Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Benjamin Kuska

Integration eines verteilten Datenspeichers in einem drahtlosen Sensornetzwerk mittels Network Coding

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Benjamin Kuska

Integration eines verteilten Datenspeichers in einem drahtlosen Sensornetzwerk mittels Network Coding

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Dirk Westhoff
Zweitgutachter: Prof. Dr. Thomas Schmidt

Abgegeben am 14. August 2011

Benjamin Kuska

Thema der Bachelorarbeit

Integration eines verteilten Datenspeichers in einem drahtlosen Sensornetzwerk mittels Network Coding

Stichworte

Sensornetzwerk, Persistenz, Verteilter Speicher, Network Coding, TinyOS

Kurzzusammenfassung

Im Verlauf der Bachelorarbeit wird ein verteilter Datenspeicher in drahtlosen Sensornetzwerken erstellt. Zur Aggregation der Daten wird das Codierungsverfahren Network Coding eingesetzt. Es wird beschrieben, wie drahtlose Sensornetzwerke aufgebaut sind und wie Network Coding funktioniert. Hierauf folgen Entwicklungen von Strategien, die die Verteilung der Daten über das Netzwerk effizient und ausfallsicher konstruieren sowie Anfragen von Daten effizient behandeln. Die Erkenntnisgewinnung erfolgt durch eine Simulation des Netzwerks mit dem Fokus auf die Datenhaltung. Das System bietet Varianten zur Rekonstruktion von Daten nach einem Desaster, bei dem ein beträchtlicher Teil der Sensorknoten ausgefallen ist. Aus den Erkenntnissen der Theorie erfolgt eine Implementierung in TinyOS. Als Referenzprojekt dient das EU geförderte Projekt WSAN4CIP.

Title of the paper

Integration of a distributed data storage in a wireless sensor network using network coding

Keywords

Sensornetwork, Persistence, Distributed Storage, Network Coding, TinyOS

Abstract

In the process of this bachelor thesis, a distributed data storage for wireless sensor networks is created. For the aggregation of the data, the coding procedure network coding is used. Description of how wireless sensor networks are built and how network coding works is given. In this thesis, strategies are worked out, to distribute data across the network efficiently and fail-safe, and to request data in an efficient way. The acquisition of knowledge takes place through a simulation of the network. The system provides options to reconstruct data after a disaster, in which the majority of sensor nodes have failed. An implementation of the theory is made in TinyOS. The reference project is the EU sponsored WSAN4CIP.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Kleinste Bauteile in einem Verbund, betrieben durch eine Batterie, überwachen ein Areal, speichern und versenden die aufgezeichneten Daten an weitere Knoten im Netzwerk. Sensorknoten eines drahtlosen Sensornetzwerks sind heute technisch zu vielem fähig. In den verschiedensten Anwendungsgebieten werden die Sensorknoten platziert, um neue Erkenntnisse über das Verhalten von Tieren, Materialien, der Umwelt und vielem anderen aufzuzeigen. Die technischen Anforderungen, die dabei an die Sensorknoten gestellt werden, sind enorm. Durch die kleine Bauweise gibt es kaum Platz um umfangreiche Ausstattung zu liefern. Die Grundkonfiguration eines Sensorknotens besteht aus einem Mikroprozessor, einigen Sensoren, einer Speichereinheit und einer Funkeinheit. Durch den Betrieb über Batterien ist zudem die Lebenszeit stark begrenzt und abhängig von der Ressourcennutzung der Software, die auf dem Sensorknoten läuft. Ist die Energie verbraucht, fällt der Sensorknoten aus und ist im Netzwerk nicht mehr zu gebrauchen. Er muss von Menschenhand ausgetauscht werden.

Ein weiterer wichtiger Punkt ist der Speicherplatz. In PC-Systemen wächst der Speicherplatz stetig an. Auf den Sensorknoten ist der Speicher jedoch stark begrenzt und nicht unbedingt persistent. Fällt ein Sensorknoten aus, ist häufig auch der aktuelle Arbeitsspeicher verloren.

