihr Votum schicken. Dieses Votum muss vom Koordinator bestätigt werden. Ab dem Zeitpunkt, an dem die Teilnehmer die geforderte Bestätigung erhalten haben, befinden sie sich in dem zu untersuchenden Unsicherheitszustand. Der Koordinator trifft dann die Entscheidung über Commit oder Abort der Transaktion und teilt dies den Teilnehmern mit. Für ihn ist die Transaktion anschließend beendet. Wenn die Teilnehmer die Entscheidung des Koordinators erhalten, so ist für sie die Transaktion ebenfalls beendet, anderenfalls verbleiben sie unsicher. Die Verwendung von Timern erlaubt dann beispielsweise, dass ein Teilnehmer in regelmäßigen Abständen versucht den Koordinator zu erreichen, um den Transaktionsausgang zu erfragen. Am Unsicherheitszustand der Teilnehmer ändert dies jedoch nichts, da dieser erst mit Erhalt des Transaktionsausgangs (Commit oder Abort) beendet werden kann.

Im Verlauf der Transaktion und insbesondere während des Unsicherheitsfensters können aus verschiedenen Gründen Fehler auftreten, z.B. aufgrund erschöpfter Energieressourcen, technischer Defekte oder Mobilität (vgl. Fehlermodell in Abschnitt 2.3). Dadurch erhalten Transaktionsteilnehmer beispielsweise den Transaktionsausgang nicht. Es kann jedoch auch dazu führen, dass Nachrichten gar nicht erst gesendet werden, weil ein Knoten gerade ausgefallen ist.

Kapitel 3

Stand der Forschung

3.1 Forschung im Bereich von MANETs

Das vorige Kapitel 2 hat bereits die wesentlichen Charakteristiken und Problematiken von MANETs aufgezeigt. Auch wenn diese Netze im Vergleich zu anderen Themenbereichen der Informatik noch ein recht neues Forschungsgebiet sind, gibt es zahlreiche Veröffentlichungen. Es gibt eine Vielzahl von unterschiedlichen Themen, z.B. die Entwicklung von Modellen für MANETs selbst, Bewegungsmodelle, Routingverfahren oder ortsbezogene Dienste. Es kann daher nur auf einige Aspekte eingegangen werden.

In diesem Kapitel sollen beispielhaft aus einigen Forschungsbereichen Ansätze vorgestellt werden. Eine besondere Herausforderung ist die Anwendungsentwicklung. Im Rahmen dieser Arbeit sollen insbesondere mobile Transaktionen betrachtet werden. Aufgrund der besonderen MANET-Charakteristiken existiert noch kein wirklicher Ansatz für Transaktionen, die ausschließlich im Ad-Hoc-Netz durchgeführt werden. Zurzeit wird bei den gängigen Lösungen noch überall eine konsolidierte Datenbank im Hintergrund angenommen, zu der zumindest zeitweise eine Verbindung besteht und die sich um die eigentliche Transaktionsdurchführung kümmert. Abschnitt 3.2 wird zunächst zwei aktuelle Ansätze vorstellen. Dort zeigt sich – wie in vielen anderen Forschungsbereichen auch –, dass insbesondere die Mobilität eine herausfordernde Charakteristik von MANETs ist. Abschnitt 3.3 zeigt daher einige Ansätze zur Vorhersage von Mobilität auf.

Anschließend werden in Abschnitt 3.4.1 einige graphentheoretische Ansätze zur Modellierung von MANETs vorgestellt. Auch hier sind zurzeit Grenzen durch die Mobilität gesetzt. Darauf aufbauend wird in Abschnitt 3.5 auf Bewegungsmodelle für Ad-Hoc-Netzwerke eingegangen. In einem nächsten Schritt zeigt Abschnitt 3.6 Ansätze für Routing-Verfahren auf, mit denen Pakete über Multihop verschickt werden können. Abschließend werden einige Schlussfolgerungen aus den einzelnen Ansätzen gezogen.

3.2 Modelle für mobile Transaktionen

Im Bereich der Forschung zu mobilen Transaktionen stellt sich heraus, dass die meisten Verfahren auf der Annahme beruhen, dass über Basisstationen eine feste Datenbank zur Verfügung steht (vgl. z.B. die Ansätze in [24, 37]). Ansätze für Transaktionsmodelle, bei denen ausschließlich mobile Geräte ohne Zugriff auf eine konsolidierte Datenbank beteiligt sind, sind bisher noch wenig erforscht.

Bei den bisherigen Modellen für mobile Transaktionen kann man zwischen zwei Klassen unterscheiden. Zum einen gibt es die Klasse, bei der mobile Geräte zum Zeitpunkt der Transaktion schwach mit der konsolidierten Datenbank verbunden sind. Ein Beispiel hierfür sind die sog. *Kangaroo-Transaktionen.* Zum anderen gibt es die Klasse, bei der mobile Geräte zum Transaktionszeitpunkt nicht mit der Datenbank verbunden sind. Erst zum Synchronisationszeitpunkt besteht eine Verbindung mit der konsolidierten Datenbank. Dann werden die auf den lokalen Datensätzen ausgeführten Transaktionen nachgeholt. Als Beispiel für diese Klasse werden die

sog. *Promotion-Transaktionen* vorgestellt.

3.2.1 Kangaroo-Transaktionen

Es wird hier davon ausgegangen, dass sich die mobilen Knoten durch Funkzellen bewegen, die jeweils durch eine Basisstation abgedeckt werden. Zudem muss zum Transaktionszeitpunkt eine Verbindung zur konsolidierten Datenbank bestehen.

Ein mobiler Rechner startet eine Kangaroo-Transaktion auf seiner aktuellen Basisstation (vgl. zu diesem Modell [24, 37]). Um die Bearbeitung der Transaktion selbst kümmern sich dann die Basisstation sowie die beteiligten Datenbanksysteme. Prinzipiell bauen Kangaroo-Transaktionen auf traditionellen Transaktionen auf. Lediglich der sog. *Handoff-Prozess*, also der Übergang von einer Funkzelle in eine andere, erfordert besondere Berücksichtigung. Dazu ist eine auf einer Basisstation gestartete Kangaroo-Transaktion in Subtransaktionen unterteilt. Subtransaktionen laufen auf einer Basisstationen im Rahmen einer sog. *Joey-Transaktion*. Bei der Bewegung von einer Funkzelle in eine andere wandert auch die Kangaroo-Transaktion mit. Aber alle noch bei der alten Basisstation laufenden Subtransaktionen werden dort weiterbearbeitet. Die neue Station startet für weitere Subtransaktionen eine neue Joey-Transaktion. Dabei muss berücksichtigt werden, dass beim Fehlschlagen einer Joey-Transaktion ggf. die gesamte Kangaroo-Transaktion zurückgesetzt werden muss.

3.2.2 Pro-Motion-Transaktionen

Pro-Motion-Transaktionen unterstützen Transaktionen durch mobile Rechner, die zum Transaktionszeitpunkt nicht mit einer konsolidierten Datenbank verbunden sind. Bei diesem Konzept ist nicht nur Serialisierbarkeit vorgesehen, sondern auch benutzerdefinierte Korrektheitskriterien.

Kernprinzip bei Pro-Motion-Transaktionen sind sog. *Compacts* (vgl. zu diesem Modell [37, 24]), die Vereinbarungen zwischen dem lokalen Datenbanksystem und dem mobilen Rechner entsprechen. Sie bestehen beispielsweise aus Zustandsinformationen, Konsistenzbedingungen, Verpflichtungen und maximaler Verweilzeit. Mit den Compacts erhält ein mobiler Rechner bestimmte Zugriffsrechte und Garantien, die mit gewissen Verpflichtungen verbunden sind. Sie regeln die Replikation von Daten auf dem mobilen Rechner. Laufen diese Rechte des Compacts ab, können die Änderungen des mobilen Rechners nicht mehr an das lokale Datenbanksystem übertragen werden.

Grundlage des Modells sind geschachtelte Split-Transaktionen. So kann ein mobiler Rechner beliebig viele Transaktionen durchführen. Hat er wieder Verbindung mit der lokalen Datenbank, können die abgeschlossenen Transaktionen an diese weitergegeben werden. Dazu werden die abgeschlossenen Transaktionen durch Split-Operationen aus dem Transaktionsbaum herausgelöst. Mit diesem Verfahren wird erlaubt, dass ggf. dieselben Daten von unterschiedlichen Anwendungen mit verschiedenen Zugriffsrechten repliziert und auch manipuliert werden. Besteht keine Verbindung zur Datenbank für den mobilen Rechner, können die Transaktionen vollständig lokal ablaufen. Bei einem globalen Commit verwenden die daran beteiligten Compacts ein Zwei-Phasen-Commit-Protokoll.

3.3 Vorhersage der Mobilität in MANETs

Bei einem MANET ist die Bewegung der mobilen Geräte eine der Haupteigenschaften. Wie bereits die Abschnitte 2.3 und 3.2 zeigen, ist dies jedoch eine der Hauptursachen für Verbindungsabbrüche. Dadurch stellt der Umgang mit Mobilität in verschiedenen Bereichen eine Herausforderung dar.

So ist z.B. unter anderem durch B. Kwak der Versuch unternommen worden, ein Maß für die Mobilität in einem MANET zu entwickeln (vgl. [33]). Damit ist jedoch keine Aussage darüber möglich, wohin sich ein Knoten bewegt oder wann die Verbindung zwischen zwei Knoten zusammenbricht. Es ist jedoch beispielsweise wünschenswert, dass nur Geräte in bestimmten Regionen eines MANETs durch eine Anwendung angesprochen werden können. Die Entwicklung solcher sog. *ortsbezogenen* Dienste ist ein weiteres Forschungsthema im Bereich Mobilität (vgl. z.B. [14]).

Daneben führt Mobilität dazu, dass Benutzer das MANET verlassen, weil sie z.B. aus einem Gebäude herausgehen. Für Anwendungen wäre es jedoch von Vorteil, zumindest eine Information über die globale Abwanderungsrate von Geräten aus dem MANET bestimmen zu können. Hilfreich wäre außerdem, für einzelne Knoten ermitteln zu können, wie lange sie voraussichtlich noch im MANET für einen oder mehrere andere Knoten erreichbar sein werden (z.B. für das Routing). Anderenfalls würde die Kommunikation zwischen den Geräten nicht mehr möglich sein. In diesem Bereich, der sich mit der Vorhersage von Mobilität befasst (engl. *mobility prediction*), unterscheiden sich die Ansätze u.a. danach, ob die ganze zukünftige Netztopologie oder nur das Verhalten einzelner Knoten bestimmt werden soll. Ebenso wird nach der Art der Mobilitätsvorhersage unterschieden, z.B. der Abwanderung aus einem MANET bzw. einer Partition darin oder der Vorhersage der Kommunikationsfähigkeit zwischen Knoten innerhalb einer Partition.

Allerdings sollte immer bedacht werden, dass die Möglichkeit zur Vorhersage der Mobilität von Knoten im Netz beschränkt sein kann. So sprechen U. Bilstrup u.a. von einem sog. *Wissenshorizont*, über den ein Knoten in einem MANET maximal verfügen kann (vgl. [11]). Dieser Horizont entspricht der Anzahl von Hops über einen gewissen Zeitraum hinweg, in dem die Kommunikation mit den *sichtbaren* Knoten möglich ist. Abhängig ist der Horizont von der Kommunikationsreichweite und insbesondere der Mobilität. Je höher die Mobilität im Netzwerk ist, desto kleiner der Wissenshorizont. Daraus ergibt sich, dass eine Vorhersage über die Netztopolgie in weiteren Entfernungen als der Wissenshorizont nicht möglich ist. Innerhalb des Horizonts hingegen können Vorhersagen getroffen werden.

Im Folgenden werden einige Ideen zur Vorhersage von Mobilität vorgestellt, da der SLS (vgl. Kapitel 4) im Rahmen dieser Arbeit um die Berücksichtigung der Mobilität von Knoten zwischen Clustern erweitert werden soll.

Viele Forschungsansätze zur Mobilitätsvorhersage sind ursprünglich im Bereich der zellenförmigen Mobiltelefonsysteme entstanden (vgl. [45]). Das fundamentale Problem ist hier die Vorhersage darüber, ob ein mobiler Nutzer eine Zelle verlässt und wohin er sich bewegen wird. Diese Information ist z.B. wichtig für die Bandbreitenreservierung und die *call admission control*, aber auch für die Anwendung z.B. bei Kangaroo-Transaktionen, wie in Abschnitt 3.2.1 beschrieben. Ein großer Teil der entwickelten Ansätze basiert auf Markovmodellen, so dass die Vorhersage über die Zukunft des Benutzers auf Grundlage von aktuellen und vorhergehenden Zuständen bestimmt wird. Dabei werden u.a. das richtige Timing, Geschwindigkeit und Informationen über die Bewegung sowie die Abstände zwischen den Basisstationen und Mobiltelefonnutzern einbezogen. Kennzeichnend für die Entwicklungen in diesem Bereich ist, dass die Vorhersagen durch das Hauptleitungssystem (engl. *backbone systems*) zentral getroffen werden. Die Benutzer kommunizieren über Basisstationen mit einem Hop Entfernung und nicht direkt miteinander. Es erfolgt keine Topologieänderung im Laufe der Zeit, sondern nur die Benutzer bewegen sich durch die Zellen (vgl. z.B. [6, 41, 17]).

Für die Anwendung im Bereich von MANETs sind einige grundsätzliche Charakteristiken aus dem Mobilfunkbereich ein Problem, weshalb viele der entwickelten Ideen nicht direkt angewendet werden können. Es existieren keine festen Basisstationen, sondern die Benutzer kommunizieren direkt oder über mehrere Hops miteinander. Bedingt durch die Mobilität der Benutzer ändert sich zudem die Topologie des MANETs unter Umständen sehr häufig im Laufe der Zeit. Diese

besonderen Eigenschaften der MANETs müssen bei der Vorhersage von Mobilität berücksichtigt werden, um sinnvolle Aussagen liefern zu können. Forschungsansätze in diesem Bereich unterscheiden sich im Wesentlichen hinsichtlich ihrer Sicht auf das MANET, ob sie das gesamte Netz oder nur Ausschnitte davon für die Vorhersagen einbeziehen:

- **Zentralisierter Ansatz:** Ziel ist es, dass Knoten über globale Netzwerk- und Topologieinformationen verfügen. Die Mobilität von Knoten kann zu globalen Topologieänderungen führen, was somit dem ganzen Netz mitgeteilt werden muss. Probleme eines solchen Ansatzes sind unter Umständen extremer Kommunikationsoverhead für die begrenzten Ressourcen der Knoten.
- **Lokaler Ansatz:** Knoten verfügen hier über eine lokale Sicht auf das MANET, z.B. über alle anderen Knoten in konstanter Hop-Entfernung. Entsprechend dieser Informationen kann dann das Verhalten des Knotens daran angepasst werden.
- **Host-zentrierter Ansatz:** Dieser Ansatz betrachtet nur den einzelnen Knoten und dessen individuelle Wahrscheinlichkeit dafür, sich von seiner aktuellen Position wegzubewegen. Informationen über die Nachbarschaft des Knotens und deren Verhalten werden nicht mit einbezogen.

Einige der Forschungsansätze, bei denen Teilaspekte für diese Arbeit bezüglich des Umgangs mit Mobilität relevant sind, werden im Folgenden kurz vorgestellt. Dabei handelt es sich in allen Fällen um sog. *proaktive* Verfahren. Diese sammeln im Laufe der Zeit Informationen in Form einer Art *Historie*, die dann für die Vorhersage mit einbezogen wird.

3.3.1 Zentralisierter Ansatz für die Vorhersage von Mobilität

Bei dem Ansatz von F. De Rosa, A. Malizia und M. Mecell (vgl. dazu [45]) wird die Vorhersage der zukünftigen Mobilität anhand der Wahrscheinlichkeiten für zukünftige Verbindung bzw. Verbindungsverlust gemacht. Es wird von einem zugrunde liegenden MANET, bestehend aus fünf bis zehn Geräten, ausgegangen. Ein zentraler Knoten, der sog. *Koordinator*, verwaltet global Mobilitäts- und Topologieinformationen, indem er alle Abstände zwischen den Geräten untereinander erfragt. Voraussetzung ist genau eine zusammenhängende Komponente[1], damit man Informationen über alle Knoten und Abstände erhält und somit für verschiedene mögliche Topologien einen wahrscheinlichen nächsten Graphen im Hinblick auf die Verbindungsqualität bestimmen kann.

Dieses Verfahren ist für eine sog. *Koordinierungsschicht* entwickelt worden und soll Anwendung im Bereich von Katastrophenszenarien finden. Ziel ist es, dass der Koordinator alle Informationen über die Abstände der Knoten im Netz sammelt und deren Mobilität vorhersagt. Damit soll festgestellt werden, welche Knoten die Verbindung zueinander verlieren werden. Mit dieser Vorhersage soll die Bewegung der Knoten gesteuert werden können, so dass immer eine zusammenhängende Netzwerkkomponente bestehen bleibt.

Vorhersagetechnik Voraussetzung ist, dass zum Zeitpunkt $t_0 = 0$ alle Knoten miteinander verbunden sind. Dies muss später auch zu jedem Zeitpunkt t_i gewährleistet werden. Zum Zeitpunkt t_i sammelt der Koordinator die Abstandsinformationen von allen Geräten zu ihren jeweiligen Nachbarn. Zusätzlich sendet jeder Knoten auch eine *Historie* von k Abstandsinformationen

[1] Der Begriff Zusammenhangskomponente stammt aus der Graphentheorie (vgl. z.B. [18]): Ein nicht leerer Graph $G = (V, E)$ heißt *zusammenhängend*, wenn er für je zwei seiner Knoten i,j mit $i, j \in V(G)$ einen i-j-Weg enthält.

$t_{i-1}, t_{i-2}, t_{i-3} \ldots, t_{i-k}$ zu seinen früheren Nachbarn mit k =Anzahl der vereinbarten Zeiteinheiten. Mit diesem Informationen wird der wahrscheinliche *nächste Verbindungsgraph* durch den Koordinator berechnet. Für alle Abstände zwischen zwei Knoten i und j werden die Abstände zwischen ihnen gewichtet aufsummiert und durch die Summe der Anzahl von Informationen geteilt. Die Gewichtung erfolgt dabei derart, dass die letzte Abstandsinformation die höchste Gewichtung erhält und die älteste die geringste. Gleichzeitig werden damit auch die vergangenen Bewegungen mit einbezogen. Aus dieser geschätzten Entfernung zwischen den beiden Knoten kann nun eine Wahrscheinlichkeit erzeugt werden, indem man diesen Wert bezüglich der technischen Übertragungsreichweite normiert.

Diese Wahrscheinlichkeiten werden für alle Abstände der Knoten zu allen ihren Nachbarn berechnet. Diese Ergebnisse können für E Knoten in einer quadratischen Matrix $M = |E| \times |E|$ verwaltet werden. In dieser quadratischen Matrix sind die Diagonalelemente alle 1, da jeder Knoten sich selbst immer erreichen kann. Alle anderen Elemente $m_{i,j}$ entsprechen der Wahrscheinlichkeit dafür, dass zwei Knoten i und j in Verbindung stehen. Zusätzlich ist ein Grenzwert β nötig, der angibt, wie hoch die Wahrscheinlichkeit für eine Verbindung mindestens sein muss, um berücksichtigt zu werden. Die Matrix repräsentiert nun den Graph des Netzwerkes zum Zeitpunkt t_{i+1}, der auf Zusammenhang überprüft werden kann. Es kann ebenfalls festgestellt werden, ob Knoten über Singlehop oder Multihop verbunden sein werden. Aufgrund der Repräsentation des MANETs können dafür Graphenalgorithmen für Zusammenhang und Wegefindung verwendet werden.

Vor- und Nachteile Sinnvoll an diesem Ansatz ist, dass die Knoten selbst die Abstandsinformationen zu ihren Nachbarn berechnen und zunächst bei sich speichern. Somit wird erst bei Anfrage durch den Koordinator eine Nachrichtenübertragung notwendig. Von Vorteil ist auch, dass dadurch ein Koordinator einen Überblick darüber erhält, welche Knoten in der nächsten Zeiteinheit mit welcher Wahrscheinlichkeit erreichbar sein werden. Dazu wird die Historie über vergangene Bewegungen mit einbezogen.

Eine Problematik bei dieser Idee ist, dass keine plötzlichen Ausfälle berücksichtigt werden (z.B. aufgrund Defekts, Ausstellen durch den Benutzer oder Energie). Außerdem wird bisher davon ausgegangen, dass es nur einen Koordinator zur Verwaltung der Mobilitätsinformationen im MANET gibt. Da dieser jedoch die Informationen ad hoc bei Bedarf erfragt, besteht die Möglichkeit, dass es in einem MANET auch mehrere solche Koordinatoren für unterschiedliche Anwendungen gibt. Außerdem wird dieses Verfahren sehr instabil, wenn man statt mit kleinen Zeitfenstern mit größeren Zeiträumen wie Minuten arbeitet – insbesondere, wenn das zugrunde liegende Bewegungsmodell auf zufälligen Richtungsänderungen basiert. Ebenso wird bei dieser Methode, insbesondere bei dichten MANETs mit hoher Knotenanzahl, der Nachrichten- und Berechnungsaufwand sehr hoch, wenn das ganze Netz berücksichtigt wird.

3.3.2 Lokaler Ansatz für die Vorhersage von Mobilität

Das Verfahren zur Mobilitätsvorhersage von W. Su, S.-J. Lee und M. Gerla (vgl. [48]) basiert auf der sog. *Link Expiration Time* (Abk. LET). Ziel der Entwicklung dieses Ansatzes ist die Verbesserung der Effizienz von Ad-Hoc-Routing-Mechanismen. Da nur Informationen von Knoten einbezogen werden, die für eine Verbindung relevant sind, zählt dies Verfahren zu den lokalen Vorhersagemechanismen.

Als Annahme wird hier vorausgesetzt, dass in vielen Netzen die Bewegung der Knoten einer gewissen Regelmäßigkeit unterliegt. Zudem wird davon ausgegangen, dass die mobilen Geräte über GPS verfügen.

Bei diesem Vorhersageverfahren werden die GPS-Positionsinformationen mit Hilfe von Piggybacking (Deutsch *Huckepack-Verfahren*) mit den Datenpaketen übertragen, während eine funk-

tionierende Verbindung besteht. Diese Daten werden zur Schätzung dafür benutzt, wann die Verbindung zwischen zwei adjazenten Knoten zusammenbrechen wird. Dazu wird davon ausgegangen, dass die Knoten die Übertragungsreichweite r haben. Eine Verbindung bricht dann zusammen, wenn sich zwei Knoten nicht mehr im gegenseitigen Übertragungsbereich befinden. Sind also für zwei Knoten i und j deren Positionen (x_i, y_i) bzw. (x_j, y_j), die Geschwindigkeiten v_i bzw. v_j sowie die jeweiligen Bewegungsrichtungen θ_i bzw. θ_j bekannt, so kann der Zusammenbruch der Verbindung berechnet werden.

Vor- und Nachteile Im Gegensatz zu dem in Abschnitt 3.3.1 vorgestellten Ansatz werden hier weniger Ressourcen benötigt. Die Positionsinformationen werden nicht mit extra Nachrichten übertragen, sondern an andere Nachrichtenpakete gehängt und über eine bestehende Verbindung verschickt. Dadurch können Nachrichten und Energiekosten gespart werden. Gleichzeitig wird nur eine lokale Sicht auf die Topologie aufgebaut, die für die aktuelle Kommunikation notwendig ist. Das Berechnungsverfahren ist einfach und durch mobile Geräte verwendbar. Durch Berücksichtigung aller Verbindungen auf einem Weg kann auch die sog. *Route Expiration Time* (Abk. RET) bestimmt werden.

Als Einschränkung ist zu nennen, dass nur ein kurzer Zeitraum in der Zukunft abgedeckt wird (d.h. es wird die kurzfristige Mobilität betrachtet, vgl. Abschnitt 2.3.2 Punkt 3). Anschließend geht die Verbindung verloren und es kann nichts darüber ausgesagt werden, ob erneut eine Kommunikationsmöglichkeit bestehen wird. So werden z.B. keine Informationen über längerfristige Bewegungsmuster der Vergangenheit mit einbezogen. Auch die Möglichkeit plötzlicher Ausfälle spielt hier keine Rolle.

3.3.3 Host-zentrierte Ansätze zur Vorhersage von Mobilität

3.3.3.1 Autonomer und host-bezogener Ansatz zur Vorhersage von Mobilität

Der Ansatz von I.-R. Chen und N. Verma (vgl. [16]) ist für zellulare Systeme im Mobilfunkbereich und für Ad-Hoc Netzwerke entwickelt worden. Dieser ist insbesondere zur Verbesserung von Handoffs und dem Bandbreitenmanagement geeignet. Jeder Knoten kann selbst die Wahrscheinlichkeit dafür berechnen, dass er sich von seiner jetzigen Position zur nächsten innerhalb eines bestimmten Zeitfensters bewegen wird. Diese Berechnung basiert auf kürzlichen und vergangenen Bewegungen, die sowohl zufällig wie regelmäßig sein können.

Annahme ist hier, dass das geographische Gebiet des kabellosen Netzwerks als Gitternetz mit $n \times m$ Feldern dargestellt werden kann. Jedes Feld erhält eine ID und die Knoten können sich nur in Nachbarfeldern bewegen und nicht diagonal. Außerdem wird bei Ad-Hoc Netzwerken davon ausgegangen, dass Knoten immer ihre Position bestimmen können: außerhalb von Gebäuden mit GPS, innerhalb z.B. mittels Kurzstreckenfunk oder Infrarotsensoren (vgl. [16]). Bei Zellennetzwerken existiert eine Basisstation pro Feld, so dass ein Knoten immer weiß, wo er sich befindet. Somit können die mobilen Knoten immer ihren Aufenthaltsort sowie die Aufenthaltsdauer dort bestimmen.

Für die Vorhersage der Mobilität werden verschiedene Algorithmen eingesetzt, die die folgenden Strukturen verwenden. Zunächst ist der sog. *Bewegungscache* (engl. *movement cache*) nötig. Dieser speichert kürzliche Bewegungsinformationen, also die Bewegung von einem Feld im Gitternetz zu einem anderen, sowie die Aufenthaltsdauer in jedem Feld. Zusätzlich enthält jeder Eintrag einen Verweis auf den zu einer Bewegung gehörenden Weg. Wege werden in der sog. *Bewegungsmuster-Tabelle* (engl. *movement pattern table*) verwaltet. Diese setzen sich aus einer Folge von Feld-IDs zusammen, die durch die Bewegung eines Nutzers von einem Feld zum anderen entstehen. Anfangs- und Endfeld eines Weges werden durch die Länge der Aufenthalts-

dauer bestimmt. Dies sind sog. *residente Felder*, d.h. der Aufenthalt darin ist größer als ein Schwellenwert t, der im Vorfeld spezifiziert werden muss.

Der Basisalgorithmus für die Mobilitätsvorhersage sieht nun zwei Schritte vor. Zunächst werden mittels *Pattern Matching* alle Wege ermittelt, die aufgrund der gesammelten Bewegungsmuster in Frage kommen. Diesen Wegen werden Gewichte zugeordnet. Je besser ein gefundener Weg passt, desto höher sein Gewicht. Daraus ergibt sich eine Menge von in Frage kommenden Wegen. Anschließend wird mit diesen Informationen die Wahrscheinlichkeit dafür berechnet, dass das aktuelle Feld i verlassen und ein Nachbarfeld j betreten wird. Dazu wird die Bayesche Formel für bedingte Wahrscheinlichkeiten verwendet zur Berechnung der Wahrscheinlichkeit dafür, dass ein Knoten im Zeitfenster t vom aktuellen Feld i in ein Nachbarfeld j wandert.

Entspricht jedoch das Endfeld eines Weges dem aktuellen Feld, so wird im ersten Schritt ggf. ein passender Weg gefunden, aus dem jedoch dann kein mögliches Folgefeld hervorgeht. Deshalb wird dieser Teil um einen Greedy-Algorithmus erweitert. Dabei werden nicht nur die perfekt passenden Wege berücksichtigt, sondern auch diejenigen, die diesen nur als Teilweg enthalten und noch weiter gehen. Zusätzlich werden alle Wege berücksichtigt, die Teile des aktuellen Weges enthalten. Auch für diese wird dann die Wahrscheinlichkeit für den nächsten Schritt berechnet. Am Ende erhält man ein Feld, das durch den Knoten am wahrscheinlichsten im Zeitraum t betreten wird.

Neben dem genannten Verfahren gibt es weitere ähnliche Ansätze, die ebenfalls eine Art Gitterstruktur zur Abbildung des Netzes verwenden. So z.B. der Ansatz von S. Kwon, H. Park und K. Lee (vgl. [34]), der sich allgemein mit kabellosen Netzen befasst. Auch hier wird zur Mobilitätsvorhersage eine Historie der Nutzerbewegung mit einbezogen.

Vor- und Nachteile Als Vorteil der zugrunde liegenden Gitternetzstruktur ist zu nennen, dass Knoten ihren Aufenthaltsort sowohl innerhalb wie außerhalb von Gebäuden bestimmen können. Zudem können sie auch die Aufenthaltsdauer dort bestimmen. Simulationsergebnisse der Autoren zeigen, dass eine verhältnismäßig gute Vorhersage der Mobilität aufgrund der proaktiven Sammlung von Bewegungsinformationen erzielt werden kann.

Nachteilig ist jedoch, dass dieses Verfahren sehr rechen- und speicheraufwändig ist. Gerade für mobile Geräte mit wenig Ressourcen kann dies problematisch werden. Insbesondere die Erweiterung um den Greedy-Algorithmus führt zu einer hohen Komplexität in Bezug auf Rechenaufwand und Platzbedarf. Aufgrund der besonderen MANET-Charakteristiken gibt es noch keinen wirklichen Ansatz für Transaktionen, die ausschließlich im mobilen Netz durchgeführt werden. Zurzeit wird bei den gängigen Lösungen noch überall eine konsolidierte Datenbank im Hintergrund angenommen, zu der zumindest zeitweise eine Verbindung besteht und die sich um die eigentliche Transaktionsdurchführung kümmert.

3.3.3.2 Lokalisierung von besuchten Orten

N. Marmasse und C. Schmandt (vgl. [36]) haben auch einen host-zentrierten Ansatz, *comMotion* genannt, entwickelt. Dabei können die mobilen Geräte ebenfalls ihre Position in der Umgebung bestimmen. Außerdem werden persönliche Informationen des Nutzers zu verschiedenen Orten mit einbezogen. Damit ist die Vorhersage darüber möglich sein, welches Ziel angesteuert wird und wann dieses erreicht wird.

Als Annahme wird auch hier vorausgesetzt, dass die Geräte über GPS verfügen. Ein Agent zur Positionsbestimmung (engl. *location learning agent*) zeichnet Bewegungsmuster auf und lernt Orte, an denen sich ein Nutzer häufig aufhält. Neue gelernte Orte werden dann im Kontext gespeichert, z.B. physikalische Position, Datum und Zeit. Zusätzlich wird vom Benutzer eine

Bezeichnung für den neuen gelernten Ort erfragt. Das System lernt somit inkrementell und benötigt keinerlei Startkonfiguration. Wichtig zu bemerken ist, dass im Gegensatz zu dem Ansatz in Abschnitt 3.3.3.1 hier nicht davon ausgegangen wird, dass auch innerhalb eines Gebäudes die genaue Position ermittelt werden kann. GPS funktioniert nicht in Gebäuden, aber wenn ein Ort öfter besucht wird (z.B. ein Nutzer in ein Gebäude geht und dort eine Weile bleibt), so wird alles in einem vordefinierten Radius um diese Position als zusammenhängender Ort betrachtet. Ein Weg ist auch hier wieder eine Route zwischen bekannten Orten, wobei zwischen zwei Orten mehrere unterschiedliche Routen bestehen können. Die Wege sind von Bedeutung, da über die Wahl eines Weges sowie die Häufigkeit der Benutzung eines Weges der nächste Ort vorhergesagt werden kann.

Für das Lernen der Wege, die Vorhersage von Zielorten und das Schätzen der Ankunftszeit werden u.a. Verfahren wie Bayes Klassifizierer und Hidden Markov Modelle eingesetzt. Auf diese wird an dieser Stelle jedoch nicht näher eingegangen, sondern dazu sei auf [36], Abschnitt 4 verwiesen.

Vor- und Nachteile Als Vorteil dieses Verfahrens ist zunächst das inkrementelle und adaptive Lernen zu nennen. Es ist kein externer Datenbestand notwendig, ebenso wenig eine Startkonfiguration. Die Geräte können jederzeit in einer neuen Umgebung mit dem Lernen anfangen. Die Daten werden ausschließlich bei dem Knoten gespeichert und es ist keine Nachrichtenbelastung des Netzwerkes erforderlich. Zudem werden nur Orte berücksichtigt, die auch besucht werden, so dass dadurch Speicherplatz gespart und Overhead vermieden werden kann.

Es ist jedoch ggf. sehr zeitaufwändig, bis ein Ort mit einer gewissen statistischen Sicherheit gelernt worden ist. So ist dies Verfahren gerade zu Beginn nur bedingt einsetzbar, wenn ein Knoten in einer neuen Umgebung noch keinen Ort gelernt hat. Außerdem ist keine Unterscheidung zwischen nah beieinander liegenden Orten möglich, da GPS nur außerhalb von Gebäuden nutzbar ist. Inwieweit das tolerabel ist, hängt von den Rahmenbedingungen und der jeweiligen Anwendung ab.

3.4 Modelle für MANETs

MANETs zeichnen sich insbesondere durch ihre Dynamik aus. Aufgrund der ggf. permanenten Mobilität hat ein Knoten im Verlauf seiner Verbindung mit dem Netz immer wieder andere Knoten in seiner Nachbarschaft. Diese kann er dann mittels Funk erreichen. Dabei müssen jedoch Problematiken wie z.B. Interferenzen berücksichtigt werden, die Einflüsse auf die Verbindungen haben können. Jeder Knoten kann zudem als Relay-Station für das Weiterleiten von Nachrichten zwischen anderen Knoten funktionieren. Die Modellierung aller Aspekte eines MANETs (vgl. Kapitel 2) ist sehr problematisch. Entsprechend gibt es nur wenige Forschungsansätze.

Eine Ansatzmöglichkeit für die Modellierung ist die Betrachtung der Netzwerktopologie basierend auf der Konnektivität[2] von Knoten. Unabhängig vom Bewegungsmuster der Knoten und der verwendeten Funktechnologie kann man zu jedem Zeitpunkt die Topologie des Netzwerkes betrachten. Einen solchen Zustand kann man beispielsweise als *Graph* betrachten. Mit Hilfe dieser Modellierung können fundamentale Eigenschaften von MANETs modelliert und formalisiert werden, z.B. Konnektivität, Gradanzahl[3] oder Anzahl der Hops.

[2] Unter Konnektivität versteht man eine Verbindung oder die Art und Weise einer Verbindung bzw. die Verbindungsdichte.

[3] Die *Gradanzahl* eines Knoten A entspricht der Anzahl seiner Nachbarn, d.h. die Anzahl der Knoten, mit denen A direkt kommunizieren kann.

3.4.1 Repräsentation eines MANETs als Graph

Zur Modellierung von MANETs als Graphen kann die Graphentheorie verwendet werden. Für die formalen Definitionen von Graphen und ihren Eigenschaften sei beispielsweise auf [18] verwiesen.

Wendet man diese Graphen-Definitionen auf MANETs an (vgl. dazu [22]), kann man ein solches Netzwerk zu einem bestimmten Zeitpunkt (aufgrund u.a. von Mobilität) formal als Graph $G = (V, E)$ modellieren. Die *Knoten* entsprechen den mobilen Geräten und die *Kanten* den Kommunikationsverbindungen zwischen den Knoten. Zwei Knoten i und j mit $i, j \in V(G)$ sind über eine Kante $e = (i, j) \in E(G)$ miteinander verbunden, wenn sie sich im gegenseitigen Übertragungsbereich befinden und Nachrichten austauschen können. Als *Nachbarn eines Knotens* $i \in V(G)$ werden alle Knoten $j \neq i \in V(G)$ bezeichnet, mit denen i direkt kommunizieren kann. Die Anzahl der Nachbarn entspricht der *Gradanzahl* des Knotens i. Ist für den Nachrichtenaustausch zwischen zwei Knoten i und j ein Routing der Pakete über mehrere Knoten $k \neq i, j \in V(G)$ nötig, so entspricht die Länge des i-j-Weges P_{ij} (mit $P_{ij} \subseteq G$ Teilgraph von G) der Anzahl von *Hops*, die die Knoten voneinander entfernt sind. Kann ein Knoten i mit keinem anderen Knoten j kommunizieren, so ist er *isoliert*. Er stellt die kleinstmögliche *Komponente*[4] K eines Graphen dar. Tritt eine Netzwerkpartitionierung auf (vgl. Abschnitt 2.3.1), so entspricht jede der auftretenden Partitionen einer Komponente K_l im Graphen G mit $l = 1, \ldots, n$ Anzahl der Komponenten von G. Innerhalb einer Komponente K_l kann ein Knoten i jeden anderen Knoten $j \neq i$ über einen i-j-Weg erreichen. Die Kommunikation zwischen Knoten aus unterschiedlichen Komponenten ist nicht möglich.

Es bleibt zu bemerken, dass Graphen nur bedingt eine Abbildung eines echten MANETs liefern können. Die folgenden Beispiele für Graph-Modelle werden zeigen, dass immer nur ein Zustand eines Graphen zu einem Zeitpunkt repräsentiert werden kann. Die Mobilität der Geräte führt zu einer Folge von Graphen, die dynamisch miteinander verbunden werden müssten, um ein realistisches Modell zu liefern. Auch Aspekte wie z.B. Interferenzen oder Abschattung, die durch Einflüsse der Umwelt entstehen, sind schwer einzubeziehen.

3.4.2 Graph-Modelle für MANETs

Für die Modellierung von MANETs als Graphen ist vor allem die Umsetzung der Verbindungen zwischen Knoten wichtig, da der Kommunikationsaustausch darauf beruht. Durch die in Abschnitt aufgezeigten Problematiken wie z.B. Mobilität oder unterschiedliche Signalausbreitung ändert sich jedoch das Netz ständig. Bei Modellen sind daher zwei wesentliche Faktoren zu berücksichtigen. In einem MANET hängt die *Verbindungswahrscheinlichkeit* sowohl vom Abstand zwischen den Knoten wie vom Fading ab. Außerdem sind Verbindungen in einem MANET *lokal korreliert*, d.h. es besteht eine Abhängigkeit der Verbindungen vom Abstand der Knoten zueinander. Eine direkte Konsequenz dieser Eigenschaft ist z.B. eine erhöhte Wahrscheinlichkeit für eine Verbindung zwischen zwei Knoten, wenn diese einen gemeinsamen Nachbarn haben.

Es gibt einige diesbezügliche Ansätze, die hier kurz beispielhaft genannt werden sollen. Als einer der ersten Ansätze sind dabei *Zufallsgraphen* zu nennen, die auf Erdös und Rényi zurückgehen (vgl. dazu [18, 22]). Bei diesem Modell ist die Verbindungswahrscheinlichkeit unabhängig vom Abstand der Knoten zueinander, d.h. diese ist gleich für jeweils zwei beliebige Knoten des Netzes. Die lokale Korrelation von Verbindungen wird bei diesem Ansatz nicht berücksichtigt.

Eine weitere Möglichkeit für eine Modellierung stellen *reguläre Gittergraphen* dar (vgl. dazu [22]). Hier werden die Knoten auf einem regulären Gitter platziert, bei dem adjazente Knoten gleich weit voneinander entfernt sind. Dadurch wird bei der Verbindungswahrscheinlichkeit zwar

[4] Eine *Komponente* eines Graphen $G = (V, E)$ ist ein Teilgraph T von G mit $T = (V' \subseteq V, E' \subseteq E)$ (vgl. [18]).

der Abstand zwischen Knoten prinzipiell mit einbezogen, allerdings wird dieser für zwei benachbarte Knoten als gleich angenommen. Auch findet die lokale Korrelation von Verbindungen keine Berücksichtigung.

Als drittes mögliches Modell sind *skalierungsfreie Graphen* zu nennen (vgl. dazu [22]). Die Bezeichnung rührt aus der Unabhängigkeit der Verteilung des Grades von der Netzwerkgröße. Wie beim Zufallsgraphen wird bei der Verbindungswahrscheinlichkeit von der Unabhängigkeit von den Abständen ausgegangen. Allerdings kann es in einem solchen Netz sog. *Hubs* geben. Diese Knoten verfügen häufiger über einen hohen Grad im Gegensatz zu den übrigen Knoten des Netzwerks. Bei diesen Hubs liegt entsprechend eine höhere Verbindungswahrscheinlichkeit vor. Trotz des erhöhten Auftritts von Clustern wird hier ebenfalls keine lokale Korrelation mit einbezogen (Ausnahmen sind bedingt soziale Netzwerke, vgl. dazu [22] S. 39).

Der *pfadlose geometrische Zufallsgraph* ist eine weitere Modellierungsvariante (vgl. dazu [22]). Dabei wird die Übertragungsreichweite R der Knoten mit einbezogen und wird als *Überdeckungsbereich* bezeichnet. Ein Knoten kann mit allen Knoten kommunizieren, die in diesem Bereich mit Radius R liegen. Das Modell resultiert in einer perfekten Abdeckung des Gebietes mit Kreisen. Die Verbindungswahrscheinlichkeit ist hier nur vom Abstand der Knoten zum Zentrum des Übertragungsbereiches abhängig, d.h. es ist derselbe für je zwei Knoten mit demselben Abstand. Durch die mit dem Modell eingeführten Abstandsabhängigkeiten wird auch die Korrelation zwischen Verbindungen berücksichtigt.

Als letztes soll hier der *lognormale geometrische Zufallsgraph* vorgestellt werden. Mit diesem wird versucht, eine noch bessere Abbildung der Realität zu gewährleisten. Dies impliziert die Berücksichtigung von mittelgroßen Funksignalstärkeschwankungen, die als lognormal verteilt angenommen werden (vgl. [22] S. 32). Dadurch kann die Verbindungswahrscheinlichkeit abhängig vom Abstand sein, die mittels einer probabilistischen Funktion berechenbar ist. Auch die lokale Korrelation wird mit einbezogen.

3.5 Bewegungsmodelle für MANETs

Bewegungsmodelle sind einer der wichtigsten Faktoren bei der Untersuchung von Anwendungen in MANETs. Dies ist darauf zurückzuführen, dass insbesondere die Mobilität ein solches Netz auszeichnet (vgl. Abschnitt 2.1). Dementsprechend beeinflusst das zugrunde liegende Bewegungsmodell auch die Performance in einem MANET wie z.B. die Paketweiterleitung mittels Routing.

Es gibt sog. *homogene* Modelle wie z.B. das Random Waypoint Bewegungsmodell oder das Random Walk Bewegungsmodell. Darin bewegen sich die mobilen Knoten homogen entlang von zufällig ermittelten Strecken. Die Simulation von Charakteristiken realer Szenarien ist mit derartigen Modellen daher schwierig. Es gibt jedoch auch einige *heterogene* Modelle, die eine Abbildung bestimmter realer Szenarien versuchen. Die Art dieser Bewegungsmodelle hängt dabei stark von dem Anwendungsbereich ab, in dem das MANET eingesetzt werden soll. So simulieren diese nur sehr spezifische Szenarien, z.B. verschiedene Verhaltensweisen von Menschen wie Gruppenmobilität oder Straßenzüge. Aspekte wie Interferenzen oder Gebäude, die eine perfekte Signalausbreitung verhindern, werden bei den Modellen nicht berücksichtigt. Die mit derartigen Bewegungsmodellen gewonnenen Ergebnisse sind daher nur für spezielle Szenarien mit den jeweiligen Rahmenbedingungen aussagekräftig.

Einige der entwickelten Bewegungsmodelle werden im Folgenden kurz vorgestellt. Das bekannteste und am meisten verwendete Modell davon ist das Random Waypoint Mobilitätsmodell. Mit dem Area Graph-based Mobilitätsmodell wird versucht, eine bessere Abbildung realer Szenarien (z.B. eines Universitätscampus) zu schaffen. Beide Modelle werden ausführlicher vorgestellt, da

diese bei den Simulationen dieser Arbeit verwendet werden. Abschließend wird noch kurz auf weitere Ansätze von Bewegungsmodellen für speziellere Szenarien eingegangen.

3.5.1 Random Waypoint Mobilitätsmodell

Das Random Waypoint Mobilitätsmodell (Abk. RWP) ist das bei Simulationen von MANETs am meisten genutzte homogene Bewegungsmodell (vgl. dazu [10]). Die Knoten können sich hier in einem vordefinierten Gebiet nahezu uneingeschränkt bewegen. Dabei erfolgt die Bewegung der Knoten entlang von Strecken von einem Punkt zu einem anderen. Wenn sich ein Knoten an einem Punkt befindet, wird zufällig ein neuer Zielpunkt im Areal bestimmt sowie die Geschwindigkeit auf dem Weg dorthin. Die berechneten Zielpunkte sind gleichverteilt über das gesamte Simulationsgebiet. Ist der Zielpunkt erreicht worden, macht der Knoten eine Pause. Dieser Zeitraum kann im Vorfeld bei der Erzeugung spezifiziert werden. Nach dieser Pause beginnt der Prozess erneut. Das Verfahren wird für jeden Knoten bis zum Simulationsende immer wieder durchgeführt.

Obwohl es das meist genutzte Bewegungsmodell für Simulationen von MANETs ist, sind einige Probleme bekannt. Analysen haben beispielsweise gezeigt, dass die räumliche Knotenverteilung nicht wirklich gleichverteilt ist. Es zeigt sich vielmehr eine Häufung im Zentrum. Dies wirkt sich insbesondere auf die Konnektivität zwischen den Knoten aus, da diese in einigen Bereichen des Areals höher als in anderen ist (vgl. [35]). Außerdem hat sich gezeigt, dass die Knotengeschwindigkeit im Verlauf der Simulation stetig abnimmt und im Schnitt nicht durchgehend konstant ist (vgl. [54]).

3.5.2 Area Graph-Based Mobilitätsmodell

Das Area Graph-Based Bewegegungsmodell (Abk. AGBMM) ist im Gegensatz zu dem zuvor vorgestellten Modell ein heterogenes Mobilitätsmodell. Hiermit wird die Modellierung realer Szenarien möglich. Häufig bilden sich in der Realität clusterartige Szenarien, z.B. auf einem Universitätscampus. In einem Gebäude wie beispielsweise einer Bibliothek oder einem Hörsaal sind größere Menschenmengen versammelt als auf dem Weg dazwischen (vgl. [12, 49] zu dem Modell).

Die Idee des Modells basiert auf einem sog. *Area-Graph* bestehend aus Knoten, die verschiedene Orte als rechteckige *Cluster* (die sog. *Areas*) mit unterschiedlichen Knotendichten repräsentieren. Dieser Graph ist gerichtet und gewichtet (vgl. dazu die Definition in Abschnitt 3.4.1). Dabei entspricht die Richtung der Kanten der Bewegungsrichtung der Knoten (z.B. Wegen zwischen Gebäuden) und die Gewichtung der Wahrscheinlichkeit dafür, dass diese Kante benutzt wird. Die Cluster haben eine relativ hohe Knotendichte, da sie beispielsweise ein Gebäude repräsentieren. Im Gegensatz dazu ist die Dichte auf den Kanten geringer. Bei der Erzeugung eines solchen Szenarios wird die Aufenthaltsdauer in einem Cluster zufällig bestimmt. Grundlage dafür ist ein für das Szenario vorgegebenes Zeitintervall für einen durchschnittlichen Aufenthalt.

Bei der Bewegung der Knoten ist zwischen der Bewegung innerhalb und außerhalb der Cluster zu unterscheiden. Innerhalb eines Clusters erfolgt die Bewegung entsprechend dem Random Waypoint Mobilitätsmodell (vgl. Abschnitt 3.5.1). Außerhalb des Clusters bewegt sich ein Knoten auf einer der Kanten, die die Cluster verbinden. Die Geschwindigkeit der Knoten auf einer Kante kann dabei durch Parameter bestimmt und an die zu simulierenden Szenarien angepasst werden.

3.5.3 Weitere Mobilitätsmodelle

An dieser Stelle soll beispielhaft darauf hingewiesen werden, dass es außer den beiden oben genannten Modelle weitere Ansätze gibt.

So wird mit dem *Manhattan Mobilitätsmodell* (Abk. MMM) die Bewegung von Knoten entlang von Straßen simuliert (vgl. [8]). Dazu bewegen sich die Knoten entlang von horizontalen und vertikalen Straßen auf der Karte einer Stadt. Die Schnittstellen entsprechen Kreuzungen, wobei die Wahrscheinlichkeit 50% dafür beträgt, dass ein Knoten auf derselben Straße bleibt. Für das Abbiegen beträgt die Wahrscheinlichkeit jeweils 25%. Da Straßenzüge und das menschliche Verhalten (u.a. an Kreuzungen) deutlich komplexer sind, ist dieses Modell häufig nicht ausreichend für die Abbildung der Realität. Auch Einflüsse der Umgebung (z.B. Häuser) auf die Signalausbreitung werden nicht berücksichtigt. Ähnlich verhält es sich bei dem *Freeway Mobilitätsmodell,* das die Abbildung von Autobahnen oder Schnellstraßen versucht (vgl. [19]).

Als Beispiel für ein Gruppenmobilitätsmodell soll hier das *Referenzpunkt-Gruppenmobilitätsmodell* (engl. *Reference Point Group Mobility Modell*, Abk. RPGM, vgl. [23]) genannt werden. Dies findet u.a. bei der Simulation von militärischen Kriegsgebieten Anwendung. Es wird von Gruppen ausgegangen, bei der jede ein logisches Zentrum, den sog. Gruppenführer, hat. Dieser bestimmt das Bewegungsverhalten der Gruppe. Jedes Mitglied wird zufällig in der Nachbarschaft des Gruppenführers platziert. Die Geschwindigkeit und Bewegungsrichtung der Mitglieder werden dabei vom Gruppenführer abgeleitet und variieren von diesen Werten mit zufälliger Abweichung. Auch dieses Modell ist nur für bestimmte Anwendungsszenarien einsetzbar.

3.6 Routing in MANETs

Die vorigen Abschnitte haben sich mit Modellen für MANETs und möglichen Bewegungsmodellen für solche Netze befasst. Ein zentraler Aspekt ist zudem die Datenübertragung zwischen Knoten. In einem Netzwerk erfolgt die zielgerichtete Weiterleitung der Daten durch *Routing-Protokolle* (vgl. z.B. [42]). Diese sollen einen Weg vom Quell- zum Zielknoten bestimmen. Ein solcher Weg sollte möglichst kurz sein und die Netzressourcen schonen, d.h. nur gering belastete Knoten auswählen. Ein Weg sollte in möglichst kurzer Zeit und mit möglichst wenig Nachrichten gefunden werden. Zudem sollten Speicher- und Rechenkapazitäten der Knoten im Netz gespart werden, d.h. die Routing-Tabellen sollten möglichst klein sein. Kleine Routing-Tabellen sind wichtig, da diese regelmäßig aktualisiert werden, z.B. wenn Knoten verschwinden bzw. neue ins Netz kommen oder sich Knoten außer Reichweite bewegen. Durch die Mobilität der Knoten in einem MANET kommen weitere Anforderungen an das Routing hinzu. Es existieren keine zentralen Instanzen wie z.B. Basisstationen, so dass die Knoten die Netztopolgie nicht mitgeteilt bekommen. Jeder Knoten muss sich diese selbst erarbeiten. Dabei stellt der ständige Topologiewechsel aufgrund von Mobilität eine besondere Herausforderung dar. Auch wechselnde Metriken der Übertragungsstrecken, z.B. bedingt durch Interferenzen, müssen berücksichtigt werden. Zudem sind, im Gegensatz zu fest verkabelten Netzen, die Ressourcen der Knoten beschränkt. Ein Routing-Protokoll sollte daher die Systemleistung und Energieressourcen der mobilen Geräte schonen.

Es gibt für MANETs eine Reihe von verschiedenen Ansätzen für Routing-Verfahren, die versuchen, die oben genannten Aspekte zu berücksichtigen. Im Folgenden werden die grundlegenden Ideen zweier Gruppen vorgestellt, ohne allerdings auf Details einzelner Protokolle konkret einzugehen. Es ist darauf hinzuweisen, dass es noch eine Vielzahl weiterer Routing-Protokolle gibt, die z.B. auch Energieaspekte mit einbeziehen.

3.6.1 Positionsbasierte Routingverfahren

Zur exakten Bestimmung der Position von Knoten im MANET werden geodätische Informationen genutzt, die man z.B. über GPS-Empfänger erhält. Auf Grundlage dieser Daten kann man dann den kürzesten und besten Weg zwischen der Quelle und dem Ziel ermitteln. Ein Beispiel für ein solches Routing-Protokoll ist *Location Aided Routing* (Abk. LAR, vgl. [31]).

3.6.2 Topologiebasierte Routingverfahren

Bei den Ansätzen der topologiebasierten Routingverfahren werden nur Informationen über die Verbindung eines Knoten zu seinen Nachbarn benötigt. Es reicht aus zu wissen, welche Knoten eine direkte Verbindung zueinander haben und welche über andere Knoten Nachrichten austauschen können. Das Wissen über die Topologie erhält ein Knoten durch das Versenden sog. *Hello*-Pakete und Antworten darauf. Die Unterscheidung bei den Routingverfahren erfolgt dahingehend, zu welchem Zeitpunkt sich ein Knoten die Informationen über die Topologie des Netzes beschafft. Werden die Informationen gesammelt, bevor die Notwendigkeit besteht, eine Nachricht zu schicken, so handelt es sich um ein *proaktives* Routing. Wenn hingegen Topologieinformationen erst ermittelt werden, wenn eine Nachricht verschickt werden soll, so spricht man von *reaktivem* Routing.

3.6.2.1 Proaktives Routing

Der Weg, über den ein Paket bei der Kommunikation transportiert werden soll, steht vorher schon fest. Diese Informationen werden im Vorfeld durch das Versenden der Hello-Pakete und der Antworten darauf ermittelt. Auf diese Art und Weise kann eine Nachricht ohne Zeitverzögerung verschickt werden. Ein Beispiel für ein solches Protokoll ist das *Destination Sequence Distance Vector Routing* (Abk. DSDV, vgl. [43]).

Problematisch ist bei diesem Ansatz, dass eine Vielzahl von Nachrichten verschickt wird, um alle möglichen Wege zu ermitteln. Je nach Bedarf werden diese jedoch ggf. nicht benötigt.

3.6.2.2 Reaktives Routing

In diesem Fall wird der Weg für Nachrichten zwischen zwei Knoten erst bei Bedarf einer Übertragung bestimmt. Bei der ersten Nachricht, die zwischen zwei Knoten verschickt wird, muss daher mit einer Zeitverzögerung gerechnet werden. Anschließend sind nur noch Kontrollpakete notwendig. Beispiele für derartige Routing-Protokolle sind z.B. das *Dynamic Source Routing* (Abk. DSR, vgl. [28]) und das *Ad-hoc On-Demand Distance Vector Routing* (Abk. AODV, vgl. [44]).

Ein Vorteil dieses Verfahrens ist der geringere Energieverbrauch. Zudem können Nachrichten gespart werden. Dafür muss allerdings beim ersten Paket mit einer Zeitverzögerung bei der Übertragung gerechnet werden.

3.6.2.3 Hybride Routingmechanismen

Bei hybriden Routingverfahren sollen die Vorteile von proaktivem und reaktivem Routing vereinigt werden. So kann z.B. in einem kleineren räumlichen Bereich ein proaktives Routing eingesetzt werden, während für weiter entfernte Gebiete ein reaktives Verfahren eingesetzt wird. Damit stehen für nahe Ziele die Wege sofort ohne Zeitverzögerung zur Verfügung. Bei weiter entfernten Knoten hingegen muss beim ersten Paket mit einer Verzögerung gerechnet werden. Ein Beispiel

für ein solches Verfahren ist z.B. das *Hybrid Routing Protocol for Large Scale Mobile Ad Hoc Networks with Mobile Backbones* (Abk. HRPLS, vgl. [39]).

3.7 Fazit

Im Bereich mobiler Transaktionen zeigen sich zurzeit noch keine wirklichen Ansätze für *echte* mobile Transaktionen. Bisher wird immer vom Vorhandensein konsolidierter Datenbanken und Basisstationen ausgegangen, zu denen zumindest temporär eine Verbindung besteht. Mit dem SLS soll im Rahmen dieser Arbeit ein Ansatz für rein mobile Transaktionen untersucht werden. Insbesondere die Verkürzung von Unsicherheitszeiten soll dabei im Vordergrund stehen. Dazu müssen die Fehlermöglichkeiten in MANETs mit einkalkuliert werden. Bisher werden bereits technische Defekte und beschränkte Energieressourcen berücksichtigt. Zusätzlich soll im Rahmen dieser Arbeit die Einbeziehung von Mobilität versucht werden. Dazu soll ein entsprechendes Verfahren entwickelt werden, um die Aufenthaltsdauer und Aufenthaltshäufigkeit zu ermitteln, um damit eine Wahrscheinlichkeit für das Verlassen eines Ortes – die sog. *Abwanderungsrate* – schätzen zu können.

Da Mobilität eine derart wichtige Rolle spielt, wird an deren Vorhersage intensiv geforscht. Allerdings sind diese Ansätze jeweils nur in Abhängigkeit vom Anwendungsgebiet einsetzbar. Es zeigt sich, dass sich viele dieser Ansätze nur mit der *kurzfristigen* Mobilität innerhalb der nächsten Sekunden oder Minuten befassen. Die sog. *langfristige* Mobilität, also wie lange sich jemand irgendwo aufhält, wird selten untersucht. Lediglich in dem Ansatz in Abschnitt 3.3.3.2 wird dies versucht. Die langfristige Mobilität spielt jedoch insbesondere dann eine Rolle, wenn die Wahrscheinlichkeit dafür gefragt ist, dass ein Nutzer einen Ort verlässt und irgendwann wieder zurückkommt.

Trotz intensiver Forschungen ist die Modellierung eines MANETs aufgrund seiner Dynamik sehr schwierig. Graphen können immer nur eine momentane Situation abdecken. Durch die Mobilität ändert sich der Zustand des Netzes und somit auch der des Graphen permanent. Es ist auch schwer die Kommunikationsfähigkeit zwischen Knoten über einen größeren Zeitraum abzuschätzen. Mit einem Graphen ist dies insbesondere deswegen problematisch, weil er nur einen momentanen Zustand darstellt.

In Simulationen kann die dynamische Änderung des Netzes z.B. durch Bewegungsmodelle beeinflusst werden. Allerdings reflektieren diese häufig nur ein bestimmtes Anwendungsgebiet und bilden die Realität nur bedingt ab. Untersuchungen mit einem Bewegungsmodell können daher in den meisten Fällen nicht als allgemeingültig angenommen werden.

Auch im Hinblick auf die Nachrichtenweiterleitung gibt es verschiedene Ansätze. Das Ziel ist auch hier, Nachrichten trotz der Mobilität möglichst schnell und kostengünstig vom Sender zum Empfänger zu transportieren. Dabei ist bei der Auswahl eines Routings häufig auch die Netzstruktur von Bedeutung.

Kapitel 4

Shared Log Space (SLS)

4.1 Grundlegende Idee des SLS

Der SLS ist ein Ansatz für Middleware in MANETs zur Erhöhung von Datenverfügbarkeit und Datensicherheit für Applikationen. Mit diesem Verfahren soll die Unsicherheitszeit von Teilnehmern beim Auftritt von Fehlern während der Commit-Phase einer Transation verkürzt werden. Dazu wird das gesamte Netzwerk als global verteilter Speicher benutzt, um den Ausgang einer Transaktion, das sog. *Transaktionslog*, für ausgefallene Teilnehmer zu sichern. Verbindet sich ein ausgefallener Teilnehmer wieder mit dem Netz, soll er die Möglichkeit haben, den Ausgang einer Transaktion auch dann zu erhalten, wenn sowohl der Koordinator wie die anderen Teilnehmer nicht mehr zur Verfügung stehen. Ziel ist es daher, die gewünschte Verfügbarkeit eines Transaktionslogs durch *Redundanz* zu garantieren. Ein Hauptmechanismus ist *Datenverteilung*, um das Transaktionslog auf mehreren Knoten im MANET zu speichern. Bei diesem Verfahren arbeiten die Knoten des Netzes sowohl bei der Verteilung wie bei der Speicherung eines Transaktionslogs kooperativ zusammen.

Um die Speicherkapazitäten der Geräte zu schonen und die Nachrichtenlast im Netzwerk möglichst gering zu halten, wird sowohl die Dauer der Speicherung wie auch der Grad der Verteilung an die Bedürfnisse der jeweiligen Anwendung und der Teilnehmer angepasst. Dazu sind zwei Parameter notwendig. Zum einen wird die Spezifikation darüber benötigt, mit welcher Wahrscheinlichkeit für ausgefallene Teilnehmer das Transaktionslog im MANET wieder auffindbar sein soll. Dieser Wert wird im Folgenden als $p_{retrieval_{app}}$ bezeichnet. Außerdem muss vereinbart werden, über welchen Zeitraum hinweg das Transaktionslog gespeichert werden soll. Dieser Zeitraum t_m, in dem das Transaktionslog mit der Wahrscheinlichkeit $p_{retireval_{app}}$ innerhalb des MANETs zur Verfügung stehen soll, wird im Folgenden als *garantierte Recoveryzeit* (engl. *Mission Time*) bezeichnet. Beide Kriterien beeinflussen hauptsächlich, wie stark die Knoten des Netzes an der Speicherung eines Transaktionslogs beteiligt sind. Da beide eng zusammenhängen, werden sie im weiteren Verlauf als *gewünschte Zuverlässigkeit* R_{SLS} bezeichnet.

Zur Garantie der geforderten Zuverlässigkeit R_{SLS} müssen die Transaktionslogs bei Ausfällen von Teilnehmern im MANET verteilt werden. Dazu werden verschiedene Verteilungsstrategien verwendet, die in den folgenden Abschnitten kurz umrissen werden.

Bei der Auswahl und Umsetzung der Verteilungsstrategien sind die in Kapitel 2 aufgezeigten Problematiken wie beschränkte Energieressourcen, begrenzte Speicherkapazitäten und Übertragungsbandbreite der mobilen Geräte zu berücksichtigen. Ebenso sollen Charakteristiken von MANETs selbst beachtet werden, z.B. unterschiedliche Dichten oder die ggf. heterogene technische Ausstattung der Geräte im Netz. Mit einer dem aktuellen Netzwerk angepassten Strategie sollen Energie (Nachrichten) und Speicher gespart werden.

Im folgenden Abschnitt 4.1.1 werden zunächst die Verteilungsstrategien einführend kurz vorgestellt. Anschließend folgt in Abschnitt 4.2 das Zuverlässigkeitsmodell mit den jeweiligen Berechnungsverfahren für die Verteilungsmechanismen des SLS. Zuletzt werden im letzten Abschnitt 4.3 die Prozessabläufe detailliert vorgestellt, die für die Simulationen implementiert werden. Ergänzend dazu wird ein einfacher Vergleichsprozessablauf für Transaktionen ohne SLS-Charak-

teristiken vorgestellt, mit dem eine Evaluation des SLS in den Simulationen möglich wird.