Die Übertragungsraten bei fest verkabelten Netzwerken steigen ebenfalls stetig an. In Sensornetzwerken hingegen sind die Übertragungsraten gering und durch die Ressourcen streng begrenzt [5, 38]. Speziell hier müssen Verfahren eingesetzt werden, die den Fluss der Daten optimieren. Um das Übertragungsmedium zu schonen, darf es keine Rückfragen bei fehlerhaften oder verlorenen Paketen geben. Der Einsatz von Verfahren zur Vorwärtsfehlerkorrektur kann die Rückfragen ausschließen. Der Sender codiert hierzu die Daten, sodass beim Verlust eines Pakets nicht ein großes, sondern mehrere kleine Löcher entstehen [55, S. 188f] (siehe auch Abbildung 1.1). Network Coding deckt genau diesen Bereich ab. Hierbei werden Pakete zusammengefasst, um so den Fluss der Daten zu optimieren [32].

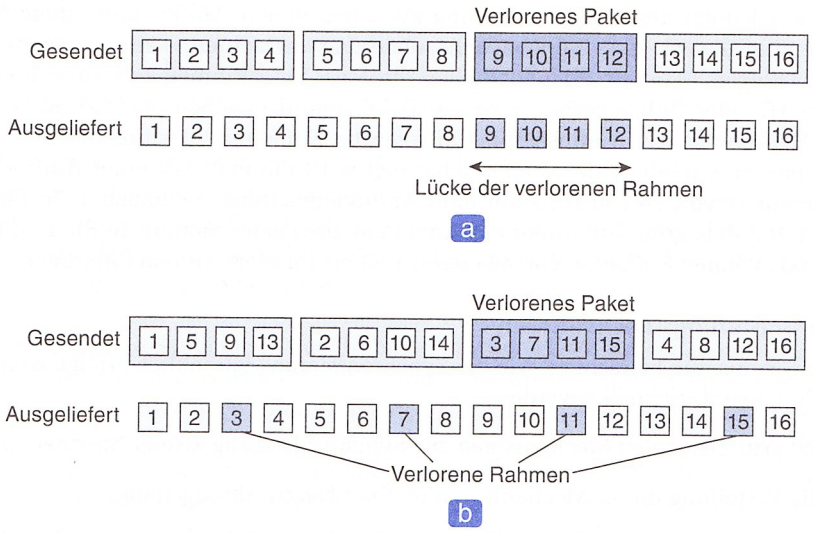

Abbildung 1.1: Auswirkung eines Paketverlusts; Quelle: [55, S. 189]

Im Folgenden werden drei Ansätze vorgestellt, mit denen ein verteilter Datenspeicher in einem drahtlosen Sensornetzwerk realisiert werden kann. Die Basis der Speicherung pro Knoten ist eine Zufallsverteilung mit definierten Randbedingungen. Jeder Knoten aggregiert die Daten zusätzlich. Der Zufall bestimmt ebenfalls das Aggregationsverhalten der Sensorknoten. Durch die Aggregation minimiert sich die Netzwerkbelastung beim Anfragen der Daten durch einen Administrator. Sämtliche Erkenntnisgewinnungen basieren auf Simulationsergebnissen.

1.1 Motivation

Das WSAN4CIP[1] - *Wireless Sensor and Actuator Networks for the Protection of Critical Infrastructures* - hat sich zum Ziel gesetzt, durch neue Technologien im Bereich Wireless Sensor and Actuator Networks (WSAN), den Schutz der kritischen Infrastrukturen (engl.: critical infrastructure [CI]) zu gewährleisten. Zur Erfüllung dieses Ziels werden alle Bereiche der WSAN betrachtet, unter anderem der Bereich der verteilten Datenspeicher. Zum Einsatz kommen unbewachte drahtlose Sensornetzwerke. Ein Administrator, der die Daten aus dem Sensornetzwerk ausliest, steht somit nur zeitlich eingeschränkt zur Verfügung. Sämtliche Daten müssen daher dauerhaft konsistent gespeichert werden. Im Bereich der verteilten Datenspeicher hat das WSAN4CIP sieben Grundpfeiler aufgestellt, die bei der Ausarbeitung neuer Technologien zu beachten sind [24]. Diese sind nachfolgend von dem Autor dieser Bachelorarbeit frei aus dem Englischen übersetzt.

[1] `http://www.wsan4cip.eu/`, Zugriffsdatum: 03.05.2011

Persistenz Gespeicherte Daten müssen auch nach einer Fehlfunktion, einem Ausfall oder einer Zerstörung von Knoten dauerhaft gespeichert sein.

Verteilung über die drahtlose Verbindung Die Übertragung über eine drahtlose Verbindung ist teuer. Daher kann die Nutzung dieses Mediums bei verteilten Datenspeichern in drahtlosen Sensornetzwerken extrem energieverschwendend sein.

Einschränkungen beim Speichern Es muss für ein gutes Verhältnis zwischen der Präzision von gespeicherten Daten über die Zeit und dem aufgewendeten Speicher gesorgt werden.

Sicherheit Durch die drahtlose Übertragung und die potenziellen physikalischen Schwächen der Knoten müssen die übertragenden Daten authentifiziert sein und verheimlicht werden.

Desaster Radius Der maximale Wirkungsradius eines Katastrophenareals, in dem Knoten ausfallen, kann in weiteren Betrachtungen geschätzt werden und sollte bei der redundanten Datenhaltung beachtet werden.

Leistungsfähigkeit von Abfragen Wird ein System häufig angefragt[2], so muss die Effektivität der Anfragen optimal sein. Das heißt, dass die Komplexität der Anfrage so gering und kostengünstig wie möglich ausgelegt sein muss. Dies gilt ebenso für die Antworten der Sensorknoten.

Netzwerktopologie Die Netzwerktopologie kann in manchen Fällen von erheblicher Bedeutung sein. Hierunter fallen insbesondere WSNs, die eine bestimmte Infrastruktur überwachen sollen oder eine bestimmte Form (z.B. quadratisch) aufweisen.

1.2 Zielsetzung

Aufbauend auf den Grundpfeilern des WSAN4CIP Anforderungskatalogs für verteilte Datenspeicher soll ein verteilter Datenspeicher entworfen werden, der auch beim Ausfall eines oder mehrerer Sensorknoten als persistent anzusehen ist *(Persistenz)*. Die redundante Haltung der Daten auf mehreren Sensorknoten kann durch die einfache Verteilung der Daten nur durch einen Rückkanal, über den das erfolgreiche Speichern des aktuellen Datenpakets bestätigt wird, gesichert werden. Dies führt zu einem erheblichen Mehraufwand in der Kommunikation und zu Mehrkosten auf dem Übertragungsmedium *(Verteilung über die drahtlose Verbindung)*. Es wird eine Speichermöglichkeit benötigt, die ohne diese Nachteile arbeiten kann. Um sämtlichen Mehraufwand auf dem Übertragungsmedium auszuschließen, wird im

[2] Die Daten werden beim „Anfragen“ von einem Administrator aus dem Netz ausgelesen. Hierdurch werden Datenmengen auf dem Übertragungskanal produziert.

Folgenden ein Verfahren erforscht, bei dem der Sender seine Daten mittels Broadcast an alle Nachbarn versenden kann. Hierdurch wird die Betrachtung der *Netzwerktopologie* überflüssig, da beim Broadcast keine Informationen über die Nachbarknoten im Netzwerk benötigt werden. Das Speicherverhalten der einzelnen Knoten wird möglichst zufällig ausfallen. Durch die Zufallsverteilung soll eine hohe Redundanz der Daten bei gleichzeitiger Minimierung des Speicheraufwandes erreicht werden. Zudem erfolgt eine Aggregation der gespeicherten Daten pro Knoten. Das Aggregationsverhalten unterliegt ebenfalls dem Zufall. Durch die Aggregation wird sich der Verkehrsfluss beim späteren Anfragen der Daten, zum Beispiel durch einen Administrator, stark verringern *(Leistungsfähigkeit von Abfragen)*.

Ziel dieser Arbeit ist es, einen verteilten Datenspeicher auf Basis der Replikation von Daten aufzubauen. Die Daten sollen dabei stückweise auf verschiedenen Knoten im Netzwerk gespeichert und aggregiert werden. Die Teilung der Gesamtdatenmenge erfolgt hierbei automatisch durch die zeitlich versetzte Aufnahme und Verteilung der Sensordaten. Dabei stellt die Menge aller aufgezeichneten Sensordaten über einen begrenzten Zeitraum die Gesamtdatenmenge dar.