4.1.1 Verteilungs- und Recoverystrategien des SLS

4.1.1.1 Verteilungsstrategien des Koordinators

Die Verteilungsstrategien versuchen die in Kapitel 2 beschriebenen Charakteristiken von MANETs beim Auftritt von Fehlern zu berücksichtigen. Hauptsächlich unterscheiden sich die Strategien im Hinblick auf den Aufwand bei der Verteilung, d.h. die Anzahl und die Auswahl der Knoten, die das Transaktionslog speichern sollen. Eine Übersicht über die Verteilungsstrategien kann der Tabelle 4.1 entnommen werden.

Zunächst wird nach der *Dichte des* MANETs unterschieden, also ob dieses dicht oder dünn mit Knoten besetzt ist. In einem dichten MANET kann angenommen werden, dass ein beliebiger Knoten für einen anderen Knoten mit großer Wahrscheinlichkeit erreichbar ist. Diese Erreichbarkeit kann für ein dünnes MANET nicht vorausgesetzt werden bzw. die Wahrscheinlichkeit dafür ist sehr gering.

Ergänzend zur Dichte des Netzes können Knoten mit sog. *Kontextbewusstsein* berücksichtigt werden. Diese sind aufgrund ihrer technischen Ausstattung in der Lage, ihre eigene Ausfallwahrscheinlichkeit zu berechnen (vgl. dazu Abschnitt 2.2 zum Systemmodell sowie Abschnitt 4.2.1). Knoten ohne Kontextbewusstsein verfügen nicht über diese Fähigkeit.

Aus diesen Überlegungen ergeben sich die folgenden Strategien für die jeweiligen Kombinationen von Dichte und Kontextbewusstsein:

1. **Dichtes MANET und Knoten mit Kontextbewusstsein (DirDissCA):** Der Koordinator kann hier aus seiner Umgebung n Knoten mit geringer Ausfallwahrscheinlichkeit bestimmen, mit denen er die gewünschten Garantien gewährleisten kann. Da angenommen wird, dass eine große Anzahl von Knoten im Netz für den Koordinator erreichbar ist, kann er diese n Knoten zum Zeitpunkt der Verteilung erreichen. Diese Strategie wird deshalb als *gerichtete Verteilung auf n ausgewählte Knoten* bezeichnet.

2. **Dichtes MANET und Knoten ohne Kontextbewusstsein (DirDissCUA):** Sind die Knoten nicht in der Lage, ihre eigene Ausfallwahrscheinlichkeit zu bestimmen, kann der Koordinator keine besonders ausfallsicheren Geräte auswählen. Er muss die Ausfallwahrscheinlichkeit global schätzen bzw. diese wird durch Konfiguration vorgegeben (wie im Fall dieser Arbeit). Damit kann er die Anzahl n von für die Verteilung notwendigen Knoten bestimmen. Aufgrund der Annahme eines dichten MANETs wird er in seiner Umgebung n Knoten mittels Broadcast zum Speichern des Transaktionslogs erreichen. Daher wird dieses Verfahren *gerichtete Verteilung auf n anonyme Knoten* genannt.

3. **Dünnes MANET (UnDirDiss):** Für ein dünnes MANET kann diese Erreichbarkeit nicht vorausgesetzt werden. Es kann also nicht davon ausgegangen werden, dass genügend ausfallsichere Knoten für die Verteilung zur Verfügung stehen. Hier wird ein modifiziertes Fluten verwendet, so dass zwar jeder (mittels Broadcast) erreichbare Knoten das Transaktionslog erhält, aber nur ein durch den Koordinator bestimmter Anteil des Netzes dieses speichert. Da in einem dünnen MANET das Kontextbewusstsein keine Rolle spielt, bezeichnet man diesen Mechanismus als *ungerichtete Verteilung*.

	dichtes MANET	**dünnes MANET**
Knoten mit Kontextbewusstsein	gerichtete Verteilung auf n ausgewählte Knoten (DirDissCA)	ungerichtete Verteilung (UnDirDiss)
Knoten ohne Kontextbewusstsein	gerichtete Verteilung auf n anonyme Knoten (DisDissCUA)	ungerichtete Verteilung (UnDirDiss)

Tabelle 4.1: Übersicht über die Verteilungsstrategien des SLS

4.1.1.2 Recoverystrategien ausgefallener Teilnehmer

Der SLS-Prozessablauf verwendet die im vorigen Abschnitt 4.1.1 eingeführten Verfahren DirDissCA, DirDissCUA und UnDirDiss zur Verteilung eines Transaktionslogs im MANET beim Auftritt von Fehlersituationen. Entsprechend der benutzten Verteilungsstrategie verwenden Teilnehmer bei einer Recovery ebenfalls unterschiedliche Methoden. In jedem Fall sendet ein ausgefallener Teilnehmer bei jedem Reocvery-Versuch eine Anfrage an den Koordinator. Zusätzlich kommen Recoverys in Abhängigkeit von der vom Koordinator verwendeten Verteilungsstrategie hinzu. Dabei ist zu beachten, dass die Recovery bei DirDissCUA und UnDirDiss unabhängig von der Netzdichte identisch ist, da die Teilnehmer keinen der Knoten mit dem gespeicherten Transaktionslog kennen.

1. **Anfrage an den Koordinator:** Dies ist die Basisstrategie, die beim Vergleichs-Prozessablauf ausschließlich verwendet wird. Bei jedem Recovery-Versuch wird diese daher zur Vergleichbarkeit auch von ausgefallenen SLS-Teilnehmern benutzt.

2. **Gerichtete Recovery:** Der Koordinator hat DirDissCA zur Verteilung verwendet. Ein Teilnehmer schickt Recovery-Anfragen an die n ausgewählten ausfallsicheren Knoten.

3. **Ungerichtete Recovery:** Bei der Verteilung hat der Koordinator DirDissCUA oder UnDirDiss angewandt. Der Teilnehmer sendet einen Single-Hop-Broadcast an seine Nachbarn.

4.2 Zuverlässigkeitsmodell des SLS

Für das Zuverlässigkeitsmodell des SLS ist zunächst die Ausfallwahrscheinlichkeit $p_{failure}$ eines mobilen Gerätes eine wichtige Voraussetzung (für die Definition eines Ausfalls sei auf das Fehlermodell in Abschnitt 2.3 verwiesen). Wie sich diese bestimmt, wird daher im Folgenden als Grundlage für die weiteren Abschnitte als erstes eingeführt. Aufbauend darauf werden die Verteilungsstrategien vorgestellt. Dabei wird jeweils die Idee des Verfahrens sowohl für die Seite des Koordinators wie für die des Teilnehmers dargestellt. Ergänzend wird auf deren individuelle Vor- und Nachteile eingegangen. Ebenso wird das jeweilige Berechnungsverfahren für die Zuverlässigkeit vorgestellt.

4.2.1 Ausfallwahrscheinlichkeit eines Gerätes

Wie bereits in Abschnitt 4.1.1 eingeführt, können Geräte mit Kontextbewusstsein aufgrund ihrer technischen Ausstattung ihre eigene Ausfallwahrscheinlichkeit $p_{failure}$ berechnen. Sind im MANET hingegen nur Geräte ohne Kontextbewusstsein vorhanden, so muss der Koordinator eine globale Ausfallwahrscheinlichkeit aufgrund der ihm zur Verfügung stehenden global geschätzten Informationen berechnen.

In beiden Fällen werden bei der Ausfallwahrscheinlichkeit verschiedene Ursachen (vgl. dazu Abschnitt 2.3) für einen Fehler mit einbezogen. Zum einen wird die Wahrscheinlichkeit dafür berücksichtigt, dass ein technischer Defekt vorliegt. Außerdem spielt die Energie der Geräte eine Rolle, d.h. von ihr hängt ab, wie groß die Wahrscheinlichkeit für einen Ausfall aufgrund von erschöpften Energieressourcen ist. Bei beiden Aspekten wird davon ausgegangen, dass die diesen Ausfällen zugrunde liegende Wahrscheinlichkeitsverteilung exponentialverteilt ist.

Es werden daher zunächst in Abschnitt 4.2.1.1 die wichtigsten Eigenschaften der Exponentialverteilung vorgestellt. Anschließend zeigt Abschnitt 4.2.1.2, wie beim SLS die Ausfallwahrscheinlichkeit eines Gerätes berechnet wird.

4.2.1.1 Exponentialverteilung

Die Exponentialverteilung ist die am weitesten verbreitete Wahrscheinlichkeitsverteilung im Bereich der Zuverlässigkeitstheorie technischer Geräte (vgl. [7] S. 42). Ursache dafür ist ihre mathematische Einfachheit sowie die Tatsache, dass ihre Anwendung zu realistischen Lebenszeitmodellen für bestimmte Arten von technischen Geräten führt.

Im Folgenden wird die Exponentialverteilung in Bezug auf die Zuverlässigkeit eines technischen Gerätes eingeführt, so dass die Annahmen in den weiteren Abschnitten direkt verwendet werden können.

Sei die Zeit bis zum Ausfall eines Gerätes (die sog. *time to failure*) der Zeitraum von der Inbetriebnahme bis zum ersten Ausfall. Der Startpunkt ist entsprechend $t = 0$. Die Zeit bis zum Ausfall kann als Zufallsvariable T (diese wird hier als stetig verteilt angenommen, vgl. [25] S. 19) interpretiert werden. Auch der Zustand eines Gerätes zum Zeitpunkt t kann durch eine Zufallszahl, die Zustandsvariable $X(t)$, wie folgt beschrieben werden:

$$X(t) = \begin{cases} 1 & \text{Das Gerät ist funktionsfähig zum Zeitpunkt } t \\ 0 & \text{Das Gerät ist nicht funktionsfähig zum Zeitpunkt } t \end{cases} \tag{4.1}$$

Eine stetige Zufallsvariable heißt *exponentialverteilt*, wenn ihre Wahrscheinlichkeitsdichte mit einem festen Parameter $\lambda > 0$ wie folgt definiert ist (vgl. [46, 25]):

$$f(t) = \begin{cases} \lambda e^{-\lambda t} & \text{für } t \geq 0 \\ 0 & \text{für } t < 0 \end{cases} \tag{4.2}$$

Die zugehörige Verteilungsfunktion $F(t)$ ist entsprechend definiert als:

$$\begin{aligned} F(t) &= P\{T \leq t\} \\ &= \int_0^t \lambda e^{-\lambda x} dx \\ &= 1 - e^{-\lambda t} \text{ für } x \geq 0 \end{aligned} \tag{4.3}$$

F(t) beschreibt die Wahrscheinlichkeit dafür, dass ein Gerät im Zeitintervall $(0, t]$ ausfällt. Im Gegensatz dazu beschreibt die Zuverlässigkeitsfunktion *R(t)* die Wahrscheinlichkeit dafür, dass

ein Gerät im Zeitintervall $(0, t]$ *nicht* ausfällt und zum Zeitpunkt t noch funktionsfähig ist (vgl. u.a. [7, 25, 46]). Die Zuverlässigkeit *R(t)* ist wie folgt beschreibbar

$$\begin{aligned} R(t) &= P(T > t) \\ &= \int_t^\infty \lambda e^{-\lambda x} dx \\ &= e^{-\lambda t} \quad \text{für } t > 0 \end{aligned} \tag{4.4}$$

Daraus ergibt sich, dass $R(t) + F(t) = 1$ ist.

Der konstante Parameter λ wird in Bezug auf die Zuverlässigkeit technischer Geräte auch als Ausfallrate λ bezeichnet. Bei dieser Verteilung ist λ der Kehrwert des Erwartungswertes und definiert als (vgl. [46] S. 161)

$$E(t) = \frac{1}{\lambda} \tag{4.5}$$

Da λ der Kehrwert des Erwartungswertes ist, kann mit Hilfe von Formel 4.5 der Parameter λ bei bekanntem Erwartungswert einfach berechnet werden.

Eine wesentliche und wichtige Charakteristik der Exponentialverteilung im Rahmen dieser Arbeit ist ihre sog. *Gedächtnislosigkeit*. Um diese Eigenschaft zu verstehen, betrachtet man z.B. eine Zufallsvariable Y, die die Gesamtlebensdauer eines Gerätes beschreibt, also die Länge des Zeitraums, bis ein Gerät ausfällt. Interessant ist nun die Wahrscheinlichkeit dafür, dass ein zu einem beliebigen Zeitpunkt t noch funktionsfähiges Gerät noch während des zukünftigen Zeitraums Δt funktionieren wird. Hat ein Gerät also bis zum Zeitpunkt t funktioniert und soll auch über den Zeitraum Δt hinweg funktionsfähig sein, so muss die Gesamtlebensdauer größer sein als $t + \Delta t$. Es gilt also (für den Beweis vgl. z.B. [46] S. 161):

$$P\{Y > t + \Delta t\} = P\{Y > \Delta t\} \tag{4.6}$$

Die Annahme einer exponentialverteilten Lebenszeit impliziert somit, dass ein bereits benutztes Gerät stochastisch betrachtet *so gut wie neu* ist, wodurch keine Notwendigkeit für dessen Ersatz besteht. Außerdem ist es u.a. für die Schätzung der Zuverlässigkeitsfunktion ausreichend, Daten im Verlauf eines Untersuchungszeitraums während der Funktion der Geräte zu sammeln. Das wirkliche Alter der Geräte ist in diesem Zusammenhang nicht von Bedeutung (vgl. dazu [25]).

4.2.1.2 Berechnung der Ausfallwahrscheinlichkeiten beim SLS

Die Berechnung von Ausfallwahrscheinlichkeiten erfolgt beim SLS in den bereits genannten zwei Fällen. Zum einen verwenden Knoten mit Kontextbewusstsein diese, um ihre eigene Zuverlässigkeit im Hinblick auf den garantierten Recoveryzeitraum t_m zu bestimmen. Aber auch der Koordinator verwendet dieses Verfahren bei DirDissCUA und UnDirDiss, um anhand ihm zur Verfügung stehender globaler Information über das MANET eine globale Ausfallwahrscheinlichkeit zu schätzen. Neben dem vereinbarten garantierten Recoveryzeitraum t_m nimmt er zusätzlich globale Ausfallraten für technische Defekte und Energie an. Die ermittelte globale Ausfallwahrscheinlichkeit wird dann für jedes der sich im Netz befindenden Geräte angenommen.

Der SLS bezieht bei der Bestimmung zwei grundlegende Aspekte mit ein: die Wahrscheinlichkeit für das Auftreten eines technischen Defekts $P_{techFailure}$ sowie die Wahrscheinlichkeit für erschöpfte Energieressourcen $P_{energyFailure}$. Mit beiden berechneten Werten kann dann eine individuelle bzw. globale Gesamtzuverlässigkeit R wie folgt bestimmt werden:

$$R = (1 - P_{techFailure}) \cdot (1 - P_{energyFailure}) \tag{4.7}$$

wobei die Unabhängigkeit der beiden Ausfallwahrscheinlichkeiten $P_{techFailure}$ und $P_{energyFailure}$ angenommen wird. Unter Verwendung der Gesamtzuverlässigkeit kann die Gesamtausfallwahrscheinlichkeit $p_{failure}$ wie folgt berechnet werden:

$$p_{failure} = 1 - R \tag{4.8}$$

Technische Ausfallwahrscheinlichkeit

Für technische Ausfälle wird im Rahmen dieser Arbeit angenommen, dass diese exponentialverteilt auftreten (vgl. u.a. [25, 38]). Bezug nehmend auf die in Abschnitt 4.2.1.1 vorgestellten Charakteristiken der Exponentialverteilung ergibt sich daher für die Ausfallwahrscheinlichkeit eines Gerätes aufgrund eines technischen Defektes:

$$P_{techFailure} = 1 - e^{-\lambda_{tech} t_m} \tag{4.9}$$

wobei λ_{tech} die zugehörige Ausfallrate darstellt und t_m der garantierte Recoveryzeitraum ist.

Zur Ermittlung der technischen Ausfallrate einer Geräteart werden in der Praxis gewöhnlich Untersuchungen durchgeführt, wie sie z.B. in [25, 38] beschrieben werden. Dabei wird für vorher definierte Zeiträume untersucht, wie viele der zu Beginn funktionsfähigen Geräte im Laufe der Betriebszeit ausfallen. Alternativ kann jedoch bei Kenntnis des Erwartungswertes für einen Ausfall das gesuchte λ_{tech} mit Hilfe der Formel 4.5 berechnet werden.

Energiebedingte Ausfallwahrscheinlichkeit

Für Ausfälle aufgrund von erschöpften Energieressourcen wird ebenfalls angenommen, dass diese exponentialverteilt sind. Die Berechnung der zugehörigen Ausfallwahrscheinlichkeit berechnet sich in Analogie zu der technischen Ausfallrate:

$$P_{energyFailure} = 1 - e^{-\lambda_{energy} t_m} \tag{4.10}$$

wobei λ_{energy} der energiebedingten Ausfallrate entspricht und t_m wiederum der garantierte Recoveryzeitraum ist. Das hier erforderliche λ ist beispielsweise auch mit der zuvor genannten Formel 4.5 ermittelbar, wenn ein entsprechender Erwartungswert für einen solchen Ausfall bekannt ist oder angenommen werden kann.

4.2.2 Verteilung in dichten MANETs

In dichten MANETs wird zwischen zwei Verteilungsstrategien unterschieden, die sich an den technischen Fähigkeiten der mobilen Geräte orientieren. Sind Knoten mit Kontextbewusstsein vorhanden, kann eine ausfallsichere Gruppe für die Verteilung ausgewählt werden. Bei Knoten ohne Kontextbewusstsein muss hingegen ein anderer Ansatz gewählt werden.

4.2.2.1 Verteilung bei Knoten mit Kontextbewusstsein (DirDissCA)

Die Idee dieser Strategie basiert darauf, dass die Knoten im MANET ihre eigene Ausfallwahrscheinlichkeit $p_{failure}$ bestimmen können. Sind in der Umgebung des Koordinators genügend Knoten mit geringer Ausfallwahrscheinlichkeit vorhanden, so kann dieses Verfahren verwendet werden. Ziel ist es, möglichst wenige, aber bezüglich der vereinbarten garantierten Recoveryzeit

t_m ausfallsichere Knoten für die Speicherung des Transaktionslogs zu verwenden. Auf diese Art wird sowohl die Netz- wie auch die Speicherlast für das MANET reduziert. Kosten entstehen erst bei der Recovery von ausgefallenen Transaktionsteilnehmern, da diese versuchen müssen, die Knoten mit gespeichertem Transaktionslog zu erreichen. Das Verfahren wird daher auch als *gerichtete Verteilung auf n ausgewählte Knoten für dichte* MANET*s* bezeichnet.

Um zu ermitteln, ob diese Verteilung möglich ist, befragt der Koordinator seine k Nachbarknoten nach ihren Ausfallwahrscheinlichkeiten für den Zeitraum t_m. Sind genügend Knoten mit niedriger Ausfallwahrscheinlichkeit zum Erreichen von R_{SLS} verfügbar, können davon die zuverlässigsten n für die Verteilung ausgewählt werden. Dazu sendet der Koordinator den Transaktionsteilnehmern die notwendigen Informationen über die ausgewählten n Knoten für die Verteilung. Bemerkt der Koordinator später den Ausfall von einem oder mehreren Teilnehmern, schickt er das Transaktionslog zu allen n Knoten. Somit können Teilnehmer diese Knoten nach einem Ausfall direkt nach dem Transaktionslog fragen. Eine erfolgreiche Recovery ist möglich, so lange noch mindestens ein Gerät von den n ausgewählten Knoten im Netz verfügbar ist. Aufgrund der expliziten Auswahl von Knoten für die Speicherung des Transaktionslogs wird diese Strategie auch als gerichtete Verteilung für dichte MANETs bezeichnet.

Berechnung der Anzahl von benötigten Knoten

Für die Berechnung der Anzahl von Knoten, die benötigt werden, um die geforderte Zuverlässigkeit R_{SLS} garantieren zu können, werden die Ausfallwahrscheinlichkeiten der k Nachbarn des Koordinators betrachtet. Ziel ist die Auswahl einer Gruppe von n Knoten mit jeweils möglichst geringer Ausfallwahrscheinlichkeit, so dass insgesamt die geforderte Zuverlässigkeit R_{SLS} erreicht wird. Dabei wird zunächst nicht die Wahrscheinlichkeit dafür berücksichtigt, dass ein Gerät nach einem Ausfall wieder in das Netz zurückkommt (z.B. durch Aufladen des Akkus).

Der Koordinator sortiert zu Beginn die Knoten nach ihren Ausfallwahrscheinlichkeiten. Anschließend beginnt er mit dem Knoten, der die geringste Ausfallwahrscheinlichkeit hat und testet, ob mit diesem bereits das gewünschte R_{SLS} erreicht werden kann. Wenn ja, wird nur dieser Knoten später für die Verteilung verwendet. Anderenfalls wird schrittweise der Knoten mit der nächstgrößeren Ausfallwahrscheinlichkeit hinzugenommen und geprüft, ob mit diesem und den bisherigen Knoten die erforderte Zuverlässigkeit erreicht werden kann. Ob das gewünschte R_{SLS} mit den k Nachbarknoten des Koordinators erreicht werden kann, berechnet sich dabei wie folgt:

$$R_{SLS}(n) = 1 - \prod_{i=1}^{n} p_{failure_i} \tag{4.11}$$

wobei $p_{failure_i}$ die Ausfallwahrscheinlichkeit des Knotens i repräsentiert und $n \leq k$ mit n der Anzahl von notwendigen Knoten für die Speicherung des Transaktionslogs ist und k die Anzahl von Nachbarn des Koordinators.

Vor- und Nachteile

Ein Vorteil dieser Strategie ist, dass kein Fluten des gesamten MANETs notwendig ist. Sowohl bei der Verteilung wie bei der Recovery kann direkt an diese n Knoten eine Nachricht verschickt werden.

Als Nachteil ist jedoch zu bemerken, dass diese Strategie von den Knoten die Fähigkeit zur Berechnung der Ausfallwahrscheinlichkeit fordert. Zudem müssen die n Knoten zum Zeitpunkt der Verteilung für den Koordinator auch erreichbar sein. Außerdem muss mindestens einer der n Knoten im Netz bei der Recovery eines ausgefallenen Teilnehmers erreichbar sein und darf nicht selbst ausgefallen oder in eine andere Partition abgewandert sein. Der Erfolg dieser Strategie hängt somit von der Erreichbarkeit der n Knoten ab.

4.2.2.2 Verteilung bei Knoten ohne Kontextbewusstsein (DirDissCUA)

Verfügen die Knoten des MANETs nicht über Kontextbewusstsein und können somit keine individuelle Ausfallwahrscheinlichkeit $p_{failure}$ bestimmen, kann der Koordinator keine besonders zuverlässigen Geräte auswählen. Er muss daher aufgrund seiner globalen Informationen über die Knoten im Netz eine globale Ausfallwahrscheinlichkeit berechnen. Damit kann er dann die Anzahl n von notwendigen Knoten schätzen, um die gewünschte Zuverlässigkeit R_{SLS} zu garantieren. Bei der Verteilung werden Single-Hop-Broadcasts verwendet, um n Knoten für die Speicherung des Transaktionslogs zu erreichen. Jeder Knoten des MANETs, der das Log speichert, sendet eine Bestätigung zurück. Der Koordinator stoppt die Verteilung dann, wenn er n Bestätigungen erhalten hat. Da keine Knoten explizit ausgewählt werden können, wird diese Strategie auch als *ungerichtete Verteilung auf n anonyme Knoten für dichte* MANET*s* bezeichnet.

Für die Recovery eines ausgefallenen Transaktionsteilnehmers bedeutet diese Strategie, dass er keinen Knoten im Netz mit dem Transaktionslog kennt. Er muss daher das MANET mit seiner Anfrage nach dem Transaktionslog fluten – in diesem Fall ist das ein Broadcast an die direkten Nachbarn. Eine erfolgreiche Recovery ist für einen ausgefallenen Knoten möglich, solange mindestens einer der n Knoten noch im Netz erreichbar ist.

Prinzipiell ist bei DirDissCUA eine Auswahl der n Knoten für die Verteilung wie bei DirDissCA möglich, z.B. könnten alle Nachbarn des Koordinators verwendet werden. DirDissCUA ist somit nur eine *Variante* von DirDissCA. Durch einen Broadcast sollen Nachrichten bei der Verteilung gespart werden (bei DirDissCA sind jeweils n Nachrichten nötig). Ein Unterschied betrifft die Recovery ausgefallener Teilnehmer. Da die Teilnehmer keinen Knoten mit dem Transaktionslog kennen, können keine Knoten direkt befragt werden. Es wird deshalb ein Broadcast an die Nachbarn verwendet.

Berechnung der Anzahl von benötigten Knoten

Bei der Berechnung der Anzahl von notwendigen Knoten wird ähnlich vorgegangen wie in Abschnitt 4.2.2.1. Es wird hier jedoch statt der individuellen Ausfallwahrscheinlichkeiten eine globale Schätzung der Ausfallwahrscheinlichkeit $p_{failure}$ für alle Knoten verwendet. Damit ist die Berechnung der Anzahl n von Knoten, die zur Speicherung des Transaktionslogs benötigt werden, wie folgt möglich:

$$\begin{aligned} R_{SLS} &= 1 - (p_{failure})^n \\ 1 - R_{SLS} &\leq (p_{failure})^n \\ \frac{ln(1 - R_{SLS})}{ln(p_{failure})} &\leq n \end{aligned} \tag{4.12}$$

wobei R_{SLS} die von der Anwendung gewünschte Zuverlässigkeit für das Wiederfinden eines Logs ist.

Vor- und Nachteile

Als Vorteil dieser Strategie ist zu nennen, dass sie unabhängig von der Fähigkeit der Geräte zur Berechnung einer individuellen Ausfallwahrscheinlichkeit ist. Gleichzeitig soll die Festlegung einer Anzahl n von benötigten Knoten verhindern, dass das gesamte MANET das Transaktionslog speichert. Damit soll eine Minimierung der Speicherlast erreicht werden. Durch die Verwendung von nur einem Broadcast bei der Verteilung sollen zudem Nachrichten gespart werden (d.h. es sind keine n Nachrichten notwendig).

Als Nachteil dieser Strategie kann sich der Broadcast erweisen. Trotz Bestimmung einer festen Knotenanzahl werden mit einem Broadcast ggf. nicht genug Knoten erreicht. Daher müssen diese Broadcasts so lange wiederholt werden, bis n Bestätigungen beim Koordinator eingetroffen sind. In dünneren Netzen kann dadurch eine erhebliche Nachrichtenlast auftreten, wenn der Koordinator versucht, die gewünschte Anzahl von Knoten zu erreichen.

Auch für die Transaktionsteilnehmer ist bei einer Recovery ein Broadcast an die direkten Nachbarn notwendig. Sie kennen keinen Knoten mit dem Transaktionslog. Wird das Transaktionslog nur auf wenige Knoten verteilt, kann auch hier eine erhöhte Nachrichtenanzahl durch wiederholte Broadcast notwendig sein, bis eine erfolgreiche Recovery möglich ist.

4.2.3 Verteilung in dünnen MANETs (UnDirDiss)

Eine Hauptproblematik in dünnen MANETs ist, dass nicht garantiert werden kann, ob ein Knoten zu einem bestimmten Zeitpunkt oder ausreichend viele Knoten mit geringer Ausfallwahrscheinlichkeit gerade erreichbar sind. Die Ansätze aus Abschnitt 4.2.2 bieten sich daher nicht an.

Da die Annahme ist, dass nur wenige Geräte für die Verteilung zur Verfügung stehen, werden die technische Ausstattung der Geräte und somit individuelle Ausfallwahrscheinlichkeiten nicht berücksichtigt. Es wird jedoch angenommen, dass der Koordinator die Gesamtanzahl von Knoten im MANET schätzen kann.

Bei dieser Strategie wird ein Prozentsatz des Netzes zur Speicherung bestimmt. Dieser Prozentsatz wird als sog. *Verteilungsgrad* d bezeichnet. Um diesen gewünschten Verteilungsgrad im MANET zu erzielen, flutet der Koordinator das Netz einmal sowohl mit dem Transaktionslog wie mit dem Verteilungsgrad. Jeder Knoten, der diese Nachricht erhält, entscheidet zufällig anhand von d, ob er das Log speichert.

Bei der Recovery muss ein ausgefallener Knoten mittels Broadcast seine k Nachbarn befragen. Die Garantie für das erfolgreiche Auffinden des Transaktionslogs ist in diesem Fall nur abhängig von diesen k Knoten.

Berechnung des benötigten Verteilungsgrades

Für die Bestimmung des Verteilungsgrades d sind als Grundlage die in Abschnitt 4.2.2.2 vorgestellten Berechnungen nötig. Der Koordinator verwendet hier wiederum eine globale Ausfallwahrscheinlichkeit $p_{failure}$ und berechnet zunächst für die gewünschte Zuverlässigkeit R_{SLS} die Anzahl n von notwendigen Knoten für die Speicherung. Zusätzlich muss er die Gesamtanzahl r von Knoten im MANET insgesamt schätzen. Mit diesen beiden Parametern kann der gewünschte Verteilungsgrad d wie folgt berechnet werden:

$$d = \frac{n}{r} \tag{4.13}$$

wobei die erreichte Zuverlässigkeit von n abhängt und der Verteilungsgrad durch Erhöhung dieser Anzahl an das gewünschte R_{SLS} angepasst werden kann (vgl. Formel 4.12).

Vor- und Nachteile

Ein Vorteil dieser Strategie ist die Unabhängigkeit davon, ob Geräte ihre individuelle Ausfallwahrscheinlichkeit bestimmen können. Die Festlegung eines Verteilungsgrades soll außerdem die Speicherkosten minimieren, da nicht jeder Knoten das Transaktionslog speichert. Durch das zufällige Speichern anhand des Verteilungsgrades hängt die Recovery für ausgefallene Geräte lediglich von ihrer direkten Umgebung ab. Außerdem ist nur ein Broadcast des Koordinators notwendig. Dessen Aufwand wird somit reduziert und auf das gesamte Netz verteilt.

Als Nachteil ist hier wiederum das Fluten zu nennen. Trotz Beschränkung der für das Speichern notwendigen Anzahl von Knoten erhält doch zunächst jedes erreichbare Gerät diese Nachricht.

4.3 Prozessabläufe für die Simulation des SLS

Zur Evaluation des SLS werden Prozessabläufe für die Simulation verwendet, die auf dem in Abschnitt 2.4.4 vorgestellten Transaktionsmodell basieren. Aufgrund der unterschiedlichen Funktionalitäten von Koordinator und Teilnehmer sind für beide Seiten Prozessabläufe notwendig. Wichtig ist dabei die Umsetzung des zuvor beschriebenen Zuverlässigkeitsmodells. Außerdem muss der Ablauf der Prozesse von Koordinator und Teilnehmer aufeinander abgestimmt sein, um einen reibungslosen Ablauf von Transaktionen zu ermöglichen.

Um die Bewertung des SLS zu ermöglichen, wird zusätzlich ein einfacher Prozessablauf verwendet, der ebenfalls auf dem in Abschnitt 2.4.4 eingeführten Transaktionsmodell beruht.

Im Folgenden werden zunächst die Prozessabläufe für Koordinator und Teilnehmer bei Verwendung des SLS vorgestellt und daran anschließend der für den Vergleich entwickelte Prozessablauf.

4.3.1 Umsetzung des Zuverlässigkeitsmodells

Wesentlich für die Umsetzung des Transaktionsmodells unter Verwendung des in Abschnitt 4.2 beschriebenen Zuverlässigkeitsmodells sind Timeouts (diese werden im Folgenden als auch *Timer* bezeichnet) und Zustände. Beide ermöglichen den Ablauf des entwickelten Transaktionsablaufs. Durch Timeouts wird gesteuert, wie lange auf Nachrichten gewartet werden soll, so dass Verklemmungen verhindert werden können. Zusätzlich ermöglichen die Zustände, dass Nachrichten nur in vordefinierten Zuständen für einen vordefinierten Zeitraum erwartet und bearbeitet werden.

4.3.2 Prozessablauf des Koordinators

Der Prozessablauf einer Transaktion bei einem Koordinator kann in zwei verschiedene Abschnitte unterteilt werden. Am Anfang der Transaktion steht die bereits vorgestellte Abstimmungsphase. Daran schließt sich die eigentliche Commit-Phase an, die selbst in vier Phasen unterteilt werden kann. Ein entsprechendes Zustandsdiagramm dazu ist in Abbildung 4.1 dargestellt. Im Folgenden werden die Abläufe der einzelnen Phasen detailliert dargestellt, da deren Unsicherheitsfenster im Rahmen dieser Arbeit schwerpunktmäßig untersucht werden soll. Es wird vorausgesetzt, dass zum einen der Koordinator bereits alle Transaktionsteilnehmer kennt und diese wiederum den Koordinator kennen. Außerdem wird davon ausgegangen, dass zwischen den teilnehmenden Knoten der Beginn der Commit-Phase vereinbart wurde (vgl. dazu Abschnitt 2.4.4).

4.3.2.1 Starten der Transaktion

Der Start einer Transaktion bis zum Ende der Abstimmungsphase wird beim Prozessablauf des SLS-Koordinators als Processing-Phase bezeichnet. Da das Unsicherheitsfenster der Commit-Phase untersucht werden soll, werden vorherige Aktionen im folgenden Zustand realisiert:

Zustand PROCESSING_PHASE: Dieser Zustand umfasst alle Aktionen der Transaktionen bis zum Ende der Commit- und dem Beginn der Commit-Phase. Einzeloperationen dieser Phase sind für die Untersuchungen dieser Arbeit nicht relevant. Die Aufenthaltsdauer in diesem Zustand steuert der ProcessingTimer (Kurzform für *Processing Phase Timer*). Durch ihn wird eine Zustandsänderung ausgelöst:

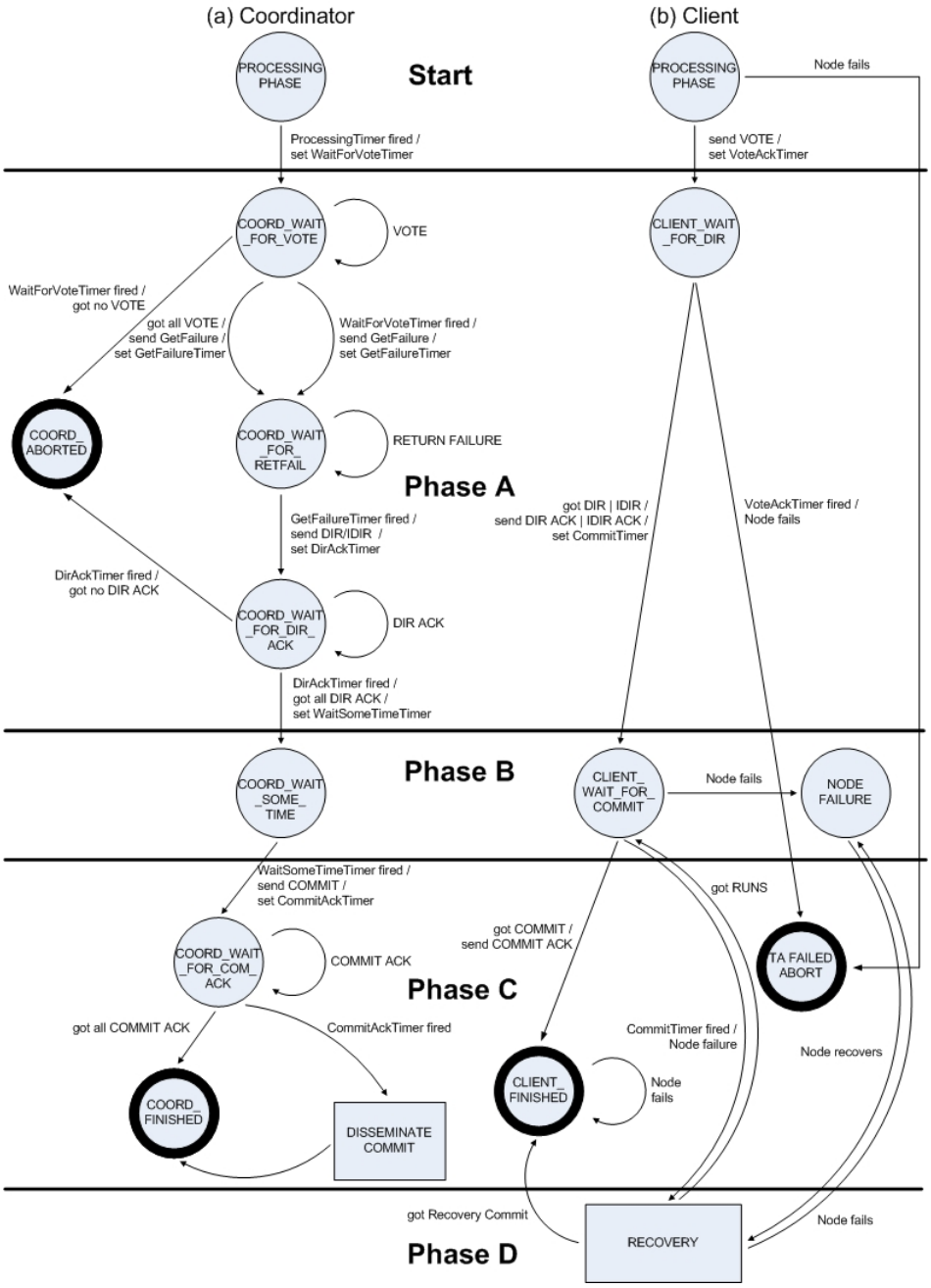

Abbildung 4.1: Zustandsdiagramm für die Prozessabläufe des Koordinators und des Klienten beim SLS

1. **Feuern des ProcessingTimers:** Der Koordinator wechselt in den Zustand COORD_WAIT_FOR_VOTE (Kurzform für *Coordinator Wait For Vote*) und wartet auf die Voten der Transaktionsteilnehmer. Gleichzeitig wird damit in Teil A der Commit-Phase übergegangen.

4.3.2.2 Commit-Phase

Die Commit-Phase beim Prozessablauf des SLS kann in vier Teilphasen untergliedert werden. Phase A umfasst den Beginn der Commit-Phase. In Phase B ermittelt der Koordinator die Transaktionsentscheidung, die den Teilnehmern in Phase C mitgeteilt wird. Der Koordinator kann in Phase C dann entweder die Transaktion abschließen oder muss beim Auftreten von Ausfällen den Transaktionsausgang im MANET verteilen. Zuletzt behandelt Phase D, wie der Koordinator bei Recovery-Anfragen von Teilnehmern reagiert.

4.3.2.2.1 Phase A: Votum der Teilnehmer Diese Phase umfasst aufseiten des Koordinators das Sammeln der Voten von den Teilnehmern, die Auswahl einer Verteilungsstrategie für den Fall von Fehlern und zuletzt die Benachrichtigung der Teilnehmer über die gewählte Strategie. Dazu werden die im Folgenden beschriebenen drei Zustände abgearbeitet. Voraussetzung ist, dass mindestens ein Transaktionsteilnehmer erreichbar ist. Anderenfalls bricht der Koordinator die Transaktion ab.

Zustand COORD_WAIT_FOR_VOTE: Der Koordinator wartet auf die Vote-Nachrichten (vgl. zu den Begriffen Abschnitt 2.4.4) der Transaktionsteilnehmer. Die maximale Aufenthaltsdauer in diesem Zustand wird durch den WaitForVoteTimer gesteuert. Ein Zustandswechsel kann durch folgende Ereignisse erfolgen:

1. **Empfang aller Vote-Nachrichten von den Teilnehmern:** Der Koordinator wechselt in den Zustand COORD_WAIT_FOR_RETFAIL (Kurzform für *Coordinator Wait For Return Failure Probability*). Gleichzeitig sendet er mittels Single-Hop-Broadcast eine GET FAILURE-Nachricht (Kurzform für *Get Failure Probability*) an seine Nachbarn und startet den dazu gehörigen GetFailureTimer (Kurzform für *Get Failure Probability Timer*).

2. **Feuern des WaitForVoteTimers**

   a) **Empfang mindestens einer Vote-Nachricht:** Hier verhält sich der Koordinator analog zu 1.), allerdings werden im weiteren Verlauf der Transaktion nur die Teilnehmer berücksichtigt, von denen ein Votum eingetroffen ist.

   b) **Ausbleiben aller Vote-Nachrichten:** Da kein Votum eingetroffen ist, nimmt der Koordinator die Transaktion als gescheitert an und geht in den Zustand COORD_ABORTED (Kurzform für *Coordinator Aborted*).

Zustand COORD_WAIT_FOR_RETFAIL: Während der Koordinator sich in diesem Zustand befindet, empfängt und speichert er die Ausfallwahrscheinlichkeiten seiner Nachbarn in Form von RETURN_FAILURE-Nachrichten (Kurzform für *Return Failure Probability).* Die Aufenthaltsdauer des Koordinators bis zum Zustandswechsel wird durch den GetFailureTimer definiert:

1. **Feuern des GetFailureTimers:** Der Koordinator berechnet anhand der von seinen Nachbarn gesendeten Ausfallwahrscheinlichkeiten, welche Verteilungsstrategie beim Auftritt von Knotenausfällen verwendet werden soll. Dabei berücksichtigt er, ob er sich in einem dichten

oder dünnen MANET befindet (Diese Information wird im Rahmen dieser Arbeit als gegeben angenommen, vgl. dazu Abschnitt 4.1.1). Ist dies entschieden, wechselt der Koordinator in den Zustand COORD_WAIT_FOR_DIR_ACK (Kurzform für *Coordinator Wait For Direction Acknowledgement*). Außerdem wird den Teilnehmern die Entscheidung mitgeteilt. Soll die gerichtete Verteilung bei Ausfällen verwendet werden, sendet der Koordinator eine DIR-Nachricht (Kurzform für *Direct Dissemination*), ansonsten eine IDIR-Nachricht (Kurzform für *Indirect* bzw. *Epidemic Dissemination*).

Zustand COORD_WAIT_FOR_DIR_ACK: Während dieses Zustands empfängt der Koordinator die Bestätigungen über den Erhalt der Verteilungsstrategien – also DIR_ACK- oder IDIR_ACK-Nachrichten (Kurzformen für *Direct Dissemination Acknowledgement* und *Indirect* bzw. *Epidemic Dissemination Acknowledgement*) der Teilnehmer. Die maximale Aufenthaltsdauer ist durch den DirAckTimer beschränkt (Kurzform für *Direction Acknowledgement Timer*). Es gibt folgende Möglichkeiten für den weiteren Ablauf der Transaktion:

1. **Feuern des DirAckTimers**

   a) **Empfang von mindestens einer DIR_ACK bzw. IDIR_ACK-Nachricht:** Der Koordinator kann seinen Prozessablauf fortsetzen. Er wechselt in den Zustand COORD_WAIT_SOME_TIME (Kurzform für *Coordinator Wait Some Time*) und setzt den Prozessablauf fort. Damit erreicht er Phase B.

   b) **Ausbleiben aller DIR_ACK bzw. IDIR_ACK-Nachrichten:** In diesem Fall geht der Koordinator in den Zustand COORD_ABORTED über und bricht die Transaktion ab, da er alle Teilnehmer als ausgefallen annimmt.

4.3.2.2.2 Phase B: Berechnung der Commit-Entscheidung Phase B besteht beim Koordinator nur aus dem Zustand COORD_WAIT_SOME_TIME, der die Entscheidungsfindung zum globalen Commit oder Abort simulieren soll.

Zustand COORD_WAIT_SOME_TIME: Dieser Zustand simuliert die Berechnung des Transaktionsausgangs. Die Aufenthaltsdauer wird durch den WaitSomeTimeTimer (Kurzform für *Coordinator Wait Some Time Timer*) bestimmt. Eine Zustandsänderung ist nur bei folgendem Ereignis möglich:

1. **Feuern des WaitSomeTimeTimers:** Der Koordinator hat eine Commit-Entscheidung[1] getroffen. Er wechselt dann seinen Zustand in COORD_WAIT_FOR_COM_ACK (Kurzform für *Coordinator Wait For Commit Acknowledgement*). Gleichzeitig sendet er eine COMMIT-Nachricht an alle Transaktionsteilnehmer. Anschließend befindet sich der Koordinator dann in Phase C.

4.3.2.2.3 Phase C: Transaktionsabschluss bzw. Verteilung Dieser Teil der Commit-Phase beinhaltet den erfolgreichen Abschluss der Transaktion bzw. die Verteilung des Transaktionslogs beim Auftritt von Fehlerfällen. Zunächst wartet der Koordinator auf die Commit-Bestätigungen

[1] Anmerkung: Im Rahmen dieser Arbeit wird immer ein COMMIT geschickt. Sollte in einer realen Anwendung die Entscheidung auf ein ABORT gefallen sein, so könnte das Verfahren ebenso verwendet werden. Die hier verwendete Vereinfachung beeinflusst nicht die Untersuchungsergebnisse, da hier ermittelt werden soll, ob Ausfälle im Unsicherheitsfenster geheilt werden können. Welche Art von Nachricht (COMMIT oder ABORT) durch einen Ausfall verloren gegangen ist, ist dabei nicht relevant. Wichtig ist, dass die Nachricht wieder gefunden werden kann.

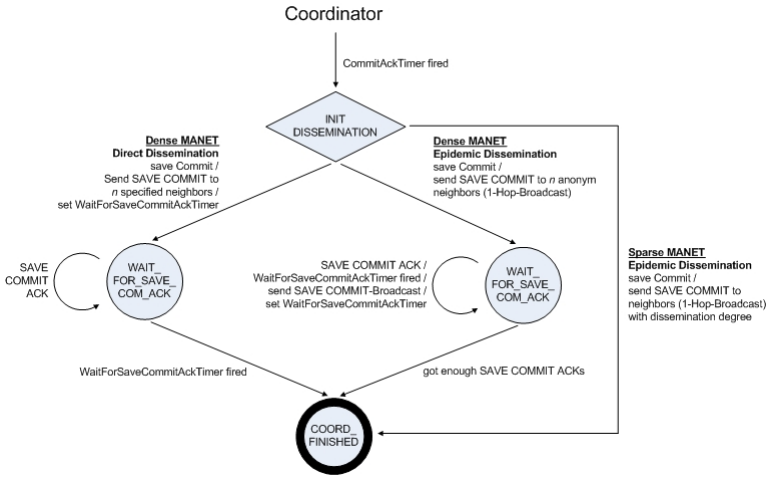

Abbildung 4.2: Zustandsdiagramm für die Verteilung eines Transaktionslogs durch den Koordinator

der Teilnehmer. Treffen alle Nachrichten ein, kann die Transaktion erfolgreich abgeschlossen werden. Anderenfalls ist die Verteilung des Transaktionslogs notwendig. Für den Koordinator wird somit die Transaktion in Phase C in jedem Fall beendet.

Zustand COORD_WAIT_FOR_COM_ACK: In diesem Zustand wartet der Koordinator auf die COMMIT_ACK-Nachrichten (Kurzform *Commit Acknowledgement*) der Teilnehmer. Der CommitAckTimer (Kurzform für *Commit Acknowledgement Timer*) bestimmt die maximale Wartezeit auf Nachrichten. Ein Zustandswechsel ist bei folgenden Begebenheiten möglich:

1. **Empfang aller COMMIT_ACK-Nachrichten:** Die Transaktion kann für den Koordinator erfolgreich beendet werden und er wechselt in den Zustand COORD_FINISHED (Kurzform für *Coordinator Finished).*

2. **Feuern des CommitAckTimers:** Einige oder alle COMMIT_ACK-Nachrichten der Teilnehmer sind ausgeblieben. Der Koordinator muss davon ausgehen, dass diese Teilnehmer ausgefallen sind. Um ihnen dennoch den erfolgreichen Abschluss der Transaktion zu ermöglichen, wird der Transaktionsausgang im MANET verteilt. Dieses Vorgehen wird im Folgenden für jede Strategie des SLS detailliert dargestellt.
   Bei der Initialisierung der Verteilung (vgl. Zustand INIT DISSEMINATION in Abbildung 4.2) wird die Strategie verwendet, die der Koordinator in Phase A ausgewählt hat. Die einzelnen Strategien werden einführend in Abschnitt 4.1.1 sowie formal in den Abschnitten 4.2.2 und 4.2.3 vorgestellt.

⇒ Verteilung in dichten MANETs bei Knoten mit Kontextbewusstsein (DirDissCA)

Der Koordinator hat in Phase A n Nachbarknoten mit geringer Ausfallwahrscheinlichkeit für die Speicherung des Transaktionslogs beim Auftritt von Knotenausfällen ausgewählt (vgl. dazu Abbildung 4.2 und Abschnitt 4.2.2.1). Er sendet an die n Knoten den Ausgang der Transaktion als SAVE COMMIT-Nachricht und wechselt in den im Folgenden beschriebenen Zustand COORD_WAIT_FOR_DISS_COM_ACK (Kurzform für *Coordinator Wait For Save Disseminated Commit Acknowledgement Timer*). Zusätzlich speichert er diese Information selbst.

Zustand COORD_WAIT_FOR_DISS_COM_ACK: In diesem Zustand wartet der Koordinator auf die Speicherbestätigungen der n Knoten, an die er das Transaktionslog geschickt hat. Die Wartezeit auf die SAVE COMMIT-Nachrichten wird durch den WaitForSaveCommitAckTimer (Kurzform für *Wait For Save Commit Acknowledgement*) definiert:

1. **Feuern des WaitForSaveCommitAckTimers:** Der Koordinator geht in den Zustand COORD_FINISHED und die Transaktion ist für ihn beendet. Sollten alle oder einige der Speicherbestätigungen ausbleiben, so verhält sich der Koordinator genauso, d.h. er verteilt den Transaktionsausgang kein zweites Mal. Evtl. sind einige oder alle der n Knoten bereits ausgefallen, so dass ein weiteres Verschicken nur die Nachrichtenlast erhöhen, aber zu keiner Erhöhung der speichernden Knotenanzahl führen würde. Die Annahme ist hier, dass in einem dichten MANET ein Knoten mit geringer Ausfallwahrscheinlichkeit sofort erreichbar ist, da er sich noch im Netz befindet.

⇒ Verteilung in dichten MANETs bei Knoten ohne Kontextbewusstsein (DirDissCUA)

Hat der Koordinator in Phase A nur Knoten ohne Kontextbewusstsein im Netz vorgefunden, so verteilt er bei Ausfällen auf n anonyme Knoten (vgl. Abbildung 4.2 und Abschnitte 4.2.2.2 und 4.2.3). Um eine Verteilung auf n Knoten zu erreichen, sendet er die SAVE COMMIT-Nachricht mittels Single-Hop-Broadcast an seine Nachbarn. Zusätzlich speichert er diese Information selbst und wechselt wie bei DirDissCA in den Zustand COORD_WAIT_FOR_DISS_COM_ACK, allerdings mit verändertem Verhalten im Vergleich zur vorigen Netzkonfiguration.

Zustand COORD_WAIT_FOR_DISS_COM_ACK: Der Koordinator wartet auf SAVE COMMIT-Nachrichten von seinen Nachbarknoten. Wenn genügend Knoten diese Nachricht geschickt haben, kann der Koordinator in den Zustand COORD_FINISHED gehen. Sollte jedoch der WaitForSaveCommitAckTimer feuern, so wird der Timer erneut gesetzt und das Transaktionslog durch einen Broadcast noch einmal verschickt. Dies wird so lange wiederholt, bis ausreichend viele Knoten das Transaktionslog gespeichert haben. Der Koordinator verbleibt im Zustand COORD_WAIT_FOR_DISS_COM_ACK, bis er n SAVE COMMIT-Nachrichten erhalten hat.

⇒ Verteilung in dünnen MANETs (UnDirDiss)

In einem dünnen MANET wird ebenfalls ein Broadcast verwendet. Zusätzlich zum Transaktionslog wird der benötigte Verteilungsgrad mit verschickt. Die Knoten im Netz, die die Nachricht erhalten, entscheiden dann zufällig anhand des Verteilungsgrades (vgl. Abbildungen 4.3 und 4.2.b sowie Abschnitt 4.2.3 für eine detaillierte Beschreibung der Strategie), ob sie die Nachricht speichern. Haben sie die Nachricht zuvor noch nicht gesehen, so schicken sie diese anschließend mittels Broadcast weiter.

4.3.2.2.4 Phase D: Recovery In der letzten Phase D hat der Koordinator lediglich eine passive Rolle. Er beantwortet nur REC_TA-Anfragen (Kurzform für *Recovery Transaction Request*) von Teilnehmern. In folgenden Situationen antwortet der Koordinator auf Recovery-Anfragen:

1. **Empfang einer REC_TA-Anfrage im Zustand COORD_WAIT_SOME_TIME** (vgl. Abbildung 4.4.b.1): Der Teilnehmer war nur kurzzeitig ausgefallen, so dass der Koordinator noch die Transaktionsentscheidung berechnet. Er antwortet mit einer RUNS-Nachricht (Kurzform für *Computation Is Running*). Dadurch weiß der recovernde Teilnehmer, dass noch keine Entscheidung über den Transaktionsausgang getroffen worden ist.

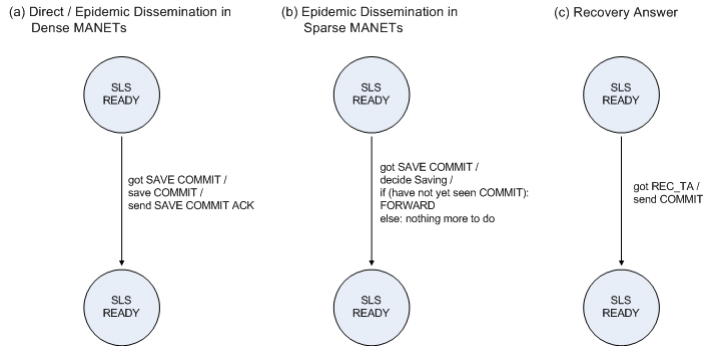

Abbildung 4.3: Zustandsdiagramm für die nicht direkt an einer Transaktion beteiligten Knoten

2. **Empfang einer REC_TA-Anfrage im Zustand COORD_WAIT_FOR_COM_ACK** (vgl. Abbildung 4.4.b.2): Der Koordinator sendet erneut ein COMMIT an den anfragenden Teilnehmer. An seinem weiteren Verhalten ändert die Anfrage nichts.

3. **Empfang einer REC_TA-Anfrage im Zustand COORD_FINISHED** (vgl. Abbildung 4.4.b.3): Auch in diesem Fall antwortet der Koordinator dem anfragenden Teilnehmer mit einem COMMIT. Wie bei 2.) wird dadurch keine Zustandsänderung beim Koordinator ausgelöst.

4.3.3 Prozessablauf des Transaktionsteilnehmers

In Analogie zum Prozessablauf des Koordinators kann auch der des Klienten in Abstimmungs- und Commit-Phase unterteilt werden. Ebenso wie beim Koordinator wird auch beim Klienten die Commit-Phase in vier Phasen unterteilt. Der Prozessablauf eines Teilnehmers ist ebenfalls in Abbildung 4.1 zu finden.

4.3.3.1 Starten der Transaktion

Bei einem Teilnehmer wird wie beim Koordinator der Start einer Transaktion bis zum Ende der Abstimmungsphase im Rahmen des Prozessablaufs als Processing-Phase bezeichnet. Auch hier ist die Commit-Phase in die bereits vorgestellten Phasen A bis D unterteilt.

Zustand PROCESSING_PHASE: Mit diesem Zustand wird der Transaktionsstart bis zum Beginn der Commit-Phase abgedeckt. Die Aufenthaltsdauer wird wiederum durch den Processing-Timer (Kurzform für *Processing Phase Timer*) gesteuert. Der Timer muss hier identisch zu dem des Koordinators sein, damit alle an der Transaktion beteiligten Knoten möglichst gleichzeitig die Commit-Phase betreten[2]. Ein Zustandswechsel tritt bei folgendem Ereignissen ein:

1. **Feuern des ProcessingTimers:** Der Transaktionsteilnehmer sendet seine Vote-Nachricht an den Koordinator. Anschließend wechselt er in den Zustand CLIENT_WAIT_FOR_DIR (Kurzform für *Client wait for Direction*, entspricht der Empfangsbestätigung des Votums).

[2] Anmerkung: Im Rahmen dieser Arbeit wird angenommen, dass die Entscheidung zum Eintrittszeitpunkt in die Commit-Phase durch die an der Transaktion beteiligten Knoten in der Wahlphase getroffen wurde.

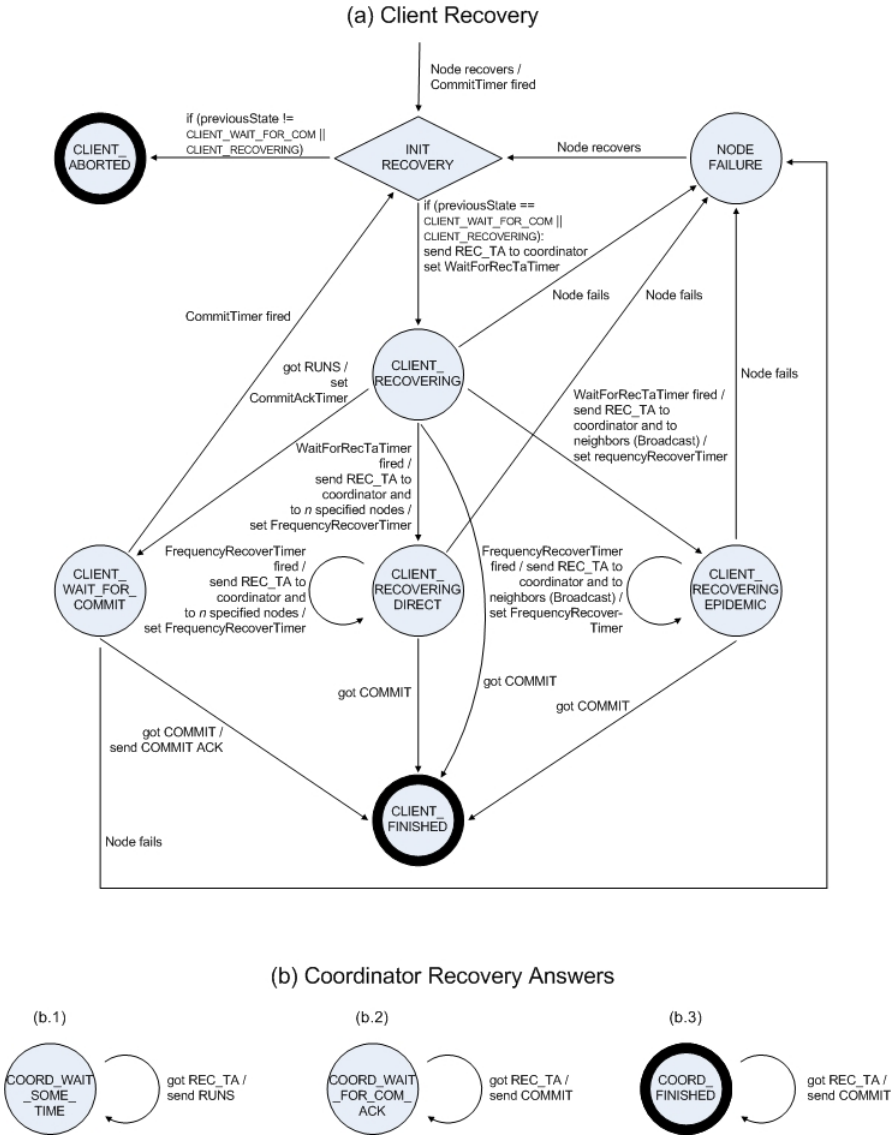

Abbildung 4.4: (a) Prozessablauf bei der Recovery eines Klienten und (b) Beantwortung von Recovery-Anfragen durch den Koordinator

2. **Ausfall des Knotens:** Der Teilnehmer bricht die Transaktion ab und geht in den Zustand CLIENT_ABORTED in Phase C über. Der Grund dafür ist, dass ohne Kenntnis über die durch den Koordinator verwendete Verteilungsstrategie keine entsprechende Recovery gestartet werden kann.

4.3.3.2 Commit-Phase

In Analogie zum in Abschnitt 4.3.2 beschriebenen Prozessablauf des Koordinators unterteilt sich auch der des Teilnehmers in die im Folgenden beschriebenen vier Phasen.

4.3.3.2.1 Phase A: Votum der Teilnehmer Phase A besteht auf der Seite des Klienten ausschließlich aus dem Warten auf die Empfangsbestätigung des gesendeten Votums. Dies impliziert auch die Information über die Verteilungsstrategie, die der Koordinator bei Knotenausfällen verwendet. Die Teilnehmer befinden sich dazu in dem im Folgenden beschriebenen Zustand.

Zustand CLIENT_WAIT_FOR_DIR: Der Teilnehmer wartet auf die Bestätigung der Vote-Nachricht vom Koordinator in Form einer DIR- oder IDIR-Nachricht. Dieser Zeitraum wird durch den WaitForDirTimer (Kurzform für *Wait For Direction Timer*, entspricht der Bestätigung der Vote-Nachricht mit der Information über die gewählte Verteilungsstrategie) definiert. Der Timer muss abgestimmt sein auf die notwendigen Operationen des Koordinators zur Bestimmung einer Verteilungsstrategie. Das Verlassen dieses Zustandes ist aufgrund der folgenden beiden Ereignisse möglich:

1. **Empfang der Bestätigung als DIR- oder IDIR-Nachricht vom Koordinator:** Der Teilnehmer setzt seinen CommitTimer (Kurzform für *Wait For Commit Timer*) und geht in den Zustand CLIENT_WAIT_FOR_COM (Kurzform für *Client Wait For Commit*) über. Damit hat der Teilnehmer Phase B erreicht.

2. **Feuern des WaitForDirTimers:** in diesem Fall ist die Bestätigung der Vote-Nachricht durch den Koordinator ausgeblieben. Der Knoten bricht die Transaktion ab und geht in den Zustand CLIENT_ABORTED über, mit dem Phase D erreicht wird. Die Begründung für diese Modellierung ist, dass ein Teilnehmer ohne Kenntnis über die verwendete Verteilungsstrategie keine adäquate Recovery starten kann.

3. **Ausfall des Knotens:** Der Teilnehmer bricht die Transaktion ab und geht in den Zustand CLIENT_ABORTED. Auch in diesem Fall verfügt er nicht über die notwendigen Informationen, um eine Recovery starten zu können.

4.3.3.2.2 Phase B: Berechnung der Commit-Entscheidung Während dieser Phase befindet sich der Klient in den beiden im folgenden beschriebenen Zuständen.

Zustand CLIENT_WAIT_FOR_COM: Der Teilnehmer wartet auf die COMMIT-Nachricht des Koordinators. Die maximale Aufenthaltsdauer in diesem Zustand ist durch den CommitAck-Timer (Kurzform für *Client Wait For Commit Timer*) bestimmt. Ein Verlassen dieses Zustandes beim Eintritt folgender Ereignisse möglich:

1. **Empfang der COMMIT-Nachricht vom Koordinator:** Der Teilnehmer sendet als Bestätigung eine COMMIT ACK-Nachricht und geht in den Zustand CLIENT_FINISHED über. Damit erreicht er Phase C.