1.3 Abgrenzungen

Diese Arbeit hat das Ziel, grundlegende Erkenntnisse im Bereich der verteilten Datenspeicher in Sensornetzwerken zu gewinnen. Hierbei können nicht alle Eckpunkte betrachtet werden. Im Wesentlichen konzentriert sich die Arbeit auf die persistente Speicherung, auch bei großen Knotenausfällen, und die effektive Abfrage der Daten nach diesem Desaster. Der Focus liegt dabei auf der Ausfallsicherheit, wobei die Umstände des Desasters keine Rolle spielen. Die drei Grundpfeiler des WSAN4CIP *Sicherheit*, *Desaster Radius* und *Einschränkungen beim Speichern* haben keine Beachtung gefunden, da sie nicht Kern der Problemstellung sind. Der Punkt *Netzwerktopologie* kann durch die Wahl des Broadcast als Sendemethode ebenfalls vernachlässigt werden.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in vier Kapitel. Zunächst werden im Kapitel 2 einige einführende Grundlagen behandelt. Es findet eine ausführliche Einleitung in WSNs und Network Coding sowie eine Erläuterung des Begriffs „verteilter Datenspeicher“ statt. Vergleiche zu Peer-to-Peer Netzwerken zeigen Verwandschaften dieser Netzwerktypen auf. Zudem wird es eine kurze Einführung in Konfidenzintervalle geben. Kapitel 3 gibt eine Einführung in die vom Autor entwickelte Simulationsapplikation, mit der die Experimente durchgeführt wurden. In Kapitel 4 folgt die Ausarbeitung des verteilten Datenspeichers in drahtlosen Sensornetzwerken mittels Network Coding und die Darstellung sowie die Erläuterung der Forschungser-

gebnisse. Das fünfte Kapitel befasst sich mit der Implementierung des Ansatzes aus Kapitel 4 in TinyOS und den Restriktionen, die hiermit verbunden sind. Die Arbeit endet mit einem Fazit und einem Ausblick auf weiterführende Arbeiten.

Kapitel 2

Grundlagen

2.1 Drahtlose Sensornetzwerke

Drahtlose Sensornetzwerke als Verbund von extrem vielen Sensorknoten zu einem Netzwerk stellen die Basis für diese Arbeit dar. In diesem Kapitel soll der Begriff „drahtlose Sensornetzwerke“ erläutert und der Grund für die Nutzung von drahtlosen Sensornetzwerken aufgezeigt werden. Nach dem prinzipiellen Aufbau solcher Netzwerke folgen allgemeine Problemstellungen und Einschränkungen bei der Nutzung drahtloser Sensornetzwerke.

2.1.1 Motivation zur Verwendung von drahtlosen Sensornetzwerken

Sensornetzwerke sind flexibel einsetzbar. Sie können Bereiche überwachen, an denen Menschen keinen Zugang, zum Beispiel durch räumliche Gegebenheiten oder aufgrund zu starker Belastung durch Kontamination, haben. Somit finden Sensornetzwerke in nahezu jedem Bereich Anwendung. Beispielsweise werden sie in der Landwirtschaft zur Begutachtung der Beschaffenheit des Erdbodens, in der Stromindustrie zur Kontrolle der Stromleitungen, in der Medizin, Müllverwertung, Forschung, Umwelt und vielen weiteren Bereichen eingesetzt. Dabei ist nebensächlich in welchem Anwendungsgebiet der Einsatz erfolgt. Die Sensorknoten werden immer dafür eingesetzt, Daten aufzuzeichnen und zu speichern. [5]

2.1.2 Aufbau

Sensorknoten, auch motes[3] genannt, werden in Sensornetzwerken zu einer Vielzahl (bis zu einigen Tausend Knoten) zusammengeschlossen. Skalierbarkeit ist eine der Hauptbedingungen für den erfolgreichen Einsatz solcher Sensornetzwerke. Da Sensornetzwerke diese enorme Größe annehmen können, müssen sie in der Regel selbstorganisierend sein. Häufig

[3] engl. mote - Stäubchen, Staubkorn oder auch Splitter [48, S. 850]

fallen Sensorknoten aufgrund erschöpfter Batterien oder schädlicher Umwelteinflüsse aus. Die ausgefallenen Sensorknoten müssen zwar von Menschenhand ausgetauscht werden, die Integration verläuft jedoch in der Regel beim Einschalten des Sensorknotens automatisiert. [38, 65]

2.1.3 Probleme und Einschränkungen

Durch die kleine Bauweise und den drahtlosen Einsatz der Sensorknoten drängen sich einige Probleme und Einschränkungen auf. So ist die Lebenszeit stark abhängig von der Batterieversorgung. Sind die Batterien aufgebraucht, kann der Sensorknoten nicht mehr arbeiten und wird somit unbrauchbar. Es muss also darauf geachtet werden, dass sämtliche Software effizient und ressourcenschonend arbeitet. Ebenso sind der Speicher und die Recheneinheit stark limitiert. Die Speichereinheit umfasst nicht mehr als bis zu 48 Kilobytes. Der Mikroprozessor taktet nur mit bis zu 8 MHz. Somit muss bei der Implementierung neben dem für den Strombedarf bedachten ressourcenschonenden Arbeiten auch auf eine effektive Speichernutzung und auf möglichst wenig komplexe Berechnungen Acht gegeben werden. [11, 19, 38, 41]

Da die Sensorknoten mittels der integrierten Funkeinheit drahtlos miteinander kommunizieren können, kann es auf dem Übertragungsweg zu Paketverlusten kommen. Die Rate des Verlusts kann je nach äußeren Einflüssen variieren. Bei der Entwicklung von Verfahren, die die Kommunikation der Sensorknoten nutzen, sollte der mögliche Paketverlust somit beachtet werden. [5]

2.1.4 Verwendete Hardware

Sämtliche Verfahren und Erkenntnisse dieser Arbeit basieren auf Sensorknoten der Bauweise *Telos-B*. Es wurden baugleiche Module von zwei verschiedenen Herstellern benutzt. Die nachfolgenden Spezifikationen beziehen sich auf die Module *tmote sky* der Firma *moteiv*. Die anderen Module sind von der Firma *Crossbow* und sind mit dieser Spezifikation deckungsgleich. [11]

Die Sensorknoten *tmote sky* haben eine Funkeinheit (Chipcon CC2420) mit einem 2.4 GHz ISM Trägersignal. Sie erreichen eine Übertragungsgeschwindigkeit von 250 kbps. Die Funkeinheit ist kompatibel mit dem IEEE 802.15.4 Standard. Die Sensorknoten erreichen eine Übertragungsreichweite von bis zu 100 Metern. Der Mikroprozessor MSP430 F1611 taktet mit 8 MHz und hat 10 KB RAM, 48 KB Flashspeicher und 128 B Informationsspeicher integriert. Auf den Modulen sind Sensoren zur Messung der Luftfeuchtigkeit, der Temperatur und des Lichts installiert. Zur Programmierung und zur Sammlung der Daten aus einem Netzwerk bieten die Module einen USB-Port. Zur Stromversorgung ist eine Halterung für

zwei AA-Batterien angebracht. Mit dieser Leistung können die Sensorknoten im Stand-by-Betrieb[4] theoretisch bis zu 17 Jahren laufen. [19, 41]

2.2 Verteilter Datenspeicher

Die Speicherung von Daten in einem Netzwerk kann auf verschiedene Arten vorgenommen werden. Der einfachste Ansatz ist ein zentraler Server, auf dem die Daten gespeichert werden, so dass jeder Client hierauf über das Netzwerk Zugriff hat. Diese Client-Server-Architektur ist für ein Sensornetzwerk nicht zu gebrauchen, da in der Regel kein zentraler Server verfügbar ist.