2. **Feuern des CommitTimers:** Es ist kein COMMIT vom Koordinator beim Teilnehmer eingetroffen. Dieser muss nun eine Recovery starten und geht in den virtuellen Zustand INIT_RECOVERY über, wo er die der Verteilung des Koordinators angemessen Recovery initiiert. Damit betritt er direkt Phase D.

3. **Ausfall des Knotens:** Durch einen Ausfall wechselt der Teilnehmer in den Zustand NODE_FAILURE. Er verbleibt somit bis zum Ende des Ausfalls in Phase B.

Zustand NODE_FAILURE: Der Knoten ist ausgefallen. Kommt er wieder zurück ins Netz, so betritt er direkt Phase D und startet eine Recovery.

4.3.3.2.3 Phase C: Transaktionsabschluss Diese Phase umfasst den erfolgreichen Transaktionsabschluss eines Teilnehmers mit dem Zustand CLIENT_FINISHED und den nicht erfolgreichen Abschluss, der aus dem Zustand CLIENT_ABORTED resultiert. Beide werden im Folgenden näher erläutert.

Zustand CLIENT_FINISHED: Der Teilnehmer konnte die Transaktion erfolgreich abschließen – entweder durch ein COMMIT gemäß dem fehlerfreien Transaktionsablauf oder durch eine erfolgreiche Recovery. Sollte es nun zu einem Knotenausfall kommen, so ist das für den Teilnehmer nicht mehr relevant und ändert nichts an seinem Zustand.

Zustand CLIENT_ABORTED: Hat der Teilnehmer keine Bestätigung seines Votums vom Koordinator erhalten oder ist während der Processing-Phase ausgefallen, so bricht er die Transaktion ab. Auch hier ändert ein Ausfall nichts mehr an seinem Zustand.

4.3.3.2.4 Phase D: Recovery Die letzte Phase des Prozessablaufs des Teilnehmers umfasst ausschließlich die Recovery, die entweder nach einem Knotenausfall gestartet wird oder nach dem Feuern des CommitTimers. Zunächst wird bei der Initiierung (INIT_RECOVERY ist ein rein modellierter Zustand, der bei der Implementation nicht umgesetzt wird) überprüft, in welchem Zustand sich der Knoten vor der Recovery befand.

Virtueller Zustand INIT_RECOVERY: Hat sich der Teilnehmer entweder im Zustand CLIENT_WAIT_COM_ACK oder im Zustand NODE_FAILURE befunden, so kann die Recovery initiiert werden. Anderenfalls geht der Teilnehmer in den Zustand CLIENT_ABORTED über und bricht die Transaktion ab (bzw. verbleibt im Zustand CLIENT_FINISHED bei bereits beendeter Transaktion). Der Grund für diese Modellierung ist – wie bereits oben genannt –, dass der Teilnehmer in diesen Zuständen nicht über die notwendigen Informationen verfügt, um eine Recovery gemäß der vom Koordinator verwendeten Verteilung zu starten. Wenn nach der Initiierung eine Recovery gestartet wird, sendet der Teilnehmer in einem ersten Schritt nur an den Koordinator eine REC_TA-Nachricht und geht in den Zustand CLIENT_RECOVERING über.

Zustand CLIENT_RECOVERING: Der Teilnehmer wartet auf eine Recovery-Antwort vom Koordinator. Die maximale Verweilzeit in diesem Zustand definiert der WaitForRecTaTimer (Kurzform für *Wait For Recovery Transaction Answer Timer*). Ein Verlassen dieses Zustandes ist durch folgende Ereignisse möglich:

1. **Empfang der COMMIT-Nachricht vom Koordinator:** Die Recovery – und damit die Transaktion – ist für den Teilnehmer dann mit Erreichen des Zustandes CLIENT_FINISHED erfolgreich abgeschlossen.

2. **Empfang einer RUNS-Nachricht vom Koordinator:** Der Knoten war nur kurz ausgefallen. Die Recovery startet zu einem Zeitpunkt, an dem der Koordinator noch keine Entscheidung getroffen hat. In diesem Fall wird der Teilnehmer mittels RUNS darüber informiert und kann wieder in den Zustand CLIENT_WAIT_FOR_COM wechseln, den CommitTimer starten und auf das reguläre COMMIT warten.

3. **Ausfall des Knotens:** Erleidet der Teilnehmer im Zustand CLIENT_RECOVERING einen Ausfall, geht er in den Zustand NODE_FAILURE über. Bei seiner Rückkehr nach dem Ausfall wird die Recovery erneut initiiert.

4. **Feuern des WaitForRecTaTimers:** Es ist keine Nachricht vom Koordinator eingetroffen. Der Teilnehmer muss mit der Recovery fortfahren, die der vom Koordinator in Phase A ausgewählten Verteilungsstrategie entspricht. Bei der Recoverystrategie wird hier lediglich zwischen gerichtetem und ungerichtetem Verfahren unterschieden.

   a) **Gerichtete Recovery (vgl. Verteilung DirDissCA in Abschnitt 4.3.2):** In einem dichten MANET bei Knoten mit Kontextbewusstsein sendet der Koordinator das Transaktionslog an n ausgewählte Knoten (vgl. Verteilung DirDissCA in Abschnitt 4.3.2). Der Teilnehmer sendet nun bei der Recovery entsprechend an diese n Knoten direkt REC_TA-Nachrichten sowie eine RECOVER REQUEST-Nachricht an den Koordinator. Anschließend wechselt er in den Zustand CLIENT_RECOVERING_DIRECT (Kurzform für *Client Recovering Direct*).

   b) **Ungerichtete Recovery (vgl. Verteilung DirDissCUA und UnDirDiss in Abschnitt 4.3.2):** In einem dichten MANET bei Knoten ohne Kontextbewusstsein sowie in einem dünnen MANET wird das Transaktionslog im Netz auf Knoten verteilt, die die Teilnehmer nicht kennen. Dementsprechend startet ein ausgefallener Teilnehmer eine ungerichtete Recovery. Dazu sendet er zusätzlich zur RECOVER REQUEST-Nachricht an den Koordinator einen Single-Hop-Broadcast mit der REC_TA-Nachricht an seine Nachbarn. Danach geht er in den Zustand CLIENT_RECOVERING_EPIDEMIC (Kurzform für *Client Recovering Epidemic*) über.

Zustand CLIENT_RECOVERING_DIRECT: Der Teilnehmer wartet auf eine COMMIT-Nachricht vom Koordinator und den n für die Verteilung ausgewählten Knoten. Die maximale Wartezeit auf den Transaktionsausgang nach einem Versuch wird durch den FrequencyRecoverTimer (Kurzform für *Client Frequency Recovery Timer*) bestimmt. Folgende Begebenheiten sind in diesem Zustand möglich:

1. **Empfang der COMMIT-Nachricht vom Koordinator oder einem der n Knoten:** Der Teilnehmer kann die Transaktion erfolgreich abschließen und geht in den Zustand CLIENT_FINISHED über.

2. **Feuern des FrequencyRecoverTimers:** Es ist keine COMMIT-Nachricht vom Koordinator oder einem der n Knoten eingetroffen. Der Teilnehmer muss dazu einen neuen Recovery-Versuch starten. Dazu sendet er wieder eine RECOVER REQUEST-Nachricht an den Koordinator sowie REC_TA-Nachrichten an die vom Koordinator ausgewählten n Knoten. Dieser Prozess wird so lange wiederholt, bis der Teilnehmer die Recovery mit einem COMMIT erfolgreich beenden kann. Bis dahin ist der Teilnehmer unsicher über den Transaktionsausgang und bleibt im Zustand CLIENT_RECOVERING_DIRECT.

3. **Ausfall des Knotens:** Kommt es im Zustand CLIENT_RECOVERING_DIRECT zu einem Ausfall, wechselt der Teilnehmer in den Zustand NODE_FAILURE. Bei der Rückkehr ins

MANET wird die Recovery erneut initiiert. Am Unsicherheitszustand über den Transaktionsausgang ändert ein Ausfall nichts.

Zustand CLIENT_RECOVERING_EPIDEMIC: Der Teilnehmer wartet auf eine COMMIT-Nachricht vom Koordinator oder einem seiner Nachbarn. Die Wartezeit auf Antworten wird auch hier durch den FrequencyRecoverTimer definiert. Die folgenden Ereignisse können in diesem Zustand eintreten:

1. **Empfang der COMMIT-Nachricht vom Koordinator oder einem der Nachbarknoten:** Mit dem Empfang eines COMMIT kann der Teilnehmer die Transaktion erfolgreich beenden und in den Zustand CLIENT_FINISHED übergehen.

2. **Feuern des FrequencyRecoverTimers:** Weder vom Koordinator noch von einem der Nachbarknoten ist eine COMMIT-Nachricht eingetroffen. Der Teilnehmer muss deshalb einen neuen Versuch starten. Er schickt daher noch einmal eine RECOVER REQUEST-Nachricht an den Koordinator sowie einen Single-Hop-Broadcast mit der REC_TA-Nachricht. Ein Teilnehmer wiederholt dieses Verfahren so lange, bis eine erfolgreiche Recovery möglich war. Bis dahin verbleibt er im Zustand CLIENT_RECOVERING_EPIDEMIC und ist unsicher über den Transaktionsausgang.

3. **Ausfall des Knotens:** Fällt ein Teilnehmer im Zustand CLIENT_RECOVERING_EPIDEMIC aus, geht er in den Zustand NODE_FAILURE über. Kommt er anschließend wieder zurück in das MANET, wird die Recovery erneut initiiert. Der Unsicherheitszustand des Teilnehmers bleibt jedoch bestehen.

4.3.4 Vergleichsprozessablauf ohne SLS

Zur Evaluation des SLS ist ein Prozessablauf ohne dessen spezielle Charakteristiken zum Vergleich notwendig. Die Grundstruktur dieses Vergleichsprozessablaufs ist daher identisch zu den in Abschnitten 4.3.2 und 4.3.3 vorgestellten Abläufen. Aufgrund der grundsätzlich ähnlichen Struktur werden sowohl Koordinator- als auch Teilnehmer-Prozessablauf im Folgenden zusammen kurz vorgestellt. Das zu diesem Prozessablauf gehörige Zustandsdiagramm ist in Abbildung 4.5 zu finden.

4.3.4.1 Starten der Transaktion

Um Vergleichbarkeit zwischen den Prozessabläufen garantieren zu können, wird auch hier der Start einer Transaktion bis zum Ende der Abstimmungsphase als Zustand PROCESSING_PHASE sowohl für den Koordinator wie für den Teilnehmer realisiert. Bei beiden ist die Commit-Phase in die bereits vorgestellten Phasen A bis D unterteilt.

Zustand PROCESSING_PHASE: Dieser Zustand umfasst den Transaktionsstart bis zum Beginn der Commit-Phase. Die Aufenthaltsdauer wird ebenfalls durch den ProcessingTimer (Kurzform für *Processing Phase Timer*) Bestimmt. Der Timer muss für Koordinator und Teilnehmer identisch sein, damit alle an der Transaktion beteiligten Knoten gleichzeitig die Commit-Phase betreten. Ein Zustandswechsel ist bei folgenden Ereignissen möglich:

1. **Feuern des ProcessingTimers:** Sowohl Koordinator wie Teilnehmer gehen in Phase A der Commit-Phase über.

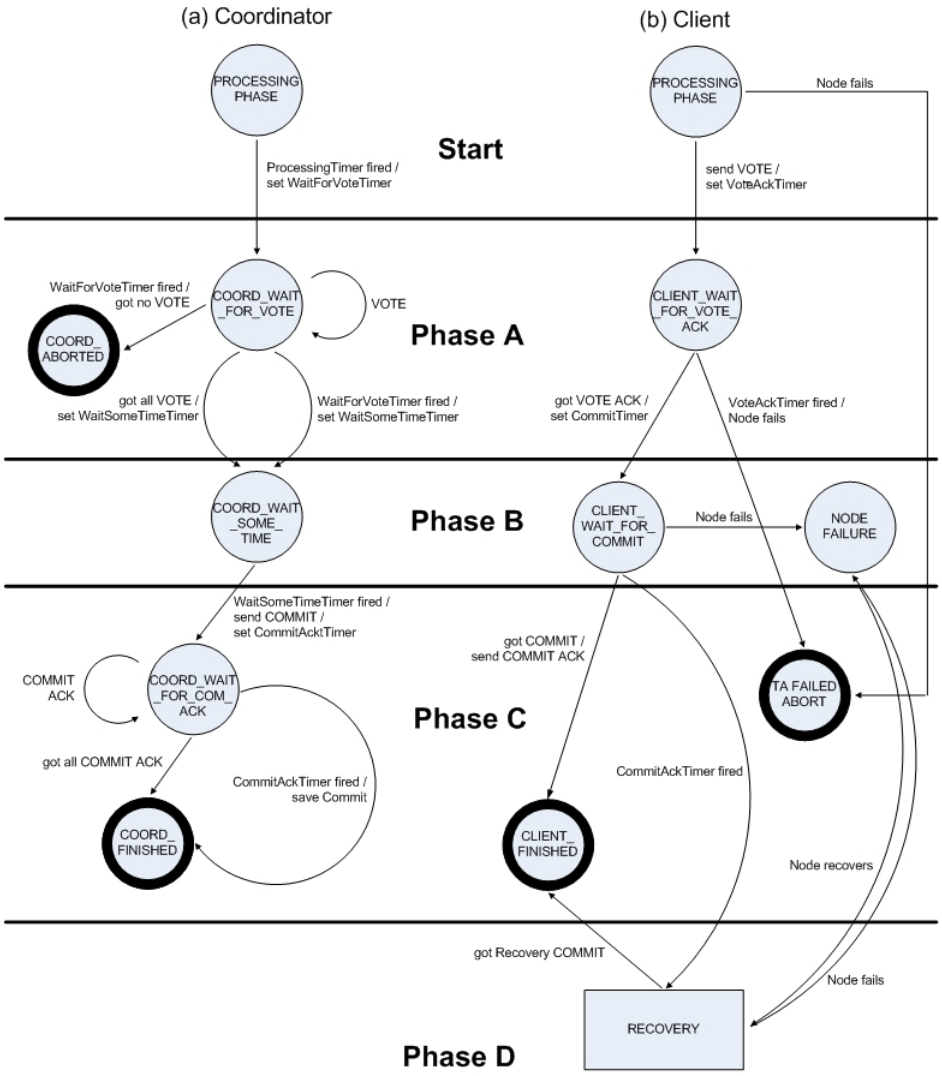

Abbildung 4.5: Vergleichsprozessablauf ohne Verwendung des SLS

a) **Koordinator:** Der Koordinator geht in den Zustand COORD_WAIT_FOR_VOTE über und wartet auf die Vote-Nachrichten der Teilnehmer.

b) **Transaktionsteilnehmer:** Ein Teilnehmer sendet seine Vote-Nachricht an den Koordinator. Anschließend wechselt er in den Zustand CLIENT_WAIT_FOR_VOTE_ACK (Kurzform für *Client Wait For Votum Acknowledgement*).

2. **Ausfall des Knotens:** Der Teilnehmer bricht die Transaktion ab und geht in den Zustand CLIENT_ABORTED in Phase C über. Der Grund dafür ist, dass ohne Kenntnis über die durch den Koordinator verwendete Verteilungsstrategie keine entsprechende Recovery gestartet werden kann.

4.3.4.2 Commit-Phase

Wie auch beim SLS-Prozessablauf ist die Entscheidungsphase in die vier Abschnitte A bis D unterteilt. Auf diese wird in den folgenden Absätzen näher eingegangen.

4.3.4.2.1 Phase A: Votum der Teilnehmer

⇒ Ablauf des Koordinators

Auf der Seite des Koordinators ist Phase A durch den Zustand COORD_WAIT_FOR_VOTE geprägt.

Zustand COORD_WAIT_FOR_VOTE: Der Koordinator wartet auf die Vote-Nachrichten der Transaktionsteilnehmer. Ein Zustandswechsel ist hier bei folgenden Begebenheiten möglich:

1. **Empfang aller Vote-Nachrichten von den Teilnehmern:** Der Koordinator sendet an die Teilnehmer eine VOTE-ACK-Nachricht (Kurzform für *Vote Acknowledgement*). Anschließend wechselt er in den Zustand COORD_WAIT_SOME_TIME. Damit hat er Phase B erreicht.

2. **Feuern des WaitForVoteTimers**

   a) **Empfang mindestens einer Vote-Nachricht:** Hier verhält sich der Koordinator analog zu 1.), allerdings werden im weiteren Verlauf der Transaktion nur die Teilnehmer berücksichtigt, von denen ein Votum eingetroffen ist.

   b) **Ausbleiben aller Vote-Nachrichten:** Den Koordinator hat keine Vote-Nachricht erreicht, so dass er in den Zustand COORD_ABORTED (Kurzform für *Coordinator Aborted*) übergeht und die Transaktion abbricht.

⇒ Ablauf des Teilnehmers

Ein Teilnehmer befindet sich in Phase A in Zustand CLIENT_WAIT_FOR_VOTE_ACK, der im Folgenden näher beschrieben wird.

Zustand CLIENT_WAIT_FOR_VOTE_ACK: Der Teilnehmer wartet auf die VOTE-ACK-Nachricht vom Koordinator. Es gibt drei mögliche Ereignisse für das Verlassen dieses Zustandes:

1. **Empfang der VOTE-ACK-Nachricht:** Der Teilnehmer wechselt in den Zustand CLIENT_WAIT_FOR_COM. Dadurch befindet er sich in Phase B.

2. **Feuern des WaitForVoteAckTimers:** Die VOTE-ACK-Nachricht vom Koordinator ist ausgeblieben. Der Teilnehmer geht vom Scheitern der Transaktion aus und wechselt in den Zustand CLIENT_ABORTED.

3. **Ausfall des Knotens:** Der Teilnehmer wechselt in den Zustand CLIENT_ABORTED und nimmt die Transaktion als gescheitert an. Dasselbe geschieht auch beim Auftreten eines Knotenausfalls in diesem Zustand (in Analogie zum Verhalten des SLS-Klienten, vgl. Abschnitt 4.3.3).

4.3.4.2.2 Phase B: Berechnung der Commit-Entscheidung

⇒ **Ablauf des Koordinators**

Für den Koordinator besteht auch hier Phase B ausschließlich aus dem im Folgenden beschriebenen Zustand.

Zustand COORD_WAIT_SOME_TIME: Dieser Zustand repräsentiert die Simulation der Entscheidungsfindung des Koordinators zum globalen Commit. Auch hier definiert der WaitSomeTimeTimer die Aufenthaltsdauer in diesem Zustand. Folgende Ereignisse sind möglich bzw. folgende Nachrichten werden verarbeitet:

1. **Feuern des WaitSomeTimeTimers:** Der Koordinator sendet die COMMIT-Nachricht an die Teilnehmer und geht in den Zustand COORD_WAIT_FOR_COM_ACK über. Anschließend befindet er sich in Phase C.

2. **Empfang einer REC_TA-Anfrage:** Erhält der Koordinator eine solche Recovery-Anfrage eines ausgefallenen Teilnehmers (vgl. Abbildung .1a), so antwortet er wie beim SLS-Prozessablauf mit einer RUNS-Nachricht. Der recovernde Teilnehmer weiß dadurch, dass noch keine Entscheidung getroffen worden ist.

⇒ **Ablauf des Teilnehmers**

Der Teilnehmer befindet sich in dieser Phase in dem im Zustand CLIENT_WAIT_FOR_COM.

Zustand CLIENT_WAIT_FOR_COM: Der Teilnehmer wartet auf die COMMIT-Nachricht des Koordinators. Die Wartezeit wird durch den CommitTimer bestimmt. Die folgenden Begebenheiten können hier zu einer Zustandsänderung führen:

1. **Empfang der COMMIT-Nachricht vom Koordinator:** Der Teilnehmer wechselt in den Zustand CLIENT_FINISHED und sendet an den Koordinator sein COMMIT ACK. Dazu geht er in den virtuellen Zustand INIT_RECOVERY über, der die Recovery initiiert. Der Teilnehmer hat dann Phase C erreicht.

2. **Feuern des CommitTimers:** Die COMMIT-Nachricht des Koordinators ist ausgeblieben und der Teilnehmer muss eine Recovery starten. Er betritt damit Phase D. Es ist kein COMMIT vom Koordinator beim Teilnehmer eingetroffen. Dieser muss nun eine Recovery starten und geht dazu in den virtuellen Zustand INIT_RECOVERY über, wo er die der Verteilung des Koordinators angemessene Recovery initiiert. Damit betritt er direkt Phase D.

3. **Ausfall des Knotens:** Durch einen Ausfall wechselt der Teilnehmer in den Zustand NODE_FAILURE. Er bleibt dann weiterhin in Phase B bis zum Ende des Ausfalls und dem Start einer Recovery.

4.3.4.2.3 Phase C: Transaktionsabschluss

⇒ **Ablauf des Koordinators**

Der Koordinator durchläuft in dieser Phase vor Abschluss der Transaktion den folgenden Zustand COORD_WAIT_FOR_COM_ACK.

Zustand COORD_WAIT_FOR_COM_ACK: Der Koordinator wartet auf die Bestätigungen über den Empfang des COMMITs von den Teilnehmern. Die Dauer der Wartezeit wird durch den CommitAckTimer beschränkt. Folgende Ereignisse können zu einer Zustandsänderung führen bzw. folgende Nachrichten werden verarbeitet:

1. **Empfang aller COMMIT_ACK-Nachrichten:** Der Koordinator kann die Transaktion erfolgreich beenden und geht in den Zustand COORD_FINISHED (Kurzform für *Coordinator Finished)* über. Damit ist für ihn die Transaktion erfolgreich abgeschlossen.

2. **Feuern des CommitAckTimers:** Einige oder alle COMMIT_ACK-Nachrichten der Teilnehmer sind ausgeblieben. Dies veranlasst den Koordinator, vor dem Wechsel in den Zustand COORD_FINISHED den Transaktionsausgang zu speichern, so dass später Recovery-Anfragen von ausgefallenen Knoten beantwortet werden können. Trotzdem ist für den Koordinator auch hier die Transaktion beendet.

3. **Empfang einer REC_TA-Anfrage:** Empfängt der Koordinator eine REC_TA-Anfrage eines Teilnehmers, so sendet er das COMMIT. An seinem Zustand ändert dies jedoch nichts.

Zustand COORD_FINISHED: Für den Koordinator ist die Transaktion abgeschlossen. Empfängt er eine REC_TA-Anfrage, so sendet er dem ausgefallenen Teilnehmer das COMMIT. Für seinen Zustand hat das jedoch keine Auswirkungen.

⇒ **Ablauf des Teilnehmers**

Wie bereits im vorigen Abschnitt erläutert, befindet sich der Teilnehmer im Zustand CLIENT_FINISHED nach dem Empfang der COMMIT-Nachricht des Koordinators und der Antwort mit COMMIT ACK.

Zustand CLIENT_FINISHED: Die Transaktion ist für den Teilnehmer erfolgreich abgeschlossen. Ein Knotenausfall ändert nichts mehr an seinem Zustand.

4.3.4.2.4 Phase D: Recovery

⇒ **Ablauf des Teilnehmers**

Auch beim Vergleichsprozessablauf umfasst die letzte Phase im Wesentlichen die Recovery eines ausgefallenen Teilnehmers (vgl. Abbildung 4.6). Diese wird entweder nach einem Knotenausfall gestartet oder nach dem Feuern des CommitTimers.

Virtueller Zustand INIT_RECOVERY: Es wird zunächst überprüft, in welchem Zustand sich der Knoten vor der Recovery befand. War das der Zustand CLIENT_WAIT_COM_ACK oder der Zustand NODE_FAILURE, so kann die Recovery initiiert werden. Bei jedem anderen Vorgängerzustand bricht der Teilnehmer die Transaktion ab und geht in den Zustand CLIENT_ABORTED (bzw. verbleibt im Zustand CLIENT_FINISHED bei bereits beendeter Transaktion). Wird eine Recovery gestartet, sendet der Teilnehmer eine REC_TA-Nachricht an den Koordinator. Der Teilnehmer geht in den Zustand CLIENT_RECOVERING über.

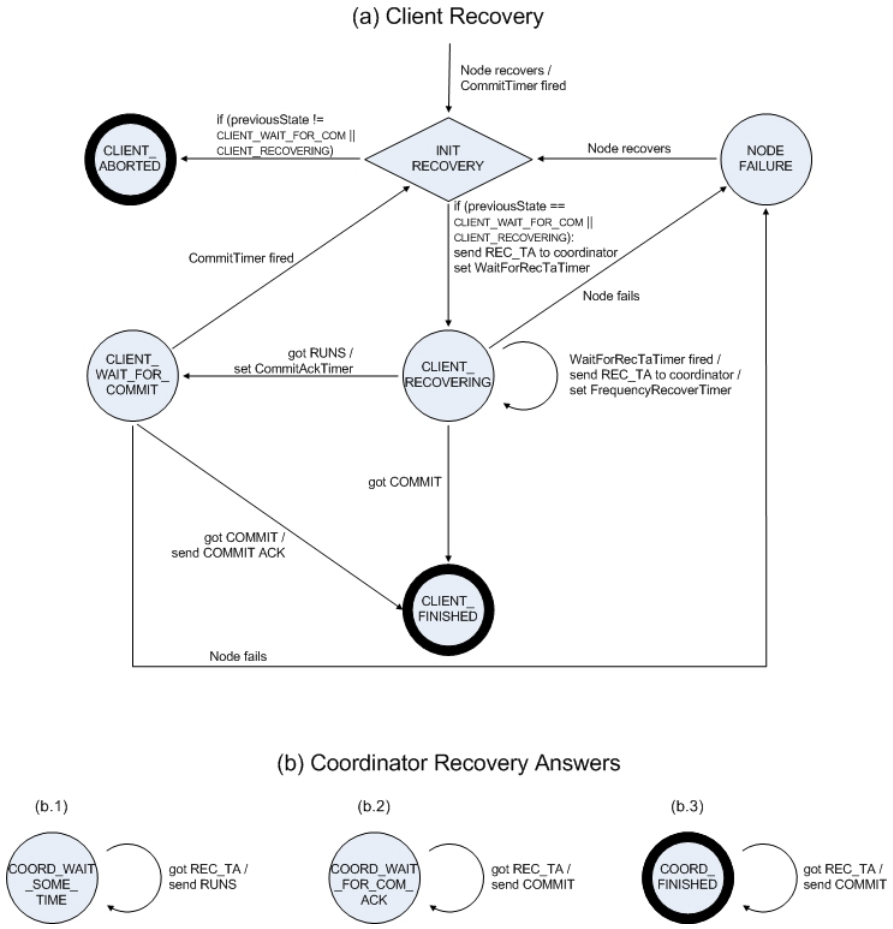

Abbildung 4.6: (a) Prozessablauf bei der Recovery eines Klienten ohne SLS und (b) Beantwortung von Recovery-Anfragen durch den Koordinator ohne SLS

Zustand CLIENT_RECOVERING: Der Teilnehmer wartet auf die Recovery-Antwort des Koordinators. Bei folgenden Ereignissen ist ein Zustandswechsel möglich:

1. **Empfang der COMMIT-Nachricht vom Koordinator:** Der Teilnehmer kann die Transaktion erfolgreich abschließen und in den Zustand CLIENT_FINISHED übergehen.

2. **Empfang einer RUNS-Nachricht vom Koordinator:** Der Teilnehmer war nur kurz ausgefallen während der Entscheidungsberechnung und geht nun wieder in den Zustand CLIENT_WAIT_FOR_COM über. Er kann mit dem regulären Prozessablauf fortfahren.

3. **Ausfall des Knotens:** Erleidet der Teilnehmer einen Ausfall, wechselt er in den Zustand NODE_FAILURE. Bei seiner Rückkehr ins MANET nach dem Ausfall wird die Recovery erneut initiiert.

4. **Feuern des WaitForRecTaTimers:** Es ist kein COMMIT vom Koordinator eingetroffen. Der Teilnehmer muss dann eine neue REC_TA-Anfrage an den Koordinator schicken. Gleichzeitig setzt er den FrequencyRecoverTimer anstatt des WaitForRecTaTimers. Solange der Teilnehmer keine Recovery-Antwort vom Koordinator erhält, bleibt er im Zustand CLIENT_RECOVERING und ist unsicher über den Transaktionsausgang. Phase D wird also auch beim Vergleichs-Prozessablauf erst bei einer erfolgreich abgeschlossenen Transaktion verlassen.

Kapitel 5

Evaluation des Shared Log Space

5.1 Netzwerksimulator NS2

Der an der UC Berkeley entwickelte *Netzwerksimulator NS Version 2* (kurz NS2, vgl. [3, 27]) ist ein objekt-orientierter, diskreter ereignis-orientierter Netzwerksimulator, geschrieben in C++ und OTcl (vgl. [4]). OTcl entspricht der Tcl-Skriptsprache mit objekt-orientierten Erweiterungen. NS2 ist der am meisten genutzte Netzwerksimulator im Bereich der MANET-Forschung (vgl. z.B. [32]). Mit NS2 können lokale, Wide Area Netzwerke sowie mobile Netzwerke simuliert werden. Es steht eine Vielzahl von Protokollen für die Simulation zur Verfügung, z.B. TCP, UDP, FTP und Telnet. Außerdem sind verschiedene Router Queue Mechanismen und eine Reihe von Routing Algorithmen implementiert. Eine Stärke neben der Vielfalt an implementierten Protokollen ist die Fähigkeit zur graphischen Darstellung von Netzwerkverkehr mit dem *Network Animator NAM* (vgl. [2]).

Zur Benutzung von NS2 müssen sog. Tcl-Skripten geschrieben werden, die die eigentliche Funktionalität der Knoten in den Simulationen definieren. Diese initiieren ein Ereignis-Steuerprogramm (engl. *event scheduler*) und bestimmen die Netzwerktopologie unter Einbeziehung von Netzwerkobjekten sowie Funktionen der Netzwerk-Setup-Modulbibliotheken. Außerdem wird festgelegt, wann Pakete verschickt werden und wann das Senden eingestellt werden soll. Soll eine Simulation mit der bestehenden Funktionalität von NS2 durchgeführt werden, so müssen OTcl-Skripten mit dem gewünschten Simulationsprogramm geschrieben werden. Wenn NS2 um neue Funktionalitäten wie z.B. Protokolle erweitert werden soll (diese werden oft auch als *Agenten* bezeichnet), muss die Gesamtarchitektur von NS2 dabei berücksichtigt werden. Dies beinhaltet die Simulatorobjekte in der OTcl-Bibliothek ebenso wie die Ereignis-Steuerprogramm-Objekte und die in C++ geschriebenen Netzwerkkomponenten der NS Simulatorbibliothek. Auf die in C++ implementierten Teile kann man mit OTcl nur durch eine in tclcl (vgl. [5]) implementierte OTcl-Verknüpfung zugreifen. Dadurch entspricht NS2 einem objekt-orientierten Tcl-Interpreter mit speziellen Simulatorbibliotheken. Bei der Simulation mit einem OTcl-Skript werden verschiedene Ergebnisdateien angelegt, die z.B. den Netzwerkverkehr protokollieren. Diese können analysiert oder mit dem Netzwerkanimator NAM graphisch verfolgt werden.

Die Bewegung von Knoten wird in sog. *Szenarien-Dateien* spezifiziert. Solche Szenarien mit Bewegungsmustern für mobile Knoten können mit dem Tool *setdest*, welches ebenfalls von NS2 angeboten wird, generiert werden. Bei der Generierung können beispielsweise die Simulationsfläche, die Anzahl von mobilen Knoten sowie deren Geschwindigkeit im Szenario vorgegeben werden. Bei den mit setdest erzeugten Szenarien bewegen sich die Knoten entsprechend dem Random Waypoint Algorithmus (vgl. dazu 3.5.1). Inzwischen stehen jedoch eine Vielzahl weiterer Bewegungsmodelle zur Verfügung. Einige davon werden kurz in Abschnitt 3.5 vorgestellt.

Für die Simulation muss in den Tcl-Skripten u.a. die Netzwerkumgebung sowie das Netzwerk selbst mit der zugrunde liegenden Szenariendatei spezifiziert werden. Außerdem ist die Erzeugung von Knoten notwendig, an die die gewünschten Agenten gebunden werden. Die Agenten selbst besitzen die gewünschte Funktionalität, so dass sie z.B. Protokolle repräsentieren können. Mit diesen Einstellungen kann dann die gewünschte Simulation gestartet werden.

5.2 Ablauf der Simulationen

Für die Simulation des SLS wird der zuvor vorgestellte Netzwerksimulator NS2 verwendet. Um den SLS damit simulieren zu können, muss dessen Funktionalität implementiert werden. Da eine möglichst große Anzahl von Transaktionen untersucht werden soll, werden MANET-Szenarien für den NS2 benötigt, welche die Bewegung von Knoten für jeden Zeitpunkt der Simulation definieren. Für diese Szenarien werden Simulationsskripten generiert, die die zu simulierenden Transaktionen enthalten. Diese können dann mit NS2 simuliert werden. Zuletzt ist zur Auswertung der Simulationen die Analyse der Log-Dateien notwendig. Der detaillierte Ablauf wird im Folgenden vorgestellt.

5.2.1 Überblick über den Simulationsablauf

Die Simulationen des SLS können in verschiedene Schritte untergliedert werden. Einen Überblick darüber bietet Abbildung 5.1. Einige der folgenden Abschnitte reflektieren einzelne Blöcke des Simulationsablaufs und werden im Folgenden entsprechend detailliert erläutert. Zunächst müssen mit Hilfe des NS2-Tools *setdest* MANET-Szenarien generiert werden. Diese beinhalten die Bewegungsmuster der mobilen Knoten. Aus den Szenarien können dann sog. *Simulations-Control-Dateien* erzeugt werden. Diese enthalten für definierte Zeitpunkte im Simulationszeitraum jeweils die Verbindungsqualität zwischen allen Knoten.

Zur Generierung der Tcl-Skripten werden sowohl die Szenarien wie die Control-Dateien und eine Properties-Datei mit den gewünschten Parametern benötigt. Außerdem ist die vorherige Implementierung eines neuen Protokolls für den SLS notwendig. Dieses muss Schnittstellen definieren, die bei der Erzeugung der Tcl-Skripten verwendet werden können. Auf Details wird im Abschnitt 5.2.4.2 eingegangen, da mit Hilfe der Properties-Datei alle für die Simulation notwendigen Parameter spezifiziert werden.

Nachdem die Tcl-Skripten erzeugt worden sind, können die eigentlichen Simulationen mit NS2 gestartet werden. Dazu müssen die vorher erstellten MANET-Szenarien zur Verfügung stehen.

Bei der Simulation der generierten Simulationsskripten werden Log-Dateien angelegt. Diese protokollieren den Ablauf der gestarteten Transaktionen. Zusätzlich werden am Ende jedes Simulationslaufs die globalen Ergebnisse in eine Log-Datei geschrieben, z.B. die Anzahl der gestarteten Transaktionen und die Anzahl der erfolgreichen Transaktionen. Da aufgrund der Variation der Parameter eine Vielzahl von Log-Dateien erzeugt werden, bietet sich die Weiterverarbeitung der Log-Dateien zur Analyse an. Dazu werden die Logs geparst und die Ergebnisse in eine Datenbank eingefügt. Dies erlaubt die Analyse der Logs mit beliebigen Berechnungen, die wiederum als Parameter spezifiziert werden. Auf diesen Teil wird detaillierter in den Abschnitten 5.2.6 und 5.2.7 eingegangen.

5.2.2 Generierung der MANET-Szenarien-Dateien

Die für die Simulationen benötigten MANET-Szenarien-Dateien werden mit dem NS2-Tool *setdest* erzeugt. Danach werden die Parameter vorgestellt, die im Rahmen dieser Evaluation verwendet werden.

5.2.2.1 Erzeugung von Szenarien mit setdest

Setdest ist ein Tool, welches zusammen mit dem eigentlichen Netzwerksimulator NS2 zur Verfügung steht (im Verzeichnis ~ns/indep-utils/cmu-scen-gen/setdest). Es handelt sich dabei um ein Konsolenprogramm, welches wie folgt mit den aufgezählten Parametern aufgerufen wird:

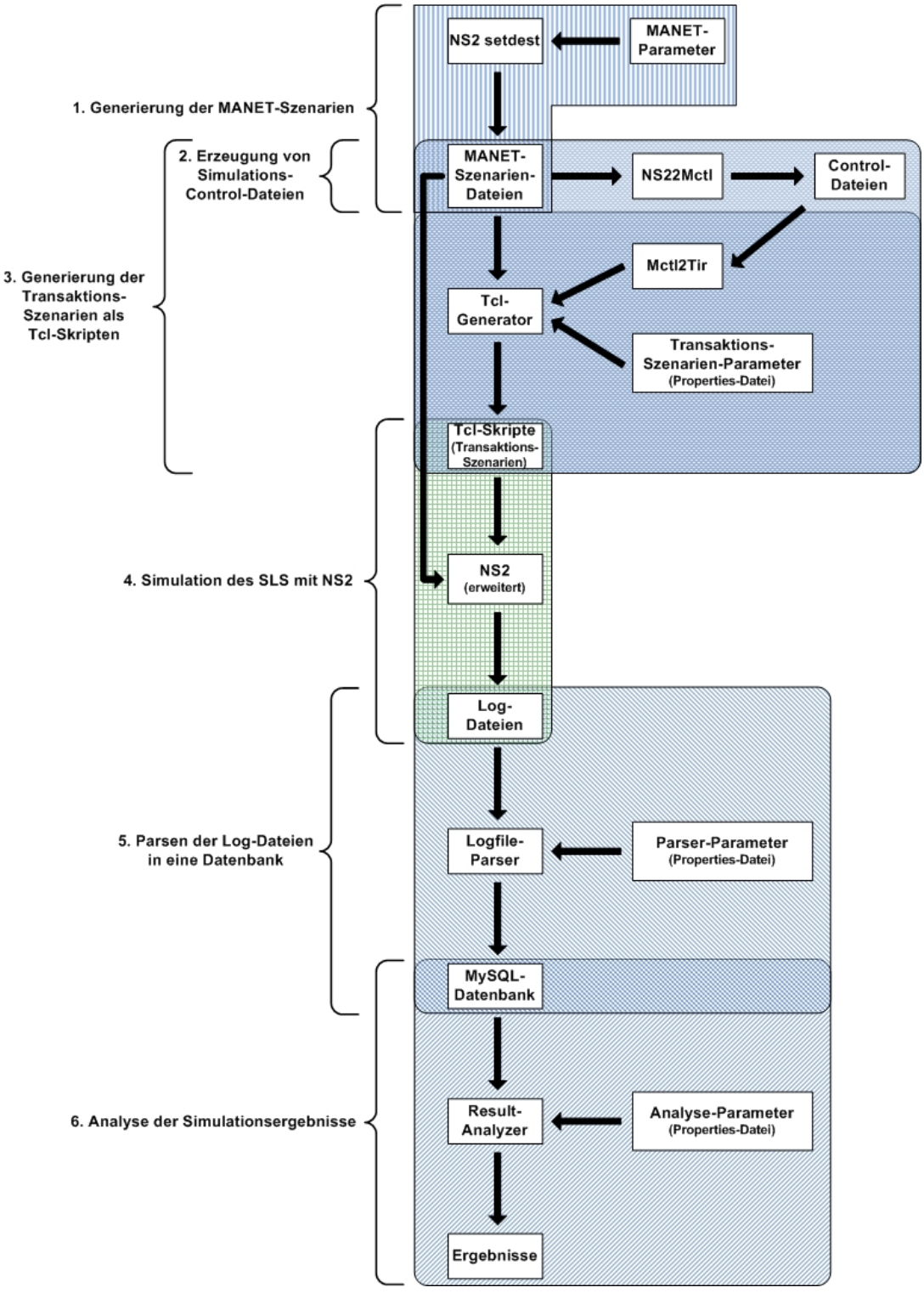

Abbildung 5.1: Übersicht über den Simulationsablauf

Knotenanzahl		**Simulationsdauer**
echte Anzahl	**erwartete Anzahl aufgrund von Ausfällen**	
10	5	500000 Sek.
20	10	500000 Sek.
30	15	50000 Sek.
40	20	50000 Sek.
50	25	50000 Sek.

Tabelle 5.1: Laufzeit der Simulationen für die jeweilige Knotenanzahl

Parameter	**Wert**
Fläche	500x500
Pausenzeit	1 Sek.
Geschwindigkeit	2 bis 10 Millisek.
Knotenanzahl	10 bis 50 Knoten
Bewegungsmodell	Random Waypoint
Signalausbreitung	Two Ray Ground

Tabelle 5.2: Allgemeine Parameter der Mobilitätsszenarien

```
./setdest [-n num_of_nodes] [-p pausetime] [-s maxspeed] [-t simulationtime] [-x maxx] [-y maxy]
> [outdir/movement-file]
```

Entsprechend den angegebenen Spezifikationen wird dann eine Szenarien-Datei generiert, in der sich die darin enthaltenen Knoten gemäß den übergebenen Parametern verhalten. Die Datei beginnt dabei mit der Ausgangsposition der Knoten am Beginn der Simulationszeit. Dann folgt die Definition der Knotenbewegungen wie z.B.

```
$ns_ at 2.000000000000 "$node_(0) setdest 90.441179033457 44.896095544010 1.373556960010"
```

Mit dieser Zeile wird beispielsweise definiert, dass sich der Knoten 0 zum Zeitpunkt 2s mit der Geschwindigkeit von 1.37 m/s zum Punkt (90.44, 44.89) bewegen wegen soll. Dasselbe geschieht für jeden Knoten während der ganzen Simulationszeit immer wieder, sobald eine Richtungsänderung erforderlich ist (anderenfalls ist mit dieser Zeile die Bewegung entlang einer Strecke definiert). Setdest generiert die Knotenbewegung dabei unter Verwendung des Random Waypoint Algorithmus (vgl. 3.5.1).

5.2.2.2 Verwendete Parameter für die MANET-Szenarien der SLS-Evaluation

Insgesamt werden sechs Szenarienmengen generiert, die jeweils mehrere Szenarien-Dateien enthalten. Da in den Szenarien mit 10 und 20 Knoten weniger Transaktionen aufgrund der geringen Dichte zustande kommen, wird eine längere Simulationszeit gewählt (vgl. Tabelle 5.1). Da dennoch bei ersten Tests keine vergleichbare Anzahl an Transaktionen gestartet werden konnte, werden für beide Netzdichten zusätzliche Szenarien für jede Szenarienmenge generiert.

Bei allen Szenarien werden, abgesehen von der Knotenanzahl und der Simulationsdauer, identische Parameter verwendet (vgl. Tabelle 5.2). Die Knoten bewegen sich dabei entsprechend dem Random Waypoint-Bewegungsmodell. Bei der Signalausbreitung wird Two Ray Ground verwendet.

5.2.3 Implementierung des SLS für NS2

In diesem Abschnitt wird auf einige wesentliche Aspekte der NS2-Implementierung und das Implementierungsmodell eingegangen. Dies betrifft jedoch nicht die grundlegende Entwicklung und grobe Struktur eines neuen Protokolls, das zur Umsetzung des SLS-Konzeptes notwendig ist. Darauf wird beispielhaft im Anhang A.2.1 eingegangen. Auf programmierspezifische Details zu C/C++ wie z.B. das Einbinden einzelner Pakete wird ebenfalls verzichtet. Nach Beleuchtung charakteristischer Aspekte bei der Implementierung werden abschließend die Schnittstellen vorgestellt, über die in den Tcl-Skripten die notwendigen Parameter für die Transaktionen wie z.B. Timer oder Ausfallzeiten bei den Agenten gesetzt werden können.

5.2.3.1 NS2-Implementierungsmodell für die Prozessabläufe

Bei der Implementierung der SLS-Prozessabläufe ist das Ziel, dass nach dem Start der Transaktion sowohl Koordinator wie Teilnehmer selbständig agieren. Dies ist insbesondere deshalb von Bedeutung, da die Dauer der Unsicherheitszustände beim Auftreten von Fehlern untersucht werden soll. Für die Simulation werden zwei neue Protokolle entwickelt, zum einen das Protokoll ohne die Funktionalitäten des SLS (der sog. *SimpleAgent*) und zum anderen das Protokoll mit den Charakteristiken des SLS (der sog. *DisseminationAgent*). Beide sind von der Grundstruktur identisch, die in Abschnitt 4.3 vorgestellt wird. Auf die Implementierungsunterschiede bei beiden Protokollen wird hier nur insoweit eingegangen, wie Besonderheiten des SLS-Prozessablaufs betroffen sind.

Das allgemeine Modell zur Implementierung der Prozessabläufe zeigt Abbildung 5.2. Um eine möglichst große Anzahl von Transaktionen starten zu können, soll ein Knoten an mehreren Transaktionen teilnehmen können. Dabei kann er jeweils unterschiedliche Funktionen haben, d.h. sowohl als Teilnehmer wie als Koordinator agieren. Zu diesem Zweck werden für beide Protokolle zwei Agenten entwickelt. Ein Agent, der sog. *SLSAgent,* repräsentiert einen Knoten und wird entsprechend einmal an jedes Knotenobjekt gebunden. Ergänzend dazu wird ein Transaktionsagent, der sog. *DisseminationAgent,* entwickelt. Dieser entspricht entweder einem Teilnehmer oder einem Koordinator und wird an einen SLSAgenten gebunden. Dieser Agent verhält sich entsprechend dem eigentlichen SLS-Prozessablauf (vgl. Abschnitt 4.3). Für jede Transaktion, an der ein Knoten teilnimmt, wird ein solcher DisseminationAgent neu erzeugt. Somit kann ein Knoten gleichzeitig für eine Transaktion der Koordinator und bei einer anderen Transaktion ein Teilnehmer sein. Jede dieser Transaktionen kann sich dabei in einem unterschiedlichen Stadium befinden. Gleichzeitig beinhaltet der SLSAgent Kernfunktionen des SLS. Er speichert die verteilten Transaktionslogs und antwortet auf Anfragen nach der Ausfallwahrscheinlichkeit des Knotens. Außerdem steuert er Ausfälle und die Recovery, indem er den Knoten und die an diesen gebundenen Transaktionsagenten darüber informiert (vgl. dazu Abschnitt 5.2.3.4). Diese Funktionalitäten sind unabhängig von einer Teilnahme an einer Transaktion und können deshalb vom SLSAgenten als Knotenobjekt zentral bearbeitet werden. Nachrichtenpakete erreichen den Knoten selbst und werden an den betreffenden Agenten weitergereicht. Jeder an einen Knoten gebundene Agent erhält einen individuellen Port, so dass Pakete den gewünschten Agenten erreichen können. Dabei hat der SLSAgent immer Port 0, da er eine Erweiterung des Knotenobjektes repräsentiert.

Für den Vergleichsprozessablauf ohne SLS wird zur Vergleichbarkeit ebenso verfahren wie in Abbildung 5.2 dargestellt. Es wird an den Knoten der sog. *SimpleSLSAgent* als Knotenobjekt gebunden. Dieser hat jedoch in diesem Fall keine Funktionalität außer dem Steuern der Knotenausfälle. Der Koordinator bzw. Teilnehmer einer Transaktion wird hier auch als eigener Agent, der sog. *SimpleAgent*, implementiert. Er verhält sich gemäß des Vergleichsprozessablaufs (vgl.

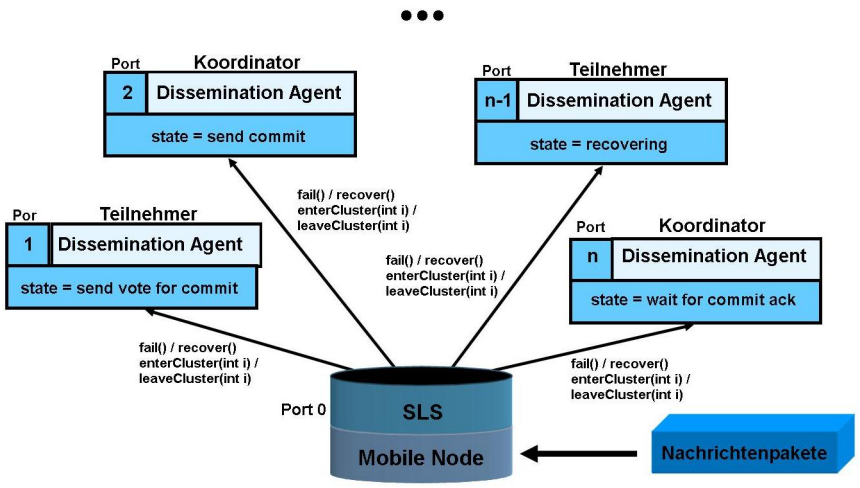

Abbildung 5.2: Implementierungsmodell für NS2 (für die Simulation des SLS und die Simulation des um Mobilitätsberücksichtigung erweiterten SLS)

Abschnitt 4.3.4).

5.2.3.2 Annahmen für die Implementierung

Es wird angenommen, dass der Zeitpunkt des Transaktionsstarts bekannt ist, an dem Koordinator und Teilnehmer einander erreichen können. Dadurch kann der SLS-Prozessablauf so implementiert werden, dass sich sowohl Koordinator wie Teilnehmer nach dem Start völlig selbständig entsprechend dem Protokoll verhalten und bei Fehlern entsprechend dem Prozessablauf eine Fehlerbehandlung starten. Ebenso wird davon ausgegangen, dass die Teilnehmer und der Koordinator der Transaktion bekannt sind. So können den Knoten jeweils die entsprechenden Informationen über die Teilnehmer und den Koordinator wie z.B. Adresse und Port vor dem Transaktionsstart mitgeteilt werden[1].

Im Rahmen dieser Simulationen wird außerdem angenommen, dass im Vorfeld für ein MANET-Szenario festgelegt wird, ob es dicht oder dünn ist. Für die Implementierung ist diese Annahme insofern wichtig, als alle Skripten jeweils mit beiden Möglichkeiten simuliert werden sollen, um die Effektivität der Strategien in Abhängigkeit von der Netzdichte untersuchen zu können. Dieser Wert wird als Parameter bei der Erzeugung der Agenten gesetzt (1 für dichtes MANET und 0 für dünnes MANET).

5.2.3.3 Implementierung der Prozessabläufe

Bei der Implementierung werden zur Steuerung der Kommunikation Zustände und Timer verwendet, die eine präzise Abbildung der SLS-Prozessabläufe ermöglichen. Diese werden bei der Beschreibung der Prozessabläufe in Abschnitt 4.3 vorgestellt. Bei der Implementierung der eingeführten Timer wird, wie in Anhang A.2.1 beschrieben, vorgegangen. Zusätzlich werden die notwendigen Zustände und Nachrichtentypen definiert. Damit kann sichergestellt werden, dass Nachrichten nur in dem Zustand verarbeitet werden, in dem ein Knoten sie erwartet.

[1] Vgl. dazu die Annahmen beim Systemmmodell und dem Transaktionsmodell in den Abschnitten 2.2 und 2.4.4, dass der gegenseitige Informationsaustausch zwischen Koordinator und Teilnehmer in der Processing-Phase geschehen ist und hier nur die Commit-Phase betrachtet wird.

Sowohl der DisseminationAgent wie der SimpleAgent umfassen zum einen den Prozessablauf des Koordinators und zum anderen den des Teilnehmers. Beim Koordinator wird bei der Erzeugung eine zusätzliche Variable definiert, die die Unterscheidung beim Ablauf des Protokolls ermöglicht. So arbeitet ein Koordinator andere Zustände ab als ein Teilnehmer und es werden jeweils verschiedene Nachrichten erwartet und beantwortet.

Für die in Abschnitt 4.3 beschriebenen Nachrichten werden entsprechende Nachrichtentypen definiert. Bei der Implementierung werden dafür jeweils entsprechende Methoden geschrieben, die für das Senden der gewünschten Nachricht zuständig sind. So ist z.B. die Methode sendVote() für das Senden der VoteNachricht auf der Seite der Teilnehmer zuständig. Auf Implementierungsdetails und Codebeispiele bei der Protokollentwicklung für NS2 wird im Anhang A.2.1 näher eingegangen.

5.2.3.4 Ausfälle von Knoten

In den Simulationen sollen verschiedene Arten von Ausfällen und deren Auswirkungen untersucht werden. Es bietet sich daher an, die Ausfälle von Knoten, basierend auf dem gewünschten Wahrscheinlichkeitsmodell, selbst zu erzeugen. Dazu sind zwei Methoden fail() und recover() beim SLS-Agenten (bzw. beim SimpleSLSAgent) notwendig (vgl. Abbildung 5.2). Die Methode fail() lässt einen Knoten zu dem Zeitpunkt, an dem sie aufgerufen wird, ausfallen. Der SLS-Agent meldet den Ausfall zunächst an das eigentliche Knotenobjekt. Da ein Knoten nicht aus dem der Simulation zugrunde liegenden Szenario entfernt werden kann und auch keine neuen Knoten erzeugt werden können, wird folgende Alternative gewählt: ein Knoten verlässt das Netz nicht und bewegt sich entsprechend dem Bewegungsmuster weiter. Während des Ausfalls reagiert er jedoch auf keine Nachricht an ihn und nimmt auch am Routing nicht teil. Darüber hinaus ist er auch für die anderen Knoten des Netzwerkes nicht sichtbar. Außerdem wird der Ausfall an alle Transaktionsagenten (je nach Prozessablauf DisseminationAgent oder SimpleAgent) gemeldet. Diese unterbrechen ihre jeweiligen Operationen und wechseln in den Zustand *ausgefallen*.

Wird beim Ende des Ausfalls durch den SLS-Agenten die recover()-Methode aufgerufen, nimmt der Knoten wieder normal an den Aktivitäten im MANET teil. Die Transaktionsagenten (je nach Prozessablauf DisseminationAgent oder SimpleAgent) überprüfen beim Aufruf ihrer recover()-Methode zunächst, in welchem Zustand sie vor dem Ausfall waren, und starten ggf. eine Recovery.

Mit diesem Verfahren können extern für einen Knoten mehrere Ausfälle mit einer beliebigen Wahrscheinlichkeitsverteilung generiert werden. Durch eine Schnittstelle zu Tcl kann der Ausfallzeitpunkt vor der Simulation auf Grundlage einer beliebigen Wahrscheinlichkeitsverteilung bestimmt werden. Dieser Wert kann dann über die Schnittstelle an NS2 übergeben werden. Auf die Schnittstellen zu Tcl wird in Abschnitt 5.2.3.6 eingegangen und ein Beispiel für deren Verwendung wird in Abschnitt 5.2.4.4 vorgestellt.

5.2.3.5 SLS-spezifische Implementierungscharakteristiken

Die Implementierung des SLS-Prozessablaufs erfordert einige zusätzliche Erweiterungen beim DisseminationAgenten, die der SimpleAgent nicht benötigt. Auf diese wird im Folgenden kurz eingegangen.

Dichte des MANETs

Im Rahmen dieser Simulationen wird im Vorfeld für ein MANET-Szenario festgelegt, ob es als dicht oder dünn betrachtet werden soll. Dazu wird die Dichte als Parameter bei der Erzeugung der Agenten übergeben (1 für dicht und 0 für dünn). Alle Skripten sollen auf diese Art mit beiden Optionen simuliert werden.

Kontextbewusstsein der Knoten im MANET

Das Kontextbewusstsein der Knoten wird wie die Dichte des MANETs mit Hilfe einer Variable bestimmt. Dieser Wert wird bei den SLS-Agenten gesetzt. Abhängig davon (1 für Kontextbewusstsein oder 0 für kein Kontextbewusstsein) antwortet ein SLS-Agent dann auf eine Anfrage nach der Ausfallwahrscheinlichkeit oder nicht.

Bestimmung der zu verwendenden Verteilungsstrategie durch den Koordinator

Prinzipiell wird bei dem in Abschnitt 4.3 vorgestellten SLS-Prozessablauf bei den Verteilungsstrategien in einem ersten Schritt zwischen dichten und dünnen MANETs unterschieden. Dieser Parameter wird im Vorfeld im Tcl-Skript gesetzt. Es wird dabei eine Variable verwendet, so dass in der Implementierung beim Auftreten von Ausfällen einfach danach unterschieden werden kann. Wird ein dünnes MANET angenommen, so berechnet der Koordinator den Verteilungsgrad und sendet damit einen Broadcast an seine Nachbarn zur ungerichteten Verteilung (vgl. dazu 4.2.3). Auf welche Art die Knoten das Transaktionslog verteilen, wird im folgenden Abschnitt beschrieben.

Bei einem dichten MANET muss etwas anders verfahren werden. Hier sendet der Koordinator zunächst einen Single-Hop-Broadcast und fragt seine Nachbarknoten nach ihrer Ausfallwahrscheinlichkeit. Hat er Antworten erhalten, berechnet er, ob er mit ihnen die von der Anwendung gewünschte Zuverlässigkeit $p_{retrieval_{app}}$ erreichen kann. Ist dies nicht der Fall, sendet er eine zweite Anfrage nach den Ausfallwahrscheinlichkeiten an seine Nachbarn. Nachdem er wiederum entsprechend dem zugehörigen Timer gewartet hat, testet der Koordinator erneut, ob die geforderte Zuverlässigkeit erreicht werden kann. Wenn nicht, wird er sich für die ungerichtete Verteilungsstrategie entscheiden und dies den Teilnehmern mitteilen. Reichen einige oder alle der Knoten aus, die sich bei ihm gemeldet haben, fügt er deren Knoten-ID in eine Liste ein und sendet sie an die Teilnehmer. Diese Information ist für die Teilnehmer ausreichend, um später eine Recovery-Anfrage an diese Knoten senden zu können.

Verteilung des Transaktionslogs in dünnen MANETs

Bei der ungerichteten Verteilung in dünnen MANETs werden die Daten einmal an alle Knoten verteilt, die mittels Single-Hop-Broadcast erreichbar sind. Dabei besteht jedoch die Möglichkeit, dass damit nur ein Teil des Netzwerkes aufgrund von Partitionierungen die Nachricht erhält. Deshalb ist es wünschenswert, dass Knoten zu einem späteren Zeitpunkt mit neuen Nachbarn die gespeicherten Transaktionslogs austauschen. Da die Annahme ist, dass sich das Netz aufgrund der Mobilität der Geräte immer wieder ändert, kann so im Laufe der Zeit ein größerer Prozentsatz von Knoten das Transaktionslog erhalten.

Um diesen höheren Grad der Verteilung zu erzielen, wird eine vereinfachte Variante des in [13] beschriebenen sog. *Adaptiven Pull-Protokolls* verwendet. Dazu werden Statusinformationen über den aktuellen Verteilungsprozess für die weitere Verteilung verwendet. Solange ein Knoten keine neuen Transaktionslogs erhalten hat, wird er keine weiteren Verteilungsmaßnahmen einleiten. Hat er jedoch neue Logs erhalten, so setzt er einen Timer für den Zeitpunkt der Weiterverteilung. Damit soll sichergestellt werden, dass er die Transaktionslogs nicht sofort wieder an Knoten schickt, von denen er diese zuvor selbst erhalten hat. Durch die zeitliche *Verschiebung* wird ermöglicht, dass sich durch Bewegung des Knotens auch dessen Nachbarschaft verändert. Dadurch kann ein Knoten die Daten an andere verteilen, die diese ggf. vorher noch nicht hatten.

5.2.3.6 Schnittstellen der Implementierung für die Parameterübergabe in den Tcl-Skripten

Um mittels eines Tcl-Skriptes Parameter an die implementierten Agenten für NS2 übergeben zu können, ist die Definition von Schnittstellen notwendig. Diese bestehen bei NS2 aus einer sog.

command()-Methode. Darin wird festgelegt, welche Parameter mit welcher Länge und welchen Werten an den Agenten übergeben werden können.

Prinzipiell kann die Anzahl der Argumente bei den Schnittstellen variieren. Für diese Arbeit werden nur zwei und drei Argumente benötigt. Bei der Argumentlänge 2 wird mit dem ersten Argument der Agent angesprochen, auf den sich der Befehl beziehen soll. Mit dem dritten Argument wird zusammen mit dem Befehl ein Parameterwert übergeben.

Für den DisseminationAgenten ist ein Ausschnitt aus dieser Methode im Folgenden dargestellt:

Listing 5.1: Code-Ausschnitt aus der command()-Methode des DisseminationAgenten

```
1  int DisseminationAgent::command(int argc, const char*const* argv) {
2          // two arguments in the tcl-script
3          if (argc == 2) {
4                  // start the client by sending vote message to the coordinator
5                  if (strcmp(argv[1], "send") == 0) {
6                          // do something
7                          ...
8                          return TCL_OK;
9                  }
10         // tell the agent to be coordinator and start transaction
11         if (strcmp(argv[1], "be-coordinator") == 0) {
12                 // set the corresponding value
13                 coordinator_ = true;
14                 return TCL_OK;
15         }
16         // if there are three arguments in the tcl-script
17         } else if (argc == 3) {
18                 // tell the participant the adress of the coordinator
19                 if (strcmp(argv[1], "set-coordinator") == 0) {
20                         // parse the given value and convert to an object
21                         Agent *coord = (Agent *)(TclObject::lookup(argv[2]));
22                         return TCL_OK;
23                 // inform the coordinator about a participant
24                 } else if (strcmp(argv[1], "add-member") == 0) {
25                         // tell the coordinator about a particpant
26                         DisseminationAgent *member = (DisseminationAgent *)(
                               TclObject::lookup(argv[2]));
27                         return TCL_OK;
28                 }
29                 // set the transaction id
30                 else if (strcmp(argv[1], "set-taId") == 0) {
31                         // parse the given value
32                         taId_ = atoi(argv[2]);
33                         return TCL_OK;
34                 }
35                 // set the global energy failure rate for the coordinator
36                 else if (strcmp(argv[1], "set-globalEnergyFailureRate") == 0) {
37                         globalEnergyFailureRate = atof(argv[2]);
38                         return TCL_OK;
39                 }
40                 // set the global technical failure rate for the coordinator
41                 else if (strcmp(argv[1], "set-globalTechFailureRate") == 0) {
42                         globalTechFailureRate = atof(argv[2]);
43                         return TCL_OK;
44                 }
45                 // set the p_retrieval desired by the application
46                 else if (strcmp(argv[1], "set-pretrieval") == 0) {
47                         pretrieval_ = atof(argv[2]);
48                         return TCL_OK;
```

```
49                  }
50                  // set the mission time
51                  else if (strcmp(argv[1], "set-missionTime") == 0) {
52                          missionTime = atof(argv[2]);
53                          return TCL_OK;
54                  }
55                  // set the number of nodes in the MANET
56                  else if (strcmp(argv[1], "set-nodeNumber") == 0) {
57                          nodeNumber = atof(argv[2]);
58                          return TCL_OK;
59                  }
60                  // set the density of the MANET
61                  else if(strcmp(argv[1], "set-Density") == 0) {
62                          int tmp = atof(argv[2]);
63                          return TCL_OK;
64                  }
65                  // set the waitSomeTimeDuration of the coordinator
66                  else if (strcmp(argv[1], "set-coord_waitSomeTimeDuration") == 0) {
67                          coord_waitSomeTimeDuration = atoi(argv[2]);
68                          return TCL_OK;
69                  }
70                  // set the time the coordinator has to wait for votes
71                  else if (strcmp(argv[1], "set-CoordVoteTimer") == 0) {
72                          coord_pingTimer = atof(argv[2]);
73                          return TCL_OK;
74                  }
75                  // this is done for all timer used by the coordinator and
76                  // the clients
77                  ...
78                  else if (strcmp(argv[1], "set-client_recoveryTimer") == 0) {
79                          client_recoveryTimer = atof(argv[2]);
80                          return TCL_OK;
81                  }
82          }
83          // If the command hasn't been processed by DisseminationAgent()::command,
84          // call the command() function of the base class
85          return (Agent::command(argc, argv));
86  }
```

Mit Hilfe dieser Schnittstellen können zu bestimmten Zeiten Transaktionen gestartet werden. Damit der Koordinator seine Teilnehmer kennt und diese wiederum den Koordinator, muss das allen zuvor mitgeteilt werden (vgl. Transaktionsmodell in Abschnitt 2.4.4). Den Teilnehmern wird die Information über ihren Transaktionskoordinator durch den Befehl *set-coordinator* (vgl. Zeilen 19-22) mitgeteilt. Damit sie die notwendige Adresse aus dem Argument erhalten, ist ein Casting des Arguments notwendig (vgl. Zeile 21). Es wird ein NS2-Agent erzeugt, der u.a. eine Adresse beinhaltet, mit der an den Koordinator eine Nachricht geschickt werden kann. Ebenso wird mit *add-member* (vgl. Zeilen 24-28) verfahren, um den Koordinator die Informationen über einen Transaktionsteilnehmer zu übergeben.

Wird dann zum Start der Commit-Phase beispielsweise für einen Teilnehmer in Tcl die Schnittstelle *send* (vgl. Zeilen 5-9) aufgerufen, so wird dieser sein Votum an den Koordinator schicken. Analog dazu muss beim Koordinator in Tcl an diesem Zeitpunkt die Schnittstelle *be-coordinator* (vgl. Zeilen 11-15) aufgerufen werden. Ab dann erwartet er das Votum der Teilnehmer und wird darauf antworten.

Auf dieselbe Art und Weise können alle benötigten Parameter an die Agenten für die Simulation übergeben werden, z.B. die Transaktion-ID, das $p_{retrieval_{app}}$, der garantierte Recoveryzeitraum t_m, die Netzdichte und die sowohl von Koordinator wie Teilnehmer benötigten Timer.

Für den SLSAgenten ist ebenso die Definition einer entsprechen Schnittstelle notwendig, aus der ein Ausschnitt im Folgenden dargestellt ist:

Listing 5.2: Code-Ausschnitt aus der command()-Methode des SLSAgenten

```
1  int SlsAgent::command(int argc, const char*const* argv) {
2          // two arguments in the tcl-script
3          if (argc == 2) {
4                  if(strcasecmp(argv[1], "fail") == 0){
5                          // calls the method to let all transaction
6                          // agents fail
7                          fails();
8                          return TCL_OK;
9                  } if(strcasecmp(argv[1], "recover") == 0){
10                         // calls the methods to let all transaction
11                         // agents recover
12                         recovers();
13                         return TCL_OK;
14                 }
15         // if there are three arguments in the tcl-script
16         } else if (argc == 3) {
17                 // add a dissemination agent to this sls agent
18                 if (strcmp(argv[1], "add-agent") == 0) {
19                         // save this information in a list
20                         disseminationAgents_->insert(disseminationAgents_->end(),(
                               DisseminationAgent*) (TclObject::lookup(argv[2])));
21                         return TCL_OK;
22                 // add the node information to this sls agent
23                 // necessary to let the node stop routing later
24                 } else if (strcmp(argv[1], "add-node") == 0) {
25                         node_ = (Node*) (TclObject::lookup(argv[2]));
26                         return TCL_OK;
27                 // set the p_retrieval of the application
28                 } else if (strcmp(argv[1], "set-pretrieval") == 0) {
29                         pretrieval_ = atof(argv[2]);
30                         return TCL_OK;
31                 }
32                 // set contextawareness of this node
33                 else if (strcmp(argv[1], "set-contextAwareness") == 0) {
34                         contextAware = atoi(argv[2]);
35                         return TCL_OK;
36                 }
37                 // set the technical failure rate of this node
38                 else if (strcmp(argv[1], "set-techFailureRate") == 0) {
39                         techFailureRate = atof(argv[2]);
40                         return TCL_OK;
41                 }
42                 // set the energy failure rate of this node
43                 else if (strcmp(argv[1], "set-energyFailureRate") == 0) {
44                         energyFailureRate = atof(argv[2]);
45                         return TCL_OK;
46                 }
47         }
48         // If the command hasn't been processed by DisseminationAgent()::command,
49         // call the command() function of the base class
50         return (Agent::command(argc, argv));
51 }
```

Wichtig bei den Schnittstellen des SLSAgenten ist zunächst die Verbindung zu Knoten und Transaktionsagenten. Erstere wird mit Hilfe des Befehls *add-node* (vgl. Zeilen 24-26) hergestellt,

wobei die Information über den Knoten auch hier als Tcl-Objekt übergeben wird. Die Information über die Transaktionsagenten erhält der SLSAgent durch die Schnittstelle *add-agent* (vgl. Zeilen 18-21). Alle an ihn gebundenen Transaktionsagenten werden in einer Liste verwaltet, damit eine Benachrichtigung über Ausfälle und Recoverys möglich ist.

Auch die Steuerung der Ausfälle erfolgt über Tcl und erfordert deshalb entsprechende Schnittstellen. Durch den Befehl *fail* (vgl. Zeilen 4-8) kann in Tcl zu einem gewünschten Zeitpunkt ein Ausfall erzeugt werden. Die Benachrichtigung des eigentlichen Knotenobjekts sowie der Transaktionsagenten wird dann durch das Protokoll übernommen. Dasselbe geschieht bei einer Recovery, die über den Befehl *recover* (vgl. Zeilen 9-13) erzeugt wird.

Zu dem Aufbau der Tcl-Skripten, in denen diese Schnittstellen verwendet werden, wird in Abschnitt 5.2.4.4 ein Beispiel vorgestellt.

5.2.4 Generierung der Transaktions-Szenarien als Tcl-Skripten für NS2

Zur Generierung der Simulationsskripten mit den Transaktionen für NS2 sind verschiedene Teilschritte notwendig. Zunächst müssen aus den MANET-Szenarien-Dateien Simulations-Control-Dateien erzeugt werden. Mit Hilfe von den Control-Dateien ist es dann möglich, die eigentlichen Simulationsskripten zu erzeugen. Nach Darstellung dieses Verfahrens wird auch in diesem Abschnitt auf die verwendeten Parameter eingegangen. Den Abschluss bilden zwei Beispiele. Zum einen wird ein Code-Fragment aus einem Tcl-Skript vorgestellt, welches die Verwendung der Schnittstellen zu NS2 und die Übergabe der Parameter demonstriert. Zum anderen werden die Berechnungen vorgestellt, die der Koordinator des SLS-Prozessablaufs auf Grundlage der ausgewählten Parameter in den Simulationen durchführt.

5.2.4.1 Erzeugung von Simulations-Control-Dateien

Aus den Szenarien können unter Verwendung des im Rahmen des CoCoDa-Projektes (vgl. [21]) entstandenen Programms *NS22Mctl* sog. *Simulations-Control-Dateien* erzeugt werden. Diese beinhalten für definierte Zeitpunkte im Simulationszeitraum jeweils die Verbindungsqualität zwischen allen Knoten. Als Programmparameter werden die zugrunde liegende Szenarien-Datei, der Name der Zieldatei, die Simulationsdauer und der Offset für den ersten Knoten in der Control-Datei benötigt.

Die Tabelle in Abbildung 5.3 zeigt beispielhaft einen Ausschnitt aus einer solchen Control-Datei. Es werden für die jeweiligen Zeitpunkte jeweils von einem Knoten A zu allen anderen Knoten (in der Tabelle Knoten B) die Verbindungen überprüft und die Verbindungsqualität dazu angegeben.

5.2.4.2 Erzeugung der Tcl-Skripten

Die Generierung von Tcl-Skripten für die NS2-Simulationen erfolgt mit einem für diese Arbeit entwickelten Java-Programm, dem sog. *Tcl-Generator* (vgl. das Paket *tclGenerator* auf beiliegender CD). Dabei sind drei wesentliche Teile zu nennen: die Übergabe von Parametern, mit denen simuliert werden soll, die Auswahl von Transaktionsteilnehmern sowie die Erzeugung von Ausfällen. Mit diesen Informationen bzw. Hilfsmitteln können Tcl-Skripte in großer Anzahl für die ausgewählten Szenarien generiert werden.

Übergabe von Parametern

Mit Hilfe einer Java-Properties-Datei werden dem Generator für die Tcl-Skripten die notwendigen Informationen zur Verfügung gestellt. Durch deren Verwendung können eine Vielzahl von

Zeitpunkt	Knoten A	Knoten B	Verbindungsqualität (%)
...	...	...	...
500.0	2	1	100
500.0	2	2	100
500.0	2	3	0
500.0	2	4	0
500.0	2	5	100
500.0	2	6	0
500.0	2	7	100
500.0	2	8	0
500.0	2	9	0
500.0	2	10	100
...	...	...	...
500.0	7	1	100
...	...	...	...

Abbildung 5.3: Ausschnitt aus einer Simulations-Control-Datei

Parametern übergeben werden. So können beispielsweise die Pfade zu den Ordnern spezifiziert werden, in dem die NS2-Szenarien liegen. Zusätzlich ist die Angabe verschiedener Parameter zur Variation der Simulationen möglich, z.B. die gewünschte Zuverlässigkeit der Anwendung $p_{retrieval_{app}}$ und der garantierte Recoveryzeitraum t_m, die verwendet werden sollen. Aber auch die Ausfallraten für die mobilen Knoten, die Timer für die einzelnen Aufenthaltsdauern in den Zuständen und die Anzahl der Teilnehmer können hier angegeben werden.

Die übergebenen Parameter werden in Arrays gespeichert und mit Hilfe von geschachtelten for-Schleifen werden die Parameter variiert, d.h. für jede Kombination wird ein entsprechendes Tcl-Skript erzeugt. Auf alle Parameter und ihre für diese Simulationen ausgewählten Werte wird im Unterabschnitt 5.2.4.3 näher eingegangen.

Auswahl von Transaktionsteilnehmern

In einem NS2-Szenario sollen (wenn möglich) mehrere hundert Transaktionen gestartet werden. Dazu müssen in den Szenarien jeweils Gruppen von mobilen Knoten gefunden werden, die zumindest zu Beginn der Transaktion die Möglichkeit zum Nachrichtenaustausch haben. Die Initiierung einer Transaktion zu diesem Zeitpunkt erfolgt dann im Tcl-Skript. Da die in NS2 implementierten Agenten nach dem Start der Transaktion selbstständig entsprechend dem implementierten Prozessablauf agieren, ist lediglich der Startzeitpunkt und die entsprechende Gruppe von teilnehmenden Knoten notwendig. Diese Auswahl muss vor bzw. während der Generierung der Tcl-Skripten vorgenommen werden, damit die Initiierung in die Skripten aufgenommen werden kann. Fehler nach dem Start der Transaktion sind Untersuchungsgegenstand und sollen entsprechend dem SLS-Prozessablauf behandelt werden.

Wie bereits im vorigen Abschnitt geschildert, sind für die Tcl-Generierung Szenarien und Control-Dateien notwendig. Mit Hilfe des ebenfalls im Rahmen des CoCoDa-Projektes (vgl. [21]) entstandenen Tools *Mctl2Tir* werden aus den Control-Dateien Gruppen von Knoten für definierte Zeitpunkte bzw. Zeitabstände extrahiert. Die Abstände ergeben sich dabei aus der Anzahl von gewünschten Transaktionen, die in der Simulation gestartet werden sollen. Entsprechend dieser Anzahl wird die Simulationszeit in Zeiträume unterteilt und zu jedem Zeitpunkt wird eine Knotengruppe ermittelt (wenn z.B. die Netzdichte dies zulässt). Zwischen den Knoten einer Gruppe

muss mit einer ebenfalls definierten Verbindungsqualität eine Verbindungswahrscheinlichkeit bestehen. Diese Knotengruppe entspricht den Teilnehmern der Transaktion, wobei der erste Knoten der Koordinator ist. Mit den Informationen über den Zeitpunkt der Verbindungsmöglichkeit und über die Knoten selbst kann dann im Tcl-Skript die Initiierung der Transaktion erfolgen.

Erzeugung von Ausfällen

Im Rahmen der Simulationen dieser Arbeit sollen die Ausfälle von Knoten aufgrund von Energie und technischem Defekt der Exponentialverteilung zugrunde liegen. Dazu ist es notwendig, eine allgemeine Ausfallrate λ_{fail} und eine Rückkehrrate $\mu_{recover}$ im Vorfeld zu spezifizieren. Dies geschieht wiederum über die Properties-Datei. Zusammen mit der Information über die Dauer der Simulationszeit können dann mehrfache Ausfälle der Knoten generiert werden. Dazu wird als Grundlage folgende Formel verwendet, um einen exponentialverteilten zufälligen Ausfallzeitpunkt z_{exp} zu generieren:

$$z_{exp} = (-1) \cdot \frac{1}{\lambda_{fail}} \cdot log(z_{uniform}) \tag{5.1}$$

wobei $z_{uniform}$ einer gleichverteilten Zufallszahl und $\frac{1}{\lambda_{fail}}$ dem Erwartungswert für den Ausfall entspricht. Für den Rückkehrzeitpunkt nach einem Ausfall wird mit $\mu_{recover}$ ebenso verfahren.

Auf diese Art können mehrere Ausfälle durch abwechselnde Berechnung eines Ausfallzeitpunktes und eines Rückkehrzeitpunktes entwickelt werden. Die Zeiten werden jeweils so lange aufeinander addiert bis die Simulationsdauer erreicht ist. Mit diesem Verfahren können für alle Knoten, basierend auf derselben Ausfall- bzw. Rückkehrrate, individuelle Knotenausfälle generiert werden. Mit den in Abschnitt 5.2.3.4 beschriebenen Schnittstellen in Tcl wird das gewünschte Ausfallverhalten der Knoten bei der Generierung der Tcl-Skripten erzeugt werden.

Dabei ist jedoch zu bemerken, dass ein Knoten ggf. zu Zeitpunkten ausgefallen ist, wo Transaktionen gestartet werden, an der der Knoten teilnehmen sollte. Ist es ein Teilnehmer, startet die Transaktion ggf. ohne den Knoten, wenn genug andere Teilnehmer zur Verfügung stehen. Sollte es sich um einen Koordinator handeln, so scheitert die ganze Transaktion. Dies kann jedoch im Vorfeld der Simulationen nicht untersucht werden. Es wird dadurch ein realistisches Verhalten der Knoten im MANET erzeugt.

5.2.4.3 Verwendete Parameter für die Tcl-Skripten der SLS-Evaluation

Für die Generierung der Tcl-Skripten sind eine Vielzahl von Parametern notwendig, auf die im Folgenden näher eingegangen wird. Dabei ist zu berücksichtigen, dass zwischen einigen der Parameter Korrelationen möglich sind.

Gewünschte Zuverlässigkeit der Anwendung $p_{retrieval_{app}}$

Im Rahmen dieser Simulationen wird die gewünschte Wahrscheinlichkeit für das Wiederfinden des Transaktionslogs durch einen ausgefallenen Knoten $p_{retrieval_{app}}$ auf 70% gesetzt. Bei der Analyse dieses Wertes im Zusammenhang mit den im Folgenden vorgestellten Parametern Ausfallrate und garantiertem Recoveryzeit t_m hat sich gezeigt, dass eine Erhöhung dieses Wertes auch zu einer Erhöhung des Knotenbedarfs bei der Verteilung führen würde. Insbesondere in dünnen Netzen mit wenig Knoten ist dies jedoch nur bedingt möglich. Wäre der Bedarf von Knoten für die Verteilung größer als die Anzahl von Knoten im Netz, kann die gewünschte Zuverlässigkeit $p_{retrieval_{app}}$ keinesfalls erreicht werden. Auf die Analyse des Zusammenspiels der drei genannten Parameter wird im Folgenden noch einmal eingegangen (vgl. dazu Abbildung 5.5).

Simulierte Ausfälle und verwendete Ausfallraten

Für die Simulation werden zusätzlich zu den Verbindungsabbrüchen aufgrund von Mobilität auch *echte* Knotenausfälle simuliert. Auf die Form der berücksichtigten Ausfälle wird im Folgenden als erstes eingegangen. Darauf aufbauend werden die zu den Knotenausfällen gehörigen Ausfallraten hergeleitet.

Ausfälle in den Simulationen Für die Simulationen werden zunächst Tcl-Skripten ohne Knotenausfälle simuliert. Dadurch treten bei diesen Szenarien ausschließlich Kommunikationsfehler aufgrund von kurzfristiger Mobilität auf.

In einem zweiten Schritt werden zusätzlich Tcl-Skripten mit Knotenausfällen aufgrund von technischem Defekt und erschöpfter Energieressourcen simuliert. Entsprechend treten bei diesen Simulationen sowohl Knoten- wie Kommunikationsfehler auf. Die Dauer dieser Knotenausfälle ist temporär. Sie werden auf Basis eines gemeinsamen Erwartungswertes für die Knoten erzeugt. So weisen alle Knoten trotz gleichem Erwartungswert individuelle Ausfall- und Rückkehrzeiten auf.

Ausfallraten bei den Simulationen Wie bereits im vorigen Abschnitt und in Kapitel 4.2.1 beschrieben, berücksichtigt der SLS zum einen die Wahrscheinlichkeit für einen technischen Ausfall (charakterisiert durch λ_{tech}) sowie Ausfälle aufgrund verbrauchter Energieressourcen (charakterisiert durch λ_{energy}). Für die Simulation des SLS müssen für beide Parameter sinnvolle Werte ausgewählt werden, so dass die Ausfälle in der gewünschten Häufigkeit auftreten.

Im Hinblick auf energiebedingte Ausfälle wird angenommen, dass ein mobiles Gerät wie z.B. ein PDA ca. eine Stunde mit einem MANET verbunden sein kann, bevor dessen Akkus wieder aufgeladen werden müssen. Auch für die Aufladezeit wird hier angenommen, dass etwa eine Stunde notwendig ist, bis die Geräte wieder über ausreichend Energie für ein Wiederverbinden mit dem MANET verfügen. Da der erwartete Wert von einer Stunde in beiden Fällen dem Erwartungswert (vgl. Kapitel 4) entspricht, ergibt sich daher $\lambda_{energyFailure} = 0.0002778$ für einen energiebedingten Ausfall und $\lambda_{energyReturn} = 0.0002778$ für die Rückkehr nach dem Aufladen des Akkus.

Ähnliche Überlegungen können für die technische Ausfallrate angestellt werden. Die Erwartung ist, dass ein mobiles Gerät ca. 278 Stunden permanent ohne einen technischen Defekt, der zum Versagen seiner Funktionalität führt und eine Reparatur erforderlich macht, betriebsfähig ist. Mit Hilfe dieses geschätzten Erwartungswertes kann man die zugehörige Ausfallrate bestimmen, die sich auf ca. $\lambda_{tech} = 0.000001$ beläuft.

Auswirkung der Ausfälle auf die Knotenanzahl im MANET Aufgrund der Ausfälle stehen im Verlauf der Simulation nicht immer alle Knoten im MANET zur Verfügung. Ein Teil der Knoten wird mit großer Wahrscheinlichkeit immer ausgefallen sein. Die *echte* Anzahl von Knoten wird also nicht zur Verfügung stehen, so dass man anhand der oben spezifierten Ausfälle eine *erwartete* Anzahl von nicht ausgefallenen Knoten im MANET zu jedem Simulationszeitpunkt schätzen muss. Für die Schätzung der erwarteten Anzahl von Knoten können Markovketten verwendet werden. Details zur Berechnung sind im Anhang zu A.1 finden.

Als Beispiel zeigt Abbildung 5.4 für ein Szenario mit 10 Knoten, wie groß die Wahrscheinlichkeit für jede der Knotenanzahlen $1, \ldots, 10$ ist. Die Wahrscheinlichkeiten $P(i)$ entsprechen der Wahrscheinlichkeit dafür, dass i Geräte ausgefallen und somit $n - i$ Knoten im Netz zur Verfügung stehen. Bei dem Beispiel ist erkennbar, dass im Verlauf der Zeit die Wahrscheinlichkeit für den Zustand $P(5)$ am größten ist. Man kann daher davon ausgehen, dass im Durchschnitt etwa 5 Knoten ausgefallen sind und 5 Knoten zur Verfügung stehen. Im Rahmen dieser Arbeit

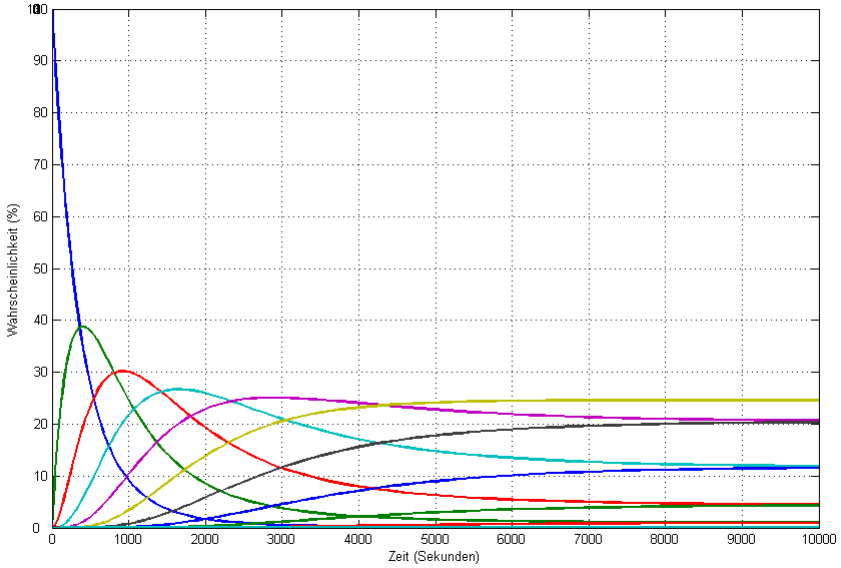

Abbildung 5.4: Wahrscheinlichkeiten *P(i)* für den Ausfall von i der n Knoten des MANETs am Beispiel von $n = 10$ Knoten

wird statt der *echten* Knotenanzahl im MANET die sog. *erwartete* Knotenanzahl angegeben. Bei den Diagrammen der Evaluationen wird ausschließlich die erwartete Knotenanzahl verwendet, da diese ein genaueres Bild des der Simulation zugrunde liegenden Netzes darstellt. Die jeweils erwartete Knotenanzahl für die verwendeten Szenarien ist ebenfalls in Tabelle 5.1 zu finden.

Garantierter Recoveryzeitraum

Bei der Simulation des SLS wird ein garantierter Recoveryzeitraum von $t_m = 3600$ Sekunden gewählt. Dieser Wert entspricht der erwarteten Ausfallzeit von einer Stunde aufgrund von Energie. Somit soll es möglich sein, dass ein Knoten nach einem energiebedingten Ausfall das von ihm benötigte Transaktionslog innerhalb von t_m findet. Dabei ist jedoch darauf hinzuweisen, dass bei der Generierung von Knotenausfällen mit $\lambda_{energy} = 0.0002778$ ein Erwartungswert verwendet wird. Somit werden sowohl Knoten vor wie nach Ablauf des garantierten Recoveryzeitraums zurückkehren (mit Varianz $\frac{1}{\lambda^2}$).

Eine Analyse der Korrelation der Werte $p_{retrieval}$, Ausfallraten und garantierter Recoveryzeitraum ist in Abbildung 5.5 dargestellt (Diese Analyse wird im Vorfeld mit Matlab [1] durchgeführt). Es hat sich gezeigt, dass mit diesen ausgewählten Parametern bei der ungerichteten Verteilung die maximale Anzahl von Knoten im Netz nicht überschritten wird. Da Netze mit 10 Knoten simuliert werden, in denen Knoten komplett ausfallen, darf also die Anzahl benötigter Knoten nicht größer als 10 werden. Bei der Wahl von bis zu 7 Teilnehmern an einer Transaktion (vgl. den übernächsten Abschnitt) muss diese Zahl so klein werden, dass die Möglichkeit der Verteilung von Transaktionslogs trotz Knotenausfällen erhalten bleibt. Dies ist für die Garantie von $p_{retrieval_{app}}$ ausschlaggebend. Würden mehr Knoten benötigt werden durch die Verteilungsstrategie, kann die gewünschte Zuverlässigkeit nicht mehr erreicht werden. Durch die erwartete Anzahl von Knoten befinden sich ggf. weniger Knoten im Netz als benötigt werden. Die Folgen

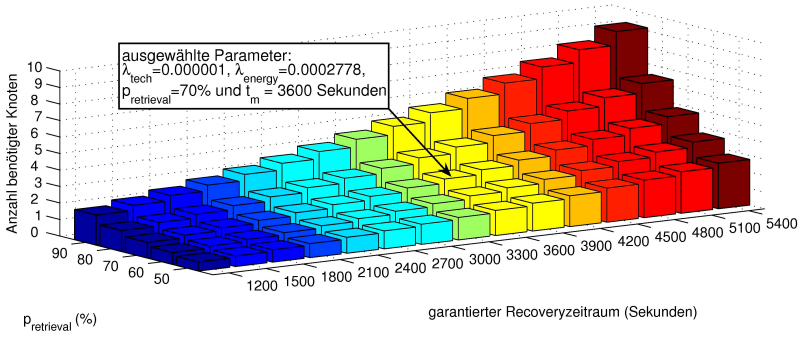

Abbildung 5.5: Untersuchung des Zusammenwirkens von Ausfallraten, $p_{retrieval}$ und garantiertem Recoveryzeitraum hinsichtlich der Anzahl von benötigten Knoten bei der Verteilung

davon müssen im Verlauf der Simulation evaluiert werden.

Timer

Die bereits in Abschnitt 4.3 bei der Darstellung der detaillierten Workflows angesprochenen Timer sind für den Ablauf der Simulationen von Bedeutung. Ihre Auswahl hängt im Wesentlichen von der Nachrichtenverzögerung (engl. *message delay*) ab, d.h. der Zeit, die ein Paket vom Sender zum Empfänger benötigt. Diese Nachrichtenverzögerung wird u.a. durch die Netzdichte und das verwendete Routing-Protokoll beeinflusst. Dadurch sind die Timer vom zugrunde liegenden Netz abhängig.

Die für die Evaluation verwendeten Timerwerte werden durch Stichproben ermittelt. Dabei wird bei sehr kleinen Werten begonnen und diese werden schrittweise erhöht, bis ein reibungsloser Ablauf bei der Kommunikation möglich ist.

Timer des SLS-Prozessablaufs Für den SLS-Workflow müssen die Timer aufeinander abgestimmt werden. Eine Übersicht der in den Simulationen verwendeten Timer ist in Tabelle 5.3 dargestellt. Betrachtet man dazu noch einmal den SLS-Prozessablauf in den Abschnitten 4.3.2 und 4.3.3, so stellt man z.B. fest, dass die Timer des Klienten nach dem Senden des Votums so groß sein müssen, dass der Koordinator seine Nachbarn nach ihren Ausfallwahrscheinlichkeiten befragen und dann eine Entscheidung treffen kann. Außerdem muss bedacht werden, dass die Transaktionsteilnehmer sich in Multihop-Entfernung zum Koordinator befinden können, so dass Nachrichten ggf. länger brauchen als ein Singlehop-Broadcast. Die Timer für die Kommunikation des Koordinators und der Teilnehmer werden daher entsprechend größer gewählt.

Weiterhin beeinflussen insbesondere die mit der Verteilung des Koordinators bzw. der Recovery der Knoten verbundenen Timer die benötigte Paketanzahl sehr stark. Ist z.B. der Timer für das Wiederholen eines Recovery-Versuches sehr klein gewählt, so schicken die Klienten innerhalb von kurzen Abständen sehr viele Nachrichten. Kann das Transaktionslog dann nicht innerhalb von kurzer Zeit gefunden werden, wird dadurch das MANET mit einer erheblichen Paketanzahl belastet. Dasselbe Problem stellt sich beim Koordinator. Wird der Timer für die Verteilung des Transaktionslogs (das ist der Zeitraum für das Warten auf die notwendige Anzahl von Acks zur Bestätigung der erfolgreichen Verteilung) zu klein gewählt, so sendet der Koordinator weiterhin regelmäßig Pakete, bis endlich die geforderte Anzahl erreicht wird. Dieser Timer beeinflusst also sehr stark die Belastung des MANETs mit Nachrichten. Wird dieser Timer allerdings zu groß

Timer	Wert (in Sekunden)
coord_pingtTimer	0.2
coord_failureTimer	0.1
coord_ackDissTimer	0.2
coord_pongAckTimer	0.2
coord_distPongAckTimer	0.2
client_distDissTimer	0.6
client_pongTimer	0.8
client_recoveryTimer	2.0
client_waitForRecIdPong	2.0
client_waitForDirRecover	5.0
client_waitForIndirRecover	5.0
sls_pushTimer	60.0

Tabelle 5.3: Timer des SLS-Workflows

Timer	Wert (in Sekunden)
coord_pingTimer	0.2
coord_pongAckTimer	0.2
client_pingAckTimer	0.6
client_pongTimer	0.8
client_recoverTimer	2.0

Tabelle 5.4: Timer des Vergleichsworkflows

gewählt, so ist möglicherweise zum Zeitpunkt der Recovery eines Teilnehmers das Transaktionslog noch nicht ausreichend im MANET verteilt worden.

Timer des Vergleichs-Prozessablaufs Die Timer des Vergleichs-Prozessablaufs sind in Tabelle 5.4 abgebildet. Diese sind weniger umfangreich, da auch der Prozessablauf an sich nicht so komplex ist (vgl. Abschnitt 4.3.4). Um einen möglichst guten Vergleich bei den Simulationen zu erhalten, werden die Timer dieses Prozessablaufs genauso gewählt wie die des SLS-Workflows.

Weitere Parameter

Neben den oben bereits ausführlich beschriebenen Parametern sind noch einige weitere zu nennen (vgl. Tabelle 5.5). Die Anzahl der Teilnehmer an einer Transaktion soll zwischen 1 und 8 (inklusive Koordinator) variiert werden. Die Transaktionsdauer, also der Zeitraum, in dem der Koordinator für die Berechnung der Commit-Entscheidung simulieren soll, beträgt eine Sekunde.

Als maximale Anzahl von Transaktionen in der Simulation wird 500 gewählt. In den Simulationsszenarien mit nur 10 oder 20 Knoten und einer Simulationszeit von 500000 Sekunden sollte alle 1000 Sekunden eine Transaktionsgruppe ermittelt werden, in Szenarien mit einer Laufzeit von 50000 Sekunden etwa alle 100 Sekunden.

Das Kontextbewusstsein der Knoten im MANET beträgt entweder 0% oder 100%. Sind in einem dichten MANET nur Knoten ohne Kontextbewusstsein im Netz vorhanden, so muss der Koordinator in jedem Fall ungerichtet verteilen. Wenn hingegen alle Knoten über Kontextbewusstsein verfügen, so hängt die Verteilungsstrategie von der aktuellen Situation im Netz ab.

Parameter	Wert
Anzahl der Teilnehmer	1, 2, 4, 6, 8
Transaktionsdauer des Koordinators	1 Sekunde
Anzahl der gewünschten Transaktionen	max. 500
Prozentsatz von Knoten mit Kontextbewusstsein	0 %, 100%
Dichte des MANETs	dicht, dünn
Maximale Verlustrate	10%

Tabelle 5.5: Weitere notwendige Parameter für die Tcl-Skripten

Hat der Koordinator ausreichend ausfallsichere Nachbarn, wird die gerichtete Verteilung verwendet, anderenfalls die ungerichtete. Für dünne MANETs spielt das Kontextbewusstsein hingegen keine Rolle.

Auf die Dichte des MANETs ist bereits kurz eingegangen worden. Dabei wird eine Variable gesetzt, anhand der der Koordinator erkennt, welche Netzdichte er in der Simulation annehmen soll.

Die maximale Verlustrate ist bei der Erzeugung der Transaktionsteilnehmer mit Hilfe des Tools *Mctl2Tir* wichtig. Anhand dieses Wertes wird entschieden, ab wann ein Knoten nicht mehr in Frage kommt bei der Transaktionsinitiierung, weil die Verbindungsqualität zu den anderen Knoten zu schlecht ist.

5.2.4.4 Beispiel 1: Ausschnitt aus einem Tcl-Skript

Nachdem die vorherigen Abschnitte die Schnittstellen von Tcl zu NS2 beschreiben sowie auf die Generierung von Tcl-Skripten eingehen, soll an dieser Stelle ein Ausschnitt aus einem solchen Skript vorgestellt werden. Dieses beinhaltet die wichtigsten Charakteristiken für die Evaluation im Hinblick auf die Prozessabläufe und die ausgewählten Parameter:

Listing 5.3: Code-Ausschnitt aus einem Tcl-Skript

```
1  ... ;# Define options
2
3  set val(prop)           Propagation/TwoRayGround          ;# radio-propagation model
4  set val(nn)             10                                ;# number of mobilenodes
5  set val(rp)             AODV                              ;# routing protocol
6  set val(lenx)           500.0                             ;# heigth of sim-area
7  set val(leny)           500.0                             ;# length of sim-area
8  set val(stop)           500000                            ;# finishing time
9  set val(path)           ../logs                           ;# path for the logfiles
10 set val(sc)             ../scen-500x500-10-2-2_10-500000  ;# corresponding scenario
11
12 ... ;# some more options
13
14 # set the propagation range to 100 m
15 Phy/WirelessPhy set RXThresh_ 1.42681e-08
16 Phy/WirelessPhy set freq_ 9.14e+08
17 Phy/WirelessPhy set Pt_ 0.281838
18
19 ... ;# initiation of the main program
20
21 # create number of mobile nodes specified above [$val(nn)]
22 for {set i 0} {$i < $val(nn) } {incr i} {
23         set node_($i) [$ns_ node]
```

```
24          ...
25  }
26
27  # load the scenario file
28  source $val(sc)
29
30  # initiate the simulation scenario: generate slsAgents and connect them to nodes
31  for {set i 0} {$i < ($val(nn)) } {incr i} {
32          $node_($i) start
33          set slsAgent_($i) [new Agent/Sls]
34          $ns_ attach-agent $node_($i) $slsAgent_($i)
35          $slsAgent_($i) add-node $node_($i)
36  }
37
38  ... ;# start giving information to the slsAgents
39
40  # given information about failure rate and context awareness to the slsAgents
41  $slsAgent_(4) set-energyFailureRate 0.0002778
42  $slsAgent_(4) set-energyRecoverRate 0.0002778
43  $slsAgent_(4) set-techFailureRate 0.000001
44  $slsAgent_(4) set-contextAwareness 0
45
46  $ns_ at 1967.5565956647476 "$slsAgent_(4) fail"
47  $ns_ at 7684.900721324137 "$slsAgent_(4) recover"
48
49  ... ;# more failures during the simulation time
50
51  $ns_ at 488926.4982093698 "$slsAgent_(4) recover"
52  $ns_ at 490762.2261582191 "$slsAgent_(4) fail"
53
54  ... ;# do the same for the rest of the slsAgents
55
56  # generate disseminationAgents and attach to the SLSAgents
57  for {set i 4} {$i <= 8 } {incr i} {
58          # create the an agent
59          set p_($i) [new Agent/Dissemination]
60          # give this agent the parameter information needed for the simulation
61          $p_($i) set-pretrieval 0.7
62          $p_($i) set-taId 993
63          $p_($i) set-globalTechFailureRate 0.000001
64          $p_($i) set-globalEnergyFailureRate 0.0002778
65          $p_($i) set-globalEnergyRecoverRate 0.0002778
66          $p_($i) set-nodeNumber 10
67          $p_($i) set-Density 1
68          $p_($i) set-missionTime 3600.0
69          $p_($i) set-coord_waitSomeTimeDuration 1
70          $p_($i) set-coord_pingTimer 0.2
71          $p_($i) set-coord_failureTimer 0.1
72          $p_($i) set-coord_ackDissTimer 0.2
73          $p_($i) set-coord_pongAckTimer 0.2
74          $p_($i) set-coord_distPongAckTimer 0.2
75          $p_($i) set-client_distDissTimer 0.6
76          $p_($i) set-client_pongTimer 1.8
77          $p_($i) set-client_recoveryTimer 2.0
78          $p_($i) set-client_waitForRecIdPongTimer 2.0
79          $p_($i) set-client_waitForDirRecoverTimeout 5.0
80          $p_($i) set-client_waitForIndirRecoverTimeout 5.0
81          $p_($i) set-client_coord_processingTimer 10.0
82  }
83
```

```
84 # attach the disseminationAgents to the slsAgents and contrariwise
85 # and attach the disseminationAgents to the nodes
86 $slsAgent_(5) add-agent $p_(4)
87 $ns_ attach-agent $node_(5) $p_(4)
88 $slsAgent_(8) add-agent $p_(5)
89 $ns_ attach-agent $node_(8) $p_(5)
90 $slsAgent_(4) add-agent $p_(6)
91 $ns_ attach-agent $node_(4) $p_(6)
92 $slsAgent_(1) add-agent $p_(7)
93 $ns_ attach-agent $node_(1) $p_(7)
94 $slsAgent_(7) add-agent $p_(8)
95 $ns_ attach-agent $node_(7) $p_(8)
96
97 # inform the particpants about the coordinator and contrariwise
98 $p_(5) set-coordinator $p_(4)
99 $p_(4) add-member $p_(5)
100 $p_(6) set-coordinator $p_(4)
101 $p_(4) add-member $p_(6)
102 $p_(7) set-coordinator $p_(4)
103 $p_(4) add-member $p_(7)
104 $p_(8) set-coordinator $p_(4)
105 $p_(4) add-member $p_(8)
106
107 # tell the coordinator when he has to start waiting for votes
108 $ns_ at 1718.0 "$p_(4) be-coordinator"
109
110 # tell the participants when they have to start sending their vote
111 $ns_ at 1718.0 "$p_(5) send"
112 $ns_ at 1718.0 "$p_(6) send"
113 $ns_ at 1718.0 "$p_(7) send"
114 $ns_ at 1718.0 "$p_(8) send"
115
116 ... ;# do the same for the other transactions and handle the end of the simulation
```

Zunächst ist das Setzen einiger allgemeiner Optionen notwendig. Dies umfasst z.B. die Simulationsfläche, die Simulationsdauer, das Routing, den Pfad für die Log-Datei sowie den Pfad zur zugehörigen MANET-Szenario-Datei (vgl. Zeilen 3-10). In Zeile 28 wird NS2 mitgeteilt, wo die spezifizierte Szenario-Datei für die Simulation zu finden ist. Außerdem ist es notwendig, die Übertragungsreichweite der mobilen Knoten zu definieren (vgl. Zeilen 14-17).

Anschließend ist die Erzeugung der mobilen Knoten selbst möglich (vgl. Zeilen 22-25). Darauf aufbauend können die SLS-Agenten ebenfalls erzeugt und an die mobilen Knoten gebunden werden (vgl. Zeilen 31-36). Diesen SLS-Agenten können für die Simulationen die notwendigen Parameter über die implementierten Schnittstellen übergeben werden (vgl. Zeilen 41-44). In diesem Fall sind das z.B. die Ausfallraten und das Kontextbewusstsein. Außerdem können den SLS-Agenten nun die extern, basierend auf der gewünschten Wahrscheinlichkeitsverteilung, erzeugten Ausfälle übergeben werden (vgl. Zeilen 46-52).

In einem nächsten Schritt werden für die ermittelten Zeitpunkte in der Simulation die Transaktionsagenten (bei der Implementierung als DisseminationAgent bezeichnet) erzeugt. Über die implementierten Schnittstellen erhalten diese die benötigten Parameter (vgl. Zeilen 57-82). Anschließend müssen diese Agenten noch an den mobilen Knoten und den SLS-Agenten gebunden werden (vgl. Zeilen 86-95).

Nun können die einzelnen Transaktionsteilnehmer übereinander informiert werden, d.h. dem Koordinator werden die Teilnehmer mitgeteilt und umgekehrt (vgl. Zeilen 98-105). Anschließend kann dem Koordinator mitgeteilt werden, ab wann er auf Voten der Teilnehmer warten soll (vgl. Zeile 108). Ebenso wird den Teilnehmern der Zeitpunkt mitgeteilt, wann sie ihr Votum schicken

sollen (vgl. Zeilen 111-124). Bei Ausführung des Skriptes durch NS2 werden dann zu den im Skript definierten Zeitpunkten die jeweiligen Aktionen durchgeführt und damit die Transaktionen gestartet. Die Koordinatoren und Teilnehmer verhalten sich entsprechend dem implementierten Prozessablauf und agieren nach dem Start selbstständig ohne weitere Eingriffe auf den Prozess aus einem Tcl-Skript heraus.

5.2.4.5 Beispiel 2: Berechnungen in den Simulationen mit den gewählten Parametern

Im Folgenden werden die Berechnungen vorgestellt, die die Knoten in den Simulationen mit den ausgewählten Parametern durchführen sollen. Diese Vorüberlegungen sind sinnvoll, um zu überprüfen, ob die ausgewählten Parameter zu den Szenarien (z.B. zur Knotenanzahl im MANET) passen.

Berechnung der Ausfallwahrscheinlichkeit eines Knotens beim SLS-Prozessablauf

Um die Anzahl von notwendigen Knoten berechnen zu können, ist zunächst die Bestimmung der Ausfallwahrscheinlichkeit nötig. Diese Berechnung wird entweder von den Knoten mit Kontextbewusstsein vorgenommen oder vom Koordinator. Letzteres ist der Fall, wenn in einem Netz vor allem Knoten ohne Kontextbewusstsein vorhanden sind, so dass global eine Ausfallwahrscheinlichkeit durch den Koordinator geschätzt werden muss.

Die Berechnung der Ausfallwahrscheinlichkeit erfolgt dabei mit den zuvor ausgewählten Ausfallraten $\lambda_{tech} = 0.000001$ und $\lambda_{energy} = 0.0002778$ und dem garantierten Recoveryzeitraum $t_m = 3600$ Sekunden unter Verwendung der Formeln zum Zuverlässigkeitsmodell aus Abschnitt 4.2 wie folgt:

$$\begin{aligned} p_{failure} &= 1 - (1 - p_{techFailure}) \cdot (1 - p_{energyFailure}) \\ &= 1 - (1 - (1 - e^{-\lambda_{tech} \star t_m}) \cdot (1 - (1 - e^{-\lambda_{energy} \star t_m}) \\ &= 1 - (e^{-0.000001 \star 3600}) \cdot (e^{-0.0002778 \star 3600}) \\ &= 0.6335 \end{aligned} \tag{5.2}$$

Da hier angenommen wird, dass die Knoten mit Kontextbewusstsein über die Informationen zu ihren Ausfallraten verfügen, werden sie also ebenso eine individuelle Ausfallwahrscheinlichkeit von 0.6335 berechnen wie der Koordinator global für das ganze Netz (vgl. zur Berechnung auch Abbildung **??**). Bezüglich des Koordinators wird angenommen, dass dieser global über die Kenntnis der Ausfallraten verfügt (z.B. durch Schätzungen). Im Rahmen dieser Arbeit werden die Ausfallwahrscheinlichkeiten dem Koordinator im Vorfeld übergeben.

Gerichtete Verteilung in dichten MANETs Bei der Berechnung der Anzahl von Knoten mit Kontextbewusstsein, die für die Verteilung notwendig sind, wird in Analogie zu Abschnitt 4.2.2.1 vorgegangen. Der Koordinator beginnt bei dem Knoten mit der geringsten Ausfallwahrscheinlichkeit und nimmt schrittweise so lange weitere Knoten hinzu, bis das geforderte $p_{retrieval_{app}}$ erreicht ist. Bei den in den Simulationen verwendeten Parametern $p_{retrieval_{app}} = 70\%$ und der zuvor berechneten Ausfallwahrscheinlichkeit $p_{failure_i} = 0.6335$ jedes Knotens ergibt sich daher für $n = 3$:

$$\begin{aligned} R_{SLS}(3) &= 1 - \prod_{i=1}^{3} p_{failure_i} \\ &= 1 - (p_{failure_1} \cdot p_{failure_1} \cdot p_{failure_3}) \\ &= 1 - (0.6335 \cdot 0.6335 \cdot 0.6335) \\ &= 0.7458 \end{aligned} \tag{5.3}$$

Es werden also bei den Simulationen für die gerichtete Verteilung drei Nachbarknoten des Koordinators mit Kontextbewusstsein benötigt. Stehen dem Koordinator drei Knoten für die Verteilung zur Verfügung, so bedeutet das: ein ausgefallener Teilnehmer kann innerhalb des vereinbarten garantierten Recoveryzeitraums von $t_m = 3600$ Sekunden mit einer Wahrscheinlichkeit von 74,58% den Ausgang der Transaktion bei einem dieser drei Knoten finden.

Ungerichtete Verteilung in dichten MANETs Um die Anzahl von notwendigen Knoten zu schätzen, wenn vorwiegend Knoten ohne Kontextbewusstsein zur Verfügung stehen, berechnet der Koordinator wie oben beschrieben zunächst eine globale Ausfallwahrscheinlichkeit $p_{failure}$. Anschließend kann er dann die folgende Berechnung (vgl. Abschnitt 4.2.2.2) durchführen, um die nötige Knotenanzahl für die geforderte Zuverlässigkeit mit $p_{retrieval_{app}} = 70\%$ zu erreichen:

$$\begin{aligned} 0.7 &= 1 - (p_{failure})^n \\ 1 - 0.7 &\leq (p_{failure})^n \\ \frac{ln(1-0.7)}{ln(p_{failure})} &\leq n \\ 2.6374 &\leq n \end{aligned} \tag{5.4}$$

Der Koordinator muss das Transaktionslog also auf mindestens 3 Knoten verteilen.

Ungerichtete Verteilung in dünnen MANETs In dünnen MANETs wird ausschließlich die ungerichtete Verteilung verwendet. Hier sind daher keine Berechnungen der Teilnehmer erforderlich, sondern Berechnungen werden ausschließlich durch den Koordinator vorgenommen. Auch bei dieser Strategie berechnet der Koordinator wie in Formel 5.2 beschrieben die globale Ausfallwahrscheinlichkeit $p_{failure}$ der Knoten. Anschließend kann wie in Formel 5.3 vorgestellt die Anzahl n von notwendigen Knoten für die Verteilung berechnet werden. Hat der Koordinator beide Werte bestimmt, kann damit der für diese Strategie benötigte Verteilungsgrad wie folgt ermittelt werden:

$$\begin{aligned} d &= \frac{n}{r} \\ &= \frac{2.6}{r} \end{aligned}$$

wobei r die Anzahl von Knoten im Netz ist (im Falle dieser Simulationen 10, 20, 30, 40 und 50). Im Rahmen dieser Arbeit werden dem Koordinator die Informationen über die Gesamtzahl von Knoten im MANET zur Verfügung gestellt. Es wird dabei davon ausgegangen, dass der Koordinator in der Realität diesen Wert selbst schätzen kann.

5.2.5 Simulation des SLS mit NS2

Nachdem NS2 um die benötigten Charakteristiken erweitert worden ist, können die erzeugten Skripten damit simuliert werden. Dazu ist lediglich der Befehl `ns <tclscript>` notwendig, wobei *tclscript* dem zu simulierenden Tcl-Skript entspricht. Zur Simulation mehrerer hundert Skripten, wie das für diese Arbeit der Fall ist, bieten sich zum Aufruf Shell-Skripten an. Diese ermöglichen es, dass alle Tcl-Skripten automatisch nacheinander simuliert werden können.

5.2.6 Parsen der Log-Dateien in eine Datenbank

Um die zum Teil großen Mengen an Log-Dateien sinnvoll analysieren zu können, werden diese mit einem für diesen Zweck entwickelten Java-Programm geparst (vgl. das Paket *logfileParser* auf beiliegender CD) und die darin enthaltenen Ergebniswerte in eine MySQL-Datenbank eingefügt. Damit können anschließend jederzeit mit den Daten weitere Analysen durchgeführt werden.

Welche Log-Dateien geparst werden sollen, wird dabei wieder über die Properties-Datei spezifiziert. Es ist möglich, eine Vielzahl von Ordnern mit Log-Dateien anzugeben, deren Inhalte dann nacheinander in die Datenbank eingefügt werden. Gleichzeitig können Angaben über das Experiment selbst gemacht werden.

Datenbankstruktur

Bei der Entwicklung einer Datenbankstruktur werden nicht nur die Inhalte der Log-Dateien berücksichtigt, sondern auch Informationen über das damit zusammenhängende Experiment. So werden zusätzlich experiment-spezifische Informationen aus den Properties und die verwendeten Parameter in die Datenbank eingefügt. Für die eigentlichen Werte wird eine individuelle Tabelle verwendet, die es ermöglicht, dass im Verlauf der Weiterentwicklung von Untersuchungen neue Werte hinzukommen. Mit dieser Methode können beliebige neue Analysewerte hinzukommen.

Die für das Parsen der Log-Dateien verwendete Datenbank umfasst mit dem folgenden Schema beschriebenen Relationen. Mit dieser Struktur ist anschließend die einfache Analyse der Log-Dateien möglich, die dann ebenfalls mit einem Java-Programm erfolgen kann (Hinweis: Primärschlüssel sind unterstrichen, Sekundärschlüssel kursiv dargestellt).

Experiment: {[ID : Integer, Titel : Varchar(50), Experimentdescription : Text, Editor : Varchar(50)]}

Parameter: {[{ID : Integer, Coord_VoteTimer : Varchar(10), Coord_FailureTimer : Varchar(10) : Coord_AckDissTimer : Varchar(10), Coord_PongAckTimer : Varchar(10), Coord_DistCommitAckTimer: Varchar(10), Client_DistDissTimer : Varchar(10), Client_CommitTimer : Varchar(10), Client_RecoveryTimer : Varchar(10), Client_WaitForRecIdPong : Varchar(10), Client_WaitForDirRecover : Varchar(10), Client_WaitForIndirRecover : Varchar(10), Client_VoteAckTimer : Varchar(10), Client_RecoverTimer : Varchar(10), pretrieval : Varchar(10), FailureProbability : Varchar(10), Movementmodell : Varchar(20), MaxLossRate : Varchar(5), Multihop : Varchar(5), SimAreaHeight : Varchar(10), SimAreaLength : Varchar(10), Maxfailureduration : Varchar(10), Minfailureduration : Varchar(10)]}

Agent: {[Agentnumber : Integer , Agent : Varchar(20), Version : Varchar(10), Creator : Varchar(30)]}

Logfile: {[ID : Integer, *Experiment : Integer*, *Agent : Integer*, *Params : Integer*, Transactions : Integer, Nodenumber : Integer, Participants : Integer, Taduration: Integer, Missiontime : Integer, Density : Integer, Contextawareness :Integer, Simulationduration : Integer, Logfile : Varchar(200)]}

Value: {[ID : Integer, *Experiment : Integer*, *Agent : Integer*, *Params : Integer*, *Logfile : Integer*, Name : Varchar(50), Value : Float]}

Die Experiment-Tabelle enthält die wesentlichsten Informationen über das Experiment selbst. Dazu gehören ein Titel, eine Beschreibung sowie der Eigentümer des Experiments.

Mit der Parameter-Tabelle werden die Parameter zu einem Experiment verwaltet. Dies umfasst vor allem die globalen Parameter, die in der Properties-Datei für alle Skripten des Experiments verwendet werden. So werden z.B. die Werte der Timer, die gewünschte Zuverlässigkeit der Anwendung $p_{retrieval_{app}}$, das Bewegungsmodell, die maximal erlaubte Verlustrate oder die Höhe und Länge des Simulationsareals darin gespeichert.

In der Agent-Relation werden die wichtigsten Angaben zu den in den Experimenten verwendeten NS2-Agenten gespeichert. Zu jedem Agenten werden daher dessen Name und Version sowie der Entwickler aufgenommen.

Durch die Logfile-Tabelle wird eine Log-Datei repräsentiert. Entsprechend werden alle individuell in dieser Log-Datei verwendeten Parameter gespeichert. So werden beispielsweise die Anzahl der Transaktionen, die Anzahl der Knoten, die Anzahl der Teilnehmer, die Transaktionsdauer und der verwendete garantierte Recoveryzeitraum aufgenommen. Aber auch Werte wie die angenommene Dichte des MANETs, der Anteil von Knoten mit Kontextbewusstsein, die Simulationsdauer sowie der Pfad der Log-Datei werden gespeichert.

Die Value-Tabelle repräsentiert die eigentlichen Ergebniswerte der Simulationen, so dass sie nur die zwei Attribute Name des Wertes und den Wert selbst umfasst. Hinzu kommen noch die Fremdschlüssel auf die entsprechende Log-Datei, aus der der Wert stammt, sowie auf die zugehörigen Einträge in der Parameter-, der Experiment- und in der Agenten-Tabelle.

Format und Inhalt Log-Dateien

Nachdem im vorigen Abschnitt kurz die Datenbankstruktur eingeführt worden ist, sollen hier einige wesentliche Werte aus den Log-Dateien vorgestellt werden. Dabei wird vor allem auf die Werte eingegangen, die später bei der Analyse eine Rolle spielen. Eine Übersicht der für die Evaluation relevanten Werte mit den jeweiligen Beschreibungen ist in Tabelle 5.7 zu finden.

Die in den Simulationen ermittelten Werte beziehen sich schwerpunktmäßig auf die Analyse des Erfolgs von Transaktionen. Dazu ist es notwendig die Anzahl von gestarteten Transaktionen zu ermitteln, aber auch die Anzahl von Transaktionen, die aufgrund von Ausfällen nicht gestartet wurden bzw. wo es zu Recoverys gekommen ist. Dementsprechend werden auch alle erfolgreichen Transaktionen und Recoverys gezählt sowie die Anzahl der erfolgreichen Recoverys, die innerhalb der garantierten Recoveryzeit t_m erfolgt sind. Zusätzlich wird auch bestimmt, wie viele Transaktionen bereits zu Beginn gescheitert sind und wie viele Recoverys nicht erfolgreich abgeschlossen werden können.

Weiterhin ist auch interessant, mit welcher Strategie am ehesten eine erfolgreiche Recovery möglich ist und wie viel Zeit dafür durchschnittlich benötigt wird. Dazu wird bei jeder erfolgreichen Recovery protokolliert, mit welcher Strategie dies möglich war.

Als letzter Bereich ist die Anzahl der benötigten Pakete protokolliert worden. So werden z.B. die Anzahl der vom Koordinator und vom Teilnehmer durchschnittlich benötigten Pakete ermittelt. Zusätzlich wird auch untersucht, wie stark die übrigen Knoten des Netzes beteiligt waren, z.B. durch das Weiterleiten eines Pakets bei der Verteilung eines Transaktionslogs oder durch das Senden von Ausfallwahrscheinlichkeiten an den Koordinator.

Wert	**Beschreibung**
ANZ_TA_START	Anzahl der gestarteten Transaktionen
ANZ_TA_SUC	Anzahl der erfolgreichen Transaktionen
ANZ_TA_SUCREC	Anzahl der erfolgreichen Recoveries
ANZ_TA_FAILREC	Anzahl der fehlgeschlagenen Recoveries
ANZ_TA_FAIL	Anzahl der gescheiterten Transaktionen
ANZ_COMMIT_TIMER_FIRED	Anzahl der von den Klienten nicht erhaltenen Commits
ANZ_CLIENT_RECOVERIES	Anzahl der von den Klienten gestarteten Recoverys
ANZ_SUCC_ASK_COORDINATOR	Anzahl der erfolgreichen Nachfragen beim Koordinator
ANZ_SUCC_DIR	Anzahl der erfolgreichen gerichteten Recoverys
ANZ_SUCC_UNDIR	Anzahl der erfolgreichen ungerichteten Recoverys
ANZ_REC_MISSIONTIME	Anzahl der erfolgreichen Recoveries innerhalb des garantierten Recoveryzeitraums t_m
TME_CLIENT_WAIT_ RECOVER_COMMIT	Durchschnittliche Unsicherheitszeit bis zum Erhalt des Transaktionslogs
ANZ_CLIENT_PACKETS	Durchschnittliche Nachrichtenanzahl der Teilnehmer
ANZ_COORD_PACKETS	Durchschnittliche Nachrichtenanzahl des Koordinators

Tabelle 5.7: Übersicht über die für die Evaluation relevanten Werte aus den Log-Dateien

5.2.7 Analyse der Simulationsergebnisse

Mit den zuvor in Abschnitt 5.2.6 vorgestellten Werten aus den Log-Dateien können die gewünschten Ergebniswerte berechnet werden. Dies geschieht ebenfalls mit einem für diesen Zweck entwickelten Java-Programm (vgl. den *resultAnalyzer* auf beiliegender CD), welches die für die Berechnung notwendigen Werte aus der Datenbank extrahiert. Welche Werte dabei berechnet werden, kann ebenfalls über die Properties-Datei angegeben werden. Für die Bewertung der Simulationsergebnisse werden verschiedene Ergebniswerte benötigt.

Abschnitt 5.2.6 hat bereits die Werte vorgestellt, die in den Simulationen ermittelt werden. Zur Untersuchung von Transaktionen bei Verwendung des SLS-Prozessablaufs werden mit diesen Werten verschiedene Aspekte untersucht. Für einige Analysewerte sind weitere Berechnungen mit den vorgestellten Werten aus Tabelle 5.7 notwendig. Diese werden im Folgenden kurz vorgestellt, damit auch sie bei der Auswertung verwendet werden können.

Anteil von Transaktionen mit Recovery

Ziel ist die Ermittlung, bei wie viel Prozent von allen gestarteten Transaktionen tatsächlich eine Recovery notwendig war. Dazu wird der Quotient aus der Anzahl der von Teilnehmern gestarteten Recoveries und der Anzahl von gestarteten Teilnehmern insgesamt gebildet:

$$\frac{ANZ\_CLIENT\_RECOVERIES}{ANZ\_TA\_START \cdot \#Clients} \tag{5.5}$$

wobei ANZ_TA_START der Anzahl von gestarteten Transaktionen (und somit Koordinatoren) entspricht und $ANZ\_TA\_START \cdot \#Clients$ der Anzahl von gestarteten Teilnehmern entspricht mit $\#Clients$ der Anzahl von Teilnehmern pro Transaktion insgesamt.

Erfolgreiche Recoverys durch Nachfragen beim Koordinator

Bei der Untersuchung des Erfolgs der Verteilungsstrategien des SLS ist es wichtig, den Anteil jeder Strategie an allen erfolgreichen Recoverys zu ermitteln. Dazu wird der Quotient aus der Anzahl von erfolgreichen Recoverys mit der jeweiligen Strategie und der Gesamtanzahl von erfolgreichen Recoverys innerhalb der Simulationszeit bestimmt. Für die erfolgreiche Nachfrage beim Koordinator als der Basis-Recovery-Strategie ergibt sich dabei:

$$\frac{ANZ\_SUCC\_ASK\_COORDINATOR}{ANZ\_TA\_SUCREC} \tag{5.6}$$

Erfolgreiche Recoverys bei gerichteter Verteilung

Mit der gleichen Überlegung kann ermittelt werden, wie viel Prozent der erfolgreichen Recoverys bei Verwendung der gerichteten Verteilungsstrategie in dichten MANETs möglich war:

$$\frac{ANZ\_SUCC\_DIR}{ANZ\_TA\_SUCREC} \tag{5.7}$$

Erfolgreiche Recoverys bei ungerichteter Verteilung

Auch für den Erfolg der ungerichteten Verteilungsstrategie kann der dazu gehörige Prozentsatz bestimmt werden wie folgt:

$$\frac{ANZ\_SUCC\_UNDIR}{ANZ\_TA\_SUCREC} \tag{5.8}$$

wobei hier zu berücksichtigen ist, dass die Recovery bei den Teilnehmern immer gleich mit einem Broadcast verläuft, während die Verteilung des Koordinators abhängig von der Dichte des Netzes ist (vgl. dazu Abschnitte 4.2.2 und 4.2.3).

Erfolgreiche Recoverys innerhalb des garantierten Recoveryzeitraums

Wichtig bei der Analyse des SLS-Prozessablaufs ist jedoch nicht allein, wie viel Prozent der erfolgreichen Recoverys mit welcher Strategie möglich waren, sondern auch, ob dies innerhalb des vereinbarten Recoveryzeitraums t_m geschehen ist. Dieser Wert berechnet sich wie folgt aus den in den Simulationen ermittelten Werten:

$$\frac{ANZ\_REC\_MISSIONTIME}{ANZ\_TA\_SUCREC} \tag{5.9}$$

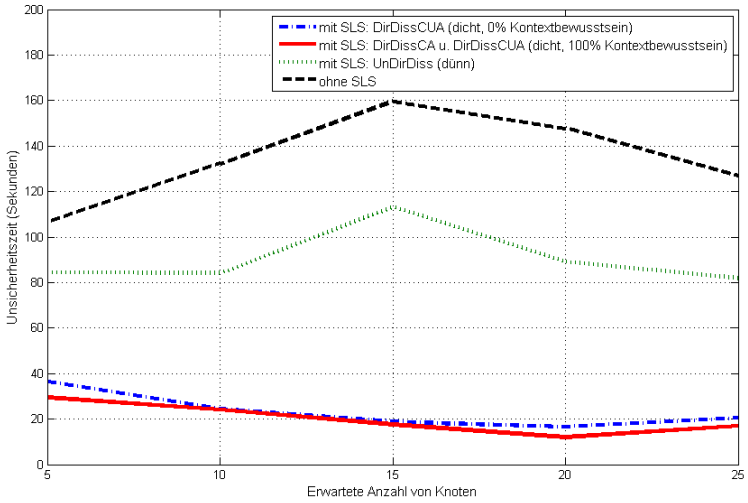

Abbildung 5.6: Durchschnittliche Unsicherheitszeit von Transaktionsteilnehmern

5.3 Ergebnisse der Simulationen

Der erste Testbereich befasst sich mit dem Hauptziel des SLS – der Reduzierung von Unsicherheitszeiten ausgefallener Transaktionsteilnehmer. Anschließend wird mit dem zweiten Testbereich betrachtet, wie viele Fehlerfälle und somit Recoverys bei beiden Prozessabläufen überhaupt auftreten und wodurch diese ausgelöst worden sind. Darauf aufbauend wird mit dem dritten Testbereich der Erfolg der einzelnen Verteilungsstrategien untersucht. Anschließend analysiert der vierte Testbereich, welcher zusätzliche Nachrichtenbedarf durch die Verteilungsstrategien des SLS erzeugt wird.