Ein anderer Ansatz ist die Verteilung der Daten auf mehrere Server, sogenannte Cluster. Hierbei wird generell zwischen zwei Arten unterschieden. Die erste Variante legt vollständige Dateien auf jeweils unterschiedlichen Servern ab. Somit wird die gesamte Datenmenge durch die Verteilung verschiedener Dateien auf verschiedene Cluster aufgeteilt. Die zweite Variante teilt jede Datei in viele Einzelteile auf und legt diese auf verschiedenen Servern ab. Fallen hier einzelne Server aus, sind die Daten einzelner Dateien nicht vollständig verloren. Beide Varianten des zweiten Ansatzes können, um die Ausfallsicherheit zu erhöhen, durch Verfahren zur Replikation der Daten erweitert werden. [55, Kapitel 11]

2.2.1 Anforderungen an verteilte Datenspeicher

Jeder verteilte Datenspeicher muss garantieren, dass die Daten permanent und korrekt gespeichert werden. Korrekt bedeutet in diesem Zusammenhang, dass die Speicherung die Daten nicht verändert. Andernfalls wären die Daten später unbrauchbar. Außerdem müssen die Daten sicher verteilt und gespeichert werden, so dass sie vor Verlust und unbefugtem Zugriff geschützt sind. Die Speicherung der Daten muss garantieren, dass, egal wie die Daten gespeichert werden, sie wieder vollständig und ohne Abweichung aus dem System extrahierbar sind. [55, Kapitel 11.6 und 11.7]

2.2.2 Ausfallsicherheit

Gegenüber einem zentralisierten Speicher bietet ein verteilter Datenspeicher den großen Vorteil von erhöhter Ausfallsicherheit. So kommt es zwar in sehr großen Netzwerken stetig zu Ausfällen einzelner Knoten, jedoch wird durch die Replikation der Daten in einem verteilten Datenspeicher garantiert, dass auch beim Ausfall einer bestimmten Anzahl an Knoten die Daten vollständig oder zumindest zu einem bestimmten Anteil erhalten bleiben. [55, Kapitel 11.7]

[4] Im Stand-by-Betrieb wird ein Sensorknoten weder senden noch empfangen oder etwas berechnen.

2.2.3 Verteilter Datenspeicher in P2P-Netzwerken

Peer-to-Peer Netzwerke (kurz P2P-Netzwerke) bekommen einen immer höheren Stellenwert im Internet. Durch die allgemein ansteigenden Bandbreiten fließen immer größere Datenmengen durch die Leitungen. Zentrale Server laufen Gefahr, zum Engpass zu werden, da sie die Datenmenge nicht ausreichend schnell für eine Vielzahl an Clients zur Verfügung stellen können. P2P-Netzwerke können diese Probleme beseitigen und sie können dazu genutzt werden, große Datenmengen zu speichern. [51, S. 10f]

2.2.3.1 Aufbau eines P2P-Netzwerks

In P2P-Netzwerken gibt es keine klassische Trennung zwischen Client und Server. Jeder Peer, der Daten aus dem Netzwerk bezieht, bietet diese automatisch wieder an. Somit ist jeder Peer sowohl Client als auch Server. Ein P2P-Netzwerk ist dementsprechend ein dezentrales Konstrukt ohne zentralen Server. Aus diesem Grund muss das P2P-Netzwerk selbstorganisierend sein. Dies geschieht durch Protokolle, über die andere Peers gefunden werden können. Es entsteht ein Overlay Netzwerk, das auf der vorhandenen TCP/IP-Struktur des Internets aufsetzt. [43]

Jeder Peer arbeitet zudem völlig autonom. Kein Peer des Netzwerks kann das Verhalten eines anderen Peers beeinflussen oder steuern. Jeder Peer kann außerdem zu jedem beliebigen Zeitpunkt dem Netzwerk beitreten oder es verlassen. Die Funktionsweise des P2P-Netzwerks ist davon nicht betroffen. [34, 43]

Sämtliche Daten in einem P2P-Netzwerk werden redundant auf allen beteiligten Peers gehalten. Die Daten bleiben somit auch nach einem Ausfall einer bestimmten, festzulegenden Anzahl an Peers reproduzierbar. [43]

2.2.3.2 Sicherheit in verteilten P2P-Datenspeichern

Durch die Dezentralisierung ergeben sich einige Einschnitte in der Sicherheit in verteilten P2P-Datenspeichern. Die Peers im Netzwerk sind zumeist unbekannt und können nicht authentifiziert werden. Zudem fließt der Datenverkehr über die unsichere TCP/IP-Struktur des Internets und ist somit diversen Angriffsmöglichkeiten ausgeliefert. Eine Verschlüsselung der Verbindung und der Daten ist daher unerlässlich. Weiterhin können zur Authentifizierung der Peers Kontaktlisten eingesetzt werden, die vertrauenswürdige Peers ausweisen. Zertifikate zur Identifizierung der Peers geben weitere Sicherheit. Auf diese Weise könnten P2P-Datenspeicher in verteilten Firmennetzwerken aufgebaut werden. Die zusätzliche Authentifizierung der Daten-Blöcke kann über kryptografische Hashes erfolgen. [34]