5.3.1 Testbereich 1: Unsicherheitszeiten der Transaktionsteilnehmer

Ziel des SLS ist die Verkürzung der Unsicherheitszeiten bei Commit-Protokollen nach dem Auftritt von Fehlersituationen. In diesem Testbereich werden daher die Unsicherheitszeiten ausgefallener Teilnehmer untersucht. Zunächst werden zu diesem Zweck die Unsicherheitszeiten ausgefallener Teilnehmer allgemein betrachtet. Anschließend wird analysiert, wie viele der erfolgreichen Recoverys innerhalb des vereinbarten garantierten Recoveryzeitraums t_m möglich sind.

5.3.1.1 Unsicherheitszeit ausgefallener Knoten

Hypothese: Durch Verwendung des SLS-Prozessablaufs wird die Unsicherheitszeit von Teilnehmern bei Fehlersituationen verkürzt. Dabei sind je nach Netzdichte die einzelnen Strategien unterschiedlich effizient, d.h. wird z.B. ein dünnes MANET angenommen, so ist die Datenverteilung mittels UnDirDiss erfolgreicher als die Verfahren DirDissCA und DirDissCUA für dichte Netze. Idealerweise sollte dadurch eine Überkreuzung entstehen, d.h. in dünneren Netzen soll UnDirDiss zu einer kürzeren Unsicherheitszeit als die anderen Verfahren führen, in dichteren Netzen soll dies mit Hilfe von DirDissCA und DirDissCUA geschehen.

Resultat: Die Ergebnisse der Simulationen sind in Abbildung 5.6 dargestellt. Zur Untersuchung der Hypothese werden Transaktionsszenarien mit dem Vergleichsprozessablauf und dem

SLS-Prozessablauf simuliert. Bei den Simulationen des SLS-Prozessablaufs wurden weitere Unterscheidungen nach Dichte und Kontextbewusstsein vorgenommen. Damit können Resultate bezüglich der einzelnen Strategien erzielt werden.

Bei allen Simulationsläufen mit dem SLS zeigt sich, dass die Unsicherheitszeiten bei Recoverys mit dem SLS deutlich kürzer sind als bei dem Vergleichsprozessablauf. Vergleicht man die Ergebnisse aus den drei Simulationsläufen mit dem SLS-Prozessablauf und den unterschiedlichen Parametern, so zeigt sich, dass bei Annahme eines dünnen MANETs und der Verwendung von UnDirDiss die längste Unsicherheitszeit besteht. Die kürzeste Unsicherheitszeit wird mit der Kombination aus DirDissCA und DirDissCUA erzielt, wenn das Netz als dicht angenommen wird und alle Knoten über Kontextbewusstsein verfügen. Wird das Netz als dicht angenommen und die Knoten haben kein Kontextbewusstsein, sind die Unsicherheitszeiten mit DirDissCUA nur leicht höher.

Beurteilung: Es wird deutlich, dass die Verteilungsstrategien des SLS dazu führen, dass die Unsicherheitszeiten von Teilnehmern nach einem Ausfall im Vergleich zu dem Prozessablauf ohne SLS deutlich verkürzt werden können. Der erste Teil der aufgestellten Hypothese wird durch die Ergebnisse also gestützt. Allerdings tritt bei den verwendeten Szenarien nicht die gewünschte Überkreuzung ein, d.h. UnDirDiss führt in dünneren Netzen nicht zu kürzeren Unsicherheitszeiten als die beiden anderen Verfahren des SLS.

Aus diesen Ergebnissen können Folgerungen im Hinblick auf die Effektivität der einzelnen Strategien gezogen werden. Alle drei Verfahren führen zwar zu einer Reduzierung der Unsicherheitszeit, allerdings muss bei einer Verteilung nach UnDirDiss mit einer um 20 bis 60 Sekunden längeren Unsicherheitszeit gerechnet werden (vgl. dazu Abbildung 5.6).

Bei UnDirDiss wird das MANET als dünn angenommen. Der Koordinator flutet beim Auftritt eines Fehlers das MANET ein einziges Mal zur Verteilung des Transaktionslogs. Mit diesem Mechanismus ist die Unsicherheitszeit kürzer als bei Verwendung des Vergleichsprozessablaufs, jedoch bei allen Knotenanzahlen länger als bei Annahme eines dichten MANETs. Grund für dieses Ergebnis ist die Wahl des Timers mit 60 Sekunden für das Pushen der Transaktionslogs. Mit diesem großen Zeitraum zwischen dem Weiterverteilen der Logs wird der benötigte Verteilungsgrad über einen längeren Zeitraum hinweg erzeugt. Dadurch ist eine erfolgreiche Recovery (insbesondere bei Kommunikationsfehlern) nicht so schnell möglich wie bei den beiden Verfahren für dichte MANETs.

Die Ursache für das bessere Ergebnis mit den gerichteten Verfahren ist darauf zurückzuführen, dass bei Annahme eines dichten MANETs mit DirDissCA und DirDissCUA schneller ein höherer Verteilungsgrad erzielt wird. Damit ist eine erfolgreiche Recovery im Durchschnitt innerhalb von 20 bis 40 Sekunden möglich. Verfügen die Knoten im Netz nicht über Kontextbewusstsein, so berechnet der Koordinator bei DirDissCUA eine Anzahl n von notwendigen Knoten zur Speicherung. Er broadcastet das Log beim Auftritt von Fehlern solange an seine Nachbarn bis er n Bestätigungen über die Speicherung erhalten hat. Der Timer für das erneute Broadcasten des Logs ist auf 0.2 Sekunden gesetzt (vgl. Tabelle 5.3). Dadurch wird öfter verteilt als bei UnDirDiss. Damit kann schneller eine große Anzahl von Knoten im Netz zur Speicherung des Transaktionslogs erreicht werden. Wenn die Knoten hingegen über Kontextbewusstsein verfügen, gibt es für den Koordinator zwei Möglichkeiten: entweder er findet in seiner Nachbarschaft eine ausfallsichere Knotengruppe (DirDissCA) oder er verteilt wie zuvor auf n anonyme Knoten (DirDissCUA). Diese Kombination aus DirDissCA und DirDissCUA bei Annahme eines dichten MANETs führt zu den kürzesten Unsicherheitszeiten zwischen 20 und 30 Sekunden. Bei einigen Fehlersituationen kann durch DirDissCA also eine Verbesserung gegenüber DirDissCUA erzielt werden. In diesen Fällen ist die Auswahl einer Gruppe von n Knoten mit geringer Ausfallwahrscheinlichkeit zur

100% Kontextbewusstsein (Kombination STRAT1 und STRAT2)		**0% Kontextbewusstsein (STRAT2)**	
ANZ_CLIENT_RECOVERIES	42	ANZ_CLIENT_RECOVERIES	34
ANZ_TA_SUCREC	42	ANZ_TA_SUCREC	34
ANZ_SUCC_ASK_COORDINATOR	5	ANZ_SUCC_ASK_COORDINATOR	1
ANZ_SUCC_DIR	21	ANZ_SUCC_DIR	0
ANZ_SUCC_UNDIR	15	ANZ_SUCC_UNDIR	33
ANZ_REC_MISSIONTIME	42	ANZ_REC_MISSIONTIME	34
TME_CLIENT_WAIT_RECOVER_PONG	3.1s	TME_CLIENT_WAIT_RECOVER_PONG	4.3s
ANZ_CLIENT_PACKETS	2.1	ANZ_CLIENT_PACKETS	2.0
ANZ_COORD_PACKETS	9.6	ANZ_COORD_PACKETS	10.0

Abbildung 5.7: Ausschnitte aus zwei Log-Dateien zu Szenarien mit einer erwarteten Anzahl von 20 Knoten und 8 Transaktionsteilnehmern

Reduzierung der Unsicherheitszeit von Knoten am effizientesten.

Zur Verdeutlichung kann man als Beispiel in Abbildung 5.7 Ausschnitte aus zwei Log-Dateien betrachten. Es werden zwei Szenarien mit 8 Teilnehmern und einer erwarteten Anzahl von 20 Knoten herausgegriffen, da bei dieser Netzdichte die kürzesten Unsicherheitszeiten auftreten (vgl. Abbildung 5.6). Einmal wird das MANET als dicht angenommen und die Knoten verfügen über Kontextbewusstsein. Im anderen Fall verfügen die Knoten nicht über Kontextbewusstsein. Somit muss der Koordinator bei der ersten Variante (linke Spalte) zwischen DirDissCA und DirDissCUA auswählen.

Die beiden beispielhaften Ausschnitte in Abbildung 5.7 stützen die obige These. Bei der Kombination aus DirDissCA und DirDissCUA in der linken Spalte zeigt sich eine kürzere Unsicherheitszeit. Die Hälfte aller Transaktionslogs wurde beim Auftritt von Fehlern mittels DirDissCA durch den Koordinator verteilt (vgl. `ANZ_SUCC_DIR`). Ausgefallene Teilnehmer konnten damit bei einem der n ausgewählten Knoten innerhalb von t_m das benötigte Log finden und die Transaktion trotz Fehler erfolgreich abschließen. Hat der Koordinator keine n ausfallsicheren Knoten gefunden, wurde auf n anonyme Knoten mittels DirDissCUA verteilt (vgl. `ANZ_SUCC_UNDIR`). Auch damit konnten ausgefallene Teilnehmer die Transaktion innerhalb von t_m erfolgreich beenden. Zusätzlich war in einigen Fällen die Anfrage an den Koordinator erfolgreich (vgl. `ANZ_SUCC_ASK_COORDINATOR`).

In der rechten Spalte von Abbildung 5.7 zeigt sich, dass auch mit DirDissCUA bei dem Beispiel-Szenario immer eine erfolgreiche Recovery innerhalb von t_m möglich war (vgl. `ANZ_REC_MISSIONTIME`). Gleichzeitig ist dabei jedoch die Unsicherheitszeit um etwa eine Sekunde höher (vgl. `TME_CLIENT_WAIT_RECOVER_PONG`). Außerdem benötigt der Koordinator im Schnitt mehr Nachrichten (vgl. `ANZ_COORD_PACKETS`). Lediglich auf der Seite der Teilnehmer werden leicht weniger Pakete benötigt als bei dem Beispiel in der linken Spalte (vgl. `ANZ_CLIENT_PACKETS`). Dies lässt sich darauf zurückführen, dass für die Recovery bei DirDissCA mehr Nachrichten notwendig sind als bei DirDissCUA (vgl. Abschnitt 4.2). Bei DirDissCUA wird pro Recovery-Versuch ein Broadcast an die Nachbarn geschickt. Im Gegensatz dazu muss bei DirDissCA bei jedem neuen Versuch eine Nachricht an den Koordinator und die n Knoten gesendet werden.

Zusammenfassend bleibt festzuhalten: *Durch alle Verteilungsstrategien des SLS kann die Unsicherheitszeit erheblich verkürzt werden, allerdings zeichnen sich Nebeneffekte dieser Verbesserung ab.* Die Verteilungsstrategien des SLS sind unterschiedlich effizient. UnDirDiss führt zur

längsten Unsicherheitszeit, die Kombination von DirDissCA und DirDissCUA zur kürzesten. Die genannten Nebeneffekte müssen bei der Bewertung berücksichtigt und daher im Folgenden näher untersucht werden. Erste Analysen in Abbildung 5.7 deuten an, dass die Effizienz den Nachrichtenbedarf zur Verteilung beeinflusst. Abschnitt 5.3.4 wird diesen Aspekt näher untersuchen.

5.3.1.2 Erfolgreiche Recoverys innerhalb des garantierten Recoveryzeitraums t_m

Hypothese: Mit dem SLS-Prozessablauf kann eine erfolgreiche Recovery innerhalb des vereinbarten garantierten Recoveryzeitraums t_m ermöglicht werden.

Resultat: Die Ergebnisse der Simulationen mit beiden Prozessabläufen sind in Abbildung 5.8 dargestellt. Bei Verwendung des SLS-Prozessablaufs ist in ca. 99% aller Fälle eine erfolgreiche Recovery innerhalb des garantierten Recoveryzeitraums t_m möglich (die Berechnung der Ergebnisse erfolgt unter Verwendung von Formel 5.9). Im Gegensatz dazu liegt bei dem Vergleichs-Prozessablauf die Erfolgsrate deutlich darunter. In den meisten Netzen beträgt diese ca. 0% bis 1%. Lediglich in Netzen mit ca. 20 Knoten können ca. 10% der Recoverys innerhalb des garantierten Recoveryzeitraums t_m erfolgreich beendet werden.

Beurteilung: Die Ergebnisse bestätigen die obige Hypothese. Wie bereits Abschnitt 5.3.1.1 zeigt, kann die Unsicherheitszeit von Teilnehmern bei dem SLS-Prozessablauf deutlich verkürzt werden. Untersucht man explizit, wie viele der erfolgreichen Recoverys innerhalb des vereinbarten Recoveryzeitraums t_m möglich sind, so sind ebenfalls starke Unterschiede zwischen den Prozessabläufen erkennbar. Bei Verwendung des SLS-Prozessablaufs sind ca. 99% der Recoverys während t_m erfolgreich. Mit dem Vergleichs-Prozessablauf können hingegen nur maximal 10% der Recoverys innerhalb von t_m beendet werden.

Dies zeigt, dass der mit den Strategien erreichte Verteilungsgrad des Logs im MANET dazu führt, dass die vor der Transaktion vereinbarten Garantien gewährleistet werden können. Es ist also nicht nur möglich, dass Transaktionsteilnehmer nach einem Ausfall kürzer unsicher sind als beim Vergleichs-Prozessablauf, sondern auch *der Erhalt des Transaktionslogs innerhalb des vereinbarten garantierten Recoveryzeitraums t_m ist möglich.* Hierbei muss allerdings darauf hingewiesen werden, dass dieser Zeitraum nicht völlig beliebig sein kann, sondern (wie in Abschnitt 5.2 beschrieben) mit den übrigen Parametern abgestimmt sein muss.

5.3.2 Testbereich 2: Auftritt von Recoverys

In diesem Testbereich wird untersucht, wie viele Recoverys bzw. Fehlersituationen bei beiden Prozessabläufen überhaupt auftreten. Es wird versucht werden, die Hauptursache von Fehlerfällen in den Simulationen zu ermitteln.

Hypothese: Bei beiden Prozessabläufen treten gleich viele Fehlersituationen und somit Recoverys auf.

Resultat: Abbildung 5.9 zeigt die Ergebnisse der Simulation, wobei für beide Prozessabläufe jeweils Transaktionsszenarien mit Knotenfehlern simuliert wurden. Für die Berechnung der Ergebniswerte wird Formel 5.5 verwendet. Es wird deutlich, dass bei dem SLS-Prozessablauf mehr Fehlerfälle auftreten als bei dem Vergleichs-Prozessablauf. Entsprechend werden beim SLS-Prozessablauf auch mehr Recoverys gestartet. Während bei einem Netz mit 5 Knoten noch keine

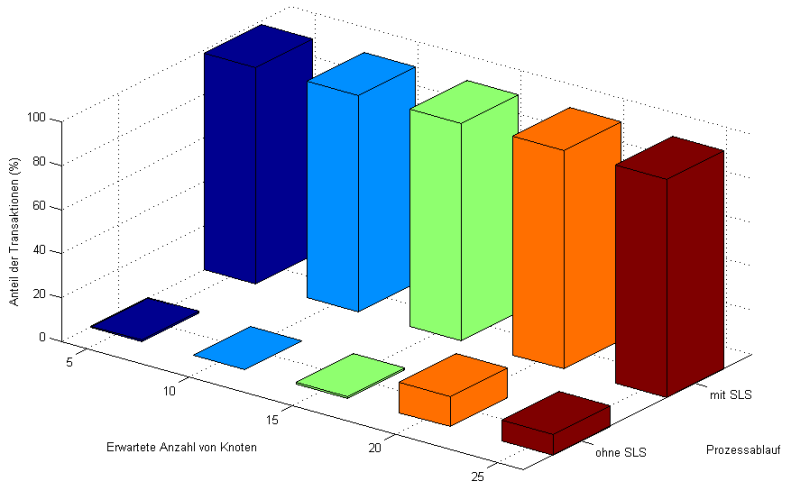

Abbildung 5.8: Erfolgreiche Recoverys innerhalb des garantierten Recoveryzeitraums in Abhängigkeit vom Prozessablauf

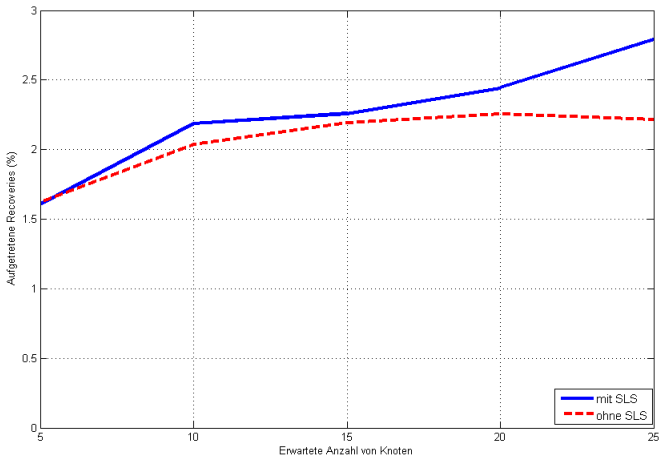

Abbildung 5.9: Auftritt von Recoverys in Abhängigkeit von Knotenausfällen und Prozessablauf

SLS-Prozessablauf			Vergleichs-Prozessablauf		
Zustand	**min. Warte-zeit**	**max. Warte-zeit**	**Zustand**	**min. Warte-zeit**	**max. Warte-zeit**
COORD_WAIT_FOR_VOTE	0.02s	0.2s	COORD_WAIT_FOR_VOTE	0.02s	0.2s
COORD_WAIT_FOR_RETFAIL	0.1s	0.1s	COORD_WAIT_SOME_TIME	1.0s	1.0s
COORD_WAIT_FOR_DIR_ACK	0.04s	0.2s			
COORD_WAIT_SOME_TIME	1.0s	1.0s			
Gesamt:	**1.16s**	**1.5s**	**Gesamt:**	**1.02s**	**1.2s**

Tabelle 5.8: Zeitdauer bis zur Commit-Entscheidung bei beiden Prozessabläufen

Unterschiede zwischen den Prozessabläufen erkennbar sind, werden in Netzen mit 10 bis 20 Knoten ca. 0.1% bis 0.2% weniger Recoverys bei dem Vergleichs-Prozessablauf gestartet. Bei einem MANET mit 25 Knoten beträgt der Unterschied sogar 0.5%.

Beurteilung: Die Ergebnisse der Simulationen können den ersten Teil obiger Hypothese nicht untermauern. Wird der SLS-Prozessablauf benutzt, so treten bei MANETs mit mehr als 5 Knoten häufiger Fehlersituationen auf als bei dem Vergleichs-Prozessablauf.

Die etwas höheren Recoveryraten beim SLS-Prozessablauf sind darauf zurückzuführen, dass dieser grundsätzlich länger ist als der Vergleichs-Prozessablauf. Dadurch können mehr Fehler auftreten. Es gibt zwei Zustände mehr: COORD_WAIT_FOR_RETFAIL und COORD_WAIT_FOR_DIR_ACK. Dabei sammelt der Koordinator zunächst die Ausfallwahrscheinlichkeiten seiner Nachbarn und wählt dann eine Verteilungsstrategie aus. Diese wird anschließend den Teilnehmern mitgeteilt. Erst nachdem diese die Nachricht mit der Information über die Verteilungsstrategie erhalten haben, beginnen sie auf das COMMIT zu warten und sind unsicher. Beim Vergleichs-Prozessablauf warten Teilnehmer hingegen direkt nach dem Senden des Votums auf das COMMIT. Folglich dauert es bei dem SLS-Prozessablauf länger bis zur eigentlichen Entscheidung der Transaktion auf COMMIT oder ABORT. Bis dahin können sich teilnehmende Knoten weiter bewegt haben, so dass Pakete nicht mehr ankommen. Dementsprechend muss eine Recovery gestartet werden. Im Gegensatz dazu wird die Transaktionsentscheidung beim Vergleichs-Prozessablauf früher getroffen. Die Knoten befinden sich dort noch in Reichweite, so dass der Transaktionsausgang die Knoten öfter erreicht und keine Recovery benötigt wird. Inbesondere die Mobilität der Knoten scheint hier bei dem längeren Prozessablauf zu mehr Kommunikationsfehlern zu führen.

Tabelle 5.8 verdeutlicht diese Überlegung. Darin werden für beide Prozessabläufe die Zustände der Commit-Phase bis zur Transaktionsentscheidung aufgeführt. Es wird jeweils unterschieden,

Anzahl der Knoten im MANET	**SLS-Prozessablauf**	**Vergleichs-Prozessablauf**
10	6.6%	3.9%
20	7.3%	5.1%
30	4.6%	5.7%
40	2.2%	5.4%
50	1.8%	3.9%

Tabelle 5.9: Anteile von Knotenfehlern an den aufgetretenen Recoverys bei beiden Prozessabläufen

wie hoch die minimale und wie hoch die maximale Wartezeit des Koordinators in jedem Zustand ist. Der Gesamtzeitraum entspricht dann dem Zeitraum, der bis zur Transaktionsentscheidung vergeht. Dabei wird von einer Nachrichtenverzögerung einer Nachricht (engl. *message delay*) von $\delta = 0.02s$ ausgegangen. Bei der mimimalen Wartezeit wird jeweils angenommen, dass die Teilnehmer direkt zu Beginn der Commit-Phase ihr Votum schicken und dies nach dem Zeitraum δ beim Koordinator eintrifft. Dies ist auch die Annahme bei den anderen Zuständen in diesem Fall, so dass dort nie ein Timer abläuft. Bei der maximalen Wartezeit hingegen wird angenommen, dass immer die Timer feuern, weil von mindestens einem Teilnehmer die benötigte Nachricht ausgeblieben ist.

Die Ergebnisse zeigen auf, dass bei Verwendung des SLS im schlechtesten Fall bei den gewählten Parametern 0.48 Sekunden (bei minimaler Wartezeit beim Vergleichsprozessablauf und maximaler Wartezeit beim SLS-Prozessablauf) später die Transaktionsentscheidung getroffen wird. Im besten Fall liegt der Unterschied zwischen den Prozessabläufen bei 0.14 Sekunden. Diese relativ große Zeitspanne von bis zu 0.48 Sekunden kann in mobilen Netzen ausreichen, um durch Bewegung der Benutzer Kommunikationsfehler zu erzeugen.

Betrachtet man dazu, wie viele Recoverys nach einem Knotenfehler getartet worden sind, wird diese Vermutung gestützt. Tabelle 5.9 zeigt die durschschnittlichen Anteile der Knotenfehler an allen Recoverys. Bei allen simulierten Transaktionsszenarien werden weniger als 10% der Recoverys nach einem Knotenfehler gestartet. Dies festigt die obige These, dass Kommunikationsfehler aufgrund von Mobilität zu mehr Fehlersituationen führen als Knotenausfälle. Zusätzlich ist festzustellen, dass in Netzen mit 5 und 10 Knoten beim SLS-Prozessablauf mehr Knotenfehler in der Commit-Phase auftreten. In einem dünneren Netz mit wenig Knoten kommt es also beim SLS-Prozessablauf durch die spätere Commit-Phase sowohl zu mehr Knoten- wie Kommunikationsfehlern.

Der Testbereich führt zu folgenden Schlussfolgerungen: *Bei Verwendung des SLS-Prozessablaufs kommt es durch zwei zusätzliche Zustände zu mehr Fehlern – und somit Recoverys – als bei dem Vergleichs-Prozessablauf. Es werden mehr als 90% aller Recoverys durch Kommunikationsfehler aufgrund der Mobilität von Knoten ausgelöst.* Dabei muss jedoch darauf hingewiesen werden, dass trotz des erhöhten Auftritts von Recoverys beim SLS-Prozessablauf diese zu 99% innerhalb von t_m erfolgreich beendet werden können (vgl. dazu Testbereich 1 in Abschnitt 5.3.1).

5.3.3 Testbereich 3: Erfolg der einzelnen Verteilungsstrategien des SLS-Prozessablaufs

Der SLS-Prozessablauf verwendet die in Abschnitt 4.1.1.1 eingeführten Verteilungsstrategien DirDissCA, DirDissCUA und UnDirDiss zur Verteilung eines Transaktionslogs im MANET beim

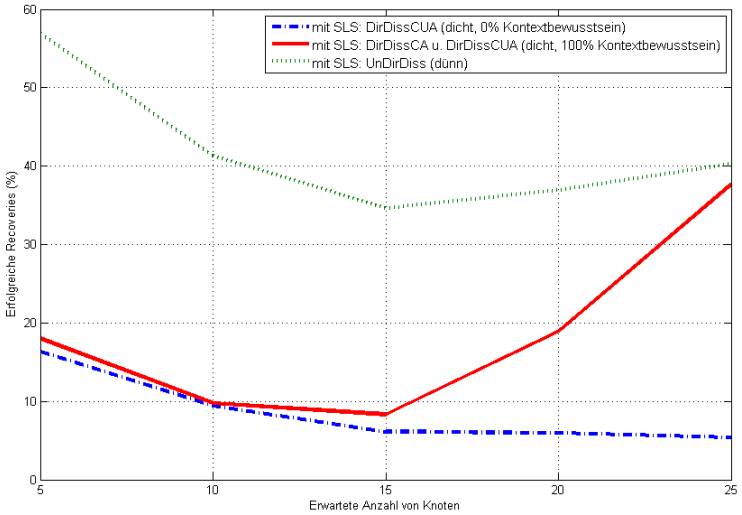

Abbildung 5.10: Erfolgreiche Recovery-Anfragen an den Transaktionskoordinator

Auftritt von Fehlern. Mit diesem Testbereich wird auf die Effizienz der einzelnen Verteilungsstrategien bei der Reduzierung der Unsicherheitszeiten eingegangen. Dazu wird analysiert, welchen Anteil die in Abschnitt 4.1.1.2 vorgestellten Recovery-Verfahren der Teilnehmer jeweils an allen erfolgreichen Recoverys haben. Zunächst werden die erfolgreichen Anfragen an den Koordinator betrachtet. Danach werden die Recovery-Strategien des SLS mit einbezogen.

5.3.3.1 Erfolgreiche Recovery-Anfragen an den Koordinator

Hypothese: In dichten MANETs sind Recovery-Anfragen an den Transaktionskoordinator erfolgreicher als in dünnen MANETs. Wenn der Koordinator nicht selbst einen Knotenfehler erlitten hat, ist er für einen ausgefallenen Teilnehmer in einem dichten MANET für die Recovery erreichbar.

Resultat: Die mit Formel 5.6 berechneten Ergebnisse der Simulationen sind in Abbildung 5.10 dargestellt. Es wurden beide Prozessabläufe simuliert. Beim SLS-Prozessablauf wurden zudem die Parameter Dichte und Kontextbewusstsein variiert.

Wird bei Annahme eines dünnen MANETs zur Verteilung UnDirDiss verwendet, liegt der Anteil der erfolgreichen Anfragen an den Koordinator bei etwa 35% bis 56% von allen erfolgreichen Recoverys. Dabei wird der höchste Prozentsatz mit 56% in einem Netz mit 5 Knoten erreicht.

Wenn von der Annahme ausgegangen wird, dass es sich um ein dichtes MANET handelt, sehen die Resultate etwas anders aus. Bei Verwendung von DirDissCUA ist der geringste Anteil zu erkennen. In Netzen mit 5 Knoten beträgt dieser ca. 16%, sonst bewegt er sich zwischen 5% und 10%. Im Gegensatz dazu zeigt sich bei der Kombination aus DirDissCA und DirDissCUA, dass der Anteil von erfolgreichen Recoverys in dünnen Netzen geringer als in dichten ist. Bei 5 und 15 Knoten bewegt sich dieser zwischen 10% und 17%. Bei 20 Knoten beträgt dieser 20% und steigt bei 25 Knoten auf 47%.

Beurteilung: Bei den Ergebnissen wird deutlich, dass Recovery-Anfragen an den Transaktionskoordinator in dichten Netzen nicht immer erfolgreicher sind als in dünnen. Der Erfolg hängt mit der vom Koordinator verwendeten Verteilungsstrategie zusammen. Somit kann die obige Hypothese nicht bestätigt werden. Die erfolgreichen Recovery-Anfragen an den Transaktionskoordinator reflektieren die in Abschnitt 5.3.1.1 untersuchten Unsicherheitszeiten von ausgefallenen Teilnehmern.

Wird das MANET als dünn angenommen und das Transaktionslog mit UnDirDiss verteilt, so ist der Anteil von erfolgreichen Recovery-Anfragen an den Koordinator mit durchschnittlich 40% verhältnismäßig groß. In dünnen Netzen mit 5 Knoten übersteigt dieser sogar 50%. Dadurch wird die Vermutung unterstützt, dass mit UnDirDiss kein so hoher Verteilungsgrad erzielt werden kann, dass mit ungerichteter Recovery das Transaktionslog schneller gefunden werden kann als beim Koordinator. Es erhalten nicht ausreichend viele Knoten im Netz innerhalb kürzester Zeit das Log zur Speicherung. Ursache dafür ist der Timer für das Pushen mit 60 Sekunden. Der Zeitraum bis zur weiteren Verteilung des Logs ist so groß, dass damit Recoverys aufgrund von Kommunikationsfehlern nicht so schnell erfolgreich beendet werden können wie bei den beiden gerichteten Verfahren. Der für die Behandlung länger andauernder Knotenfehler ausgelegte Pushtimer ist daher für kurzfristige Kommunikationsfehler nicht angemessen.

Wenn hingegen das MANET als dicht angenommen wird und die Knoten nicht über Kontextbewusstsein verfügen, wird DirDissCUA angewandt. Dann ist der Anteil von erfolgreichen Koordinatoranfragen an allen Recoverys am niedrigsten. Er beträgt nur in Netzen mit 5 Knoten etwa 16% und liegt ansonsten unter 10%. Wie bereits die Untersuchung der Unsicherheitszeiten in Abschnitt 5.3.1 zeigte, kann mit diesem Verfahren eine sehr kurze Unsicherheitszeit erzielt werden. Durch die wiederholten Broadcasts des Koordinators bei DirDissCUA wird ein sehr hoher Verteilungsgrad des Logs im MANET erreicht. Dadurch wird das Log bei einer Recovery schneller mit der ungerichteten Recovery bei einem der aktuellen Nachbarn als beim Koordinator gefunden. Entsprechend ist der Anteil an erfolgreichen Koordinatoranfragen sehr gering über alle Netzdichten hinweg.

Lediglich bei Annahme eines dichtes MANETs und Knoten mit Kontextbewusstsein deutet sich eine Entwicklung gemäß obiger Hypothese an. Hier wird je nach Vorhandensein von ausfallsicheren Knoten entweder DirDissCA oder DirDissCUA angewandt. Dabei wird deutlich, dass die Ergebnisse in Netzen mit 5 bis 15 Knoten in etwa den vorherigen Resultaten entsprechen, wo ausschließlich DirDissCUA benutzt wird. In Netzen mit 20 und 25 Knoten hingegen kommt es zu einem Anstieg des Erfolgs der Koordinatoranfragen. Hier wird das Verhalten beider Strategien widergespiegelt. In dünnen Netzen werden nicht immer ausreichend viele Knoten gefunden, die eine geringe Ausfallwahrscheinlichkeit besitzen. Entsprechend muss der Koordinator mit DirDissCUA auf n anonyme Knoten verteilen, wodurch ein hoher Verteilungsgrad erzielt wird. Wird das Netz hingegen dichter (bei 20 und 25 Knoten), so kann DirDissCA angewendet werden. Dann können ausgefallene Teilnehmer bei der gerichteten Recovery direkt Anfragen an die n Knoten sowie den Koordinator richten. Dabei sind mit 20% bzw. 30% Recovery-Anfragen an den Koordinator erfolgreicher als bei alleiniger Verwendung von DirDissCUA. Bei DirDissCA ist der Koordinator in diesen Fällen schneller erreichbar als die n Knoten ist.

Abschließend bleibt festzuhalten: *Der Anteil der erfolgreichen Recovery-Anfragen an den Koordinator reflektiert die Effizienz der einzelnen Verteilungsstrategien im Hinblick auf die verkürzten Unsicherheitszeiten – je mehr Knoten im Netz das Transaktionslog speichern, desto größer ist die Wahrscheinlichkeit, dass dieses bei einem anderen Knoten als dem Koordinator schneller gefunden werden kann.*

5.3.3.2 Anteil der einzelnen Strategien an den erfolgreichen Recoverys

Hypothese: Sowohl in MANETs mit wie ohne Knotenfehler ist in dichten Netzen die gerichtete Recovery erfolgreicher und in dünnen Netzen die ungerichtete Recovery. Es kommt also zu einer Überschneidung bei beiden Verfahren.

Resultat: In Abbildungen 5.11.a und 5.11.b sind die Resultate der Simulationen mit und ohne Knotenfehler dargestellt. Dabei wird jeweils aufgezeigt, wie viel Prozent aller erfolgreichen Recoverys aufgrund welcher Verteilungsstrategie möglich gewesen sind. Dabei werden alle Recovery-Strategien – Anfrage an den Koordinator, gerichtete und ungerichtete Recovery (vgl. Abschnitt 4.1.1.2) – betrachtet. Für die Berechnungen der Ergebnisse werden die Formeln 5.6, 5.7 und 5.8 verwendet.

Zunächst werden Netze ohne Knotenfehler betrachtet, d.h. dass nur Kommunikationsfehler zu Recoverys führen. Der größte Anteil erfolgreicher Recoverys ist bei Netzen mit 10 bis 30 Knoten aufgrund der Verteilung mit DirDissCUA oder UnDirDiss gegeben. Der Anteil der erfolgreichen ungerichteten Recovery liegt immer über 60%. In einem Netz mit 40 Knoten sinkt dieser Anteil auf ca. 42%, bei 50 Knoten liegt er knapp über 30%. Bei den erfolgreichen Anfragen an den Koordinator zeigt sich, dass dieser Anteil in einem Netz mit 20 Knoten mit ca. 20% am schlechtesten ist. In MANETs mit 40 und 50 Knoten steigt der Anteil auf 55% bzw. 67%. Die Anteile der erfolgreichen Recoverys aufgrund der gerichteten Verteilung betragen bei 20 Knoten 13% und bei 30 Knoten 12%. In allen anderen Fällen liegt deren Anteil unter 10%.

Treten Knotenausfälle im MANET auf, bewegt sich der Anteil erfolgreicher Recoverys durch Verteilung mittels DirDissCUA und UnDirDiss zwischen 56% und 65%. Insgesamt erkennt man eine fallende Tendenz des Anteils der ungerichteten Recovery. Im Gegensatz dazu steigt der Anteil erfolgreicher Koordinatoranfragen. Bewegt sich dieser Wert bei 5 bis 15 Knoten noch zwischen 10% und 20%, so steigt er auf etwa 68% bei 25 Knoten im MANET. Auch bei der gerichteten Recovery steigt der Erfolgsanteil, allerdings weniger stark als bei den Koordinatoranfragen. Während bei 5 Knoten noch 0% der Recoverys damit abgeschlossen werden konnten, sind es zuletzt bei 25 Knoten etwa 14%.

Beurteilung: Bei der Betrachtung der Resultate sowie der Abbildungen 5.11.a und 5.11.b wird deutlich, dass bei den gewählten Parametern die obige Hypothese nicht bestätigt werden kann. Weder in Szenarien mit Auftritt von Knotenfehlern noch in Szenarien ohne Knotenfehler stellt sich die erwartete Überschneidung zwischen gerichteter und ungerichteter Recovery ein.

Wie bereits Abschnitt 5.3.2 aufzeigt, haben Knotenfehler nur einen geringen Einfluss auf Recoverys im Gegensatz zu Kommunikationsfehlern (vgl. dazu auch Tabelle 5.9). Allerdings ist beim Auftreten von Knotenfehlern das Netz dünner als ohne derartige Fehler. Entsprechend verschiebt sich die Überkreuzung zwischen erfolgreicher Anfrage beim Koordinator und ungerichteter Recovery weiter nach rechts. Ansonsten verhalten sich beide Recoverystrategien im Wesentlichen gleich.

Unterschiede sind jedoch bei der gerichteten Recovery erkennbar. Treten keine Knotenfehler auf, sind bei Netzen mit 20 und 30 Knoten ca. 13% bis 12% der erfolgreichen Recoverys mit dem gerichteten Verfahren möglich. Bei allen anderen Knotenzahlen sinkt dieser Anteil auf unter 10%. Wenn hingegen Knotenfehler auftreten, kommt es zu einer Steigerung mit zunehmender Netzdichte von 0% bis hin zu ca. 14%.

Diese unterschiedlichen Resultate sind auf die Netzdichte zurückzuführen. Treten keine Knotenfehler auf, so sind alle Knoten verfügbar und können bei der Verteilung verwendet werden. Entsprechend führt dies zu dem erhöhten Anteil des Erfolgs der gerichteten Recovery bei Netzen mit 20 und 30 Knoten. Da der Einfluss der Knotenfehler auf die Strategien sehr gering ist (vgl.

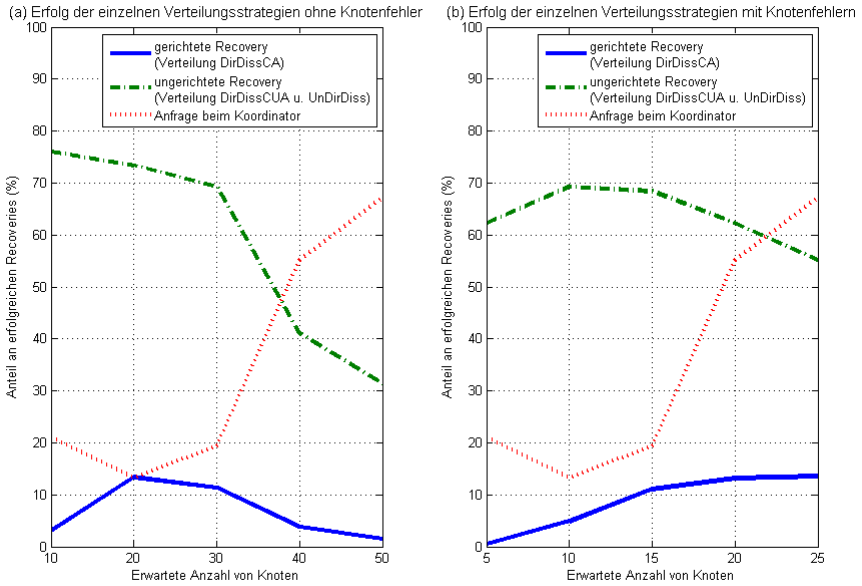

Abbildung 5.11: Erfolg der einzelnen Verteilungsstrategien mit Knotenausfällen im MANET-Szenario

Abschnitt 5.3.2), führen Kommunikationsfehler dazu, dass in dichteren MANETs ohne Auftritt von Knotenfehlern DirDissCA und somit die gerichtete Recovery wenig Erfolg haben.

Beim Auftritt von Knotenfehlern in den Szenarien stellt sich die Situation anders dar. Bedingt durch die Knotenfehler ist die Dichte der Szenarien reduziert. Dadurch wird in dünnen Netzen DirDissCA selten eingesetzt. In der Reichweite des Koordinators befinden sich nicht ausreichend viele Knoten mit geringer Ausfallwahrscheinlichkeit. Je dichter die Netze werden, desto größer wird der Anteil des Erfolgs der gerichteten Recovery. Der Koordinator findet ausreichend viele ausfallsichere Knoten für DirDissCA. Dies verdeutlicht auch, dass zwar nur wenige Recoverys durch Knotenfehler ausgelöst werden, die Einbeziehung der Ausfallwahrscheinlichkeiten bei der Verteilung jedoch sinnvoll ist. Vergleicht man die Ergebnisse der Simulationen mit und ohne Ausfälle miteinander, so zeigt sich, dass bei echten Knotenfehlern die Einbeziehung von Ausfallwahrscheinlichkeiten eine wirkungsvolle Unterstützung bietet. Durch diese Einschätzung werden offensichtlich Knoten ausgewählt, die während des garantierten Recoveryzeitraums t_m mit großer Wahrscheinlichkeit zur Verfügung stehen werden. Dies erklärt den ansteigenden Anteil der gerichteten Recovery beim Auftritt von Knotenfehlern.

Gleichzeitig ist in beiden Fällen zu erkennen, dass der Anteil der ungerichteten Recovery sehr hoch ist. Insbesondere in Netzen mit 5 bis 15 bzw. 10 bis 30 Knoten sind damit die besten Erfolge zu erzielen. Ursache dafür ist die bessere Verteilung des Transaktionslogs durch DirDissCUA und UnDirDiss. Diese Vermutung hat sich bereits in den vorherigen Testbereichen angedeutet (vgl. z.B. Abschnitt 5.3.1 zur Untersuchung der Unsicherheitszeiten). Mit beiden Verfahren wird ein so hoher Verteilungsgrad erzielt, dass ein Teilnehmer bei der Recovery das Transaktionslog schneller in seiner direkten Umgebung als beim Koordinator erhält. Insbesondere mit DirDissCUA wird das Transaktionslog auf sehr viele Knoten im MANET verteilt, wie die vorherigen Testbereiche zeigen. Mit Testbereich 4 in Abschnitt 5.3.4 wird dieser Aspekt näher untersucht.

Folgenden Schluss kann man aus den Ergebnissen ziehen: *Je dichter das* MANET *ist, desto besser funktionieren beim Auftritt von Kommunikations- und Knotenfehlern Verfahren wie Anfragen*

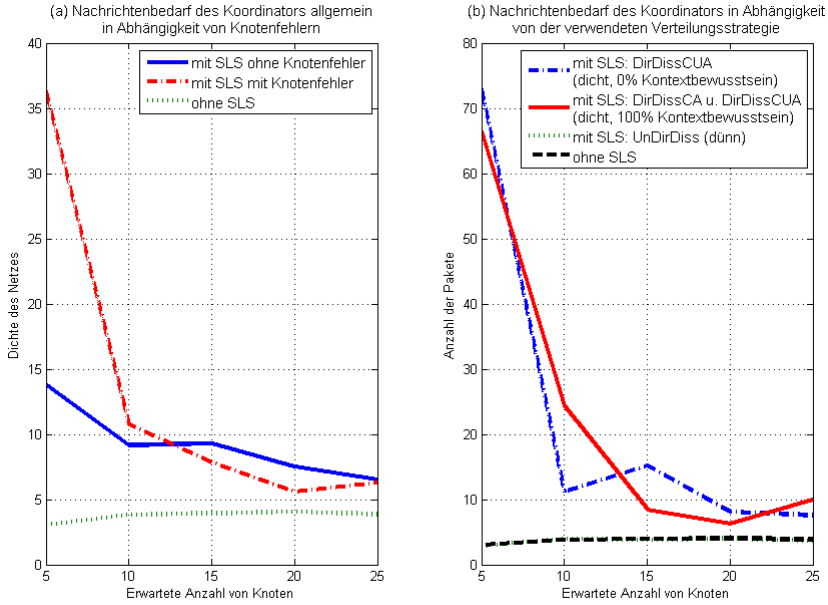

Abbildung 5.12: Vergleich des Nachrichtenbedarfs des Koordinators allgemein

an den Koodinator und die gerichtete Recovery, die individuelle Ausfallwahrscheinlichkeiten mit einbezieht. Bei den gewählten Parametern ist deren Erfolg jedoch trotzdem schlechter als der der ungerichteten Recovery. Der Grund dafür ist der unterschiedliche Verteilungsgrad des Transaktionslogs im MANET.

5.3.4 Testbereich 4: Nachrichtenaufwand der Verteilungsstrategien

Der vierte Testbereich geht auf die Kosten ein, die durch die Reduzierung bei den Unsicherheitszeiten verursacht werden. Dazu wird hier der durchschnittliche Nachrichtenaufwand von Koordinator und Teilnehmer bei beiden Prozessabläufen verglichen.

5.3.4.1 Nachrichtenanzahl des Koordinators

Hypothese: Die vom Koordinator benötigte Anzahl von Nachrichten ist beim SLS-Prozessablauf aufgrund der Erweiterungen höher als beim Vergleichsprozessablauf.

Resultat: Die Abbildungen 5.12a und 5.12.b verdeutlichen die Ergebnisse für die Simulation beider Prozessabläufe mit und ohne Knotenfehler (treten keine Knotenfehler auf, so ist die reale Knotenanzahl doppelt so groß und wird in Klammern dazu angegeben, vgl. Abschnitt 5.2.4.3).

Zunächst zeigt Abbildung 5.12.a die Unterschiede durch den Auftritt von Knotenfehlern. Es zeigt sich, dass beim Vergleichsprozessablauf keine Unterschiede des Nachrichtenverbrauchs bei Simulationen mit oder ohne Knotenfehler auftreten. Der Transaktionskoordinator beim Vergleichsprozessablauf benötigt bei jeder Netzdichte ca. 3 Nachrichtenpakete für seine Aufgaben.

Beim SLS-Prozessablauf hingegen werden Unterschiede deutlich. Wenn im MANET keine Knotenfehler auftreten, so benötigt der Koordinator in einem Netz mit 10 (20) Knoten ca. 14 Pakete und bei 20 (40) bis 50 (100) Knoten weniger als 10 Pakete. Anders stellt sich die Situation beim Auftritt von Knotenfehlern dar. In einem Netz mit 5 Knoten benötigt der Koordinator ca. 36

Pakete. Befinden sich 10 Knoten im Netz, so braucht der Koordinator ca. 11 Pakete. In Netzen mit 15 bis 25 Knoten reduziert sich der Nachrichtenverbrauch auf durchschnittlich ca. 5 bis 7 Nachrichten.

Da beim SLS-Prozessablauf beim Auftritt von Knotenfehlernen in Netzen mit wenig Knoten durch den Koordinator sehr viele Nachrichten benötigt werden, ist eine dahingehende Analyse sinnvoll. Abbildung 5.12.b zeigt die Ergebnisse des Vergleichs- und des SLS-Prozessablaufs. Bei Letzterem wird unterschieden im Hinblick auf die verwendete Verteilungsstrategie. Wird UnDirDiss benutzt, so entspricht die Anzahl der benötigten Nachrichten mit 3 Paketen der des Vergleichs-Prozessablaufs. Bei DirDissCUA ist der größte Nachrichtenbedarf erkennbar. In einem Netz mit 5 Knoten werden vom Koordinator im Schnitt ca. 72 Pakete verschickt. Bei Netzdichten von 15 bis 25 Knoten bewegt sich der Bedarf zwischen 10 und 15 Nachrichten. Wird die Kombination aus DirDissCA und DirDissCUA angewendet, reduziert das den Verbrauch (im Vergleich zur ausschließlichen Verwendung von DirDissCUA). Bei einem Netz mit 5 Knoten benötigt der Koordinator ca. 66 Pakete und bei 10 Knoten etwa 25 Pakete. Haben die Szenarien eine Dichte von 15 bis 25 Knoten, sendet der Koordinator durchschnittlich ca. 5 bis 10 Nachrichtenpakete.

Beurteilung: Abbildung 5.12.a verdeutlicht, dass der Koordinator des SLS-Prozessablaufs mehr Nachrichten benötigt als der Vergleichs-Prozessablauf. Damit wird die obige Hypothese bestätigt. Dies ist auch stimmig zu den Überlegungen in Abschnitt 5.3.2, da der SLS zwei zusätzliche Zustände beinhaltet und bei beiden Nachrichten verschickt, einmal an seine Nachbarn zur Abfrage der Ausfallwahrscheinlichkeit und einmal die Information über die ausgewählte Verteilungsstrategie an alle Transaktionsteilnehmer. Im Schnitt müssen daher $x+1$ Nachrichten beim SLS mehr verschickt werden, wobei x der Anzahl von Teilnehmern entspricht und die 1 der Anfrage nach den Ausfallwahrscheinlichkeiten der Nachbarn mittels Broadcast entspricht. Bei bis zu 8 Teilnehmern (vgl. dazu Tabelle 5.5), muss ein Koordinator daher ggf. ca. 9 Nachrichten mehr verschicken (vorausgesetzt, dass alle 8 Teilnehmer den Koordinator mit ihrem Votum erreicht haben). Treten Knotenfehler auf, ist zusätzlicher Nachrichtenaufwand in Abhängigkeit von der jeweiligen Strategie notwendig. Die Ergebnisse sind daher stimmig zu dem erwarteten Mehraufwand aufgrund der zusätzlichen Zustände.

Weiterhin sind die Unterschiede des Nachrichtenbedarfs des Koordinators im Hinblick auf Knotenfehler zu betrachten (vgl. Abbildung 5.12.a). Treten keine Knotenfehler auf, so werden in dünnen Netzen mit 10 (20) Knoten ca. 14 Nachrichten mehr benötigt als beim Vergleichs-Prozessablauf. Bei den anderen Dichten beträgt der Mehraufwand nur ca. 5 bis 10 Nachrichten. Der Aufwand bewegt sich hier also in einem erwartungsgemäßen Bereich aufgrund der zwei zusätzlichen Zustände.

Wenn Knotenfehler auftreten, sind die Netze während der Simulation deutlich dünner besetzt. Entsprechend werden in einem MANET mit 5 Knoten durchschnittlich ca. 36 Pakete benötigt. Es stehen wenier Knoten für die Speicherung zur Verfügung, die nicht an der Transkation beteiligt sind, so dass die Verteilung mehr Nachrichten erfordert. Bei den anderen Netzdichten sind die Ergebnisse ähnlich zu den Resultaten ohne Knotenfehler – zum Teil werden sogar weniger Nachrichten benötigt. Dies ist zum Beispiel bei den Knotenanzahl von 15 bis 25 der Fall. Der erhöhte Nachrichtenbedarf in dünnen MANETs und der verringerte Bedarf in dichteren Netzen erklärt die beobachtete Verbesserung bei der Unsicherheitszeit in Abschnitt 5.3.1. Um den benötigten Verteilungsgrad des Transaktionslogs im MANET zu erzielen, werden mehr Nachrichten benötigt als beim Vergleichs-Prozessablauf. Es liegt auch die Vermutung nahe, dass je nach Netzdichte die Verteilungsstrategien unterschiedlich gut funktionieren und Einfluss auf den Nachrichtenbedarf haben.

Abbildung 5.12.b zeigt die vom Koordinator benötigten Nachrichten in Abhängigkeit von der

verwendeten Verteilungsstrategie auf. Bei UnDirDiss benötigt der Koordinator ähnlich viele Nachrichten wie beim Vergleichs-Prozessablauf. Dies ist darauf zurückzuführen, dass eine Befragung der Nachbarn nach deren Ausfallwahrscheinlichkeit hier nicht notwendig ist. Der Koordinator arbeitet mit globalen Schätzungen. Für die Verteilung bei Knotenfehlern ist nur ein Broadcast notwendig.

Wird DirDissCUA angewandt, ist ein deutlicher Nachrichtenanstieg in dünnen Netzen erkennbar. Dies ist auf die Verteilung bei Knotenfehlern zurückzuführen. Der Koordinator broadcastet alle 0,2 Sekunden das Transaktionslog, bis er die benötigten n Bestätigungen erhalten hat. Mit diesem zusätzlichen Nachrichtenaufwand wird ein hoher Verteilungsgrad erzielt, so dass ausgefallene Teilnehmer schnell das Transaktionslog finden und nur kurz unsicher sind.

Die Kombination von DirDissCA und DirDissCUA führt zu einer Verbesserung im Vergleich zur reinen Nutzung von DirDissCUA. Lediglich in einem Netz mit 10 Knoten werden mehr Pakete benötigt. Bei allen anderen Netzdichten führt die zusätzliche Möglichkeit mit DirDissCA zu dieser Reduktion. Während bei DirDissCUA die Broadcasts so lange wiederholt werden, bis n Bestätigungen eingetroffen sind, wird bei DirDissCA zur Verteilung das Transaktionslog nur einmal an die n ausgewählten Knoten geschickt (in dichten Netzen ist die Annahme, dass diese erreichbar sind, so dass keine Wiederholung notwendig ist). Bei dichteren Netzen ist die Auswahl ausfallsicherer Knoten möglich. Dies führt insbesondere in dichteren Netzen ab 15 Knoten zur Nachrichtenverringerung und somit benötigt der Koordinator insgesamt weniger Nachrichten.

Diese Analyse des Nachrichtenbedarfs der einzelnen Strategien stützt also die Folgerungen bei der Untersuchung des Erfolgs der einzelnen Verteilungsstrategien. Je dichter das Netz ist, desto mehr steigt der Anteil der gerichteten Recovery an allen erfolgreichen Recoverys (vgl. Abschnitt 5.3.3.2). In dünnen Netzen hingegen funktionieren Verfahren wie DirDissCUA und UnDirDiss besser. Insbesondere DirDissCUA führt zu einem hohen Verteilungsgrad, der die in Abschnitt 5.3.1 beobachtete Verkürzung der Unsicherheitszeit erklärt.

Abschließend bleibt für diesen Untersuchungsteil festzuhalten: *Der Nachrichtenbedarf des Koordinators spiegelt die Effizienz der SLS-Verteilungsstrategien bei der Reduzierung der Unsicherheitszeiten sowie am Anteil der erfolgreichen Recoverys wider. Je dünner das Netz ist, desto mehr Nachrichten benötigt der Koordinator zur Verteilung – insbesondere bei DirDissCUA. Je dichter das* MANET *wird, desto besser funktionieren gerichtete Verfahren wie DirDissCA und DirDissCUA, so dass der Nachrichtenverbrauch verringert wird. Dies führt zur Gewährleistung der vereinbarten Garantien wie Abschnitt* 5.3.1 *zeigt (d.h. erfolgreiche Recovery im vereinbarten Zeitraum t_m).*

5.3.4.2 Nachrichtenanzahl der Klienten

Hypothese: Die von den Transaktionsteilnehmern benötigte Anzahl von Nachrichten wird durch die Verteilungsstrategien des SLS-Prozessablaufs im Vergleich zu dem Vergleichs-Prozessablauf erhöht.

Resultat: Abbildung 5.13 verdeutlicht die Unterschiede im Nachrichtenbedarf zwischen dem Vergleichsprozessablauf und dem SLS-Prozessablauf mit Knotenausfällen sowie ohne Knotenausfälle.

Der Transaktionsteilnehmer des Vergleichs-Prozessablaufs benötigt im Schnitt bei jeder Netzdichte ca. 3 bis 4 Nachrichten. Dies ist sowohl mit wie ohne Knotenfehler im MANET gegeben.

Für den Teilnehmer des SLS-Prozessablaufs sehen die Resultate etwas anders aus. Treten keine Knotenfehler auf, werden durchschnittlich immer ca. 4 bis 5 Nachrichtenpakete benötigt. Kommt es in den Szenarien jedoch zu Knotenfehlern, stellt sich die Situation aufseiten des Teilnehmers wie beim Koordinator anders dar. In Netzen mit 5 bis 15 Knoten werden von den Teilnehmern

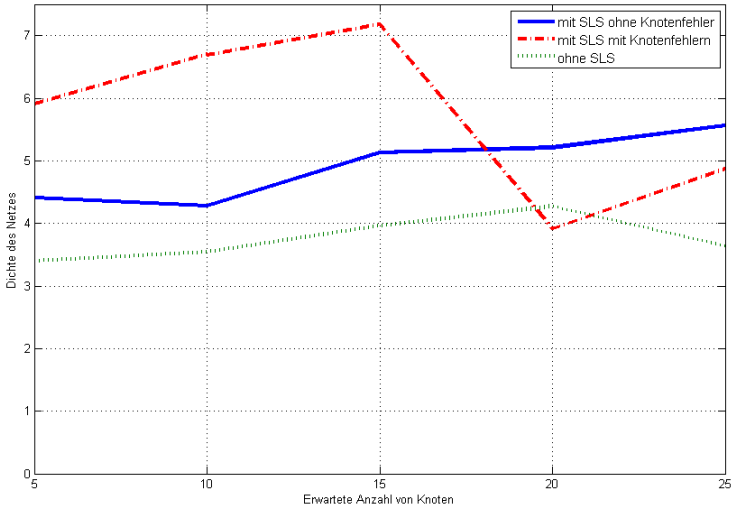

Abbildung 5.13: Vergleich des Nachrichtenbedarfs der Transaktionsteilnehmer allgemein

ca. 6 bis 7 Nachrichten benötigt. Wenn das Netz 20 Knoten enthält, reduziert sich der Bedarf auf etwa 3.5 Pakete, bei einer Netzdichte von 25 Knoten benötigt ein Teilnehmer ca. 5 Nachrichten.

Beurteilung: Die Resultate der Simulationen in Abbildung 5.13 zeigen, dass auch die Teilnehmer des SLS-Prozessablaufs bei einer Recovery mehr Nachrichten benötigen als beim Vergleichs-Prozessablauf. Die obige Hypothese kann damit bestätigt werden.

Beim SLS-Prozessablauf ist zusätzlich wieder eine Betrachtung im Hinblick auf die Wirkung von Knotenfehlern sinnvoll. Treten keine Knotenfehler auf, so benötigt ein SLS-Teilnehmer nur etwa eine Nachricht mehr als der des Vergleichs-Prozessablaufs.

Kommt es im Netz hingegen zu Knotenfehlern, stellt sich die Situation etwas anders dar. In dünnen MANETs mit 5 bis 15 Knoten werden etwa doppelt so viele Pakete wie beim Vergleichs-Prozessablauf benötigt. Befinden sich 20 bzw. 25 Knoten im Netz, so wird nur etwa eine Nachricht mehr benötigt. Damit ist der Verbrauch geringer als ohne den Auftritt von Knotenfehlern.

Der Mehraufwand an Nachrichten ist auf die Struktur der Recoveryverfahren zurückzuführen. Bei einer Recovery sendet ein Teilnehmer zunächst pro Versuch eine Anfrage an den Koordinator. Ergänzend dazu wird bei der ungerichteten Recovery ein Single-Hop-Broadcast an die Nachbarn gesendet. Bei der gerichteten Recovery werden Anfragen an die n vom Koordinator ausgewählten Knoten geschickt. Der in den Simulationen aufgetretene Mehraufwand entspricht also dem entworfenen SLS-Prozessablauf.

Insgesamt betrachtet reflektieren auch diese Ergebnisse die Effizienz der Verteilungsstrategien bei der Reduzierung der Unsicherheitszeit (vgl. Abschnitt 5.3.1). Durch die Verteilungsstrategien hat der Koordinator das Transaktionslog sehr häufig im Netz verteilt (vgl. Abschnitt 5.3.4.1) . Im Schnitt tritt daher nur der Mehraufwand an Nachrichten ein, der durch die Recovery-Strategien zu erwarten ist.

Die Ergebnisse dieser Analyse führen zu folgendem Schluss: *Transaktionsteilnehmer des SLS-Prozessablaufs brauchen etwa ein bis drei Nachrichten mehr als die des Vergleichs-Prozessablaufs. Dieser Mehraufwand ist auf die Recovery-Verfahren zurückzuführen, da ein Recovery-Versuch beim SLS-Prozessablauf mindestens eine zusätzliche Nachricht erfordert. Aufgrund des erzielten*

hohen Verteilungsgrades des Transaktionslogs durch die SLS-Strategien ist der Nachrichtenaufwand erwartungsgetreu. Die gewünschten Garantien können im Hinblick auf die Unsicherheitszeiten nach dem Auftritt von Fehlern gewährleistet werden.

5.3.5 Schlussfolgerungen

Die Ergebnisse der Simulationen in Abschnitt 5.3.1 zeigen, dass durch den SLS die Unsicherheitszeit von Transaktionsteilnehmern nach einem Ausfall deutlich verkürzt werden kann. Zudem können die Garantien zur erfolgreichen Recovery innerhalb eines vereinbarten Zeitfensters t_m zu etwa 99% bei den ausgewählten Parametern gewährleistet werden. Es zeigen sich jedoch Nebeneffekte dieser Verbesserung von Unsicherheitszeiten ausgefallener Teilnehmer.

Die Untersuchungen verdeutlichen, dass mehr als 90% aller Recoverys durch Kommunikationsfehler ausgelöst werden. Der größte Teil aller Fehlersituationen wird durch Kommunikationsfehler und nicht durch Knotenfehler ausgelöst.

Durch den SLS-Prozessablauf kommt es außerdem zu mehr Fehlern und somit zu mehr Recoverys als beim Vergleichs-Prozessablauf. Die zusätzlichen Fehlersituationen beim SLS-Prozessablauf entstehen durch zwei zusätzliche Zustände. Dadurch betreten Transaktionsteilnehmer später das Unsicherheitsfenster als beim Vergleichs-Prozessablauf. Entsprechend kommt es durch die kurzfristige Mobilität der Knoten häufiger zu Kommunikationsfehlern, da sich die Teilnehmer bis zum Unsicherheitsfenster weiter voneinander bewegt haben, als dies beim Vergleichs-Prozessablauf der Fall ist.

Der Anteil der einzelnen Verteilungsstrategien des SLS an den erfolgreichen Recoverys und somit an der Reduzierung der Unsicherheitszeiten ist unterschiedlich hoch. Insbesondere mit DirDissCUA sowie der Kombination aus DirDissCA und DirDissCUA können die Unsicherheitszeiten erheblich verkürzt werden. Besonders effizient ist dabei DirDissCUA. Mit diesem Verfahren wird das Transaktionslog durch wiederholte Broadcasts des Koordinators auf viele Knoten des Netzes verteilt. Durch diese Methodik ist sowohl in dichten wie in dünnen MANETs eine Reduzierung von Unsicherheitszeiten möglich. Somit ist dieses Verfahren nicht nur für dichte MANETs geeignet (vgl. zur Beschreibung des Verfahrens Abschnitt 4.2.2.2), sondern auch für dünne Netze.

DirDissCA wird nur in seltenen Fällen eingesetzt, so dass deren Anteil an den erfolgreichen Recoverys sehr gering ist. Es stehen nur selten ausreichend viele ausfallsichere Knoten zur Verteilung des Transaktionslogs bei den gewählten Parametern zur Verfügung. Die Simulationen zeigen, dass dieses Verfahren insbesondere für dichte MANETs geeignet ist, in denen die Knoten über Kontextbewusstsein verfügen. Damit verhält sich diese Strategie entsprechend den Überlegungen bei der Entwicklung (vgl. Abschnitt 4.2.2.1).

Mit UnDirDiss kommt es zur längsten Unsicherheitszeit von allen SLS-Verteilungsstrategien, allerdings ist diese immer noch deutlich kürzer als beim Vergleichs-Prozessablauf. Der Koordinator verteilt das Log nur einmal mittels Broadcast an seine Nachbarn. Die Knoten, die das Log erhalten, speichern dieses und senden es nach 60 Sekunden weiter (vgl. für Details Abschnitt 4.2.3). Mit diesem Verfahren wird über einen längeren Zeitraum hinweg der benötigte Verteilungsgrad des Transaktionslogs im Netz erzielt als mit DirDissCUA, wodurch es zur längeren Unsicherheitszeit kommmt. Für eine erfolgreiche Recovery ist daher ein längerer Zeitraum notwendig. Unterstützt wird diese Beobachtung dadurch, dass bei Anwendung dieser Strategie der Anteil der erfolgreichen Recovery-Anfragen höher als bei DirDissCUA ist. Die Simulationen führen hier zu dem Schluss, dass diese Strategie in dünnen MANETs nicht wie gewünscht zu besseren Resultaten als DirDissCUA im Hinblick auf die Unsicherheitszeiten ausgefallener Teilnehmer führen. Entgegen der Erwartung haben Kommunikationsfehler eine größere Auswirkung auf Fehlersituationen als Knotenfehler. Hier ist eine Anpassung des Timers notwendig, so dass das Transaktionslog nach einem Fehlerfall schneller im Netz verteilt wird.

Der Nachrichtenbedarf des Koordinators spiegelt die Effizienz der Verteilungsstrategien bei der Reduzierung von Unsicherheitszeiten wider. Bei der Verwendung von UnDirDiss benötigt der Koordinator des SLS-Prozessablaufs ebenso viele Nachrichten wie beim Vergleichs-Prozessablauf. Im Gegensatz dazu werden bei Anwendung von DirDissCUA sowie der Kombination aus DirDissCA und DirDissCUA deutlich mehr Pakete notwendig. Dieser Zusatzbedarf ist auf die Verteilungsstrategien zurückzuführen. DirDissCUA erfordert das wiederholte Broadcasten des Koordinators, bis er die erforderliche Anzahl von Bestätigungen erhalten hat. Gerade in dünnen Netzen dauert es aufgrund der geringen Anzahl von verfügbaren Knoten ggf. lange, bis dieses Ziel erreicht ist. Gleichzeitig erhalten auch Knoten das Transaktionslog, die sich eventuell nicht in der Übertragungsreichweite für das direkte Senden von Nachrichten befinden. Dementsprechend kann ein hoher Verteilungsgrad bei DirDissCUA erzielt werden, der zu der deutlichen Verkürzung der Unsicherheitszeit ausgefallener Teilnehmer führt.

Abschließend bleibt bezüglich der Evaluation des SLS festzuhalten, dass durch die entwickelten Verteilungsstrategien die Unsicherheitszeit von Transaktionsteilnehmern nach Fehlersituationen deutlich verkürzt werden kann. Allerdings ist diese Verbesserung zum Teil mit einem erheblichen zusätzlichen Nachrichtenaufwand verbunden. Außerdem haben sich bei den gewählten Parametern nicht alle Strategien in ihrer bisherigen Form als angemessen erwiesen. Für DirDissCA stehen in vielen Fällen nicht genug Knoten mit geringer Ausfallwahrscheinlichkeit zur Verfügung. Und mit UnDirDiss kann insbesondere in dünnen MANETs nicht innerhalb von wenigen Sekunden der gewünschte Verteilungsgrad erzielt werden, da der Timer zur Weiterverteilung zu groß zur Behandlung von Kommunikationsfehlern ist. DirDissCUA liefert die besten Ergebnisse bei der Verkürzung der Unsicherheitszeiten, ist jedoch auch mit dem höchsten Nachrichtenaufwand verbunden.

Kapitel 6

Erweiterung des SLS um Mobilitätsberücksichtigung

6.1 Motivation und Zielsetzung

Bei den bisherigen Untersuchungen wurden nur Szenarien betrachtet, in denen sich die Geräte innerhalb eines festen Bereichs - eines sog. *Clusters* - bewegen. In der Realität laufen oder fahren Benutzer jedoch zwischen verschiedenen Orten bzw. Clustern[1] hin und her (vgl. dazu auch die Charakteristiken von MANETs in Kapitel 2). Auf einem Universitätscampus halten sich Menschen beispielsweise abwechselnd in Instituten, Bibliotheken, Cafeterien, einer Mensa oder an anderen öffentlichen Plätzen auf und haben überall unterschiedlich lange Aufenthaltsdauern. Wie jedoch bereits die Auseinandersetzung mit Ausfällen in MANETs und das Fehlermodell in Abschnitt 2.3 sowie die Evaluation des SLS in Abschnitt 5.3 gezeigt haben, ist Mobilität ein wesentliches Problem bei der Kommunikation in einem mobilen Netzwerk. Der Begriff *Mobilität* meint hier die sog. *langfristige* Mobilität (vgl. dazu auch Abschnitt 3.7), d.h. die Aufenthaltsdauern in bestimmten Clustern sowie die Bewegung zwischen den Clustern. Diese ist deshalb von Bedeutung, da die vereinbarte garantierte Recoveryzeit bei Transaktionen nicht nur Sekunden umfassen kann, was in den Bereich der sog. *kurzfristigen* Mobilität fallen würde, sondern ggf. Minuten oder Stunden.

Bei der Durchführung einer Transaktion wäre es in den oben beschriebenen Szenarien hilfreich, wenn auch die Wahrscheinlichkeit dafür mit einbezogen würde, dass ein Gerät den aktuellen Cluster im Verlauf der garantierten Recoveryzeit t_m verlässt. Ein Koordinator sollte bei einer Transaktion nicht nur die Wahrscheinlichkeiten für Ausfälle aufgrund eines technischen Defekts oder erschöpfter Energieressourcen mit einbeziehen, sondern auch die Wahrscheinlichkeiten für eine sog. *Abwanderung* aus dem momentanen Cluster während der garantierten Recoveryzeit. Ziel ist es daher, die Fehlerbehandlung des SLS in MANETs derart zu erweitern, dass bei der Bestimmung der Zuverlässigkeiten sowohl durch den Koordinator wie durch die Transaktionsteilnehmer die Wahrscheinlichkeit für eine Abwanderung mit berücksichtigt wird.

Die im Kapitel 3 zu aktuellen Forschungsgebieten vorgestellten Ansätze zur Vorhersage von Mobilität sind für die Bedürfnisse des SLS nicht direkt anwendbar. Ein zentralisierter Ansatz, wie in Abschnitt 3.3.1 beschrieben, wird insbesondere für dichte MANETs zu Mehrkosten u.a. bezüglich der Nachrichtenlast führen. Lokale Ansätze aus Abschnitt 3.3.2 sind ebenfalls für kurzfristige Mobilität sinnvoll, jedoch nicht für langfristige. Der von I.-R. Chen u.a. vorgeschlagene Ansatz aus Abschnitt 3.3.3 ist eventuell für langfristige Mobilität anwendbar. Er ist jedoch sehr rechenaufwändig für kleine mobile Geräte, die solche Berechnungen nur bedingt durchführen können. Dagegen besteht die Möglichkeit, ein Verfahren zu entwickeln, das dem Ansatz von N. Marmasse aus Abschnitt 3.3.3 ähnlich ist. Ein großer Teil der in Abschnitt 3.3 beschriebenen Mechanismen kann jedoch zurzeit keine Vorhersagen im Hinblick auf die benötigte langfristige Mobilität treffen.

[1] Es wird im weiteren Kapiteld vielfach von *Clustern* anstatt von Aufenthaltsorten gesprochen, um zu verdeutlichen, dass damit beliebige Orte, Gebäude oder Plätze gemeint sein können, wo sich Gruppen von Menschen mit mobilen Geräten zusammenfinden können.

Um die Wahrscheinlichkeit für die Abwanderung aus einem Cluster bestimmen zu können, ist es erforderlich, dass Geräte die Anzahl ihrer Aufenthalte darin sowie deren jeweilige Dauer ermitteln. Mit diesen Informationen können die Geräte in die Lage versetzt werden, ihre individuelle Abwanderungswahrscheinlichkeit für einen Cluster zu schätzen. Diese Informationen sollten dem Koordinator bei der Auswahl einer Verteilungsstrategie zur Verfügung stehen. Gleichzeitig ist wünschenswert, dass die Nachrichtenlast des MANETs durch diese Erweiterung nicht erhöht wird. Außerdem muss berücksichtigt werden, dass es sich bei der ermittelten Aufenthaltsdauer häufig nur um eine Schätzung aufgrund der betrachteten Messungen handeln kann. Es sollte daher möglichst auch eine Bewertung der Schätzung im Hinblick auf Genauigkeit und Sicherheit vorgenommen werden.

Der SLS soll um ein entsprechendes Verfahren erweitert und anschließend simulativ evaluiert werden. Für die Evaluation wird das in Abschnitt 3.5.2 vorgestellte Area Graph-Based Mobilitätsmodell verwendet (vgl. [12, 49]), mit dem cluster-basierte Szenarien für NS2 generiert werden können. Zunächst wird auf das entwickelte Verfahren zur Schätzung der Abwanderungswahrscheinlichkeit aus Clustern eingegangen. Um zu untersuchen, ob dieses die gewünschte Funktionalität leisten kann, wird es im Vorfeld gesondert durch eine Simulation evaluiert. Anschließend wird im SLS die neue Funktionalität hinzugefügt. Die Ergebnisse dieser Simulationen bilden den Abschluss dieses Kapitels.

6.2 Bestimmung der Abwanderungswahrscheinlichkeit

Das im Rahmen der Arbeit entwickelte Verfahren zur Bestimmung der Abwanderungswahrscheinlichkeit aus Clustern erfordert zunächst einige Annahmen. Anschließend wird das Verfahren selbst vorgestellt. Darauf aufbauend wird auf einige wichtige Aspekte für die entsprechende NS2-Implementierung und die Simulationen eingegangen.

6.2.1 Ergänzende Annahmen zum Systemmodell

Eine Annahme bei der Entwicklung einer Methode zur Bestimmung der Abwanderungswahrscheinlichkeit aus Clustern ist, dass dieser die Exponentialverteilung (vgl. zu dieser Verteilung auch Abschnitt 4.2.1.1) zugrunde liegt. Von Bedeutung ist dabei die Eigenschaft der Gedächtnislosigkeit, d.h. dass die Wahrscheinlichkeit für eine Abwanderung zu jedem Zeitpunkt gleich ist. Es sind keine Kenntnisse über vergangene Zustände notwendig.

Weiterhin wird hier angenommen, dass mobile Geräte auch in realistischen Umgebungen jederzeit sowohl außerhalb wie innerhalb von Gebäuden ihre Position bestimmen können (vgl. z.B. [16]).

Neben diesen eher grundsätzlichen Annahmen sind einige weitere zu nennen. Diese sind keine notwendigen Einschränkungen, sondern vereinfachen im Rahmen dieser Arbeit lediglich die Untersuchung des entwickelten Verfahrens.

Zunächst wird hier angenommen, dass Transaktionen nur in Clustern stattfinden. Damit verbunden wird auch eine Recovery nur in dem Cluster gestartet, in dem die Transaktion stattgefunden hat. Anderenfalls müssten auch Informationen über Verbindungen zwischen den Clustern sowie die Bewegungsrichtung der Knoten berücksichtigt werden. Eine entsprechende Weiterentwicklung des hier vorgestellten Verfahrens ist grundsätzlich möglich, allerdings zunächst nicht berücksichtigt worden.

Bei den hier durchgeführten Experimenten verwenden die Knoten außerdem alle das kartesische Koordinatensystem. Der Grund dafür ist die Verwendung des in Abschnitt 3.5.2 vorgestellten Area Graph-Based Mobilitätsmodells, das von denselben räumlichen Vorgaben ausgeht und auf

dieser Grundlage die Szenarien für NS2 generiert. Außerdem bewegen sich die Knoten zunächst nur im 2-dimensionalen Raum, d.h. beispielsweise die Höhe oder Stockwerke von Gebäuden werden nicht berücksichtigt. Vereinfachend wird hier auch von quadratischen Clustern ausgegangen, deren Ränder senkrecht bzw. waagerecht sind[2].

6.2.2 Idee des Verfahrens

Die Idee zur Bestimmung der Abwanderungswahrscheinlichkeit aus einem Cluster basiert im Wesentlichen auf den folgenden Aspekten. Geräte im MANET müssen bestimmen können, wo sie sich befinden. Dazu werden allen Geräten dieselben räumlichen Informationen über die Cluster gegeben, in denen sie sich bewegen. Mit diesen Informationen in Form einer Art *Karte* können sie dann den Cluster ermitteln, in dem sie sich gerade befinden. Dadurch ist es möglich, dass die Geräte berechnen, wie lange sie in einem Cluster sind und wann sie diesen wieder verlassen. Ebenso kann mit dieser Methodik jedes Gerät zählen, wie oft es sich in einem Cluster aufhält, und somit letztlich seine Abwanderungswahrscheinlichkeit schätzen.

6.2.2.1 Berechnung der individuellen Abwanderungsrate durch mobile Geräte

Liegt der Wahrscheinlichkeit für die Abwanderung die Exponentialverteilung zugrunde, so ist die Schätzung des zugehörigen Parameters λ notwendig. Jeder Knoten hat ggf. mehrere Aufenthalte in einem Cluster und kann seine jeweilige Aufenthaltsdauer in dem Cluster bestimmen. Dazu wird ihm eine Karte zur Verfügung gestellt, anhand deren er seinen aktuellen Aufenthaltsort regelmäßig überprüfen kann. Ein Knoten kann damit die Anzahl seiner Aufenthalte pro Cluster und die durchschnittliche Aufenthaltsdauer darin bestimmen.

Diese von den Knoten ermittelten Daten kann man als Stichprobe aus einer Grundgesamtheit (hier eine Reihe von Daten aus einem bestimmten Untersuchungszeitraum) betrachten. Aus dieser Stichprobe kann man mit Hilfe statistischer Methoden Rückschlüsse auf die Gesamtheit als Ganzes – also die wirkliche Abwanderungsrate im Fall dieser Arbeit – ziehen. Hier ist das zentrale Problem, aus den beobachteten Daten auf den unbekannten Parameter λ der Exponentialverteilung zu schließen. Um einen einzelnen Parameter abschätzen zu können, ist die Anwendung eines sog. *Punktschätzers* sinnvoll, der eine Abschätzung der gesuchten Größe liefert. Eine Möglichkeit für einen solchen Schätzer ist die sog. *Maximum-Likelihood-Methode* bzw. *Methode der maximalen Stichprobenwahrscheinlichkeit*. Diese liefert eine einzelne Größe als Abschätzung für das gesuchte λ (vgl. [46]). Da dies der für die Knoten gewünschten Funktionalität zur Schätzung ihrer individuellen Abwanderungsrate entspricht, wird diese Methode im Folgenden vorgestellt.

Maximum-Likelihood-Schätzverfahren für den Parameter λ der Exponentialverteilung

Eine Stichprobenfunktion, mit der man den Wert eines unbekannten Parameters schätzen kann, wird als Schätzfunktion oder *Schätzer* von θ bezeichnet. Der durch die Schätzfunktion ermittelte Wert heißt entsprechend *Schätzung* (vgl. [46] S. 210).

Das Maximum-Likelihood-Schätzverfahren ist eine Schätzfunktion, die die *plausibelste Schätzung* liefert.

Seien also $X_1, \ldots, , X_n$ unabhängige exponentialverteilte Zufallsvariablen, die einen gemeinsamen unbekannten Erwartungswert θ haben. Diese Zufallszahlen werden nun gemessen und

[2] Bei Annahme eines Rechteckes R ist die Prüfung dahingehend, ob sich ein Punkt p darin befindet, in konstanter Zeit möglich (vgl. Abschnitt 6.2.3.1). Wird hingegen ein Polygon mit mehr Ecken angenommen, so sind komplexere Verfahren notwendig, z.B. aus dem Bereich der algorithmischen Geometrie (vgl. [30]).

entsprechen der Stichprobe aus einer Grundgesamtheit. Anhand dieser Werte soll nun der Parameter θ geschätzt werden. Für die n Zufallsvariablen ist die gemeinsame Wahrscheinlichkeitsdichte gegeben wie folgt (vgl. [46] S. 210), wobei θ aus den Messwerten $X_1, \ldots, X_n$geschätzt werden soll:

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) &= f_{X_1}(x_1) f_{X_2}(x_2) \ldots f_{X_n}(x_n) \\ &= \frac{1}{\theta} e^{-\frac{x_1}{\theta}} \frac{1}{\theta} e^{-\frac{x_2}{\theta}} \ldots \frac{1}{\theta} e^{-\frac{x_n}{\theta}}, \quad 0 < x_i < \infty,\ i = 1, \ldots, n \\ &= \frac{1}{\theta} e^{-\sum_1^n \frac{x_i}{\theta}}, \quad 0 < x_i < \infty,\ i = 1, \ldots, n \end{aligned} \tag{6.1}$$

wobei $f_{X_i}(x_i)$ die Wahrscheinlichkeitsdichtefunktion der Exponentialverteilung ist.

Da bei der Exponentialverteilung der Parameter λ der Kehrwert des Erwartungswertes ist, also $\lambda = \frac{1}{\theta}$, ergibt sich der Schätzer bzw. die gesuchte Likelihood-Funktion als:

$$\begin{aligned} L(\lambda) &= \prod_{i=1}^{n} f_{X_i}(x_i) \\ &= \prod_{i=1}^{n} \lambda e^{-\lambda x_i} \\ &= \lambda^n e^{-\lambda \sum_{i=1}^n x_i} \end{aligned} \tag{6.2}$$

und die logarithmierte Likelihood-Funktion als:

$$\begin{aligned} L'(\lambda) &= n \cdot \ln(\lambda) \\ &= \lambda \cdot \sum_{i=1}^{n} x_i \end{aligned} \tag{6.3}$$

wobei n jeweils der Anzahl von Messwerten bzw. dem Stichprobenumfang entspricht.

Einen Schätzwert für λ erhält man, indem man das Maximum der Funktion 6.3 bestimmt. Dazu wird die Ableitung nach λ berechnet und diese mit 0 gleichgesetzt:

$$\begin{aligned} 0 = \frac{\vartheta L'(\lambda)}{\vartheta \lambda} &= \frac{n}{\lambda} - \sum_{i=1}^{n} \lambda_i \\ &\Rightarrow \lambda' = \frac{n}{\sum_{i=1}^n x_i} \end{aligned} \tag{6.4}$$

wobei λ' anschließend dem gesuchten Schätzwert für λ entspricht.

Schätzung der Abwanderungsrate mit einem Maximum-Likelihood-Schätzer

Nimmt man die Überlegungen aus dem vorhergehenden Abschnitt als Grundlage, kann die gewünschte Abwanderungsrate mit Hilfe des Erwartungswertes für die Aufenthaltsdauer in einem Cluster geschätzt werden. Dazu wird die Anzahl der Aufenthalte pro Cluster benötigt sowie die durchschnittliche Aufenthaltsdauer darin (vgl. zur Ermittlung dieser Werte den Beginn dieses Abschnitts 6.2.2.1). Damit kann die Abwanderungsrate λ_j für die Aufenthaltsdauer in einem Cluster C_j wie folgt in direkter Analogie zu Formel 6.4 geschätzt werden als:

$$\lambda'_j = \frac{n_j}{\sum_{i=1}^{n_j} T_{i,j}} \tag{6.5}$$

wobei n_j der Anzahl der Aufenthalte im Cluster j mit $j = 1 \ldots m$ Anzahl der Cluster und $T_{i,j}$ Dauer des i-ten Aufenthaltes im Cluster j ist. Es wird deutlich, dass sich die geschätzte Abwanderungsrate λ'_j durch Mittelwertwertbildung aus den Aufenthaltsdauern der Knoten in einem Cluster ergibt.

Als Konsequenz folgt, dass die Aufenthaltsdauern in einem Cluster bestimmen, wie lange es dauert, bis ein annäherend korrekter Wert für das wirkliche Verhalten geschätzt worden ist. Je kürzer Knoten in einem Cluster sind und je schneller sie zurückkehren, desto schneller stehen ausreichend Werte zur Verfügung, die der echten Abwanderungsrate nahe kommen. Bleiben Knoten jedoch lange in einem Cluster und benötigen längere Zeit bis zur Rückkehr, wirkt sich das auch darauf aus, wie lange es bis zur Annäherung an die *echte* Abwanderungsrate dauert. Ob daher eine geschätzte Abwanderungsrate angenommen werden kann, sollte davon abhängen, wie genau und sicher der Schätzwert unter Berücksichtigung der bisherigen Aufenhalte ist. Auf diesen Aspekt wird daher im folgenden Abschnitt näher eingegangen.

6.2.2.2 Statistische Sicherheit des geschätzten Parameters λ'

Mit der zuvor vorgestellten Maximum-Likelihood-Methode wird für die echte Abwanderungsrate eines Knotens anhand der betrachteten Aufenthalte in einem Cluster ein Schätzwert λ' ermittelt. Dieser Wert liefert zunächst noch keine Aussage über die Genauigkeit und Sicherheit der Schätzung. Für die Verwendung bei der Bestimmung der Ausfallwahrscheinlichkeit eines Knotens ist es jedoch wichtig, dass die Abwanderungsrate ein größtmögliches Maß an Genauigkeit bzw. Vertrauenswürdigkeit besitzt. Dabei muss berücksichtigt werden, dass nicht nur die Abwanderungsrate – also hier der Mittelwert λ einer exponentialverteilten Grundgesamtheit – unbekannt ist, sondern auch deren Varianz σ^2. Eine explizite Aussage über die Richtigkeit der Aussage ist daher auf Basis der vorhandenen Informationen nicht möglich. Das aufgrund der bisherigen Aufenthalte geschätzte λ' kann immer noch (sowohl nach wenigen wie nach einer Vielzahl von Aufenthalten) stark vom wirklichen, aber unbekannten λ abweichen.

Für die Bewertung wäre daher z.B. hilfreich, eine Annäherung c an das *echte* λ zu finden, so dass mit *großer (statistischer) Sicherheit* gilt: $c \leq \lambda$. Eine ähnliche Problemstellung besteht bei der Bestimmung eines Vertrauens- oder Konfidenzintervalls für Schätzwerte. Von Bedeutung ist für das Schätzen der Abwanderungsrate insbesondere die untere Grenze des Intervalls. Denn bei dieser kann davon ausgegangen werden, dass die echte Abwanderungsrate mit einer vereinbarten statistischen Sicherheit γ mindestens so groß ist. Auf die Bestimmung eines Vertrauens- oder Konfidenzintervalls für den Mittelwert der Exponentialverteilung (und damit für die gesuchte Abwanderungsrate λ) wird daher im Folgenden näher eingegangen.

Bestimmung eines Vertrauens- oder Konfidenzintervalls für den unbekannten Parameter λ bei unbekannter Varianz σ^2

Sei $X_1, \ldots, X_n$ eine Stichprobe aus einer exponentialverteilten Gesamtheit. In Abschnitt 6.2.2.1 wird gezeigt, dass Formel 6.5 mit λ' die plausibelste Schätzung für die echte Abwanderungsrate λ darstellt. Dabei ist nicht davon auszugehen, dass man damit bereits das echte λ erhält, aber es wird *nahe dran* sein. Anstatt nun eine Punktschätzung zu machen, kann man auch ein Intervall $[c_l, c_r]$ bestimmen, das den gesuchten Parameter mit einer gewissen vorgegebenen Wahrscheinlichkeit γ umschließt.

Die vorgegebene Wahrscheinlichkeit γ wird auch als *statistische Sicherheit* bzw. *Vertrauens-* oder *Konfidenzniveau* bezeichnet. Entsprechend wird $\alpha = 1-\gamma$ die sog. *Irrtumswahrscheinlichkeit* genannt. Man kann dann die statistische Sicherheit $\gamma = 1 - \alpha$ wie folgt interpretieren: Die Intervallgrenzen c_l und c_r hängen von den konkreten Stichproben $x_1, \ldots, x_n$ ab und verändern sich – ebenso wie der Schätzwert λ' — von Stichprobe zu Stichprobe. Bestimmt man also 100

Vertrauensintervalle für den unbekannten Parameter λ aus 100 zufälligen Stichproben, dann enthalten ca. $100 \cdot \gamma$ Intervalle den *wahren* Wert und nur ca. $100 \cdot \alpha$ enthalten den wahren Wert *nicht*. Somit wird dabei in ca. $100 \cdot \gamma$ Fällen eine richtige und in ca. $100 \cdot \alpha$ Fällen eine falsche Entscheidung getroffen.

Die Intervallgrenzen c_l und c_r hängen dabei also nicht nur von den Stichproben ab, sondern auch von dem vorgegebenen Vertrauensniveau γ. Hier gilt, dass je größer γ gewählt wird, desto größer die Vertrauensintervalle werden. Bei einer großen geforderten statistischen Sicherheit γ wird also das dazugehörige Vertrauensintervall relativ breit werden. Als ein Maß für die Genauigkeit der Parameterschätzung kann dann die Länge des Vertrauensintervalls $l = c_r - c_l$ dienen. Je größer der Umfang der Stichprobe n wird, desto kleiner wird das Vertrauensintervall l. Die Auswahl eines geeigneten Konfidenzniveaus hängt dabei vom konkreten Problem ab (vgl. dazu [40]).

Um ein Konfidenzintervall für die unbekannte Abwanderungsrate λ mit $\gamma = 1 - \alpha$ zu erhalten, ist die Bestimmung eines Konfidenzintervalls für den unbekannten Mittelwert θ der Exponentialverteilung notwendig. Da $\lambda = \frac{1}{\theta}$ gilt, ist damit auch das Konfidenzintervall für das unbekannte λ bestimmbar. Wie in Abschnitt 6.2.2.1 gezeigt, ist der Maximum-Likelihood-Schätzer von θ das Stichprobenmittel $\sum_{i=1}^{n} \frac{X_i}{n}$, mit $X_1, \ldots, X_n$ unabhängig exponentialverteilten Zufallsvariablen. Für die Angabe eines Konfidenzintervalls für θ (und damit auch λ) sind im Vorfeld zwei weitere Annahmen notwendig. Zunächst kann gezeigt werden, dass $\sum_{i=1}^{n} X_i$ eine Gammaverteilung mit den Parametern n und $\frac{1}{\theta}$ hat (vgl. Folgerung dazu [46] S. 168). Wegen der Beziehung zwischen der Gamma- und der χ^2-Verteilung folgt daraus (vgl. dazu [46] S. 169-172)

$$\frac{2}{\theta} \sum_{i=1}^{n} X_i = \chi^2_{2n} \tag{6.6}$$

Sei nun ein beliebiges $\alpha \in (0, 1)$ gegeben. Dann gilt (vgl. zum Folgenden [46] S. 242)

$$p\left\{ \chi^2_{1-\alpha/2,2n} < \frac{2}{\theta} \sum_{i=1}^{n} X_i < \chi^2_{\alpha/2,2n} \right\} = 1 - \alpha \tag{6.7}$$

bzw. durch Umformung

$$P\left\{ \frac{2\sum_{i=1}^{n} X_i}{\chi^2_{\alpha/2,2n}} < \theta < \frac{2\sum_{i=1}^{n} X_i}{\chi^2_{1-\alpha/2,2n}} \right\} = 1 - \alpha \tag{6.8}$$

Mit diesen Annahmen ist dann das $100(1-\alpha)$-Prozent-Konfidenzintervall für θ gegeben durch

$$\theta \in \left(\frac{2\sum_{i=1}^{n} X_i}{\chi^2_{\alpha/2,2n}},\ \frac{2\sum_{i=1}^{n} X_i}{\chi^2_{1-\alpha/2,2n}} \right) \tag{6.9}$$

Sei nun $\theta_l = \frac{2\sum_{i=1}^{n} X_i}{\chi^2_{\alpha/2,2n}}$ die untere Grenze des Konfidenzintervalls und $\theta_r = \frac{2\sum_{i=1}^{n} X_i}{\chi^2_{1-\alpha/2,2n}}$ die obere Grenze, so dass sich daraus $\theta \in \{\theta_l,\ \theta_r\}$ ergibt. Dann folgt daraus mit der Annahme $\lambda = \frac{1}{\theta}$

$$\lambda \in \{\lambda_l,\ \lambda_r\} \tag{6.10}$$

Die *wahre* Abwanderungsrate λ liegt also mit einem Vertrauen von $\gamma \cdot 100\%$ in dem Intervall $\{\lambda_l,\ \lambda_r\}$.

Konservative Abschätzung der Abwanderungsrate λ

Mit dem zuvor beschriebenen Verfahren kann ein Konfidenzintervall für die Abwanderungsrate λ bestimmt werden. Für die Anwendung beim SLS ist insbesondere die untere Grenze des Intervalls $c = \lambda_l$ von Bedeutung. Diese sollte für das vereinbarte Konfidenzniveau γ die Eigenschaft $c \leq \lambda$ liefern. Somit erhält man mit Konfidenz γ eine Annäherung c an λ, so dass man davon ausgehen kann, dass c *mindestens* der gesuchten Abwanderungsrate entspricht. Das ist vor allem deshalb wichtig, da das echte λ nicht nur sehr viel größer sein kann als c, sondern ggf. genau diesem entsprechen kann. Bei Verwendung von c als Schätzung für λ kann man daher von einer *konservativen Schätzung* für λ sprechen. Wie groß die Abweichung vom echten λ ist, hängt dabei zum einem vom Stichprobenumfang ab und zum anderen vom ausgewählten Konfidenzniveau. Für die Anwendung beim SLS ist es wichtig, dass der Koordinator vor allem solche Knoten berücksichtigt, die ihre Abwanderungsrate mit größtmöglicher Sicherheit angeben können. Da sich mit dem vorgestellten Verfahren jeder weiteren Stichprobe eine größere Genauigkeit ergibt, scheint diese Methode angemessen zu sein.

Bevor eine Erweiterung des SLS vorgenommen wird, ist eine Simulation und Evaluation dieser Strategie im Vorfeld sinnvoll. Auf diese wird im folgenden Abschnitt 6.3.3 näher eingegangen.

Hinweis: Im Folgenden wird die *echte* Abwanderungsrate mit λ bezeichnet. Die Schätzung der echten Abwanderungsrate mit dem Maximum-Likelihood-Schätzer wird mit λ' abgekürzt und die untere Grenze des Konfidenzintervalls für das geschätzte λ' wird λ'_{conf} genannt.

6.2.3 Implementierung für NS2

Um zu untersuchen, ob mit der in Abschnitt 6.2.2 vorgestellten Methode die Abwanderungsraten aus Clustern wirklich ermittelt werden können, wird zunächst eine entsprechende Implementierung für NS2 evaluiert (zu NS2 vgl. Kapitel 4 und 5).

Für die Implementierung des entwickelten Verfahrens wird für den SLS ein eigener Agent verwendet. An jeden Knoten im NS2-Szenario wird ein solcher Agent gebunden. Jedem Agenten wird vor Beginn der Simulation die *Karte* des Szenarios zur Verfügung gestellt, wie im folgenden Abschnitt beschrieben ist. Anschließend kann die Abwanderungsrate bestimmt werden. Darauf wird in Abschnitt 6.2.3.2 näher eingegangen.

6.2.3.1 Einbeziehung einer Karte zur Positionsbestimmung

Basierend auf der Annahme, dass Knoten ihre eigene Position bestimmen können, wird durch das Einbeziehen einer Karte die Bestimmung des aktuellen Clusters möglich. Da dies die notwendige Voraussetzung dafür ist, dass die Knoten ihre Aufenthaltsdauer in einem Cluster ermitteln können, werden den Knoten die räumlichen Dimensionen der Cluster im Vorfeld zur Verfügung gestellt.

Mit diesen Informationen kann ein Knoten überprüfen, in welchem der ihm bekannten Cluster er sich befindet. Nimmt man an, dass alle Knoten das kartesische Koordinatensystem verwenden, können Cluster als eine Karte dargestellt werden, wie dies z.B. in Abbildung 6.1 der Fall ist. Die gewählte Darstellung ist lediglich eine Möglichkeit für die Umsetzung, die sich jedoch bei Implementierung für NS2 als sinnvoll erwiesen hat.

Bei der hier verwendeten Repräsentation der Cluster wird davon ausgegangen, dass jeder Knoten zunächst einen festen sog. *Referenzpunkt* (x_1, x_2) im Koordinatensystem hat. Dieser stellt die linke untere Ecke des Clusters dar. Hinzu kommen die *Länge* und die *Breite* des Clusters. Ein quadratischer Cluster j kann somit durch das 3-Tupel $((x_1, x_2), \text{Länge}, \text{Höhe})$ vollständig beschrieben werden. Das Verschieben eines Clusters ist dann sehr einfach durch Veränderung des Referenzpunktes möglich.

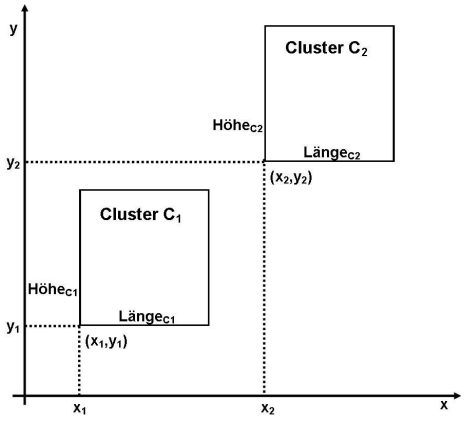

Abbildung 6.1: Repräsentation von Clustern im kartesischen Koordinatensystem

```
1  bool isInCluster(double currentX, double currentY,
2                   double clusterX, double clusterY,
3                   double clusterLength,
4                   double clusterHeight){
5
6          // Knoten befindet sich im gegebenen Cluster
7          if(currentX > clusterX &&
8             currentX < (clusterX+clusterLength) &&
9             currentY > currentY &&
10            currentY < (clusterY+clusterHeight)){
11            return true;
12         }
13
14         // Knoten befindet sich nicht im gegebenen Cluster
15         else return false;
16 }
```

Abbildung 6.2: Algorithmus zur Prüfung auf Position innerhalb eines Clusters

Mit den Informationen über die Lage eines Clusters j in der Ebene kann jeder Knoten k in konstanter Zeit prüfen, ob er sich darin befindet. Dazu wird berechnet, wie sich seine aktuelle Position zu den Dimensionen von j verhält. Es ist dazu nur nötig, die Koordinaten (x_k, y_k) der aktuellen Position des Knotens k mit den Koordinaten von j zu vergleichen, wie in Abbildung 6.2 dargestellt. Dies Verfahren muss zur Bestimmung des aktuellen Clusters für alle in der Karte definierten Cluster durchgeführt werden.

Zur Übergabe der Karte an NS2 als ein Parameter wird diese zu einem String formatiert, in dem Cluster wie folgt *codiert* sind: *x-y-Länge-Höhe*. Die Trennung der Cluster erfolgt durch Leerzeichen. Mit diesen Informationen legt jeder Knoten zunächst ein Objekt zu dem Cluster an. In diesem werden dann die Anzahl der Besuche sowie die Summe der Aufenthaltsdauern gespeichert.

6.2.3.2 Ermittlung der Abwanderungsrate

Durch einen Parameter bei der Implementierung kann spezifiziert werden, wann die Messungen zur Ermittlung der Abwanderungsrate beginnen sollen (in der Regel wird das mit dem Start der Simulation sein). Zunächst weiß ein Knoten nicht, in welchem Cluster er sich befindet. Daher muss

er alle Cluster mit der in Abschnitt 6.2.3.1 vorgestellten Methode überprüfen, ob er sich darin befindet. Hat er den entsprechenden Cluster C_j gefunden, merkt er sich diesen Eintrittszeitpunkt t_{enter}. In regelmäßigen Abständen (hier 5 Sekunden) wird überprüft, ob sich der Knoten noch in C_j befindet. Wenn ja, muss kein anderer Cluster überprüft werden. Wird allerdings festgestellt, dass sich der Knoten nicht mehr im angenommenen Cluster C_j befindet, müssen die anderen Cluster C_i mit $i = 1 \ldots m$ und $i \neq j$ überprüft werden. Evtl. kommt kein Cluster in Frage, wenn sich ein Knoten auf dem Weg zwischen zwei Clustern befindet. Wurde ein Cluster verlassen, wird außerdem der Zeitpunkt t_{leave} bestimmt. Damit kann ein Knoten für den Cluster C_j die Anzahl der Aufenthalte erhöhen und zusätzlich die Dauer des i-ten Aufenthaltes als $T_{i,j} = t_{leave} - t_{enter}$ auf die Summe aller Aufenthaltsdauern addieren.

6.2.4 Simulation

Für die Simulation des entwickelten Verfahrens zur Bestimmung der Abwanderungsrate λ durch die Knoten des MANETs ist zunächst die Auswahl eines Anwendungsszenarios notwendig. Nicht jedes Szenario ist geeignet, um das entwickelte Verfahren zu evaluieren. Dieses sollte möglichst so gewählt werden, dass damit auch die Simulation des um dieses Verfahren erweiterten SLS durchgeführt werden kann.

6.2.4.1 Auswahl von Szenarien und Bestimmung der Abwanderungsraten

Um die Idee des entwickelten Verfahrens zu evaluieren, ist ein Szenario mit zwei Clustern ausreichend. Dies entspricht einem Modell, in dem das Abwandern eines Knoten der Bewegung von einem Cluster in einen beliebigen anderen entspricht. Das gesamte restliche Netz wird dabei als zweiter Cluster repräsentiert. Die Entfernung zwischen den Clustern wird dabei so gewählt, dass Knoten aus dem einen Cluster keinen in dem anderen Cluster erreichen können. Im Hinblick auf Knotengeschwindigkeit und Pausenzeit werden die in Abschnitt 5.2 vorgestellten Parameter übernommen. Lediglich die Dauer der Simulationen wird für diese *Vorsimulation* auf 300000 Sekunden erhöht, um zu untersuchen, ob sich die Knoten entsprechend dem jeweiligen Erwartungswert für jeden Cluster verhalten. Daher ist die Betrachtung eines längeren Zeitraums als in den bisherigen Simulationen sinnvoll.

Es werden die in den nächsten beiden Abschnitten vorgestellten Szenarien ausgewählt, für die dann entsprechend Szenariendateien für NS2, basierend auf dem Area Graph-Based Mobilitätsmodell, generiert werden (vgl. dazu Abschnitt 3.5.2).

Szenario 1: Zwei Cluster mit unterschiedlichen Aufenthaltsdauern

Für die Simulation und Evaluation des entwickelten Verfahrens wird das folgende Katastrophenszenario angenommen. Ein solches Szenario kann z.B. ein Gebäude oder eine Ortschaft nach einem Feuer, einer Überflutung oder einem Erdbeben sein. In dieses Gebiet laufen die Helfer hinein und verlassen es nach ca. einer Viertelstunde wieder. Nach ca. 45 Minuten außerhalb dieses Gebietes kommen die Helfer wieder zurück.

Hier wird das Katastrophengebiet als ein Cluster C_1 betrachtet und die restliche Umgebung außerhalb als ein zweiter Cluster C_2. Die Wahrscheinlichkeit für die Bewegung von einem Cluster in den anderen (die sog. Abwanderung) wird als exponentialverteilt angenommen. Der durchschnittliche Erwartungswert für die Aufenthaltsdauer in C_1 beträgt 15 Minuten, so dass sich für das entsprechend für die Abwanderungsrate $\lambda_{C_1} = 0.001111111$ ergibt. Für den Cluster C_2wird als durchschnittlicher Erwartungswert für die Aufenthaltsdauer 45 Minuten angenommen, was der Abwanderungsrate $\lambda_{C_2} = 0.00037037$ entspricht (für die Berechnungen vgl. das Beispiel in Abschnitt 5.2.4.5).

Szenario 2: Zwei Cluster mit gleicher Aufenthaltsdauer

Eine andere Möglichkeit für ein Katastrophenszenario sieht so aus, dass die erwartete Aufenthaltsdauer in beiden Clustern gleich ist. Die Helfer laufen z.B. in ein Gebäude oder ein Katastrophengebiet für ca. 15 Minuten hinein und halten sich dann außerhalb ebenfalls ca. 15 Minuten auf. Es wird hier ebenfalls angenommen, dass die Abwanderungswahrscheinlichkeit von dem einen Cluster C_1 in den anderen Cluster C_2 exponential verteilt ist. Aufgrund dieser Annahmen ergeben sich die Abwanderungsraten $\lambda_{C_1} = 0.001111111$ und $\lambda_{C_2} = 0.001111111$.

6.2.4.2 Auswahl von Konfidenzniveaus

In Abschnitt 6.2.2.2 wird darauf hingewiesen, dass das Konfidenzintervall länger wird, wenn das gewünschte Konfidenzniveau γ größer wird. Tests haben jedoch gezeigt, dass für die Ermittlung der Abwanderungsrate keine großen Unterschiede bei den Konfidenzniveaus $\gamma_1 = 90\%$, $\gamma_2 = 95\%$ und $\gamma_3 = 99\%$ sichtbar werden. Im Rahmen dieser Arbeit wird deshalb für die weiteren Simulationen ein Konfidenzniveau von $\gamma = 95\%$ gewählt.

6.2.4.3 Resultate der Simulationen

Abbildung 6.3 verdeutlicht, dass im Laufe der Zeit mit zunehmender Anzahl von Tests eine gute Annäherung für die echte Aufenthaltsdauer in einem Cluster geschätzt wird. In den ersten ca. 50000 Sekunden ist dieser Schätzwert aufgrund einer ungenügenden Anzahl von Tests noch sehr ungenau. Die Zeit bis zur Ermittlung einer verhältnismäßig guten Genauigkeit kann bis zu 100000 Sekunden betragen, wie Simulationen gezeigt haben.

Wie Abbildung 6.4 darstellt, kann aufgrund der Generierung von exponentialverteilten Zufallsvariablen auch ein vom Erwartungswert abweichender Mittelwert der Aufenhalte entstehen. Der dazugehörige Knoten bewegt sich während der gesamten Simulationszeit nie dem Erwartungswert entsprechend, sondern hält sich ggf. durchschnittlich länger oder kürzer in einem Cluster auf. Prinzipiell entspricht dies einem realistischen Verhalten, da in der Regel davon auszugehen ist, dass ein realer Benutzer auch ein individuelles Bewegungsverhalten aufweist.

Mit Hilfe der unteren Grenze des Konfidenzintervalls $\{\theta_l, \theta_r\}$ für den geschätzten Mittelwert θ' soll eine gewisse statistische Sicherheit gewährleistet werden. Für die von einem Knoten ermittelte durchschnittliche Aufenthaltsdauer θ' sollte also mit der statistischen Sicherheit γ gelten: $\theta_l \leq \theta$, wobei θ_l die untere Grenze des Konfidenzintervalls und θ die echte Aufenthaltsdauer ist. Betrachtet man dazu beide Abbildungen 6.3 und 6.4, so stellt man fest, dass dadurch im ersten Fall die vom Knoten gelieferten Ergebnisse etwas schlechter werden. Die zweite Abbildung zeigt jedoch, dass mit der untere Grenze des Konfidenzintervalls der echte Erwartungswert besser bestimmt wird als mit dem geschätzten Mittelwert. Bei diesen Fällen bietet das Konfidenzintervall also eine bessere Abschätzung der Knotenbewegung als der geschätzte Mittelwert.

6.2.5 Schlussfolgerungen

Wie die Simulationsergebnisse zeigen, ist es nicht immer möglich, einen exakten Wert für die Abwanderungsrate λ zu bestimmen. Prinzipiell zeigen die Ergebnisse jedoch, dass die Knoten im Laufe der Zeit in der Lage sind, eine recht gute Annäherung zu schätzen. Es treten sowohl Fälle auf, in denen der gelernte in etwa dem *echten* Wert entspricht, wie auch Fälle, bei denen der gelernte Wert entweder größer oder kleiner ist. Die Verwendung der unteren Grenze c des Konfidenzintervalls λ'_{conf} als konservative Abschätzung bietet daher eine gute Möglichkeit, einen Wert zu verwenden, bei dem man davon ausgehen kann, dass das wirkliche λ mindestens so groß ist.

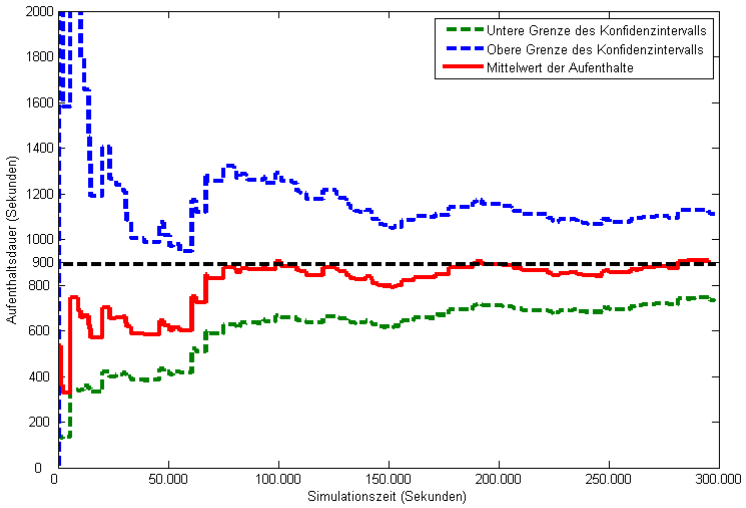

Abbildung 6.3: Annähernd exakte Schätzung der Aufenthaltsdauer in einem Cluster mit einem Erwartungswert von 900 Sekunden

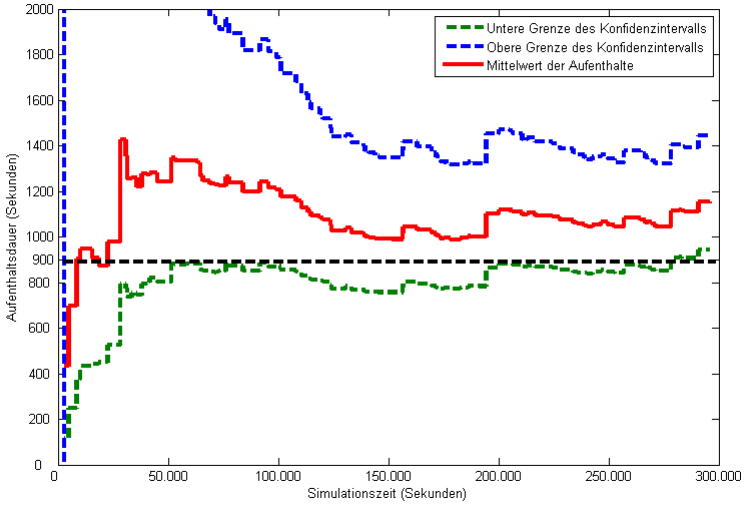

Abbildung 6.4: Ungenaue Schätzung der Aufenthaltsdauer in einem Cluster mit einem Erwartungswert von 900 Sekunden

Außerdem ist festzuhalten, dass nach dem bisherigen Simulationszeitraum eine gewisse Abweichung bestehen bleibt. In diesen Fällen sind noch nicht genügend Stichproben vorhanden, um einen genaueren Wert für λ zu liefern. Hier kann ggf. eine weitere detaillierte Untersuchung Auskunft darüber liefern, welchen Einfluss die erwartete Aufenthaltsdauer auf den Zeitraum hat, der notwendig ist, um einen möglichst genauen Wert für λ zu schätzen.

Für die Verwendung der bisherigen Parameter in den Simulationen mit dem erweiterten SLS deutet das Ergebnis an, dass es bei den beiden gewählten Szenarien grundsätzlich möglich ist, im Laufe der Simulationszeit eine gute Annäherung an die *echten* Abwanderungsraten zu schätzen. Es bleibt jedoch zu bedenken, dass gerade in den ersten 50000 bis 100000 Sekunden die geschätzten Werte ggf. noch weit entfernt vom echten Verhalten der Knoten sind. Durch Verwendung von λ'_{conf} wird dieses Problem noch verstärkt. Gleichzeitig kann jedoch mit der vereinbarten Konfidenz γ davon ausgegangen werden, dass der echte Wert mindestens so groß wie der geschätzte ist.

Der Vorteil bei der Verwendung dieser Methode ist, dass der Koordinator Knoten mit zu kurz geschätzter Aufenhaltsdauer ablehnen wird. Mit instabilen Knoten wird für den vereinbarten garantierten Recoveryzeitraum t_m das $p_{retrieval_{app}}$ nicht gewährleistet werden können. Das bedeutet, erst wenn genügend Knoten eine ausreichend lange Aufenthaltsdauer mit gewisser statistischer Sicherheit garantieren können, wird der Koordinator die gerichtete Verteilung verwenden. Bis dahin befinden sich die Knoten in einer Art *Lernphase*. Die Dauer dieser Phase richtet sich nach der statistischen Sicherheit der Schätzung, die sich mit jedem neuen Aufenhalt (also quasi einer weiteren Stichprobe, vgl. dazu Abschnitt 6.2.2) verändern kann. Das Lernverhalten und die Dauer des *Lernens* sind also wie das Bewegungsverhalten ein individueller Prozess. Dies führt dazu, dass ein Knoten jederzeit in das MANET hinzukommen und an Transaktionen teilnehmen kann. Es ist keine Vorkonfiguration im Vorfeld notwendig für das *Lernen* der Abwanderungsrate. Sobald der Knoten so viele Informationen über sein Bewegungsverhalten gesammelt hat, dass er mit der vereinbarten statistischen Sicherheit eine Schätzung abgeben kann, wird er vom Koordinator berücksichtigt werden. Dann kann der Koordinator diesen Knoten bei der Verteilung auf eine Gruppe von n ausfallsicheren Knoten verwenden. Ist die Schätzung zu unsicher, so wird angenommen, dass der Knoten über kein Kontextbewusstsein verfügt.

6.3 Erweiterung des SLS um die neue Funktionalität

Zur Ergänzung des SLS um die neue Funktionalität ist es zunächst sinnvoll, einige Annahmen zu treffen, die auch die spätere Implementierung beeinflussen. Anschließend werden in Abschnitt 6.3.2 die wesentlichen Erweiterungen an den Prozessabläufen vorgestellt. Für die Simulation des um die Schätzung der Abwanderungsrate erweiterten SLS ist auch die Erweiterung der Implementierung notwendig. Auf einige Details dazu wird in Abschnitt 6.3.3 eingegangen.

Generell ist darauf hinzuweisen, dass der grundlegende Simulationsablauf aus Abschnitt 5.2 beibehalten wird. In diesem und im folgenden Abschnitt 6.4 wird nur auf notwendige Veränderungen bzw. Ergänzungen eingegangen.

6.3.1 Annahmen und Voraussetzungen

Eine der wichtigsten Annahmen dieses Teils ist, dass Transaktionen nur innerhalb von Clustern stattfinden. Da im Rahmen dieser Erweiterung davon ausgegangen wird, dass jeder Knoten seine aktuelle Position bestimmen kann und über die Clusterdimensionen informiert ist, bedeutet dies keine Einschränkung. Damit zusammenhängend erfolgen Broadcasts sowie die Weiterleitung von Nachrichten nur dann, wenn ein Knoten sich in einem Cluster befindet. Somit sollen alle

mit einer Transaktion zusammenhängenden Aktionen auf den dazugehörigen Cluster beschränkt werden. Gleichzeitig erfolgt der Nachrichtenaustausch prinzipiell nur innerhalb von Clustern, was außerdem dazu führen soll, dass kein unnötiger Datentransfer stattfindet. Ebenso wird hier vorausgesetzt, dass Recoverys nur in dem Cluster gestartet werden, in dem zuvor die Transaktion stattgefunden hat. Dies ist insbesondere deshalb sinnvoll, da bei einer Recovery in dem nicht zur Transaktion gehörenden Cluster je nach Aufenthaltsdauer darin eine enorme Nachrichtenlast die Folge ist. Es bestünde jedoch wenig Aussicht auf Erfolg, da dort keine Daten verteilt werden.

6.3.2 Ergänzung der SLS Prozessabläufe

Basierend auf den Überlegungen aus Abschnitt 6.3.1 können für die einzelnen Prozessabläufe des Koordinators, der Teilnehmer und der Assistenzknoten (sog. SLS-Knoten) jeweils einige konkrete Änderungen formuliert werden, die dann implementiert werden.

6.3.2.1 Prozessablauf des Koordinators

Auf der Seite des Koordinators ist die wichtigste Erweiterung, dass dieser bei der Berechnung einer Ausfallwahrscheinlichkeit zusätzlich die Abwanderungsrate berücksichtigt. Bei DirDissCA in dichten MANETs ist es dementsprechend erforderlich, dass ihm die Knoten diese Information zur Verfügung stellen. Im Fall von DirDissCUA und UnDirDiss hingegen muss er global für den aktuellen Cluster eine Abwanderungsrate schätzen. Im Rahmen dieser Arbeit wird ihm dazu der echte Wert zur Verfügung gestellt.

Zur Einbeziehung der Abwanderungsrate muss die Formel 4.8 aus Abschnitt 4.2.1.2 für die Berechnung der Ausfallwahrscheinlichkeit im garantierten Recoveryzeitraum t_m entsprechend angepasst werden:

$$p_{failure} = 1 - (1 - e^{\lambda_{tech} \cdot t_m}) \cdot (1 - e^{\lambda_{energy} \cdot t_m}) \cdot (1 - e^{\lambda_{mobile} \cdot t_m}) \tag{6.11}$$

wobei $\lambda_{mobile} = c$, also der unteren Grenze des gesuchten Konfidenzintervalls entspricht. Diese Berechnung wird (wie auch in den bisherigen Simulationen) für die Ausfallwahrscheinlichkeiten der einzelnen Teilnehmer vorgenommen und für die globale Ausfallwahrscheinlichkeit aller Teilnehmer.

6.3.2.2 Prozessablauf der Teilnehmer

Bei den Teilnehmern an einer Transaktion wird dahingehend eine Erweiterung vorgenommen, dass diese sich beim Start einer Transaktion den aktuellen Cluster merken. Kommt es dann im Verlauf der Transaktion zu einem Ausfall, so können sie anschließend überprüfen, ob sie im *richtigen* Cluster sind für die Recovery. Diese wird nur in dem Cluster gestartet, in dem die Transaktion gestartet wurde. Befinden sich die Teilnehmer nicht in dem zur Transaktion gehörigen Cluster, so setzen sie z.B. beim Feuern ihres Recovery-Timers diesen wieder neu. Damit bleibt der Unsicherheitszustand erhalten, aber eine Nachricht wird erst gesandt, wenn sich der Knoten in dem Cluster befindet, in dem Aussicht auf Erfolg für die Recovery besteht.

6.3.2.3 Assistenzknoten (sog. *SLS-Knoten*)

Die SLS-Knoten, also die Knoten im MANET, die nicht direkt an der Transaktion beteiligt sind, tragen am meisten für die Bestimmung der Abwanderungsrate bei. Sie lernen mit dem in Abschnitt 6.2.2.1 vorgestellten Verfahren die Anzahl ihrer Aufenthalte in jedem Cluster und die Summe der Aufenthalte darin. Mit diesen Informationen können sie dann λ'_{conf} bestimmen. Dazu verwenden die Knoten Formel 6.9, die in Abschnitt 6.2.2.2 hergeleitet wird. Erhalten diese

Knoten dann eine Anfrage nach ihrer Ausfallwahrscheinlichkeit von einem Koordinator, können sie ihm diesen Wert zusammen mit den bereits bekannten Ausfallraten λ_{tech} und λ_{energy} zusenden. Liegt zu Beginn der Simulationszeit noch keine Information über die Aufenthaltsdauer im aktuellen Cluster vor, so antworten die Knoten nicht auf die Anfrage des Koordinators, da sie ihre Ausfallwahrscheinlichkeit nicht vollständig bestimmen können. Ihr Verhalten entspricht dann dem von Knoten ohne Kontextbewusstsein.

Wichtig für das Verhalten dieser Knoten ist außerdem, dass Datenpakete vom Koordinator, die für die Verteilung verschickt werden, nur dann weitergeleitet werden, wenn sie sich noch in einem Cluster befinden. Dasselbe gilt, wenn zusätzlich die gespeicherten Transaktionslogs regelmäßig *gepusht* werden. Damit soll verhindert werden, dass Nachrichten über die Verbindung zwischen den Clustern in weitere Cluster gelangen.

6.3.3 Implementierung für NS2

Bei der Erweiterung der Implementierung für NS2 um die Positionsbestimmg und die Schätzung einer Abwanderungsrate durch die einzelnen Knoten wird im Wesentlichen die Idee aus Abschnitt 6.3.2 umgesetzt. Die Teile für die in Abschnitt 6.3.3 vorgestellte Vorsimulation zur Bestimmung der Anzahl der Aufenthalte in einem Cluster und der Dauer der Aufenthalte darin können dabei wieder verwendet werden. Für die Teilnehmer an einer Transaktion sind nur marginale Änderungen notwendig. Sie müssen lediglich zusätzlich speichern, in welchem Cluster die Transaktion gestartet wird. Außerdem muss jeweils überprüft werden, ob man sich im relevanten Cluster befindet, wenn eine Nachricht geschickt wird. Diese Information erhalten sie von ihrem SLS-Agenten.

Die wesentlichsten Erweiterungen sind bei den SLS-Agenten notwendig. Diese prüfen regelmäßig, wo sie sich befinden. Findet eine Änderung statt, z.B. Eintritt oder Verlassen eines Clusters, so speichern sie zum einen selbst diese Information für die Berechnung der Abwanderungsrate. Zusätzlich benachrichtigen sie auch alle bei ihnen angemeldeten Transaktionsagenten über die Veränderung. Dies geschieht in Analogie zum dem in Abschnitt 5.2.3.4 vorgestellten Verfahren, so dass das bisherige Implementierungsmodell entsprechend erweitert werden kann (vgl. dazu Abbildung 5.2). Der SLS-Agent ruft bei jedem der Transaktionsagenten je nach Ereignis die entsprechende Methode auf. Betritt er einen Cluster, wird die Methode `void enterCluster(int newCluster, double globalChurn)` verwendet. Dabei werden die Transaktionsagenten über den neuen Cluster und die globale Abwanderungsrate informiert. Diese Information ist für die Teilnehmer irrelevant, jedoch für den Koordinator einer Transaktion wichtig. Ähnlich wird beim Verlassen eines Clusters mit der Methode `void leaveCluster(int leftCluster, int newLocation, double globalChurn)` verfahren. Die Agenten werden darüber informiert, welcher Cluster verlassen wurde und welches der neue Aufenthaltsort ist. Zusätzlich wird für die Koordinatoren wieder die globale Abwanderungsrate übergeben.

Zusätzlich muss durch die SLS-Agenten die untere Grenze des Konfidenzintervalls berechnet werden. Diese Berechnungen werden bei Anfrage eine Koordinators nach der Ausfallwahrscheinlichkeit des Knotens vorgenommen. Dies hat den Vorteil, dass der Koordinator keine zusätzlichen umfangreichen Berechnungen durchführen muss. Basierend auf den bisher ermittelten Informationen zu der Anzahl von Aufenthalten in einem Cluster und der Summe der Aufenthaltsdauern kann die untere Grenze des Konfidenzintervalls bestimmt werden. Bei den Vorsimulationen von Abschnitt 6.2.4 zeigt sich bereits, wie viele Aufenthalte im Simulationszeitraum pro Cluster etwa auftreten und welches Konfidenznievau sich anbietet als geeigneter Schätzwert für die Abwanderungsrate. Dementsprechend wird eine ausreichende Menge von Werten der Chi-Quadrat-Verteilung generiert und den SLS-Agenten statisch in einem Array zur Verfügung gestellt. Abhängig von der Anzahl der bisher aufgetretenen Aufenthalte können dann die relevanten Werte

verwendet werden. Durch das Bereitstellen dieser Informationen kann der Rechenaufwand für die Knoten minimiert werden.

6.4 Simulation des erweiterten SLS

Vor den eigentlichen Simulationen ist die Auswahl von Parametern notwendig. Darauf wird in diesem Abschnitt als erstes eingegangen. Anschließend werden die Ergebnisse der mit diesen Parametern durchgeführten Simulationen des erweiterten SLS vorgestellt.

6.4.1 Auswahl geeigneter Parameter

Zur Simulation des erweiterten SLS müssen wie in Kapitel 5 einige Parameter im Vorfeld bestimmt werden. Ein Teil der zuvor verwendeten Parameter kann wieder verwendet werden. Bei einigen müssen jedoch Veränderungen vorgenommen. Dies betrifft zum einen die MANET-Szenarien, zum anderen den garantierten Recovery-Zeitraum t_m sowie die Anzahl der Teilnehmer. Auf deren Auswahl wird im Folgenden näher eingegangen.

6.4.1.1 Szenarien und Abwanderungsraten

Bei der Simulation des erweiterten SLS werden die beiden in Abschnitt 6.2.4.1 vorgestellten Szenarien verwendet. Durch die unterschiedlichen erwarteten Aufenthaltsdauern bei beiden Szenarien wird die Bewegungsdynmaik zwischen den Clustern beeinflusst, was zusammen mit dem garantierten Recoveryzeitraum t_m von Bedeutung ist (vgl. Abschnitt 6.4.1.3).

Die Anzahl der Knoten wird im Vergleich zur Simulation in Kapitel 5 verdoppelt, da quasi das Szenario selbst *verdoppelt* wird durch zwei Cluster, d.h. es werden Szenarien mit 20, 40, 60, 80 und 100 Knoten simuliert. Ebenso wird als Simulationszeit bei allen Szenarien 200000 Sekunden verwendet. Dies ist notwendig, damit die Knoten über ausreichend Zeit verfügen, um eine akzeptable Abwanderungsrate für ihr Verhalten zu ermitteln.

Interessant ist bei der Analyse, welchen Einfluss die Aufenthaltsdauern in den Clustern auf die Durchführung von Transaktionen und die Verteilung von Transaktionslogs beim Auftreten von Ausfällen haben. Aus den beiden Szenarien ergeben sich aufgrund der Annahmen die bereits zuvor eingeführten Abwanderungsraten, die hier wieder verwendet werden. Wichtig bei den Szenarien ist, dass diese die erfolgreiche Durchführung von Transaktionen ermöglichen. Auch für den Fall der Verteilung von Transaktionslogs aufgrund von Ausfällen muss die erwartete durchschnittliche Knotenanzahl in jedem Cluster $n(C_j)$ berücksichtigt werden, wobei C_j mit $j = 1, \ldots, m$ und m die Anzahl von Clustern im MANET ist. Die erwartete Knotenanzahl kann mit folgender Formel berechnet werden (vgl. [49]):

$$n(C_i) \quad = \quad N \cdot \frac{f(C_i)}{\sum_{j=1}^{m} f(C_j)} \tag{6.12}$$

wobei $f(C_j) = e(C_j) \cdot t(C_j)$ mit $e(C_j)$ die Anzahl von ausgehenden Kanten des Clusters C_j, $t(C_j)$ die durchschnittliche Verweildauer im Cluster C_j und N die Anzahl von Knoten im MANET ist.

Szenario 1

Im ersten Szenario beträgt die erwartete Aufenthaltsdauer (Abk. $E(Aufenthalt)$) für den Cluster C_1 15 Minuten , was einem λ_{C_1} von 0.00111111 entspricht. Die entsprechende erwartete durchschnittliche Aufenthaltsdauer im zweiten Cluster beträgt 45 Minuten mit einem λ_{C_2} von 0.00037037.