2.2.3.3 Mögliche Integrationen von verteilten Datenspeichern in P2P-Netzwerken

Eine mögliche Integration eines verteilten Datenspeichers in P2P-Netzwerken beschreiben Mark Lillibridge et al. in [34]. Sie bauen eine kooperative Backup-Lösung über das Internet auf, die im Wesentlichen durch die Partnerschaft zwischen Peers funktioniert. Die Partnerschaft ist symmetrisch. Wenn also Peer A Daten bei Peer B speichert, so kann Peer B ebenfalls Daten bei Peer A speichern. Jeder Peer baut dabei eine virtuelle Festplatte auf, deren physikalischer Speicher auf alle Partner-Peers verteilt ist. Zum Schutz der Daten vor Verlust gehen die Autoren auf die geografische Lage der Peers ein und ziehen dabei Vergleiche zu herkömmlichen off-site Backup-Verfahren wie etwa der Sicherung auf Magnetbändern. Einen zusätzlichen Schutz vor Ausfällen stellen die Autoren durch den Einsatz von Reed-Solomon erasure-correcting codes her. Dabei werden m redundante Datenblöcke aus den k ursprünglichen Datenblöcken erzeugt und im Netzwerk verteilt. Hierbei gilt: $m \geq k$. Aus jeder beliebigen Menge von k der m Datenblöcke können die ursprünglichen Datenblöcke wiederhergestellt werden. Um einen entsprechenden Nutzen aus der verteilten Datenhaltung zu ziehen, müssen sich alle Partner zu einer garantierten Uptime verpflichten.

Einen anderen Ansatz haben Cox et al. mit ihrem *Pastiche* genannten System in [10] beschrieben. Ein wesentlicher Unterschied zum Ansatz von Lillibridge et al. ist die Datenhaltung. Dabei versucht *Pastiche* identische Daten nicht mehrfach in das System einzuspielen, sondern den verschiedenen Peers ein Nutzungsrecht auf diesen Daten einzurichten. Hierdurch kann der Speicherbedarf drastisch reduziert werden, da komplette System-Backups zu großen Teilen identische Systemdaten beinhalten. Jedoch muss das explizite Löschen von Daten hierbei einer genaueren Betrachtung unterzogen werden. Möchte ein Peer A Daten löschen um seine Quota zu bereinigen, so müssen zunächst die Nutzungsrechte der Datei geprüft werden. Haben noch andere Peers Nutzungsrechte an diesen Daten, darf das System nur das Nutzungsrecht des Peers A von der Datei entfernen.

2.2.3.4 Gegenüberstellung von verteilten Datenspeichern in Sensornetzwerken und in P2P-Netzwerken

P2P-Netzwerke unterscheiden sich hauptsächlich in den vorhandenen Ressourcen zu Sensornetzwerken. Die Bandbreite des Übertragungsmediums und der Speicherplatz sind in P2P-Netzwerken nahezu unbegrenzt. Sensornetzwerke hingegen unterliegen hier starken Einschränkungen, durch den Einsatz auf Basis von Batterien und ihrer kleinen Bauweise. Dennoch kann aus den Erkenntnissen des P2P-Forschungsgebiets einiges für die Entwicklung von verteilten Datenspeichern in Sensornetzwerken gelernt werden.

In einem Sensornetzwerk fallen stetig Sensorknoten aus. Die aufgezeichneten und auf verschiedenen Sensorknoten verteilt gespeicherten Informationen des Sensornetzwerks dürfen trotzdem nicht verloren gehen. In P2P-Netzwerken ist diese Problemstellung durch den stetigen Ein- und Austritt von Peers ebenfalls gegeben. Es müssen also in beiden Netz-

werktypen Verfahren zur persistenten Datenhaltung entwickelt werden, die auch beim Wegfall von mehreren Knoten problemlos funktionieren.