Knoten-anzahl	**Cluster C_1 mit $\lambda_{C_1} = 0.001111111$**		**Cluster C_2 mit $\lambda_{C_2} = 0.00037037$**	
	***echte* Knotenanzahl**	**erwartete Knotenanzahl aufgrund von Ausfällen**	***echte* Knotenanzahl**	**erwartete Knotenanzahl aufgrund von Ausfällen**
20	5	2.5	15	7.5
40	10	5	30	15
60	15	7.5	45	22.5
80	20	10	60	30
100	25	12.5	75	37.5

Tabelle 6.1: Durchschnittliche Anzahl von Knoten pro Cluster in Szenario 1

Knoten-anzahl	**Cluster C_1 mit $\lambda_{C_1} = 0.001111111$**		**Cluster C_2 mit $\lambda_{C_2} = 0.001111111$**	
	***echte* Knotenanzahl**	**erwartete Knotenanzahl aufgrund von Ausfällen**	***echte* Knotenanzahl**	**erwartete Knotenanzahl aufgrund von Ausfällen**
20	10	5	10	5
40	20	10	20	10
60	30	15	30	15
80	40	20	40	20
100	50	25	50	25

Tabelle 6.2: Durchschnittliche Anzahl von Knoten pro Cluster in Szenario 2

Basierend auf diesen Werten ergeben sich für dieses Szenario die in Tabelle 6.1 aufgeführten Werte für die durchschnittliche Anzahl von Knoten in jedem Cluster pro Knotenanzahl im MANET. Es wird deutlich, dass gerade in MANETs mit 20 und 40 Knoten die erwartete Anzahl von Knoten im Cluster C_1 mit 2.5 bzw. in C_2 mit 7.5 Knoten sehr gering ist. Dies muss bei der Auswahl der Anzahl von Teilnehmern berücksichtigt werden. Gleichzeitig wird bei den Simulationen darauf zu achten sein, wie viele Transaktionen im Cluster C_1 überhaupt gestartet werden können bei der erwarteten Dichte (in C_1 wird dies beispielsweise in einem Netz mit einer echten Knotenzahl von 20 Knoten nicht möglich sein). Cluster C_2 bietet diesbezüglich eine höhere erwartete Anzahl von Knoten, so dass dort die Durchführung von Transaktionen und die Verteilung von Logs möglich sein sollten.

Szenario 2

Bei Szenario 2 ist die erwartete Aufenthaltsdauer sowohl für Cluster C_1 wie für Cluster C_2 15 Sekunden. Entsprechend gilt für die Abwanderungsraten $\lambda_{C_1} = \lambda_{C_2} = 0.001111111$.

Es zeigt sich, dass in diesem Fall die erwartete Anzahl von Knoten in beiden Clustern gleich ist. Es wird daher bei den Simulationen darauf zu achten sein, wie sich die Anzahl von gestarteten Transaktionen in jedem Cluster bei diesem Szenario verhält.

6.4.1.2 Gewünschte Zuverlässigkeit der Anwendung $p_{retrieval_{app}}$

Als von der Anwendung gewünschte Zuverlässigkeit $p_{retrieval_{app}}$wird werden in den Simulationen in Kapitel 5 70% gewählt. Es wird derselbe Wert ausgewählt, da dann ein Vergleich im Hinblick darauf möglich ist, wie sich der Bedarf an Knoten für die Verteilung eines Transaktionslogs entwickelt, wenn bei der Ausfallwahrscheinlichkeit ein weiteres Kriterium hinzukommt.

6.4.1.3 Auswahl eines garantierten Recoveryzeitraums

Basierend auf den obigen Annahmen und ausgewählten Parametern kann man nun im Vorfeld mit Hilfe von Matlab untersuchen, welche garantierten Recoveryzeiträume möglich sind. Neben den bereits in Kapitel 5 verwendeten Ausfallraten $\lambda_{tech} = 0.000001$ und $\lambda_{energy} = 0.0002778$ kommen nun die Abwanderungsraten für die jeweils verwendeten Cluster hinzu. Insbesondere bei dem ersten Szenario, in dem die Abwanderungsraten für beide Cluster unterschiedlich sind, muss der garantierte Recoveryzeitraum t_m so gewählt werden, dass dieser mit der in Abschnitt 6.4.1.1 berechneten durchschnittlichen Knotenanzahl in beiden Clustern vereinbar ist[3]. Konkret bedeutet das, dass nach Abzug der Teilnehmer an einer Transaktion noch Knoten für die Verteilung in diesem Cluster zur Verfügung stehen sollten für den Fall, dass es zu Fehlern kommt und der Ausgang der Transaktion verteilt werden muss.

Betrachtet man den ersten Cluster C_1 mit $\lambda_{C_1} = 0.001111111$, so ergibt sich daraus, dass bei Berücksichtigung der durchschnittlichen Knotenanzahl in C_1 der garantierte Recoveryzeitraum t_m deutlich kleiner als in den Simulationen in Kapitel 5 gewählt werden muss. Die Ergebnisse der Matlab-Berechnungen für diesen Cluster sind in Abbildung 6.5 zu finden. Es wird deutlich, dass bei Berücksichtigung der Abwanderungsrate von λ_{C_1} deutlich mehr Knoten notwendig sind, um *nur* einen garantierten Recoveryzeitraum von $t_m = 1200$ Sekunden zu ermöglichen, als dies in Kapitel 5 für $t_m = 3600$ Sekunden der Fall war.

Im Vergleich dazu ist bei einem höheren Wert für die erwartete Aufenthaltsdauer in Cluster C_2 mit $\lambda_{C_2} = 0.00037037$ theoretisch auch ein höherer garantierter Recoveryzeitraum möglich. Wie Abbildung 6.6 zeigt, sind weniger Knoten notwendig, um bei $t_m = 1200$ Sekunden die gewünschten Garantien zu erzielen.

Für die Simulationen wird ein garantierter Recoveryzeitraum von $t_m = 1200$ Sekunden gewählt. Dadurch ergibt sich für Szenario 2 immer $E(Aufenthalt) < t_m$, während bei Szenario 1 für Cluster C_1 $E(Aufenthalt_{C_2}) > t_m$ gilt. In Bezug auf Cluster C_2 wäre durchaus ein größerer Zeitraum möglich, allerdings würde dies zu Problemen in Cluster C_1 führen (vgl. Abbildungen 6.5 und 6.6). Durch die Auswahl dieses Recoveryzeitraums besteht die Möglichkeit, dass in Cluster C_1 beispielsweise auch Erwartungswerte für die Aufenthaltsdauer in einem Cluster mit einer gewissen Abweichung toleriert werden (bis ca. 100 Sekunden). Ist die Abweichung zu hoch, so wird die Ausfallwahrscheinlichkeit so groß, dass der Knoten bei der Verteilung nicht berücksichtigt wird. Höhere Ausfallwahrscheinlichkeiten führen zu einer größeren Anzahl von notwendigen Knoten. Eine Art Beschränkung bzw. *Toleranz* stellt hier wiederum die erwartete Anzahl von Knoten im Cluster dar. Werden mehr Knoten benötigt als vorhanden sind, kommt es zu einem Fluten des Netzen mit der ungerichteten Verteilung. Dadurch wird verhindert, dass von der echten Abwanderungsrate zu stark abweichende Werte verwendet werden.

[3] Für die Berechnungen der Anzahl von notwendigen Knoten für die Verteilung eines Transaktionslogs sei auf Abschnitt 4.2 und die Beispiele in Abschnitt 5.2.4.5 hingewiesen.

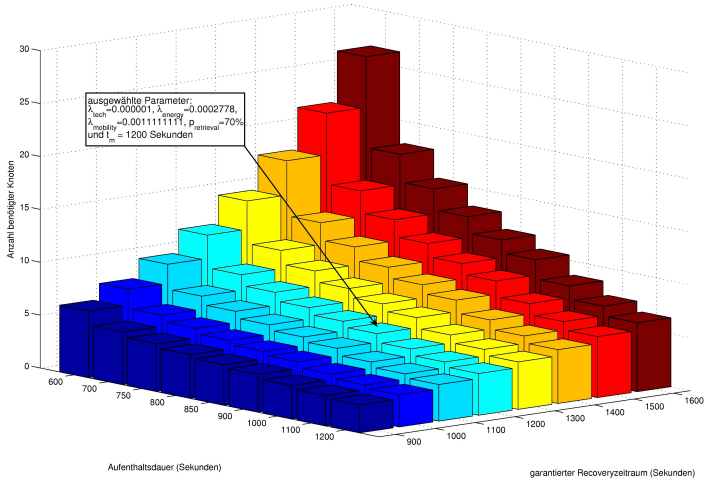

Abbildung 6.5: Benötigte Knotenanzahl für die Verteilung eines Transaktionslogs in Abhängigkeit vom garantierten Recoveryzeitraum und der erwarteten Aufenthaltsdauer im Cluster C_1

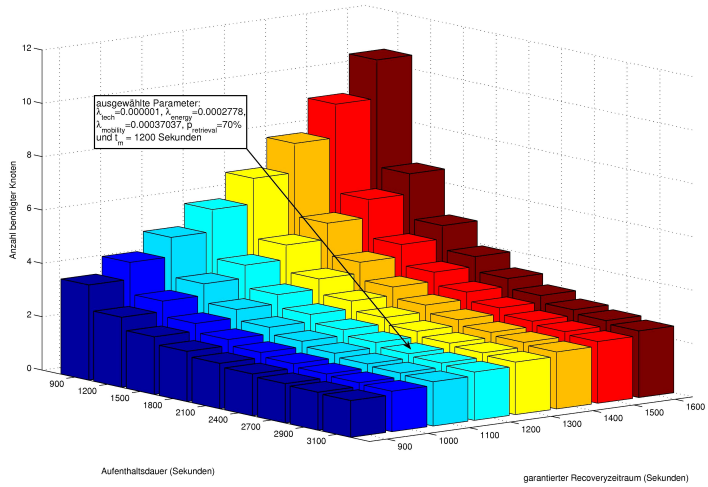

Abbildung 6.6: Benötigte Knotenanzahl für die Verteilung eines Transaktionslogs in Abhängigkeit vom garantierten Recoveryzeitraum und der erwarteten Aufenthaltsdauer im Cluster C_2

Wert	Beschreibung
ANZ_TA_CLUSTER_0	Anzahl der gestarteten Transaktionen in Cluster 0
ANZ_TA_CLUSTER_1	Anzahl der gestarteten Transaktionen in Cluster 1
ANZ_CLIENT_START_RECOVERY_-RONG_CLUSTER	Anzahl der außerhalb des Transaktionsclusters gestarteten Transaktionen

Tabelle 6.4: Übersicht über die für die Analyse neu hinzugekommenen Werte in den Log-Dateien

6.4.1.4 Anzahl der Teilnehmer

Aufgrund der zum Teil sehr geringen Netzwerkdichte bei Szenarien (vgl. Tabellen 6.1 und 6.2) werden in den Simulationen nur Transaktionen mit 3 und 4 Teilnehmern simuliert. Anderenfalls würden beispielsweise bei Szenario 1 im Cluster C_1 für Transaktionen im Schnitt nicht genügend Knoten zur Verfügung stehen.

6.5 Erweiterung der Log-Dateien und der Analyse

Bei der Erweiterung der NS2-Implementierung werden zusätzlich die in Tabelle 6.4 dargestellten Werte gemessen. Damit soll es möglich sein, die Anzahl der gestarteten Transaktionen pro Cluster zu ermitteln. Zusätzlich kann bestimmt werden, wie viele der Recoverys außerhalb des Transaktionsclusters gestartet werden.

Anteil der pro Cluster gestarteten Transaktionen

Mit der folgenden Formel kann der Anteil von Transaktionen pro Cluster bestimmt werden

$$\frac{ANZ\_TA\_CLUSTER\_0}{ANZ\_TA\_START} \tag{6.13}$$

wobei für Cluster 1 einfach $ANZ\_TA\_CLUSTER\_1$ verwendet wird.

Anteil der außerhalb des Transaktionsclusters gestarteten Recoverys

Zur Ermittlung des Anteils von Recovery, die außerhalb des Transaktionsclusters gestartet werden, wird die folgende Formel verwendet

$$\frac{ANZ\_CLIENT\_START\_RECOVERY\_WRONG\_CLUSTER}{ANZ\_CLIENT\_RECOVERYS} \tag{6.14}$$

wobei hier das Ergebnis unabhängig von den jeweiligen Clustern ist.

6.6 Resultate der Simulationen

Zur Evaluation des um Mobilitätsberücksichtigung erweiterten SLS werden wie in Kapitel 5 verschiedene Testbereiche betrachtet. Dabei sollen verschiedene Fragestellungen beantwortet werden. Zunächst soll ermittelt werden, ob bei Verwendung von λ'_{conf} die Zuverlässigkeit R_{SLS}

gewährleistet werden kann. Außerdem soll untersucht werden, welche Auswirkung die Berücksichtigung der Abwanderungsrate bei den simulierten Szenarien hat. Abschließend soll beetrachtet werden, welche *Kosten* die Berücksichtigung der Abwanderungsrate mit sich bringt, d.h. wie sich z.B. der Nachrichtenverbrauch darstellt.

Zur Untersuchung dieser Aspekte werden jeweils die Ergebnisse für die beiden gewählten Szenarien verglichen. Bei allen Simulationen kommt es zu Knotenfehlern aufgrund beschränkter Energieressourcen und technischem Defekt. Es wird bei der Evaluation ein Vergleich zwischen der Verwendung von λ'_{conf} und dem echten λ durchgeführt. *Echtes* λ meint in diesem Zusammenhang, dass den Knoten das ihrem Verhalten wirklich zugrunde liegende λ zur Verfügung gestellt wird. Es wird im Folgenden mit λ bezeichnet. Das λ'_{conf} entspricht dabei einer *konservativen Abschätzung* des Abwanderungsverhaltens. Der Schätzwert λ_{mobile} wird dem Koordinator in diesem Fall bei der Anfrage nach der Ausfallwahrscheinlichkeit zusammen mit λ_{tech} und λ_{energy} zugesandt. Wichtig bei der Evaluation ist, dass durch die Erweiterung des SLS nur Knoten mit Kontextbewusstsein und somit die Verteilung DirDissCA betroffen sind.

In einem ersten Schritt werden in Abschnitt 6.6.1 die Unsicherheitszeiten ausgefallener Transaktionsteilnehmer untersucht. Es wird darauf eingegangen, wie viel Prozent der Recoverys innerhalb des vereinbarten garantierten Recoveryzeitraums t_m abgeschlossen werden können. Anschließend wird betrachtet, welche Fehler Recoverys verursachen. Der dritte Testbereich in Abschnitt 6.6.3 zeigt den Erfolg der einzelnen Verteilungsstrategien des SLS unter Verwendung der Mobilitätsberücksichtigung auf. Da sich die Erweiterung des SLS vor allem auf die Knoten mit Kontextbewusstsein auswirkt, wird die gerichtete Verteilung auf n ausfallsichere Knoten in einem dichten MANET bei Knoten mit Kontextbewusstsein im Vordergrund stehen. Abschließend wird in Abschnitt 6.6.4 untersucht, wie hoch der Nachrichtenaufwand bei Verwendung des erweiterten SLS von Koordinator und Teilnehmern ist.

6.6.1 Testbereich 1: Unsicherheitszeiten der Transaktionsteilnehmer

In diesem Bereich soll die Unsicherheitszeit ausgefallener Transaktionsteilnehmer bei den beiden Szenarien untersucht werden. Dies umschließt auch den Anteil der erfolgreichen Recoverys innerhalb des vereinbarten garantierten Recoveryzeitraums t_m. Dabei wird analysiert, ob mit der Verwendung von λ'_{conf} das Auffinden des Transaktionslogs innerhalb von t_m mit der vereinbarten Wahrscheinlichkeit $p_{retrieval_{app}}$ möglich ist. Dazu wird der Vergleich zwischen λ'_{conf} und λ durchgeführt. Da von der Erweiterung nur die Knoten mit Kontextbewusstsein betroffen sind, wird die Verteilungsstrategie DirDissCA im Zentrum der Untersuchungen stehen.

6.6.1.1 Unsicherheitszeit ausgefallener Knoten

Hypothese: Die Verwendung von λ'_{conf} als konservative Schätzung bei DirDissCA führt zu kürzeren Unsicherheitszeiten als die Anwendung des echten λ.

Resultat: In Abbildungen 6.7.a und 6.7.b werden für beide Szenarien die Unsicherheitszeiten nach den jeweils verwendeten Verteilungsstrategien unterschieden.

Bei beiden Szenarien führt die Annahme eines dünnen MANETs und somit die Verwendung von UnDirDiss zur längsten Unsicherheitszeit und schwankt zum Teil erheblich. In Szenario 1 liegt diese zwischen 140 und 270 Sekunden, in Szenario 2 zwischen 175 und 312 Sekunden.

Wird angenommen, dass das Netz dicht ist und die Knoten kein Kontextbewusstsein haben, so kann mit ausschließlicher Verwendung von DirDissCUA bei Szenario 2 generell die kürzeste Unsicherheitszeit erzielt werden, die sich zwischen 12 und 62 Sekunden bewegt. Ähnlich sieht es bei Szenario 1 aus. Nur bei 10 und 20 Knoten ist die Unsicherheitszeit höher als bei der Annahme,

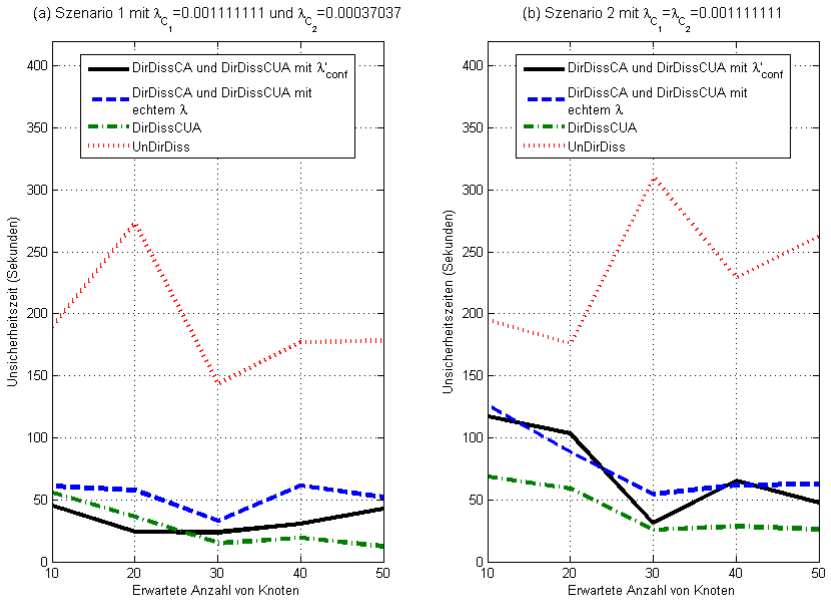

Abbildung 6.7: Durchschnittliche Unsicherheitszeit von Teilnehmern unterschieden nach Kontextbewusstsein der Knoten und angenommener Netzdichte

dass die Knoten im Netz über Kontextbewusstsein verfügen. Hier beträgt diese mindestens 25 und maximal 70 Sekunden.

Wenn das MANET als dicht angenommen wird und die Knoten über Kontextbewusstsein verfügen, wird die Kombination aus DirDissCA und DirDissCUA verwendet, wobei jeweils einmal λ'_{conf} und einmal das echte λ verwendet werden. Die damit erzielten Unsicherheitszeiten variieren bei beiden Szenarien.

In Szenario 1 kann mit diesen Strategien und λ'_{conf} bei Netzen mit 10 und 20 Knoten die kürzeste Unsicherheitszeit erzielt werden, die sich zwischen 25 und 40 Sekunden bewegt. Ist das Netz hingegen dichter, so bringt die Kombination bei Szenario 1 keine Verbesserung, sondern die alleinige Anwendung von DirDissCUA ist mit 20 bis 25 Sekunden erfolgreicher. Die Kombination aus DirDissCA und DirDissCUA bei Verwendung des echten λ führt hier zur längsten Unsicherheitszeit beim Vergleich der gerichteten Verfahren.

Bei Szenario 2 sehen die Resultate etwas anders aus. Hier wird nur mit der Verwendung von DirDissCUA die kürzeste Unsicherheitszeit erzielt über alle Netzdichten hinweg. Die Unsicherheitsheiten bei der Anwendung von DirDissCA und DirDissCUA mit λ'_{conf} bzw. dem echten λ schwanken. In Netzen mit 10, 30 und 50 Knoten führt die Verwendung von λ'_{conf} zu besseren Resultaten. Enthält das Netz 80 Knoten, so sind die Unsicherheitszeiten nahezu identisch. Lediglich bei 40 Knoten sind die Unsicherheitszeiten mit λ'_{conf} etwa 20 Sekunden höher.

Beurteilung: Durch die Erweiterung um Mobilitätsberücksichtigung werden nur die Fähigkeiten der Knoten mit Kontextbewusstsein verändert. Deshalb ist die Entwicklung der Unsicherheitszeiten bei Annahme eines dichten MANETs und der Verwendung von DirDissCA mit λ'_{conf} und echtem λ zu beobachtens. Dabei zeigt sich, dass die obige Hypothese bestätigt werden kann.

Wichtiges Ergebnis bei Szenario 1 ist, dass bei Verwendung der Kombination aus DirDissCA und DirDissCUA mit λ'_{conf} deutlich kürzere Unsicherheitszeiten erzielt werden können als mit λ.

Sei für alle Knoten n_i ihre Ausfallwahrscheinlichkeit $p_{failure_i}$ mit $i = 0, \ldots, k$. Für alle n_i seien weiterhin $\lambda_{tech} = 0.000001$ sowie $\lambda_{energy} = 0.0002778$ gegeben. R_{SLS} ist vom Koordinator zu gewährleisten, wobei das gewünschte $p_{retrieval} = 0.7$ ist und der vereinbarte garantierte Recoveryzeitraum $t_m = 1200s$ beträgt.
Dann ergibt sich für die Bestimmung der n notwendigen bei der Verteilungsstrategie in Abhängigkeit von λ'_{conf} und echtem λ:

Verwendung von λ'_{conf}	**Verwendung des echten λ**
Jeder Knoten n_i hat ein individuelles λ'_{conf} ermittelt. Damit ergeben sich die folgenden Ausfallwahrscheinlichkeiten: $p_{failure_1} = 0.8114$, $p_{failure_2} = 0.8403$, $p_{failure_3} = p_{failure_4} = 0.8711$, $p_{failure_5} = 0.8555$, $p_{failure_6} = 0.8256$, $p_{failure_7} = 0.8870$ und $p_{failure_8} = 0.9031$	Jeder Knoten n_i kennt das echte λ, so dass alle Knoten dieselbe Ausfallwahrscheinlichkeit $p_{failure_i} = 0.8114$ haben.
Für R_{SLS} ergibt sich daraus: $R_{SLS} = 1 - (p_{failure_1} \cdot \ldots \cdot p_{failure_8})$ $R_{SLS} = 1 - (0.8114 \cdot \ldots \cdot 0.9031) = 0.7031$	Für R_{SLS} ergibt sich daraus: $R_{SLS} = 1 - (p_{failure_1} \cdot \ldots \cdot p_{failure_6})$ $R_{SLS} = 1 - (0.8114 \cdot \ldots \cdot 0.8114)$
$\Rightarrow$ Es werden 8 Knoten benötigt.	**$\Rightarrow$ Es werden 6 Knoten benötigt.**

Abbildung 6.8: Beispiel zur Bestimmung der n notwendigen Knoten bei der Verteilungsstrategie DirDissCA mit λ'_{conf} sowie mit dem echten λ

Außerdem kann in dünnen Netzen mit 10 und 20 Knoten mit der Kombination aus DirDissCA und DirDissCUA generell die kürzeste Unsicherheitszeit erzielt werden. Die Unsicherheitszeit bei der Anwendung von DirDissCUA ist in diesem Fall länger. In dichteren Netzen mit 30 bis 50 Knoten führt hingegen die Benutzung von DirDissCUA allein zu den besten Resultaten.

Bei Szenario 2 sind keine kürzeren Unsicherheitszeiten mit der Kombination der gerichteten Verfahren möglich, wie dies bei Szenario 1 der Fall ist. Allerdings zeigt sich auch hier, dass in den meisten Untersuchungen (ausgenommen eine Netzdichte von 20 Knoten) die Verwendung von λ'_{conf} zu besseren Resultaten führt als das echte λ bei DirDissCA.

Die kürzere Unsicherheitszeit bei der Kombination aus DirDissCA und DirDissCUA mit λ'_{conf} im Vergleich zum echten λ wird durch verschiedene Faktoren beeinflusst. Ein wichtiger Aspekt ist die benötigte Anzahl von Knoten. Mit λ'_{conf} wird eine konservative Abschätzung für die Abwanderungswahrscheinlichkeit verwendet. Dies führt dazu, dass für die Garantie von R_{SLS} mit $p_{retrieval_{app}} = 0.7$ und garantiertem Recoveryzeitraum $t_m = 1200s$ mehr Knoten verwendet werden müssen. Ist mit λ'_{conf} die Abwanderungswahrscheinlichkeit sehr hoch, wird eine größere Anzahl von Knoten für die Verteilung benötigt, als wenn den Knoten das echte λ bekannt ist. Abbildung 6.8 zeigt eine Beispielrechnung, wobei die individuellen Abwanderungsraten zum Teil kleiner oder in einem Fall gleich dem echten Wert für λ sind. Es wird deutlich, dass bei DirDissCA mit λ'_{conf} eine größere Anzahl von Knoten zur Garantie von R_{SLS} notwendig ist. Es ergeben sich daraus zwei Möglichkeiten. Entweder es steht die notwendige Anzahl von Knoten mit Kontextbewusstsein für DirDissCA zur Verfügung. Hier ist der Knotenbedarf mit λ'_{conf} entweder gleich oder größer. Oder aber der Koordinator findet nicht ausreichend viele Knoten und greift deshalb auf DirDissCUA zurück. Da hier der Koordinator für die Schätzung das echte λ verwendet, ist bei beiden Varianten der Bedarf identisch. Aber bei Verwendung von λ als individueller Schätzung

ist dies seltener notwendig, da weniger Knoten für DirDissCA benötigt werden. DirDissCA kann also häufiger bei λ als bei λ'_{conf} eingesetzt werden. Wenn λ'_{conf} angewendet wird, führt dies also zu einem gleich großen oder größeren Verteilungsgrad d als mit λ, also gilt $d_{\lambda'_{conf}} \geq d_{\lambda}$.

Bereits an dem Beispiel deutet sich der Zusammenhang zur Aufenthaltsdauer in den Clustern an. Bei Szenario 1 befinden sich ca. $\frac{3}{4}$ aller Knoten im Cluster C_2 mit der längeren Aufenthaltsdauer (vgl. Tabelle 6.1). Somit stehen dort so viele Knoten für die Verteilung zur Verfügung, dass DirDissCA mit λ'_{conf} verwendet werden kann. Durch den längeren Aufenthalt in dem Cluster sind zudem Knoten zum Zeitpunkt einer Recovery innerhalb vom t_m möglich .

In Szenario 2 führt der höhere Knotenbedarf mit λ'_{conf} dazu, dass DirDissCA seltener angewendet werden kann. Hier ist in beiden Clustern die Anzahl der Knoten etwa gleich verteilt (vgl. Tabelle 6.2). Anhand des Beispiels wird deutlich, dass in dünnen Netzen DirDissCA nicht immer angewendet werden kann. Befinden sich z.B. nur 5 Knoten in einem Cluster und führt ein Koordinator mit 3 Teilnehmern eine Transaktion durch, stehen für die Verteilung mit DirDissCA nicht mehr ausreichend viele Knoten zur Verfügung. Hier muss dann DirDissCUA angewendet werden.

Abschließend bleibt festzuhalten: *Mit λ'_{conf} kann bei der Kombination von DirDissCA und DirDissCUA eine kürzere Unsicherheitszeit als mit dem echten λ erzielt werden. Durch die konservative Abschätzung werden mehr Knoten für die Verteilung benötigt, so dass für den Verteilungsgrad $d_{\lambda'_{conf}} \geq d_{\lambda}$ gilt. Es zeigt sich außerdem der Einfluss der Aufenthaltsdauern in den Clustern, der für die Anzahl von Knoten pro Cluster von Bedeutung ist.*

6.6.1.2 Erfolgreiche Recoverys innerhalb des garantierten Recoveryzeitraums t_m

Hypothese: Recoverys können bei Berücksichtigung der Abwanderungsrate λ'_{conf} innerhalb des garantierten Recoveryzeitraums t_m mit der Warscheinlichkeit $p_{retrieval_{app}} = 70\%$ erfolgreich beendet werden.

Resultat: Die Simulationen des SLS unter Verwendung von λ'_{conf} bei Szenario 1 zeigen, dass ca. 94% bis 96% aller Recoverys innerhalb des vereinbarten garantierten Recoveryzeitraums t_m erfolgreich abgeschlossen werden können. Die Ergebnisse für Szenario 2 zeigen, dass hier bei Benutzung von λ'_{conf} 92% bis 94% aller Recoverys innerhalb des vereinbarten garantierten Recoveryzeitraums erfolgreich sind. Bei Szenario 1 ist die Erfolgsrate ca. 1% bis 4% höher als bei Szenario 2.

Beurteilung: Es wird deutlich, dass bei beiden Szenarien über 90% der Recoverys innerhalb von t_m erfolgreich beendet werden können. Damit bestätigt sich die Hypothese. Eine Erfolgsrate von etwa 99% wie in den Simulationen von Abschnitt 5.3 kann jedoch nicht erzielt werden. Dies ist darauf zurückzuführen, dass durch die Mobilität zwischen den Clustern bei beiden Szenarien Fälle auftreten, bei denen Knoten erst nach Ablauf des vereinbarten Recoveryzeitraums wieder in den Transaktionscluster kommen. Entsprechend ist eine erfolgreiche Recovery erst später möglich. Dies ist bei beiden Szenarien der Fall. Bei Szenario 1 liegt die Erfolgsrate im Schnitt um 1% höher. Der Grund dafür ist der durchschnittlich wesentlich längere Aufenthalt in Cluster C_2 mit $\lambda_{C_2} = 0.00037037$, wo sich zudem immer $\frac{3}{4}$ aller Knoten befinden. Treten in diesem Cluster Ausfälle auf, können Recoverys schneller erfolgreich beendet werden. Sowohl Transaktionsteilnehmer wie Knoten mit dem gespeicherten Transaktionslog halten sich dort im Schnitt länger auf, als das bei den Clustern von Szenario 2 der Fall ist. Die erwartete Aufenthaltsdauer innerhalb dieses Clusters ist dabei mit 2700 Sekunden länger als der garantierte Recoveryzeitraum mit 1200 Sekunden.

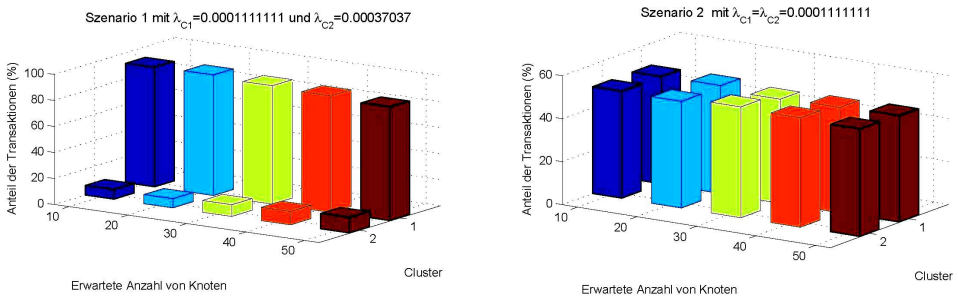

Abbildung 6.9: Anzahl der pro Cluster gestarteten Transaktionen bei den beiden gewählten Szenarien

Dies ist z.B. bei kurzzeitigen energiebedingten Ausfällen oder Verbindungssabbrüchen innerhalb des Clusters aufgrund von Mobilität der Fall.

Bei Szenario 2 hingegen ist mit 900 Sekunden die erwartete Aufenthaltsdauer kürzer. Hier ist eine erfolgreiche Recovery nur möglich, wenn zum Zeitpunkt der Verteilung Knoten für die Speicherung mit einbezogen werden, die den Cluster nicht vor Ablauf von t_m verlassen. Die Erfolgsrate der Recoverys zeigt, dass dies bei mehr als 90% der Recoverys der Fall ist. Dennoch wird deutlich, dass hier 1% bis 4% der Recoverys weniger innerhalb von t_m möglich sind als bei Szenario 1. Somit kann t_m größer als die erwartete Aufenthaltsdauer von Knoten im Cluster sein, es ist jedoch zu erwarten, dass dieser Wert nicht beliebig größer sein kann.

Zusammenfassend bleibt festzuhalten: *Mehr als 90% aller Recoverys können innerhalb des vereinbarten garantierten Recoveryzeitraums t_m beendet werden. Es deutet sich ein Einfluss der Aufenthaltsdauern an. Die erfolgreiche Recovery innerhalb von t_m ist nur dann möglich, wenn die Aufenthaltsdauer in einem Cluster und t_m sinnvoll gewählt werden.*

6.6.2 Testbereich 2: Auftritt von Recoverys in cluster-basierten Szenarien

Mit diesem Testbereich soll der Einfluss der Abwanderungen auf die Anzahl von Fehlerfällen – und damit zusammenhängend auf die Recoverys – betrachtet werden. Zu klären ist, wie hoch der Anteil von Fehlerfällen aufgrund von Abwanderungen aus dem Transaktionscluster ist.

6.6.2.1 Auftritt von Recoverys

Die Simulationsresultate zeigen, dass der Anteil von Recoverys aufgrund von Abwanderungen aus dem Transaktionscluster bei den gewählten Szenarien niedriger ist als der Anteil von Recoverys aufgrund von Knotenfehlern, die durch beschränkte Energie oder technischen Defekt verursacht werden. Der größte Teil aller Recoverys wird durch lokale (kurzfristige) Mobilität innerhalb des Transaktionsclusters ausgelöst.

Es zeigt sich bei den Ergebnissen in Abbildung 6.10, dass mehr als 70% aller Recoverys durch Kommunikationsfehler verursacht werden. Der Anteil von Recoverys aufgrund von Abwanderungen aus dem Transaktionscluster und Knotenfehlern durch beschränkte Energie ist deutlich geringer.

Bei Szenario 1 werden durchschnittlich 2% bis 5% aller Recoverys aufgrund einer Abwanderung aus dem Transaktionscluster ausgelöst. In Szenario 2 variiert dieser Anteil zwischen 1% und 4%.

Der Anteil von Recoverys aufgrund von energiebedingten Knotenfehlern ist bei beiden Szenarien höher. Bei Szenario 1 variiert dieser Anteil zwischen 10% und 17%. Für Szenario 2 sind

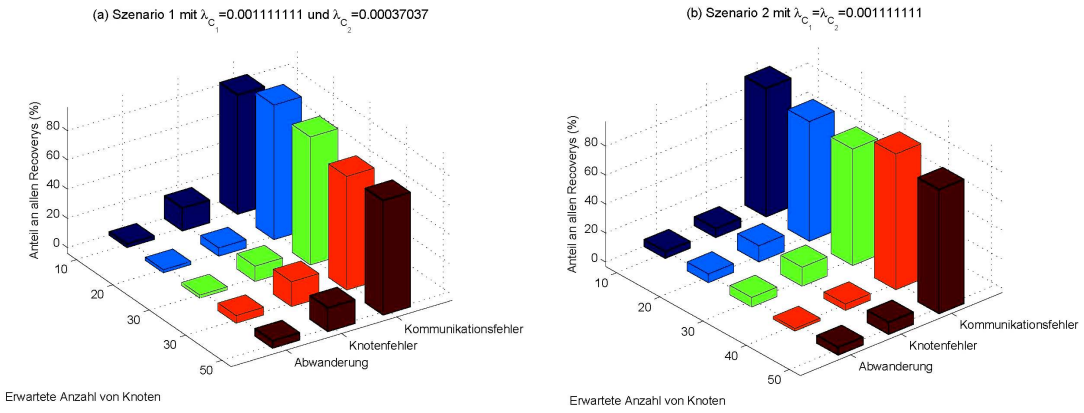

Abbildung 6.10: Ursachen für Recoverys bei den beiden gewählten Szenarien

diese Werte niedriger. Der Anteil von Recoverys aufgrund von Knotenfehlern bewegt sich hier zwischen 4% und 11%.

Wie zuvor in den Simulationen in Abschnitt 5.3 zeigt sich, dass auch bei diesen Szenarien Kommunikationsfehler durch lokale Mobilität die größte Fehlerursache bei der Durchführung von Transaktionen sind. Deren Anteil an allen Recoverys beträgt mehr als 70%.

6.6.3 Testbereich 3: Erfolg der einzelnen Verteilungsstrategien

Durch die Mobilitätsberücksichtigung zwischen den Clustern werden insbesondere die Fähigkeiten der Knoten mit Kontextbewusstsein erweitert. Entsprechend wird sich diese Erweiterung bei der Verwendung von DirDissCA auswirken. Dazu wird der Erfolg von DirDissCA in beiden Szenarien betrachtet, wobei bei Szenario 1 in Cluster C_2 (dort werden die meisten Transaktionen gestartet, vgl. Abbildung 6.6.2) $t_m < E(Aufenthalt_{C_2})$ und bei Szenario 2 $t_m > E(Aufenthalt)$ für beide Cluster gilt. Ziel ist es, den Einfluss der Verwendung von λ'_{conf} in Zusammenhang mit dem garantierten Recoveryzeitraum t_m im Vergleich zur Anwendung des echten λ aufzuzeigen.

Zur Untersuchung werden die einzelnen Strategien auch hier wieder aus dem Blickwinkel ausgefallener Teilnehmer, d.h. durch Analyse der Recoverystrategien, betrachtet (vgl. Abschnitt 4.1.1.2).

Abbildung 6.11 verdeutlicht die Ergebnisse für beide Szenarien unter Verwendung des λ'_{conf} bei DirDissCA.

Bei Szenario 1 ist eine Überschneidung zwischen ungerichteter Recovery und Anfrage bei dem Koordinator erkennbar. Der Anteil der ungerichteten Recovery an allen erfolgreichen Recoverys variiert zwischen 60% bei 10 Knoten und 30% bei 50 Knoten. Entsprechend steigen die erfolgreichen Anfragen an den Koordinator von 36% bei 10 Knoten auf 65% bei 50 Knoten im MANET. Bei der gerichteten Recovery bewegt sich der Anteil zwischen 5% und 17%, wobei der Anteil von 17% bei einem Netz mit 30 Knoten erzielt wird.

Für Szenario 2 sehen die Resultate etwas anders aus. Der größte Anteil aller erfolgreichen Recoverys mit 50% bis 60%) erfolgt durch die ungerichtete Recovery. Die Anfragen an den Koordinator liegen darunter und bewegen sich zwischen ca. 38% und 45%. Hier ist der Erfolg der gerichteten Recovery deutlich geringer als bei Szenario 1. Bei 10 Knoten wird dieses Verfahren nicht eingesetzt und steigt auf 4% bei 50 Knoten im Netz.

Um zu ermitteln, welchen Einfluss die Verwendung des echten λ hat, zeigt Abbildung 6.12 den

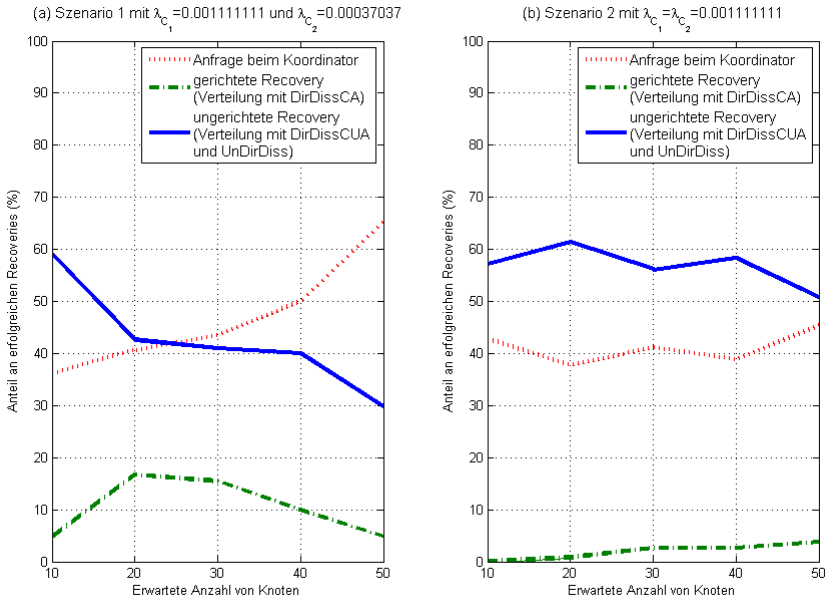

Abbildung 6.11: Erfolg der einzelnen Verteilungsstrategien bei den gewählten Szenarien

Vergleich der Strategien für Szenario 2 sowohl mit λ'_{conf} (vgl. Abbildung 6.12.a) wie mit λ (vgl. Abbildung 6.12.b). Die Ergebnisse mit λ'_{conf} beschreibt der vorige Absatz. Ergänzend dazu wird nun das echte λ betrachtet. Der Anteil der gerichteten Recovery ist hier höher und bewegt sich zwischen 12% und 19%. Gleichzeitig sinkt der Anteil der ungerichteten Recovery auf 40% bis 46%. Der Anteil der Anfragen an den Koordinator ist nahezu identisch.

Beurteilung: Es zeigt sich ein deutlicher Unterschied zwischen den beiden Szenarien bei Verwendung von λ'_{conf}. Bei Szenario 1 ist ein deutlich höherer Anteil der gerichteten Recovery erkennbar. Dieses Ergebnis verdeutlicht die Verwendung von DirDissCA bei der Kombination aus DirDissCA und DirDissCUA. Hier sind ausreichend viele ausfallsichere Knoten für die Verteilung vorhanden.

Im Gegensatz dazu ist bei Szenario 2 der Anteil der gerichteten Recovery bei der Verwendung von λ'_{conf} sehr gering. In diesem Szenario wird die Strategie DirDissCA kaum verwendet.

Diese Ergebnisse lassen den Einfluss der Gestalt der gewählten Szenarien erkennen. Der echte Aufenthalt der Knoten und die individuell über λ'_{conf} ermittelte Abwanderungswahrscheinlichkeit spielen eine Rolle. Ebenso hängt damit u.a. der Parameter t_m zusammen.

Bei Szenario 1 werden viele Transaktionen in Cluster C_2 gestartet (vgl. dazu Abbildung 6.9). Dort halten sich im Schnitt etwa $\frac{3}{4}$ aller Knoten (vgl. Tabelle 6.1) auf und die Dauer des Aufenthalts ist mit ca. 45 Minuten deutlich länger als in Cluster C_1. Entsprechend ist die Netzdichte in C_2 höher als in C_1. Dies führt dazu, dass die Auswahl von einer ausreichenden Anzahl von ausfallsicheren Knoten trotz erhöhtem Bedarf durch λ'_{conf} (vgl. Beispiel in Abbildung 6.8) möglich ist. Die Strategie DirDissCA kann somit häufiger verwendet werden, was an dem Anteil der erfolgreichen gerichteten Recoverys erkennbar ist. Ein weiterer Faktor ist der garantierte Recoveryzeitraum t_m, der deutlich kleiner als der erwartete Aufenthalt im Cluster C_2 ist. Dadurch sind sowohl der Koordinator wie auch die übrigen Knoten mit dem gespeicherten Transaktionslog länger für eine Recovery verfügbar. Da ein Großteil der Recoverys durch kurzfristige Mobilität

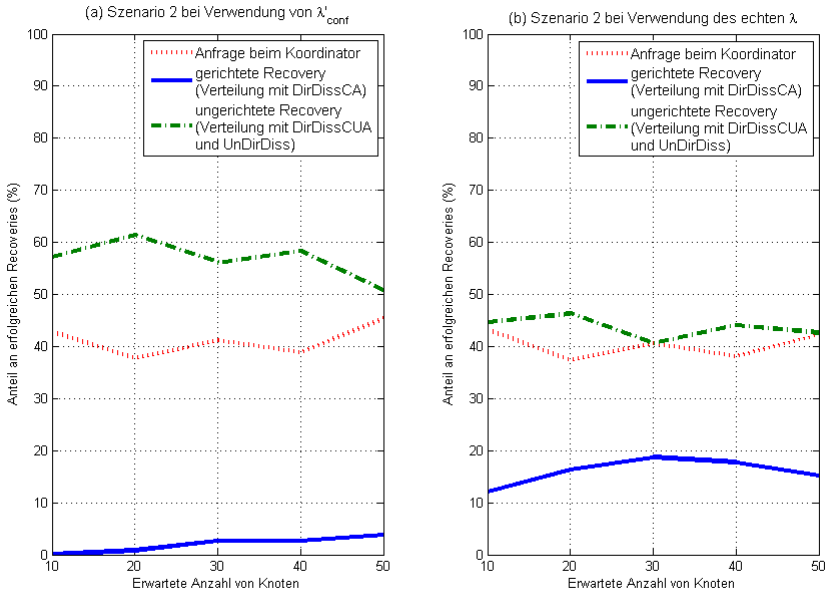

Abbildung 6.12: Vergleich des Erfolgs der einzelnen Verteilungsstrategien bei Verwendung der unteren Grenze des Konfidenzintervalls für das geschätzte λ' und bei Verwendung des echten λ am Beispiel von Szenario 2

ausgelöst wird (vgl. Abschnitt 6.6.2.1), ist dies ein wesentlicher Aspekt.

Durch die Konfiguration bei Szenario 2 stellt sich hier die Situation etwas anders dar. Hier befinden sich in beiden Clustern durch identische erwartete Aufenthaltsdauer etwa gleich viele Knoten (vgl. Tabelle 6.2) und es werden jeweils gleich viele Transaktionen gestartet (vgl. Abbildung 6.9). Dies führt zu einer höheren Mobilitätsdynamik zwischen den Clustern. Hinzu kommt, dass der garantierte Recoveryzeitraum t_m mit 1200 Sekunden höher ist als die erwartete Aufenthaltsdauer in einem Cluster, die 900 Sekunden beträgt. Somit ist die Wahrscheinlichkeit für die Abwanderung von Knoten mit Transaktionslog vor Ablauf von t_m sehr hoch. Die Folge ist ein geringerer Erfolg der gerichteten Recovery und somit der Verteilung mit DirDissCA. Es sind zu wenige Knoten vorhanden, so dass mit ihnen bei dieser geringen Netzdichte nicht die geforderte Garantie von $p_{retrieval} = 0.7$ für $t_m = 1200$ Sekunden gewährleistet werden kann.

Betrachtet man dazu bei Szenario 2, wie sich der Anteil der einzelnen Strategie bei Verwendung des echten λ entwickelt (vgl. Abbildung 6.12.b), so wird die obige Vermutung gestützt. Stehen den Knoten mit Kontextbewusstsein die Informationen über die echten erwarteten Aufenthaltsdauern zur Verfügung, steigt der Anteil der gerichteten Recoverys (und somit der Verteilung mit DirDissCA) deutlich an. Dies verdeutlicht, dass bei Verwendung von λ'_{conf} mehr Knoten notwendig sind als bei Benutzung des echten λ, wie bereits das Berechnungsbeispiel in Abbildung 6.8 erkennen lässt. Wird das echte λ verwendet, so stehen bei Szenario 2 häufiger ausreichend viele Knoten für die Verteilung zur Verfügung als bei λ'_{conf}, da weniger Knoten benötigt werden.

Es bleibt abschließend festzuhalten: *Die Aufenthaltsdauern in den Clustern der gewählten Szenarien sowie die Verwendung von λ'_{conf} bzw. λ zusammen mit dem garantierten Recoveryzeitraum t_m beeinflussen die Verwendung von DirDissCA. Je länger der Aufenthalt und je dichter das* MANET *ist, desto besser funktioniert dieses Verfahren. Durch die Verwendung von $\lambda'_{conf1312-1321}$ werden von DirDissCA für die Verteilung mehr Knoten benötigt als bei Anwendung des echten λ.*

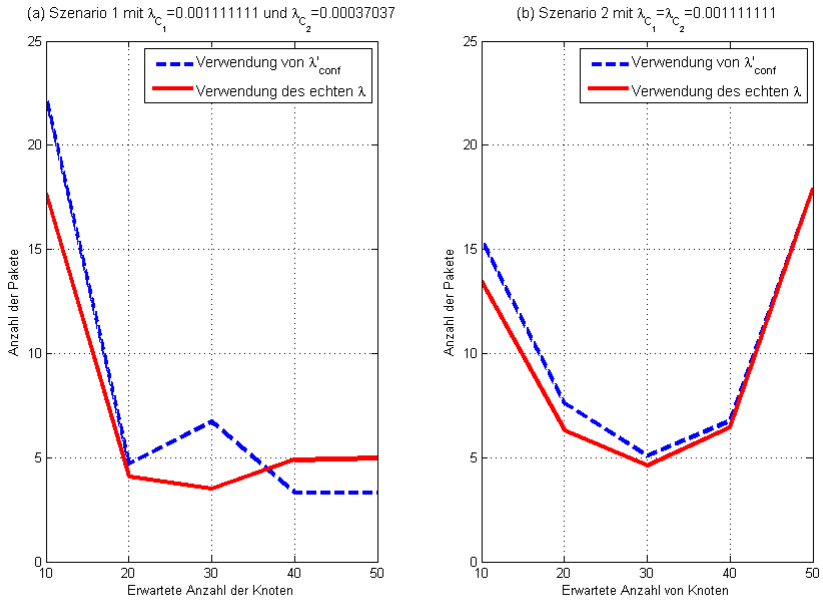

Abbildung 6.13: Vergleich der vom Koordinator benötigten Nachrichtenpakete bei den gewählten Szenarien

Entsprechend wird die Anwendung von DirDissCA auch durch die Wahl von t_m beeinflusst. Wird t_m größer als die erwartete Aufenthaltsdauer gewählt und λ'_{conf} verwendet, erhöht dies den Knotenbedarf zusätzlich. Es muss also ein sehr dichtes MANET *vorhanden sein, damit ausreichend viele Knoten zur Gewährleistung von R_{SLS} für die Speicherung des Transaktionslogs bei DirDissCA zur Verfügung stehen. Den größten Anteil an den erfolgreichen Recoverys bei den gewählten Szenarien hat die ungerichtete Recovery, bei der das Transaktionslog mittels DirDissCUA bzw. UnDirDiss verteilt wird.*

6.6.4 Testbereich 4: Nachrichtenaufwand der Verteilungsstrategien

In diesem Testbereich soll der Nachrichtenbedarf von Koordinator und Teilnehmern analysiert werden. Zu untersuchen ist, wie hoch der zusätzliche Nachrichtenbedarf bei Verwendung von λ'_{conf} ist.

6.6.4.1 Nachrichtenanzahl des Koordinators

Hypothese: Durch die Verwendung von λ'_{conf} als konservative Abschätzung der Abwanderungsrate werden mehr Nachrichten durch den Koordinator versandt als bei Verwendung des echten λ.

Resultat: Abbildung 6.13 zeigt für beide Szenarien den Nachrichtenbedarf des Koordinators allgemein. In fast allen Fällen sind dabei mit Verwendung von λ'_{conf} mehr Nachrichten notwendig als bei Anwendung des echten λ.

Bei Szenario 2 in Abbildung 6.13.b zeigt sich, dass mit dem echten λ immer weniger Nachrichten durch den Koordinator benötigt werden. Sind 10 Knoten im Netz, benötigt der Koordinator ca. 13 Pakete, bei 20 Knoten etwa 6 Pakete, bei 30 Knoten etwa 5 und bei 40 etwa 7 Nachrichten.

In einem Netz mit 50 Knoten versendet der Koordinator im Schnitt 18 Pakete. Hier sind die Resultate für λ'_{conf} und λ identisch. In den anderen Fällen wird bei Verwendung von λ'_{conf} jeweils etwa ein Paket mehr verschickt.

Für Szenario 1 sehen die Ergebnisse etwas anders aus. Auch wird in Netzen mit 10, 20 und 30 Knoten bei Verwendung von λ'_{conf} mit DirDissCA ein erhöhter Nachrichtenbedarf erkennbar. In Netzen mit 10 Knoten werden ca. 22 Pakete versendet, bei 20 Knoten etwa 5 und bei 30 Knoten ca. 7 Nachrichten. Wird hingegen das echte λ eingesetzt, so werden bei den gleichen Knotendichten nur ca. 18 Pakete, 4 Pakete und 3 Pakete benötigt. Lediglich bei den Knotenzahlen 40 und 50 zeigen sich Unterschiede zu Szenario 2. Hier ist die Verwendung von λ'_{conf} weniger aufwändig. Es werden bei beiden Dichten ca. 3 Pakete durch den Koordinator verschickt. Im Gegensatz dazu werden bei Benutzung des echten λ etwa 5 Nachrichten versandt.

In Abbildung 6.14 wird für Szenario 1 der Nachrichtenbedarf nach Strategien aufgeschlüsselt, wobei bei dem linken Diagramm (a) λ'_{conf} und bei dem rechten Diagramm (b) das echte λ verwendet wird. Da nur die Fähigkeiten der Knoten mit Kontextbewusstsein erweitert werden, zeigen sich nur Unterschiede bei der Verwendung von DirDissCA. Die Ergebnisse für DirDissCUA und UnDirDiss sind bei beiden Abschätzungen der Abwanderungsrate identisch. Mit UnDirDiss versendet der Koordinator konstant 3 Nachrichten und bei DirDissCUA variiert der Nachrichtenbedarf zwischen 4 und 53 Paketen.

Unterschiede sind bei der Kombination aus DirDissCUA und DirDissCA zu erkennen. Wird λ'_{conf} verwendet, so benötigt der Koordinator bei 10 Knoten ca. 31 Pakete, bei 20 Knoten 6 Pakete, bei 30 Knoten 16 Pakete und bei 40 sowie 50 Knoten 4 Nachrichten. Im Gegensatz dazu werden bei Anwendung von λ bei 10 Knoten 13 Pakete, bei 20 und 30 Knoten 4 Pakete sowie bei 40 Knoten 11 Pakete und bei 50 Knoten 12 Nachrichten versandt.

Beurteilung: Die obige Hypothese kann bestätigt werden. Wie bereits vorherige Untersuchungen zeigen (vgl. z.B. Berechnungsbeispiel in Abbildung 6.8), werden durch Verwendung von λ'_{conf} bei DirDissCA mehr Knoten benötigt als mit λ. Entsprechend kann seltener DirDissCA eingesetzt werden. Stattdessen muss DirDissCUA benutzt werden. Somit ist auch der Nachrichtenbedarf des Koordinators höher. Bereits die Simulationen in Abschnitt 5.3 haben den erhöhten Nachrichtenbedarf dieses Verfahrens gezeigt. Allerdings liegt der Unterschied durchschnittlich nur bei 5 bis 10 Nachrichten. Lediglich in Netzen mit 10 Knoten ist der Nachrichtenbedarf um etwa 22 Pakete höher. Die Ursache dafür ist, dass generell zu wenig Knoten zur Verfügung stehen (vgl. Tabellen 6.1 und 6.2), so dass der Koordinator über einen längeren Zeitraum hinweg das Transaktionslog immer wieder verteilen muss. Sind ausreichend Knoten vorhanden (ab 20 Knoten im Netz), so ist der Nachrichtenanstieg geringer.

In dünnen Netzen mit 10 bis 30 Knoten kann bei Szenario 1 daher DirDissCA selten eingesetzt werden, so dass der Koordinator DirDissCUA verwendet. Bereits die erwartete Anzahl von Knoten (vgl. Tabelle 6.1) deutet dies an. Erst in Netzen mit 40 und 50 Knoten stehen genügend Knoten zur Verfügung, so dass im Hinblick auf den Nachrichtenbedarf eine Verbesserung deutlich wird. Dann stehen ausreichend viele ausfallsichere Knoten für DirDissCA zur Verfügung, so dass Nachrichten gespart werden können (vgl. Abbildung 6.14.b).

Da bei Verwendung des echten λ weniger Knoten für DirDissCA benötigt werden (vgl. Beispielrechnung in Abbildung 6.8), kann dies in den dünneren Netzen eher eingesetzt werden und führt zu einer Reduzierung des Paketbedarfs. Es wird häufiger DirDissCA eingesetzt, welche weniger Nachrichten als DirDissCUA benötigt. Zudem werden von DirDissCA bei Verwendung von λ'_{conf} immer mehr Nachrichten als bei Anwendung von λ benötigt. Es kommt daher hier in jedem Fall zu einem erhöhten Paketbedarf des Koordinators. In dichteren Netzen hingegen werden die Resultate mit λ'_{conf} besser. Hier werden entweder sehr ausfallsichere Knoten ermittelt, so

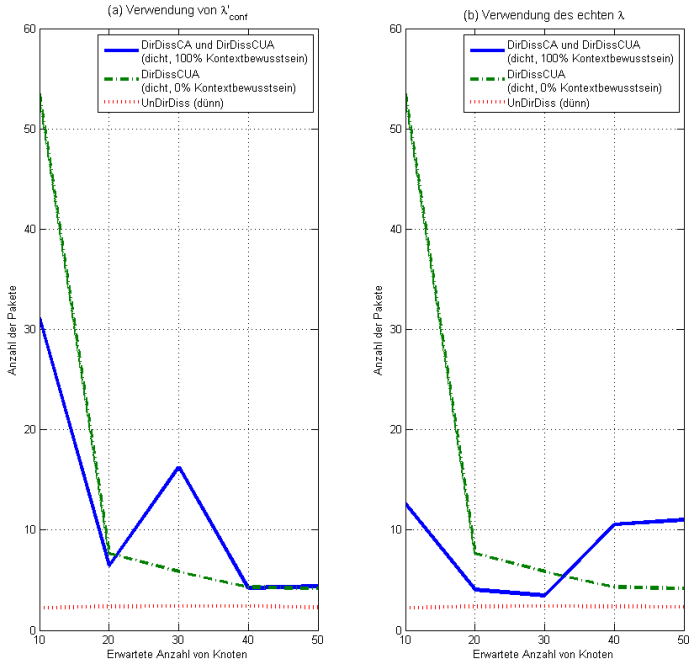

Abbildung 6.14: Vergleich der vom Koordinator benötigten Nachrichtenpakete bei Verwendung der unteren Grenze des Konfidenzintervalls für das geschätzte λ' und bei Verwendung des echten λ am Beispiel von Szenario 1

dass der Nachrichtenbedarf sinkt, oder aufgrund der höheren Netzdichte kann mit DirDissCUA schneller die benötigte Anzahl von Knoten bei der Verteilung erzielt werden.

Die Ergebnisse für Szenario 2 zeigen außerdem den Einfluss der Aufenthaltsdauern auf. In diesem Fall führt die Verwendung von λ immer zu den besseren Resultaten. Wie bereits in den vorherigen Testbereichen festgestellt, ist hier eine höhere Mobilitätsdynamik zwischen den Clustern gegeben. Zudem befinden sich in beiden Clustern immer etwa gleich viele Knoten (vgl. Tabelle 6.2), während bei Szenario 1 der Cluster C_2 etwa $\frac{3}{4}$ aller Knoten enthält. Somit ist bei Szenario 2 zum einen die Netzdichte nicht hoch genug, um mit λ'_{conf} ausreichend viele ausfallsichere Knoten für die Verteilung zu ermitteln. Es muss daher auf DirDissCUA zurückgegriffen werden. Zum anderen werden durch λ'_{conf} in der Regel bei DirDissCA mehr Knoten als mit λ benötigt. Dadurch ist auch hier bei Verwendung von DirDissCA mit einem erhöhten Nachrichtenbedarf zu rechnen.

Abschließend bleibt festzuhalten: *Durch Verwendung von λ'_{conf} kommt es bei der Kombination von DirDissCA und DirDissCUA zu einem leicht erhöhten Nachrichtenbedarf (in den Simulation im Bereich von 5 bis 10 Nachrichten). Dies ist zum einen auf den erhöhten Knotenbedarf von DirDissCA bei Verwendung von λ'_{conf} im Vergleich zum echten λ zurückzuführen. Zum anderen zeigt dies, dass bei nicht ausreichender Anzahl von ausfallsicheren Knoten häufiger auf DirDissCUA zurückgegriffen wird, bei der seitens des Koordinators mehr Nachrichten benötigt werden.*

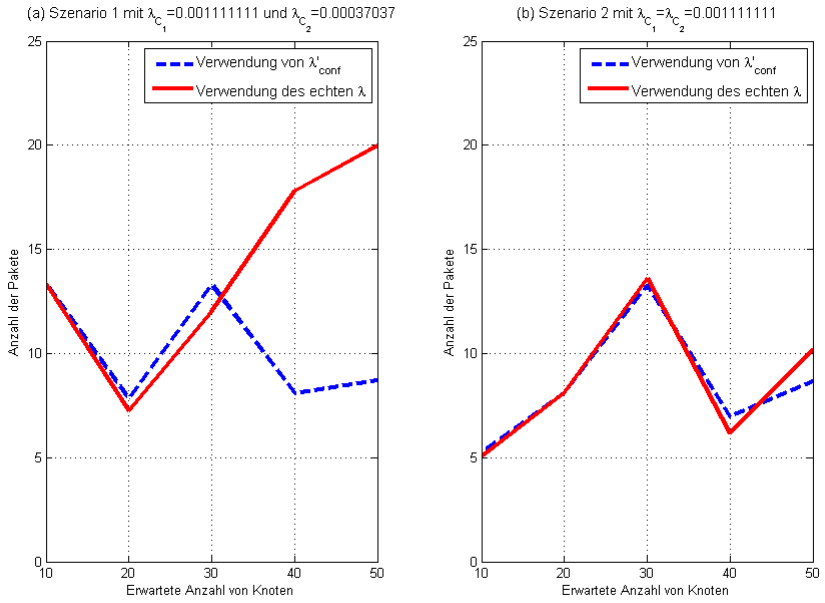

Abbildung 6.15: Vergleich der von den Teilnehmern benötigten Nachrichtenpakete bei den gewählten Szenarien

6.6.4.2 Nachrichtenanzahl des Klienten

Durch die Verwendung von λ'_{conf} wird der Nachrichtenbedarf der Teilnehmer bei den gewählten Szenarien nicht erhöht, sondern bleibt gleich oder wird geringer.

Abbildung 6.15 verdeutlicht den durchschnittlichen Nachrichtenbedarf von Transaktionsteilnehmern bei den beiden gewählten Szenarien.

Bei Szenario 1 ist der Nachrichtenbedarf für Netzdichten von 10, 20 und 30 Knoten bei λ und λ'_{conf} identisch. Die Anzahl variiert zwischen 7 und 20 Paketen. Befinden sich 40 und 50 Knoten im Netz, liegt der Bedarf bei etwa 8 Nachrichten, wenn λ'_{conf} angewendet wird. Im Gegensatz dazu führt die Anwendung von λ bei 40 Knoten zu einem Bedarf von 17 und bei 50 Knoten zu einem Bedarf von 20 Paketen.

Die Resultate für Szenario 2 sind für λ'_{conf} und λ fast idenisch. Hier bewegt sich die benötigte Nachrichtenanzahl zwischen 5 und 13 Paketen.

Der Nachrichtenbedarf der Teilnehmer bleibt bei Verwendung von λ'_{conf} bei beiden gewählten Szenarien gleich oder verringert sich. In dünneren Netzen werden bei Szenario 1 und 2 höchstens eine Nachricht mehr versandt. Gleichzeitig zeigt Szenario 1, dass in dichten Netzen bei Verwendung von λ'_{conf} der Nachrichtenbedarf der Teilnehmer verringert werden kann. Dies ist wiederum auf die Charakteristik des Szenarios zurückzuführen. Es befinden sich in Cluster C_2, wo die meisten Transaktionen stattfinden, auch $\frac{3}{4}$ aller Knoten (vgl. Tabelle 6.1), weshalb dort die Netzdichte höher ist (vgl. dazu vorherige Abschnitte). Da zudem bei DirDissCA mit λ'_{conf} ein höherer Knotenbedarf besteht (vgl. Beispiel in Abbildung 6.8) bzw. auf DirDissCUA zurückgegriffen wird, kann in jedem Fall ein höherer Verteilungsgrad erzielt werden. Für den Verteilungsgrad gilt also $d_{\lambda'_{conf}} \geq d_{\lambda}$.

Dies führt zu einer Reduzierung des Nachrichtenbedarfs beim Teilnehmer. Bei Szenario 2 ist dies aufgrund der Konfiguration nicht immer möglich, da die Netzdichte geringer als bei Szenario 1 ist. Erst bei einer Netzdichte von 50 Knoten zeigt sich eine Nachrichtenreduzierung der

Teilnehmer bei der Verwendung von λ'_{conf}.

6.7 Schlussfolgerungen

Die Simulationen des erweiterten SLS haben gezeigt, dass bei der Verwendung von λ'_{conf} als konservative Abschätzung der Aufenthaltsdauer bei den Verteilungsstrategien DirDissCA und DirDissCUA kürzere Unsicherheitszeiten als mit dem echten λ erzielt werden können. Die mit R_{SLS} vereinbarten Garantien können gewährleistet werden. Ursache dafür ist zum einen der erhöhte Bedarf an Knoten für die Speicherung des Transaktionslogs aufgrund der konservativen Abschätzung mit λ'_{conf}. Zum anderen beeinflusst der garantierte Recoveryzeitraum t_m den Knotenbedarf im Zusammenspiel mit der Aufenthaltsdauer. Wenn t_m deutlich größer als die Aufenthaltsdauer ist, werden noch einmal mehr Knoten benötigt, um die geforderte Garantie von R_{SLS} zu erfüllen. Dies gilt sowohl für λ'_{conf} wie für das echte λ.

Aus diesem gestiegenen Bedarf an Knoten für die Speicherung des Transaktionslogs resultieren zwei Folgen. Zum einen wird DirDissCA nur dann verwendet, wenn die geforderte und somit nun höhere Anzahl von ausreichend ausfallsicheren Knoten mit Kontextbewusstsein für die Verteilung zur Verfügung steht. Wegen des gestiegenen Knotenbedarfs ist der Anteil dieser Strategie daher geringer. Auf der anderen Seite wird in allen anderen Fällen – d.h. wenn nicht genug ausfallsichere Knoten mit Kontextbewusstsein gefunden werden – DirDissCUA verwendet. Diese Strategie benötigt mehr Nachrichten als DirDissCA. Beide Varianten führen allerdings zum selben Ergebnis: einem gleichen oder höheren Verteilungsgrad mit $d_{\lambda'_{conf}} \geq d_{\lambda}$ und somit einer kürzeren Unsicherheitszeit von ausgefallenen Teilnehmern.

Wie bereits erwähnt, spielt auch die Aufenthaltsdauer in Clustern eine Rolle. Diese beeinflusst die Anzahl von Knoten pro Cluster und damit auch die Anzahl von Transaktionen darin. Von diesen Charakteristiken sind auch die Verteilungsstrategien betroffen. Sind viele Knoten in einem Cluster vorhanden, so ist eine bessere und effizientere Verteilung des Transaktionslogs möglich. Dann kann beispielsweise mit DirDissCA eine nachrichtensparende Strategie vom Koordinator benutzt werden. Darüber hinaus führt ein längerer Aufenthalt von Knoten in einem Cluster dazu, dass ausgefallene Teilnehmer schneller ihre Recovery innerhalb von t_m erfolgreich beenden können.

Die Effizienz der Verteilungsstrategien wird ebenfalls von Aufenthaltsdauern in den Clustern beeinflusst. Je dichter das MANET ist, desto besser funktionieren gerichtetete Verteilungs- bzw. Recoverystrategien wie DirDissCA, Anfrage an den Koordinator sowie die gerichtete Recovery. Da die Verwendung von λ'_{conf} mehr Knoten für die Verteilung benötigt als das echte λ, wird DirDissCA in dünneren Netzen seltener verwendet. Der Einsatz von λ'_{conf} eignet sich eher für dichte MANETs.

Zusätzlich zeigen die Untersuchungen, dass Fehler aufgrund von Abwanderung aus einem Cluster nur einen geringen Anteil an allen Fehlersituationen haben. Der Anteil von Abwanderungsfehlern ist noch geringer als der von Knotenfehlern. Die Hauptursache von Recoverys sind Kommunikationsfehler, ausgelöst durch kurzfristige Mobilität. Langfristige Mobilität stellt also bei der Berücksichtigung von Fehlern nur eine untergeordnete Rolle dar.

Der Nachrichtenbedarf des Koordinators und Teilnehmers beim SLS-Prozessablauf reflektiert diese Beobachtungen. Bei der Verwendung von λ'_{conf} benötigt der Koordinator nur wenig mehr Nachrichten als bei der Benutzung des echten λ. Wie oben gezeigt, werden entweder für die Verteilung mit DirDissCA und λ'_{conf} mehr Knoten benötigt oder es wird DirDissCUA angewandt. Bei den Teilnehmern bleibt der Bedarf an Nachrichten entweder gleich oder verringert sich in dichten MANETs. Der Grund dafür ist der gleiche bzw. höhere Verteilungsgrad, der das Auffinden des Transaktionslogs mit weniger Aufwand ermöglicht.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde der bestehende SLS durch Simulation mit dem Netzwerksimulator NS2 evaluiert. Ziel des SLS ist die Reduzierung von Unsicherheitszeiten beim Auftritt von Fehlersituationen während der Durchführung von Transaktionen in MANETs. Zu diesem Zweck müssen im Vorfeld durch die Transaktionsteilnehmer die gewünschten Garantien vereinbart werden. Dies betrifft z.B. den garantierten Recoveryzeitraum t_m oder die Wahrscheinlichkeit für das erfolgreiche Wiederauffinden des Transaktionslogs $p_{retrieval}$ innerhalb von t_m (zusammen als Zuverlässigkeit R_{SLS} bezeichnet).

Zur Verteilung des Transaktionslogs nach dem Auftritt einer Fehlersituation in der Commit-Phase werden durch den Koordinator die drei Verteilungsstrategien DirDissCA, DirDissCUA und UnDirDiss verwendet. Diese berücksichtigen die Wahrscheinlichkeiten für Knotenfehler aufgrund von Energie und technischem Defekt. Es wird grundsätzlich zwischen dichten und dünnen MANETs sowie Knoten mit und ohne Kontextbewusstsein unterschieden. Die ersten beiden gerichteten Verfahren DirDissCA und DirDissCUA sind für dichte MANETs entwickelt worden. Bei DirDissCA versucht der Koordinator auf eine spezielle ausfallsichere Gruppe von n Knoten mit Kontextbewusstsein zu verteilen. Im Fall von DirDissCUA wird hingegen nur die benötigte Anzahl von n Knoten ermittelt. Während beim ersten Verfahren das Transaktionslog direkt an die n Knoten gesendet werden kann, sind bei DirDissCUA wiederholte Broadcasts notwendig. Das Verfahren UnDirDiss ist für dünne MANETs entwickelt worden. In diesem Fall wird das Transaktionslog zusammen mit dem vom Koordinator berechneten Verteilungsgrad mit einem einzigen Broadcast verteilt. Die empfangenden Knoten senden das Log selbst in Abhängigkeit vom Verteilungsgrad mittels Broadcast weiter.

An den Verteilungsstrategien orientieren sich auch die Recoverystrategien ausgefallener Transaktionsteilnehmer. Jeder Knoten sendet bei jedem Recoveryversuch eine Anfrage an den Koordinator. Ergänzend dazu wird eine gerichtete Recoveryanfrage an die n Knoten geschickt, wenn der Koordinator zu Verteilung DirDissCA verwendet hat. Eine ungerichtete Recovery wird durchgeführt, wenn der Koordinator das Transaktionslog entweder mit DirDissCUA oder UnDirDiss verteilt hat. Bei beiden Recoveryverfahren muss der ausgefallene Teilnehmer Broadcasts verwenden.

Bei der Evaluation wurde deutlich, dass mit Hilfe des SLS sowohl in dünnen wie in dichten MANETs eine Reduzierung der Unsicherheitszeiten von Teilnehmern nach Fehlersituationen in der Commit-Phase möglich war. Zudem konnten 99% aller Recoverys innerhalb des Recoveryzeitraums t_m erfolgreich abgeschlossen werden. Die vereinbarten Garantien konnten also gewährleistet werden.

Ein weiteres Resultat war die Fehlerursache für das Starten einer Recovery. Nur ein sehr geringer Teil aller Recoverys wurde durch Knotenfehler ausgelöst. Als Hauptursache mit einem Anteil von mehr als 90% stellten sich Kommunikationsfehler heraus, die durch die kurzfristige Mobilität der mobilen Geräte entstanden. Aus den Kommunikationsfehlern resultierten bei dem SLS-Prozessablauf mehr Fehlersituationen als beim Vergleichsprozessablauf, da ersterer zwei

zusätzliche Zustände im Protokoll beinhaltet.

Als besonders effiziente Verteilungsstrategien müssen DirDissCUA sowie die Kombination von DirDissCUA mit DirDissCA hervorgehoben werden. Das gerichtete Verfahren DirDissCA wurde jedoch selten eingesetzt, d.h. es standen nur in wenigen Fällen ausreichend ausfallsichere Knoten für die Strategie zur Verfügung. Somit ist diese Strategie – wie bei der Entwicklung angedacht (vgl. Kapitel 4) – nur für dichte MANETs geeignet, da dort ausreichend viele ausfallsichere Knoten mit Kontextbewusstsein gefunden werden können.

DirDissCUA hingegen ist die effizienteste Verteilungsstrategie. Durch die wiederholten Broadcasts bei der Verteilung wird ein hoher Verteilungsgrad d erzeugt. Die Unsicherheitszeit ausgefallener Teilnehmer wird deutlich verkürzt, da diese das gesuchte Transaktionslog mit nur wenigen Recoveryversuchen bei einem ihrer Nachbarn finden. Es zeigte sich, dass dieses Verfahren nicht nur in dichten MANETs zu guten Ergebnissen führt, wie bei der Entwicklung geplant war (vgl. Kapitel 4), sondern auch in dünnen MANETs.

Der Erfolg von DirDissCUA wird im Nachrichtenbedarf des Koordinators reflektiert. Er benötigt im Vergleich zu den anderen Verfahren eine erhebliche Anzahl mehr an Nachrichten. Dies resultiert aus häufigen Broadcasts innerhalb von 0.2 Sekunden, wie sie bei DirDissCUA eingesetzt werden. Durch den auf diese Art erzeugten hohen Verteilungsgrad d können die kürzeren Unsicherheitszeiten erzielt werden.

Mit UnDirDiss sind ebenfalls kürzere Unsicherheitszeiten entstanden als bei Anwendung des Vergleichsprozessablaufs, allerdings sind diese länger als bei den beiden vorherigen Varianten. Die Ursache dafür ist der Verteilungsmechanismus. Es wird durch den Koordinator nur ein Broadcast verwendet. Alle Knoten, die das Transaktionslog erhalten, verteilen es erst nach 60 Sekunden mittels adaptiven Push weiter. Da sich Kommunikationsfehler aufgrund von kurzfristiger Mobilität als Hauptproblem erwiesen haben, ist das Transaktionslog nicht ausreichend schnell im Netz verteilt. Somit sind Teilnehmer länger unsicher. Das Verfahren ist daher mit dem momentanen zeitlichen Abstand zwischen Broadcasts nicht ausreichend effizient in dünnen Netzen.

Zweites Hauptziel war die Erweiterung des SLS um Berücksichtigung der Mobilität zwischen Clustern in einem MANET. Konkret sollte die Wahrscheinlichkeit für das Verlassen eines Clusters geschätzt werden. Diese Abwanderungswahrscheinlichkeit sollte bei den Knoten mit Kontextbewusstsein als ein weiterer Faktor neben den Wahrscheinlichkeiten für verbrauchte Energie und technischen Defekt mit einbezogen werden. Damit soll R_{SLS} auch in cluster-basierten Szenarien gewährleistet werden können. Wichtig war, dass das Netz nicht zusätzlich durch Nachrichten belastet wird und Geräte ihre individuelle Abwanderungswahrscheinlichkeit ohne komplexe Berechnungen selbst bestimmen können.

Das entwickelte host-zentrierte Verfahren berücksichtigt diese Faktoren. Anhand einer *Karte* der Cluster können die Knoten in regelmäßigen Abständen ihren Aufenthalt bestimmen. Dadurch kann jeder Knoten bei jedem Besuch eines Clusters die Aufenthaltsdauer darin berechnen. Es sind des weiteren keine zusätzlichen Nachrichten für das Verfahren notwendig. Bei Anfrage des Koordinators nach der Ausfallwahrscheinlichkeit seiner Nachbarn senden die Knoten zusätzlich ihre geschätzte Abwanderungsrate aus seinem aktuellen Cluster mit. Durch Anwendung eines Maximum-Likelihood-Schätzers kann durch jeden Knoten seine individuelle Abwanderungsrate λ für einen Cluster geschätzt werden. Untersuchungen dieses Verfahrens durch Simulationen haben gezeigt, dass nicht immer eine exakte Schätzung der echten Abwanderungsrate λ möglich ist. Deshalb verwenden die mobilen Knoten die untere Grenze des Konfidenzintervalls ihres geschätzten λ, bezeichnet als λ'_{conf}. Bei diesem Wert kann mit statistischer Sicherheit γ davon ausgegangen werden, dass das echte λ mindestens so groß ist. Erste Simulationen dieser konservativen Abschätzung unter Verwendung des Area Graph-Based Mobilitätsmodells zeigten, dass dieses Verfahren sinnvoll ist, insbesondere bei großen Abweichungen der Schätzung von λ.

Wichtig bei dem entwickelten Verfahren ist außerdem, dass keine Vorkonfiguration der Knoten notwendig ist. Knoten mit Kontextbewusstsein können jederzeit neu in ein MANET kommen. Solange sie nicht mit der vereinbarten statistischen Sicherheit eine Angabe über ihre Abwanderungswahrscheinlichkeit machen können, werden sie für die Verteilung durch den Koordinator nicht berücksichtigt. Erst wenn ihre geschätzte Wahrscheinlichkeit ausreichend genau ist, werden sie für die Verteilung bei DirDissCA einbezogen.

Anhand der Simulationen wird deutlich, dass bei Verwendung von λ'_{conf} kürzere Unsicherheitszeiten als mit λ erreicht werden. Außerdem hat sich gezeigt, dass die mit R_{SLS} vereinbarten Garantien in cluster-basierten Szenarien gewährleistet werden konnten.

Die Resultate führten zu dem Ergebnis, dass mit DirDissCA bei Verwendung von λ'_{conf} in Kombination mit DirDissCUA kürzere Unsicherheitszeiten erreicht werden konnten als mit dem echten λ. Ursache dafür ist der gleiche oder erhöhte Verteilungsgrad mit $d_{\lambda'_{conf}} \geq d_{\lambda}$, der erzielt wird. Dies ist darauf zurückzuführen, dass DirDissCA mit λ'_{conf} aufgrund der konservativen Abschätzung einen höheren Knotenbedarf für die Verteilung hat als mit dem echten λ. Je dichter das MANET ist, desto besser funktioniert DirDissCA bzw. desto häufiger wird dieses Verfahren eingesetzt. In dünneren Netzen wird hingegen DirDissCUA angewandt, womit ein höherer Verteilungsgrad als mit DirDissCA erzielt wird.

Bei der Analyse des Nachrichtenbedarfs des Koordinators ist keine drastische Erhöhung durch die Verwendung von λ'_{conf} gegenüber der Anwendung des echten λ zu beobachten. In einigen Fällen zeigte sich sogar eine Reduzierung. Ähnlich sieht es bei den Teilnehmern aus. Hier ist in dichten Netzen sogar eine deutliche Nachrichtenreduzierung möglich. Dieses Ergebnis ist auf den erhöhten Verteilungsgrad $d_{\lambda'_{conf}}$ zurückzuführen.

Abschließend bleibt festzuhalten, dass mit Hilfe des SLS die Unsicherheitszeit von Teilnehmern nach Fehlersituationen erheblich verkürzt werden kann. Dabei ist jedoch zu berücksichtigen, durch die Folge dieser Effizienz ein leicht gestiegener Nachrichtenbedarf ist (in den Simulation beträgt diese Erhöhung etwa 5 bis 10 Nachrichten). Außerdem wurde gezeigt, dass mobile Geräte in der Lage sind, ihre eigene Abwanderung aus Clustern in MANETs selbst zu schätzen, ohne dass das Netz mit zusätzlichen Nachrichten belastet wird. Bei Verwendung der unteren Grenze des Konfidenzintervalls für die geschätzte Abwanderungsrate kann aufgrund eines erhöhten Verteilungsgrades die Unsicherheitszeit gegenüber der Anwendung des echten Wertes verbessert werden. Der erhöhte Nachrichtenbedarf ist also dahingehend akzeptabel, dass eine bessere Verteilung erzielt werden werden. Es zeigte sich zudem in den Simulationen, dass nicht Knotenfehler oder Abwanderungen aus einem Cluster Hauptursache für Fehlersituationen bei Transaktionen sind, sondern Kommunikationsfehler aufgrund von kurzfristiger Mobilität.

7.2 Ausblick

Trotz der erzielten Verbesserungen in Bezug auf die Verkürzung von Unsicherheitszeiten ausgefallener Teilnehmer bei Transaktionen in MANETs sind Überlegungen zu weiteren Veränderungen sinnvoll. Dies betrifft zum einen Aspekte des SLS, die im Rahmen dieser Arbeit nicht detaillierter untersucht werden konnten. Zum anderen haben sich Problematiken des bestehenden SLS gezeigt, bei denen eine Weiterentwicklung förderlich sein kann.

Verfeinerung der Parameter, insbesondere der Timer

Der SLS und der dazugehörige Prozessablauf zeichnen sich durch eine Vielzahl von Parametern aus. Im Rahmen dieser Evaluation sind einige dieser Parameter fest ausgewählt worden aufgrund von Vorberechnungen und Tests und wurden nicht mehr verändert. Dies betrifft u.a.

die zwei grundlegenden Parameter für die Zuverlässigkeit des SLS: den garantierten Recoveryzeitraum t_m und die gewünschte Garantie für den Erhalt des Transaktionslogs innerhalb dieser Zeit $p_{retrieval}$. Beide Parameter haben erheblichen Einfluss auf die Funktion des SLS haben, da sie den Bedarf an Knoten für die Verteilung bestimmen.

Bisher ist von einem großen Einfluss der Knotenausfälle auf Fehlersituationen in MANETs ausgegangen worden. Da jedoch die Simulationen gezeigt haben, dass Kommunikationsfehler das Hauptproblem darstellen, sollten insbesondere in Bezug auf t_m Untersuchungen durchgeführt werden. Der Anteil von Knotenfehlern mit langer Abwesenheit ist sehr gering, daher kann t_m kleiner gewählt werden (hier wurden 3600 Sekunden = 1 Stunde sowie 1200 Sekunden = 20 Minuten verwendet). Dazu ist ebenfalls eine Analyse dahingehend sinnvoll, wie groß das Zeitfenster der Verbindungsabbrüche überhaupt ist. Dementsprechend kann dann ein angemessener Wert für t_m ausgewählt werden. Ein kürzeres Zeitfenster für den Recoveryzeitraum kann den Bedarf an Knoten für die Verteilung – wie bereits Analysen in Abschnitt 6.4.1.3 gezeigt haben – deutlich verringern, so dass Nachrichten und Speicher gespart werden können.

Auch die Timer des Prozessablaufs wurden für die Simulationen dieser Arbeit fest ausgewählt und sind nicht weiter verändert worden. Es sollte zum einen konkret untersucht werden, welcher Zusammenhang zwischen der Größe der Timer und dem Auftreten von Kommunikationsfehlern besteht. Da der SLS-Prozessablauf länger als der Vergleichsprozessablauf ist und dadurch mehr Fehlersituationen auftreten, ist dies ein wichtiger Aspekt (vgl. Abschnitt 5.3.2).

Zum anderen beeinflussen die Timer direkt den Nachrichtenbedarf des Koordinators und des Teilnehmers. Bei DirDissCUA beeinflusst der coord_DistPongAckTimer den Abstand zwischen den Broadcasts zur Verteilung auf die n anonymen Knoten. In den Simulationen wurden 0.2 Sekunden verwendet. Zu untersuchen wäre hier, ob mit einem größerem Zeitfenster ebenfalls eine deutliche Reduzierung der Unsicherheitszeiten erzielt werden kann. Die gleichen Überlegungen gelten für die Recoverytimer des Teilnehmers. Sie bestimmen, in welchen Zeitabständen ein Teilnehmer einen Recoveryversuch startet. Im Rahmen dieser Simulationen wurden für den ersten Versuch 2 Sekunden und für jeden weiteren 5 Sekunden gewählt. Es bleibt zu untersuchen, inwieweit mit größeren bzw. kleineren Werten für diese Timer das Unsicherheitsfenster bei Fehlersituationen beeinflusst werden kann.

Ähnliche Überlegungen gelten für die Strategie UnDirDiss und die Verwendung des Push-Protokolls. Es wurde in den Untersuchungen ein Push-Intervall von 60 Sekunden verwendet. Grundlage war wiederum die Annahme von Knotenfehlern als Hauptursache von Recoverys. Entsprechend kann zur Behandlung von Kommunikationsfehlern nicht schnell genug der benötigte Verteilungsgrad erzeugt werden. Auch hier sollte eine konkrete Evaluation des Timers vorgenommen werden.

Erweiterung der gerichteten Verteilungsstrategien für *dichte* MANETs

Die Verteilungsstrategien des SLS sind bisher zur Behandlung von Knotenfehlern entwickelt worden. Kommunikationsfehler finden bisher keine Berücksichtigung. Es bieten sich daher für die Strategien diesbezügliche Erweiterungen an. Das bedeutet, dass insbesondere bei den beiden Verfahren für dichte MANETs die Wahrscheinlichkeit für Verbindungsabbrüche mit einbezogen werden sollte. Bei einem dichten MANET wird die Verteilungsstrategie durch den Koordinator nach einem Broadcast ausgewählt. Mögliche Erweiterungen für die SLS-Strategien stellen z.B. Ideen der Ansätze [53, 15, 52] oder des in Abschnitt 3.3.2 vorgestellten Verfahrens dar. Gleichzeitig empfiehlt sich eine Gewichtung der Ausfallwahrscheinlichkeiten. Da Kommunikationsfehler das Hauptproblem darstellen, sollte die Wahrscheinlichkeit für Verbindungsabbrüche während der Kommunikation höher bewertet werden als die Wahrscheinlichkeit für Knotenfehler.

Ziel ist bei den gerichteten Verfahren für dichte MANETs die Bestimmung einer Gruppe von

Knoten zur Speicherung des Transaktionslogs. Jeder Knoten mit Kontextbewusstsein kann regelmäßig seine eigene Position und seine Geschwindigkeit bestimmen. Wenn der Koordinator mit einem Broadcast die Knoten seiner Umgebung in dichten MANETs nach ihrer Ausfallwahrscheinlichkeit befragt, können ihm diese Informationen mit zur Verfügung gestellt werden. Er kann zusammen mit seinen Positions- und Bewegungsdaten z.B. die in Abschnitt 3.3.2 dargestellte LET berechnen. Es können dann die n notwendigen Knoten ausgewählt werden, zu denen voraussichtlich zum Verteilungszeitpunkt (dieser kann vom Koordinator anhand seiner Timer bestimmt werden) noch eine Verbindung besteht. In Abhängigkeit von der verwendeten Strategie (DirDissCA) können weitere Ausfallwahrscheinlichkeiten – z.B. Energie oder Defekt – mit einbezogen werden. Zur Behandlung von Kommunikationsfehlern sollte jedoch die Gewichtung der Vorhersage der kurzfristigen Mobilität, also beispielsweise der LET, höher sein, damit die erfolgreiche Verteilung sichergestellt werden kann. Für DirDissCUA bietet sich an, dass die Transaktionslogs weiter im Netz verteilt werden, z.B. mit einem der oben genannten Ansätze, um einen möglichst hohen Verteilungsgrad zu erzielen. Die n Knoten stellen quasi eine Art *Weiterleitungsgruppe* dar. Alternativ können diese n Knoten bei der Weiterverteilung in Analogie zur Strategie UnDirDiss verfahren und regelmäßig das Push-Protokoll verwenden (dessen Zeitfenster kürzer als bisher gewählt werden sollte). Den Teilnehmern sollten ergänzend dazu nicht nur bei DirDissCA die Informationen über die n Knoten mitgeteilt werden, sondern auch bei DirDissCUA. Somit kann bei beiden Verfahren durch die Teilnehmer eine gerichtete Recovery an die n Knoten angewandt werden. Ergänzend dazu sollte bei Anwendung von DirDissCUA durch die Teilnehmer zusätzlich in größeren Abständen eine ungerichtete Recovery starten. Die Annahme ist, dass die Verteilung des Transaktionslogs im ganzen Netz mittels des Push-Protokolls länger dauert.

Die Information über die unter Berücksichtigung der LET ausgewählten n Knoten (sowohl bei DirDissCA wie bei DirDissCUA) bietet nun die Möglichkeit, Kommunikationsfehler aufgrund von kurzfristiger Mobilität zu behandeln. Das Einbeziehen der gewichteten Ausfallwahrscheinlichkeit aufgrund von Energie, Defekt oder Abwanderung bei DirDissCA sowie die Verteilung mittels Broadcast im Netz bei DirDissCUA ermöglicht zusätzlich den erfolgreichen Transaktionsabschluss beim Auftritt von Knotenfehlern.

Ausnutzung der Informationen über Bewegung zwischen Clustern

Bisher wird die Information über die Cluster von den Knoten mit Kontextbewusstsein genutzt, um ihre individuelle Abwanderungswahrscheinlichkeit zu schätzen. Zurzeit finden Transaktionen innerhalb eines Clusters (sog. Transaktionscluster) statt. Recoverys werden nur innerhalb des zugehörgen Transaktionsclusters unterstützt. Das entwickelte Verfahren bietet jedoch die Möglichkeit für Erweiterungen.

Durch die Speicherung der Informationen über den Wechsel zwischen Clustern, können auch Wege *gelernt* werden. Damit zusammenhängend kann bei komplexeren Szenarien ermittelt werden, welche Wege häufiger verwendet und welche Cluster häufiger als andere besucht werden. Aus diesen Informationen können Knoten langfristig selbständig einen gewichteten Graphen über ihr Bewegungsverhalten in einem bestimmten Gebiet, welches durch verschiedene Cluster und Wege definiert ist, lernen. (Zur Reduzierung des Rechenaufwandes bei der Positionsbestimmung in komplexeren Szenarien können z.B. Algorithmen wie Breitensuche eingesetzt werden.) Diese Daten können verwendet werden, damit ein Knoten im Cluster C_i eine Aussage darüber machen kann, welchen Cluster C_j er als nächstes mit großer Wahrscheinlichkeit betreten wird. Wird diese Information dem Koordinator ebenfalls zur Verfügung gestellt bei der Sammlung der Ausfallwahrscheinlichkeiten, kann diese an die Teilnehmer weitergegeben werden. Verlässt ein Teilnehmer z.B. während der Transaktion den Cluster, kann er aufgrund der Informationen zu den n Knoten und ihrem Bewegungsverhalten entscheiden, ob sich einer dieser Knoten mit großer

Wahrscheinlichkeit auch in seinem aktuellen Cluster aufhält. Somit kann einem ausgefallenen Teilnehmer die erfolgreiche Recovery in einem anderen Cluster ermöglicht werden.

Anhang A

Ergänzendes Material

A.1 Berechnung der erwarteten Anzahl von Knoten im MANET

Aufgrund der Ausfälle stehen im Verlauf der Simulation nicht immer alle Knoten im MANET zur Verfügung. Ein Teil der Knoten wird mit großer Wahrscheinlichkeit immer ausgefallen sein. Die *echte* Anzahl von Knoten wird also nicht zur Verfügung stehen, so dass man anhand der oben spezifierten Ausfälle eine *erwartete* Anzahl von nicht ausgefallenen Knoten im MANET zu jedem Simulationszeitpunkt schätzen muss.

Annahme dafür ist, dass alle Knoten mit gleicher Wahrscheinlichkeit aufgrund von Energie und technischem Defekt ausfallen. Zur Modellierung eines MANETs bestehend aus n Knoten mit Ausfällen bieten sich Markovketten an[1]. Das dazu gehörige Markovmodell umfasst *n+1* Zustände. Im Zustand 0 ist kein Gerät ausgefallen, im Zustand 1 ein Gerät usw. Im Zustand n schließlich sind alle Geräte ausgefallen. Die Zustandsübergänge vom Zustand i zum Zustand *i+1*, wobei $i = 0, \ldots, n$, entsprechen den Wahrscheinlichkeiten für einen weiteren Ausfall, bzw. vom Zustand i zum Zustand *i-1* der Reparatur eines Gerätes. Die Wahrscheinlichkeit für den Ausfall eines Gerätes beträgt (ausgehend vom Zustand 0) $n \cdot \lambda$, für einen weiteren Geräteausfall (ausgehend vom Zustand 1) $(n-1) \cdot \lambda$ usw. Bei der Reparatur verhält es sich analog. Ausgehend vom Zustand n beträgt die Wahrscheinlichkeit für die Reparatur eines Geräte $n \cdot \mu$, für die Reparatur eines weiteren Gerätes $(n-1) \cdot \mu$ usw. Ziel ist es nun, den Zustand i zu bestimmen, für den die Wahrscheinlichkeit $P(i)$ im Verlauf der Simulationen am größten ist. Dabei entspricht das i dann der Anzahl von ausgefallenen Geräten und die Anzahl der verfügbaren Geräte x beträgt entsprechend $n - i$.

Für die Berechnung wird die Wahrscheinlichkeit für jeden Zustand der Markovkette (für Details zu Markovprozessen sei z.B. auf [25, 7] verwiesen) bestimmt. Für viele Anwendungen geschieht dies in Abhängigkeit von der Zeit t. Hier möchte man abschätzen, wie viele Knoten durchschnittlich ausgefallen sind. Hierzu ist zunächst die folgende Zustandsübergangsmatrix notwendig:

$$A = \begin{pmatrix} -n\lambda & \mu & 0 & 0 & \ldots & 0 \\ n\lambda & -(\mu + (n-1)\lambda) & 2\mu & ... & 0 & 0 \\ 0 & (n-1)\lambda & -(2\mu + (n-2)\lambda) & ... & 0 & 0 \\ \ldots & 0 & (n-2)\lambda & ... & (n-1)\mu & \ldots \\ ... & ... & 0 & ... & -((n-1)\mu + \lambda & n\mu \\ 0 & 0 & 0 & 0 & \lambda & -n\mu \end{pmatrix} \tag{A.1}$$

wobei die a_{ik}, $k = 0, 1, \ldots, j-1, j+1, \ldots, n$ die jeweilige Übergangsraten darstellen. Die benötigten Zustandsberechnungen können wie folgt aufgeschrieben werden (Annahme ist hier, dass der Initialzustand 0 vorausgesetzt wird zu Beginn der Simulation):

$$A \cdot \vec{P}(t) = \dot{P}(t) \tag{A.2}$$

[1] Anmerkung: Hier spielen Verbindungen zwischen den Knoten keine Rolle. Es wird nur die Anzahl von Knoten bei der Modellierung betrachtet.

wobei $\dot{P}(t) = -P_j(t) \sum_{k=0,k\neq j}^{n} a_{jk} + \sum_{k=0,k\neq j}^{n} P_k(t) \cdot a_{kj}$. Mit der weiteren Annahme, dass die Summe aller Zustandswahrscheinlichkeiten 1 sein muss, kann man alle Zustandswahrscheinlichkeiten bestimmen. Diese stellen eine Menge von linearen Differentialgleichungen dar, die z.B. mittels Laplace-Transformation gelöst werden können. Auf Details wird hier nicht näher eingegangen, sondern auf [25] S. 214 ff. verwiesen. Die Berechnungen können dann z.B. mittels Matlab (vgl. [1]) vorgenommen werden.

Als Beispiel zeigt Abbildung 5.4 für ein Szenario mit 10 Knoten, wie groß die Wahrscheinlichkeit für jeden Zustand der Markovkette ist. Es ist erkennbar, dass im Verlauf der Zeit die Wahrscheinlichkeit für den Zustand $P(5)$ am größten ist. Man kann daher davon ausgehen, dass im Durchschnitt etwa 5 Knoten ausgefallen sind und 5 Knoten zur Verfügung stehen. Im Rahmen dieser Arbeit wird statt der *echten* Knotenanzahl im MANET die sog. *erwartete* Knotenanzahl angegeben. Bei den Diagrammen der Evaluationen wird ausschließlich die erwartete Knotenanzahl verwendet, da diese ein genaueres Bild des der Simulation zugrunde liegenden Netzes darstellt. Die jeweils erwartete Knotenanzahl für die verwendeten Szenarien ist ebenfalls in Tabelle 5.1 zu finden.

A.2 Änderungen und Erweiterungen an NS2

A.2.1 Grundlegendes Vorgehen bei der Entwicklung eines neuen NS2-Protokolls

Für die Umsetzung des SLS-Konzeptes ist die Entwicklung eines neuen Protokolls notwendig. Dazu sind drei wesentliche Teilstücke notwendig, auf die hier beispielartig kurz eingegangen wird. Die Code-Fragmente stammen dabei aus dem für diese Arbeit entwickelten Protokoll für den SLS-NS2-Agenten.

Deklaration einer Datenstruktur für den neuen Header

Als erstes ist die Definition einer neuen Datenstruktur für den Paketheader notwendig. Damit ist dann der Transport der notwendigen Daten möglich. Die Deklaration erfolgt in einer Header-Datei, während die dazu notwendigen Zugriffsfunktionen sowie die Bindung zu Tcl in der entsprechenden C++-Datei implementiert werden müssen. Hier ein Ausschnitt aus der in dieser Arbeit verwendeten Datei *hdr_ dissemination.h:*

```
...
struct hdr_dissemination {
// message type
message_type type;

// transaction id
int taId;
// technical failure rate of a node
double techFailureRate;
// energy failure rate  of a node
double energyFailureRate;
// churn rate
double churnRate;
// the mission time
double missionTime;
// neighbors used in case of the direct dissemination
std::vector<agentAddr>* neighbors;
...
// Header access methods
```

```
static int offset_;
// required by PacketHeaderManager
  inline static int& offset() {
    return offset_;
  }
  inline static hdr_dissemination* access(const Packet* p) {
    return (hdr_dissemination*) p->access(offset_);
  }
};
...
```

Für den Zugriff auf den Header beim Erhalt einer Nachricht muss in C++ die dazu gehörige access()-Methode implementiert werden (in diesem Fall in der *hdr_ dissemination.cc*).

Mit der hier verwendeten Datenstruktur können u.a. die Adresse des Koordinators sowie die Adresse des Absenders angegeben werden. Außerdem existieren z.B. Felder für die Transaktionsnummer, die Ausfallraten sowie für eine Liste von Knoten, die vom Koordinator an die Klienten für die gerichtete Verteilung verschickt werden.

Wichtig ist insbesondere ein Feld für den Nachrichtentyp. Durch unterschiedliche Nachrichtentypen können Knoten später entscheiden, ob eine Nachricht sie in einem Zustand erreicht, in dem sie diese erwarten.

Deklaration von Agenten

Die bei NS2 verwendeten sog. *Agenten* beinhalten die eigentliche Funktionalität des Protokolls. Auch hier ist zunächst eine Header-Datei notwendig, in der alle Variablen, Methoden und Timer spezifiert werden. Zusätzlich können darin auch Zustände für Knoten spezifiziert werden, mit denen der in Kapitel VERWEIS vorgestellte Prozessablauf gesteuert werden kann.

Mit der entsprechenden C++-Datei wird dann der Ablauf des eigentlichen Protokolls implementiert. So können z.B. Methoden implementiert werden zum Senden von Nachrichten. Mit statischen Klassenvariablen können dann einzelne Ereignisse gezählt werden für die spätere Analyse. Beispielhaft hier ein Ausschnitt aus der verwendeten Klasse *disseminationAgent.cc*:

```
static class DisseminationClass : public TclClass {
  public:  DisseminationClass() : TclClass("Agent/Dissemination") {}
           TclObject* create(int, const char*const*) {
             return (new DisseminationAgent());
           }
} class_dissemination;
...
/** global counter for the experiments */
int DisseminationAgent::anz_ta_start = 0;
int DisseminationAgent::anz_ta_suc = 0;
int DisseminationAgent::anz_ta_sucrec = 0;
...
/** sending of the vote by the clients */
int DisseminationAgent::vote() {
  Packet* pkt = allocpkt();
  hdr_ip* iph = hdr_ip::access(pkt);
  iph->saddr() = here_.addr_;
  iph->daddr() = coord_addr_;
  iph->dport() = coord_port_;
```

```
  hdr_dissemination* disshdr = hdr_dissemination::access(pkt);
  disshdr->type = PING;
  disshdr->taID = myTaId;
  ...
  return TCL_OK;
}
...
```

Wichtig in dieser Klasse sind insbesondere die Methoden command() und recv(). Erstere ist von Bedeutung für die Parameterübergabe in Tcl. Das folgende Beispiel verdeutlicht, wie Informationen von Tcl in C++ verarbeitet werden können:

```
else if (argc == 3) {
...
  // get information about another agent from tcl
  else if (strcmp(argv[1], "add-member") == 0) {
    DisseminationAgent *a = (DisseminationAgent *)(TclObject::lookup(argv[2]));
    return TCL_OK;
  }
  // set the transaction id
  else if (strcmp(argv[1], "set-taId") == 0) {
    taId_ = atoi(argv[2]);
    return TCL_OK;
  }
...
```

Als erstes wird überprüft, wie viele Argumente in Tcl übergeben wurden (im Beispiel sind das 3). Anschließend werden die Argumente überprüft. Wird eine Übereinstimmung gefunden, so werden die entsprechenden Operationen durchgeführt. Im oberen Beispiel wird beim ersten Beispiel mit *add-member* verdeutlicht, dass der übergebene Wert einem anderen Agenten entsprechen soll. Deshalb wird das dritte Argument auf den dazugehörigen Agenten gecastet. Beim zweiten Beispiel wird genauso mit Integer-Werten verfahren. Anschließend wird dem Tcl-Interpreter mitgeteilt, dass die Operationen erfolgreich durchgeführt wurden.

Mit der recv-Methode können Nachrichten unterschieden werden. Wie zuvor dargestellt, werden unterschiedliche Nachrichtentypen verwendet. Kommt ein Paket an, so kann dann auf den definierten Header zugegriffen und nach Art der Nachricht unterschieden werden wie folgt:

```
hdr_ip* iph = hdr_ip::access(pkt);
if (iph->saddr() != here_.addr_) {
  hdr_dissemination* disshdr = hdr_dissemination::access(pkt);
  switch(disshdr->type) {
    case RETFAIL: {
      if(state == COORD_WAIT_FOR_RET_FAIL){
        // do something
        ...
      // message is unexpected
      }else wrongStateError(state, pkt);
    }
    break;
...
```

Zunächst wird auf den IP-Header eines Paketes zugegriffen. Mit den Informationen daraus wird sichergestellt, dass man keine Nachricht verarbeitet, die man selbst geschickt hat. Außerdem kann auf den Paket-Header des eigenen Protokolls zugegriffen werden. Durch Definition von

Nachrichtentypen wie oben beschrieben kann nun unterschieden und es können entsprechende Operationen durchgeführt werden. Beim obigen Beispiel wird eine Nachricht vom Typ RETFAIL nur verarbeitet, wenn sich der Koordinator im dazugehörigen Zustand COORD_WAIT_FOR_RET_FAIL befindet. Entsprechend kann man für jede Nachricht entsprechende Operationen definieren beim Eintreffen einer Nachricht.

Zur Steuerung der Dauer, die Knoten in einem bestimmten Zustand bleiben sollen bzw. die sie auf Nachrichten warten sollen, können Timer verwendet werden. Diese werden zunächst auch in der Header-Datei definiert:

```
class PongTimer : public TimerHandler {
  public:
    PongTimer(DisseminationAgent* a) : TimerHandler() {
      a_ = a;
    }
    virtual void expire(Event* e);
  protected:
   DisseminationAgent* a_;
};
```

Anschließend kann dann in der C++-Datei die dazugehörige Funktionalität implementiert werden. Zum einen muss die oben aufgeführte Methode expire() überschrieben werden. Danach kann dieser Timer gesetzt werden, und wenn er abgelaufen ist, wird die dazugehörige Methode aufgerufen. Als Beispiel eine verkürzte Timer-Implementierung:

```
/** Commit Timer of a client */
void CommitTimer::expire(Event* e) {
  ...
  // start the recovery process
  a_->recovery();
  ...
}
```

Verwendung des Protokolls in Tcl

Um mit dem entwickelten Protokoll simulieren zu können, müssen in Tcl entsprechende Agenten erzeugt werden. Daher soll hier kurz die Verwendung in Tcl aufgezeigt werden. Auf die eigentliche Struktur der Tcl-Skripte wird hier nicht eingegangen, der folgende Ausschnitt zeigt jedoch beispielhaft die Erzeugung mobiler Knoten in Tcl. An diese werden dann Instanzen des entwickelten Agenten gebunden:

```
# create nodes
for {set i 0} {$i < ($val(nn)) } {incr i} {
  $node_($i) start
  set slsAgent_($i) [new Agent/Sls]
  $ns_ attach-agent $node_($i) $slsAgent_($i)
  $slsAgent_($i) add-node $node_($i)
}
...
# generate Dissemination agents and attach to SLS agents
for {set i 29152} {$i <= 29155 } {incr i} {
  set p_($i) [new Agent/Dissemination]
  # set some parameter and timer
```

```
  $p_($i) set-taId 6547
  $p_($i) set-globalTechFailureRate 0.000001
  $p_($i) set-globalEnergyFailureRate 0.0002778
  ...
}
...
# attach the agents to nodes
$slsAgent_(15) add-agent $p_(29153)
$ns_ attach-agent $node_(15) $p_(29153)
...
# tell the node who the coordinator is
$p_(29153) set-coordinator $p_(29152)
# add transaction participants
$p_(29152) add-member $p_(29153)
...
# tell the coordinator to start and wait for messages
$ns_ at 9352.0 "$p_(29152) be-coordinator"
# tell the clients to start the protocol
$ns_ at 9352.0 "$p_(29153) send"
...
```

Im obigen Beispiel wird bereits die Struktur des Implementierungsmodells deutlich. Zunächst werden die eigentlichen mobilen Knoten erzeugt. Hier werden daran direkt sog. *slsAgenten* gebunden, auf die im Folgenden näher eingegangen wird. Anschließend werden sog. *DisseminationAgents* erzeugt, an die verschiedene Parameter übergeben werden. Diese werden dann jeweils an den zu ihnen gehörenden mobilen Knoten gebunden. Nachdem das geschehen ist, wird den Teilnehmern mitgeteilt, welcher Knoten der Koordinator ist. Zuletzt kann das Protokoll zu einem bestimmten Zeitpunkt gestartet werden. Dazu werden beim entsprechenden Zeitpunkt in der Simulation die relevanten Methoden aufgerufen.

A.2.2 Anpassung von NS2 an die Erweiterungen

A.2.2.1 Veränderungen für die neuen Agenten

Als erstes müssen die Pakettypen der entwickelten Agenten in der Konfiguration von NS2 hinzugefügt werden. Dazu muss die Datei ~/common/packet.h modifiziert werden, die u.a. die Paketprotokoll-IDs (z.B. PT_TCP, PT_TELNET usw.) enthält. Hier müssen in *enum packet_ t* die folgenden Deklarationen für den *DisseminationAgent* bzw. den *SimpleAgent* ergänzt werden:

```
PT_DISSEMINATION
PT_SIMPLE
```

Zusätzlich dazu muss in der *p_ info()* das Folgende eingetragen werden:

```
name_[PT_DISSEMINATION]="dissemination"
name_[PT_SIMPLE]="simple"
```

In einem nächsten Schritt müssen Änderungen in der Datei ~/tcl/lib/ns-default.tcl vorgenommen werden. Mit den folgenden Einträgen wird die Standard-Paketgröße für die neuen Agenten definiert:

```
Agent/Dissemination set packetSize_ 64
Agent/Sls set packetSize_ 64
Agent/Simple set packetSize_ 64
Agent/SimpleSls set packetSize_ 64
```

Nun muss noch die Datei ~/tcl/lib/ns-packet.tcl modifiziert werden, so dass NS2 die Header der neuen Agenten kennt. Dazu muss der folgende Code ergänzt werden:

```
foreach prot {
    # Common:
    Common Flags
    IP \# IP
    ...
    Dissemination \#Header for our Dissemination Packets
    Simple \#Header for the Simple Agent Packets
}
```

Abschließend muss noch das *Makefile* erweitert werden. Hier müssen die zu den neuen Agenten gehörenden Objekte eingetragen werden:

```
OBJ_CC = \
    ...
    dissemination/disseminationAgent.o \
    dissemination/slsAgent.o \
    dissemination/simpleAgent.o \
    dissemination/simpleSlsAgent.o \
\$(OBJ\_STL)
```

Nach den durchgeführten Änderungen sollten ein *make clean* und anschließend eine *make* gemacht werden. Danach sollte NS2 die neuen Agenten einbinden und korrekt funktionieren.

A.2.2.2 Ausfälle von Knoten

Zur Erzeugung von Ausfällen müssen alle Komponenten eines mobilen Knotens darüber informiert werden. Abschnitt 5 zeigte u.a. bereits, wie die Anwendungsagenten (*DisseminationAgent*, *SlsAgent*, *SimpleAgent* und *SimpleSlsAgent*) benachrichtigt werden. Für die Benachrichtigung der Knoten müssen auch die Dateien ~/mobile/mobileNode.h und ~/mobile/mobileNode.cc angepasst werden. Danach muss dies im *God*-Objekt ergänzt werden, damit ein ausgefallener Knoten z.B. bei der Nachbarbestimmung für das Fluten während des Ausfalls nicht berücksichtigt wird.

In der Datei ~/mobile/mobileNode.h müssen wie folgt zwei Methoden und eine Variable definiert werden. Diese können dann vom *SlsAgent* und vom *SimpleSlsAgent* aufgerufen werden:

```
...
// methods
void setFailure(bool fail);
bool getFailure();
...
// variable
bool failed;
```

Nach der Definition müssen beide Methoden in der ~/mobile/mobileNode.cc wie folgt implementiert werden:

```
void MobileNode::setFailure(bool fail){
    failed = fail;
}
bool MobileNode::getFailure(){
    return failed;
}
```

Nachdem der Knoten angepasst ist, muss nun das *God*-Objekt in ~/mobile/god.cc verändert werden. Für diese Arbeit betrifft dies nur die *IsNeighbor()*-Funktion, die beim adaptiven Fluten verwendet wird:

```
bool God::IsNeighbor(int i, int j) {
    assert(i<num_nodes && j<num_nodes);
    //printf("i=%d, j=%d\n", i,j);
    if (mb_node[i]->energy_model()->node_on() == false ||
        mb_node[j]->energy_model()->node_on() == false ||
        mb_node[i]->energy_model()->energy() <= 0.0 ||
        mb_node[j]->energy_model()->energy() <= 0.0 ||
       (mb_node[j]->getFailure() == true) ||
       (mb_node[i]->getFailure() == true)) {
            return false;
    }
    vector a(mb_node[i]->X(), mb_node[i]->Y(), mb_node[i]->Z());
    vector b(mb_node[j]->X(), mb_node[j]->Y(), mb_node[j]->Z());
    vector d = a - b;
    if (d.length() < RANGE)
           return true;
    else
           return false;
}
```

wobei *RANGE* der zu berücksichtigenden Übertragungsreichweite entspricht (vgl. dazu den folgenden Abschnitt A.2.2.3).

In einem letzten Schritt muss noch das Routing-Protokoll angepasst werden. Dies ist ebenfalls ein Objekt, das an jeden Knoten auf Port 255 gebunden ist. Zunächst wird dazu in ~/aodv/aodv.h folgende Variable eingefügt:

```
bool    routing;
```

Anschließend wird die dazugehörige Implementierung in ~/aodv/aodv.cc geändert. Das umfasst die Einführung einer neuen Schnittstelle zu Tcl:

```
...
if(argc == 2){
...
     if(strcasecmp(argv[1], "routing_on") == 0){
        routing = true;
        return TCL_OK;
    }
```

```
    if(strcasecmp(argv[1], "routing_off") == 0){
        routing = false;
        return TCL_OK;
    }
...
```

Mit der oben eingeführten Variable `routing` können die Methoden des Routing-Protokolls durch einfache Schleifen angepasst werden. Wenn `routing` auf `true` gesetzt ist, arbeitet das Protokoll. Ist es auf `false` gesetzt, so werden keine Nachrichten weitergeleitet. Das Stoppen bzw. erneute Starten des Routings kann dann in Tcl wie folgt gesteuert werden:

```
// stop the routing
$ns_ at time1 "[$node_(i) agent 255] routing_off"
// start the routing
$ns_ at time2 "[$node_(i) agent 255] routing_on"
```

wobei *time1* und *time2* jeweils zwei beliebigen Zeitpunkten während der Simulationen entsprechen mit *time2* > *time1* und *node_ (i)* einem beliebigen Knoten *i*. Mit *agent* wird der Agent auf Port 255 – also hier der Routing-Agent – angesprochen.

A.2.2.3 Einstellen der Übertragungsreichweite

Das Einstellen der Übertragungsreichweite der mobilen Knoten ist an zwei Stellen notwendig. Zunächst muss die Übertragungsreichweite in den Tcl-Skripten eingestellt werden. Dazu wird in jedem Tcl-Skript die Antenne der Knoten eingestellt wie folgt:

```
Antenna/OmniAntenna set Gt_ 1.5
Antenna/OmniAntenna set Gr_ 1.5
```

wobei *Gt_* der Übertragungsantennenhöhe (engl. *transmit antenna height*) und *Gr_* der Sende-Antennenhöhe (engl. *receive antenna height*) entspricht. Anschließend muss die kabellose Übertragungsreichweite der Antennen eingestellt werden. Im Folgenden wird diese auf 100 Meter eingestellt:

```
Phy/WirelessPhy set RXThresh_ 1.42681e-08; #100m
Phy/WirelessPhy set freq_ 9.14e+08
Phy/WirelessPhy set Pt_ 0.281838
```

Dabei entspricht *Pt_* der Übertragungsstärke (engl. *transmit power*), *freq_* der Frequenz und *RXThresh_* dem Empfangsschwellenwert (engl. *receiving threshold*). Die Werte können dazu mit dem zu NS2 gehörigen Tool ~/indep-utils/propagation/thresh.cc berechnet werden. Um die gewünschten Werte zu berechnen, ruft man das Programm mit dem zu verwendenden Ausbreitungsmodell und der benötigten Entfernung als Parameter auf (im einfachsten Fall). Als Ausbreitungsmodelle können FreeSpace, TwoRayGround oder Shadowing gewählt werden (vgl. dazu [3]). Zusätzlich können dort Einstellungen an der Antenne vorgenommen werden (die dann in den Tcl-Skripten eingetragen werden müssen), was jedoch für diese Arbeit nicht relevant ist.

Nun muss noch die Übertragungsreichweite bei NS2 selbst eingestellt werden im sog. *God-Objekt*, welches jeden Knoten und die Abstände zwischen allen Knoten kennt. Damit werden die jeweiligen Nachbarn eines Knotens bestimmt. Dies ist notwendig, wenn die Anzahl der Nachbarn

von NS2 bestimmt wird. Im Fall dieser Arbeit wird die Anzahl der Nachbarn beim adaptiven Pushen in dünnen MANETs benötigt. Die gewünschte Übertragungsreichweite, die von den Knoten genutzt wird, muss daher konsistent zu der sein, die bei NS2 eingestellt ist. Normalerweise beträgt diese 250 Meter, so dass dies verändert werden muss. Zur Einstellung der Reichweite müssen die Dateien ~/mobile/god.h und ~/indep-utils/cmu-scen-gen/setdest/calcdest.cc angepasst werden. Bei beiden Dateien muss die statische Variable *RANGE* auf 100 gesetzt werden, um eine Übertragungsreichweite von 100 Metern einzustellen.

A.3 Verwendung des Log-Datei-Parsers und des Log-Datei-Analyzers

In Kapitel 5 sind die grundlegenden Eigenschaften des entwickelten Tcl-Skripten-Generators, des Log-Datei-Parsers sowie des Analyse-Tools bereits vorgestellt worden. Im Folgenden soll kurz auf die Benutzung dieser Tools eingegangen werden. Zu Beginn wird auf den sog. *Database-Manager* eingegangen. Mit diesem kann die für die Simulationen benötigte Tabellenstruktur angelegt werden.

Wichtig bei der Verwendung der Tools ist, dass für ein Experiment immer dieselbe Properties-Datei verwendet wird. Ein Beispiel für eine Properties-Datei wird in Abschnitt **??** vorgestellt.

A.3.1 Database-Manager

Folgende Optionen stehen bei der Benutzung des Database-Managers zur Verfügung:

```
// Benutzer (immer notwendig)
-u User
// Passwort (immer notwendig)
-p Password
// Anlegen der Tabellen
-c
// Löschen existierender Tabellen
-d
// Ändern einer Tabelle durch Hinzufügen einer neuen Spalte
-a [table name] add [column] [data type]
// Ändern einer Tabelle durch Modifizierung des Datentyps einer Spalte
-a [table name] modify [column] [datatype]
```

Mit Hilfe dieser Optionen können beispielsweise die folgenden Operationen durchgeführt werden:

```
java DatabaseManager -u Tester -p testpassword -a testtable add testcolumn text
\
```

Mit dem obigen Aufruf wird die Spalte *testcolumn* vom Typ *text* zur Tabelle *testtable* hinzugefügt.

A.3.2 Log-Datei-Parser und Analyzer

Für die Benutzung des Parsers sind die folgenden Optionen möglich:

```
// Starten des Parsens
java -jar ParserManger.jar 0 <properties file>
// Starten der Analyse
java -jar ParserManger.jar 1 <properties file>
```

Dabei ist zu beachten, dass die verwendete Properties-Datei korrekte Informationen enthält. Dies bezieht sich u.a. auf die Experiment-Informationen, die verwendeten Agenten sowie die angegebenen Pfade. Anderenfalls ist das reibungslose Parsen der Log-Dateien nicht möglich. Ebenso muss für die Analyse dieselbe Properties-Datei verwendet werden. Wird dies nicht gemacht, können keine Ergebnisse zu den vorher geparsten Log-Dateien ermittelt werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] The mathworks. accelerating the pace of engeneering and science. `www.mathworks.com`.

[2] Network animator nam. http://www.isi.edu/nsnam/nam/.

[3] The network simulator ns-2. user information.

[4] Otcl - object tcl extensions (`http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/`).

[5] Tclcl `http://otcl-tclcl.sourceforge.net/tclcl/`.

[6] I. Akyildiz, J. McNair, J. Ho, H. Uzunalioglu, and W. Wang. Mobility management in next-generation wireless systems. *Mobility Management in Next-Generation Wireless Systems (Proceedings of the IEEE)*, 87:1347–1384, 1999.

[7] J. D. Andrews and T. R. Moss. *Reliability and Risk Assessment.* Professional Engineering Publishung, London and Bury St. Edmunds, UK, 2002.

[8] F. Bai, N. Sadagopan, and A. Helmy. Important: A framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 2:825– 835, 2003.

[9] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Company, 1987.

[10] Christian Bettstetter, Hannes Hartenstein, and Xavier Pérez-Costa. Stochastic properties of the random waypoint mobility model. *Wireless Networks*, 10(5):555–567, 2004.

[11] Urban Bilstrup, Bertil Svensson, and Per-Arne Wiberg. Knowledge horizon dynamic limitations in a wireless ad hoc network. In *Swedish National Computer Networking Workshop*, Stockholm, Schweden, 2003.

[12] Sven Bittner, Wolf-Ulrich Raffel, and Manuel Scholz. The area graph-based mobility model and its impact on data dissemination. *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, 2005.

[13] Joos-Hendrik Böse, Frank Bregulla, Katharina Hahn, and Manuel Scholz. Adaptive data dissemination in mobile ad-hoc networks. *A.B. Cremers, R. Manthey, P. Martini, V. Steinhage (Hrsg.): INFORMATIK 2005 - Informatik LIVE!*, September 2005, Passau, Deutschland.

[14] T. Camp, L. Wilcox, and J. Boleng. Location information services in mobile ad hoc networks. *EEE International Conference on Communications, 2002. ICC 2002.*, 5:3318 – 3324, 2002.

[15] Julien Cartigny, David Simplot, and Ivan Stojmenovic. Localized minimum-energy broadcasting in ad-hoc networks. 2002.

[16] Ing-Ray Chen and Naresh Verma. Simulation study of a class of autonomous host-centric mobility prediction algorithms for wireless cellular and ad hoc networks. *ANSS '03: Proceedings of the 36th annual symposium on Simulation*, 2003.

[17] Sunghyun Choi and Kang G. Shin. Adaptive bandwidth reservation and admission control in qos-sensitive cellular networks. *IEEE Trans. Parallel Distributed Systems*, 13(9):882–897, 2002.

[18] Reinhard Diestel. *Graphentheorie.* Springer-Verlag, Berlin and Heidelberg, 2006.

[19] Bhavyesh Divecha, Ajith Abraham, Crina Grosnan, and Sugata Sanyal. Impact of node mobility on manet routing protocols. *Journal of Digital Information Management*, 5(1):19–24, 2007.

[20] Ahmed K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers, San Maeo, California, USA, 1995.

[21] Database Freie Universität Berlin, Computer Science Department and Information Systems Group. Cooperation,communication data (cocoda), 2005-2006.

[22] Ramin Heckmat. *Ad-hoc Networks: Fundamental Properties and Network Topologies.* Springer, P.O. Box 17, 3300 AA Dordrecht, The Netherlands, 2006.

[23] Xiaoyan Hong, Mario Gerla, Guangyu Pei, and Ching-Chuan Chiang. A group mobility model for ad hoc wireless networks. *MSWiM '99: Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, 1999.

[24] Hagen Höpfner, Can Trker, and Brigitta König-Ries. *Mobile Datenbanken und Informationssysteme. Konzepte und Techniken.* dpunkt.verlag, Heidelberg, 2005.

[25] Arnljot Hoyland and Marvin Rausand. *System Reliability Theory. Models and Statistical Methods.* John Wiley & Sons, Inc., United States of America, 1994.

[26] IETF. Mobile ad-hoc networks (manet). http://www.ietf.org/html.charters/manet-charter.html.

[27] WPI Worcester Polytechnic Institute. Ns by example.

[28] David B. Johnson, David A. Maltz, and Josh Broch. Dsr: the dynamic source routing protocol for multihop wireless ad hoc networks. 2001.

[29] Alfons Kemper and André Eickler. *Datenbanksysteme. Eine Einführung.* Oldenburg Wissenschaftsverlag GmbH, 2004.

[30] Rolf Klein. *Algorithmische Geometrie.* Springer-Verlag, Berlin, Heidelberg, 2005.

[31] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2000.

[32] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. Manet simulation studies: The incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4):50–61, October 2005.

[33] Byung-Jae Kwak, Nah-Oak Song, and L. E. Miller. A mobility measure for mobile ad hoc networks. *IEEE Communications Letters*, 7:379 – 381, 2003.

[34] Sedong Kwon, Hyunmin Park, and Kangsun Lee. Systems modeling and simulation: Theory and applications. *Systems Modeling and Simulation: Theory and Applications*, 3398/2005:419–428, 2005.

[35] P. Lassila, E. Hyytia, and H. Koskinen. Connectivity properties of random waypoint mobility model for ad hoc networks, 2004.

[36] Natalia Marmasse and Chris Schmandt. A user-centered location model. *Personal Ubiquitous Comput.*, 6(5-6):318–321, 2002.

[37] Bela Mutschler and Gnther Specht. *Mobile Datenbanksysteme.* Springer-Verlag Berlin, Heidelberg, 2004.

[38] Milton Ohring. *Reliability and Failure of Electronic Materials and Devices.* Academic Press, 1998.

[39] Ashish Pandey, Md. Nasir Ahmed, Nilesh Kumar, and P. Gupta. Hybrid routing protocol for large scale mobile ad hoc networks with mobile backbones. *IEEE International Conference on High Performance Computing, HIPC*, 2006.

[40] Lothar Papula. *Mathematik für Ingenieure und Naturwissenschaftler.* Vieweg Verlag, Braunschweig/Wiesbaden, 1994.

[41] Pubudu N. Pathirana, Andrey V. Savkin, and Sanjay Jha. Mobility modelling and trajectory prediction for cellular networks with mobile base stations. *MobiHoc '03: Proceedings of the 4th ACM International Symposium on Mobile ad hoc Networking & Computing*, 2003.

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

[43] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, 1994.

[44] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. *Second IEEE Workshop on Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99*, 1999.

[45] Fabio De Rosa, Alessio Malizia, and Massimo Mecella. Disconnection prediction in mobile ad hoc networks for supporting cooperative work. *IEEE Pervasive Computing*, 4(3):62–70, 2005.

[46] Sheldon M. Ross. *Statistik für Ingenieure und Naturwissenschaftler.* Elsevier GmbH, Mnchen, 2006.

[47] Jochen Schiller. *Mobilkommunikation. Techniken für das allgegenwärtige Internet.* Addison-Wesley, München, Germany, 2000.

[48] W. Su, S. Lee, and M. Gerla. Mobility prediction in wireless networks. *MILCOM 2000. 21st Century Military Communications Conference Proceedings*, 1:491–495, 2000.

[49] Danny Tschirner. Experimentelle und analytische untersuchung des area graph-based mobility models. Master's thesis, Freie Universität Berlin, 2006.

[50] Gottfried Vossen and Margret Gross-Hardt. *Grundlagen der Transaktionsverarbeitung.* Addison-Wesley (Deutschland) GmbH, Bonn, Paris, Reading, Menlo Park, New York, Don Mills, Wokingham, Amsterdam, Milan, Sydney, Tokyo, Singapore, Madrid, San Juan, Seoul, Mexico City, Taipei, 1993.

[51] Gerhard Weikum. *Transaktionen in Datenbanksystemen. Fehlertolerante Steuerung paralleler Abläufe.* Addison-Wesley Verlag (Deutschland) GmbH, Bonn, Reading, Menlo Park, New York, Don Mills, Wokingham, Amsterdam, Sydney, Singapore, Tokyo, Madrid, San Juan, 1988.

[52] Jie Wu and Fei Dai. Efficient broadcasting with guaranteed coverage in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 4(3):259–270, 2005.

[53] Hui Xu, Manwoo Jeon, Shu Lei, Jinsung Cho, and Sungyoung Lee. Localized broadcast oriented protocols with mobility prediction for mobile ad hoc networks. *Next Generation Teletraffic and Wired/Wireless Advanced Networking*, 2006.

[54] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 2:1312– 1321, 2003.