Ein weiterer Punkt ist die Optimierung des Datenflusses. Dieser ist in Sensornetzwerken durch die geringe Bandbreite des Übertragungsmediums und die eingeschränkte Lebensdauer durch den Betrieb mit Batterien stark begrenzt. In P2P-Netzwerken hingegen sollte der Datenfluss aufgrund der immensen Datenmengen optimiert werden.

Es bleibt festzuhalten, dass sich das Verhalten von P2P-Netzwerken und Sensornetzwerken in einigen Forschungsbereichen prinzipiell überschneidet. So können die Forschungsergebnisse des einen Netzwerktyps häufig auf den anderen Netzwerktyp übertragen werden.

2.3 Network Coding

Der Begriff Network Coding umfasst Algorithmen, die der Flussoptimierung in Netzwerken dienen. Der Informationsfluss stellt die maximale Übertragungsgeschwindigkeit in einem Netzwerk dar und wird durch die langsamste Komponente begrenzt [54, Kapitel 4.1.4]. Ein Verfahren, um den Informationsfluss zu steigern, ist das Kombinieren von mehreren Paketen. Hierbei werden die eingehenden Informationen miteinander verschmolzen, eine zusätzliche Information *(die besagt, welche Klartext-Informationen in das Paket eingegangen sind)* angehängt und das so entstandende Paket weitergesendet. Zusätzlich kann mittels dieses Vorgehens gezeigt werden, dass es vom Vorteil gegenüber der Replikation von Daten ist, wenn mehrere Empfänger eine gleiche Menge von Informationen erhalten. Durch die Kombination der Daten kann Bandbreite gespart werden. Somit schonen Verfahren zur Optimierung des Informationsflusses in Netzwerken die Ressourcen des Übertragungsmediums. [4] Abbildung 2.1 stellt den Informationsfluss in einem kleinen Netzwerk dar. Das Netzwerk kann

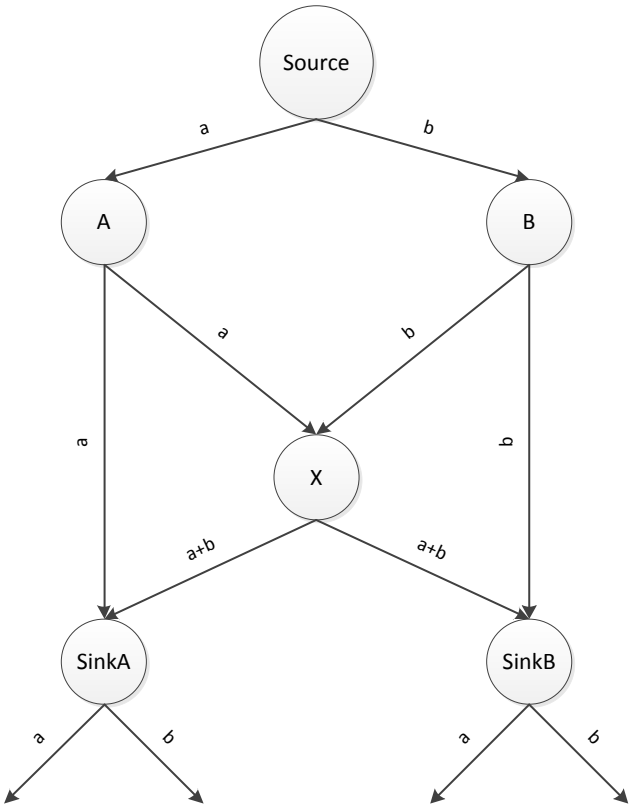

Abbildung 2.1: Darstellung der Kombination von Paketen mittels Network Coding

als gerichteter Graph angesehen werden. Hierbei ist $Source$ die Informationsquelle, die eine Information a und eine Information b sendet. Diese zwei unterschiedlichen Pakete gehen an zwei unterschiedliche Empfänger A und B. Die Knoten $SinkA$ und $SinkB$ sind die Empfänger und sollen jeweils beide Informationen erhalten. Knoten A sendet Paket a an alle folgenden Knoten $\{X, SinkA\}$, B sendet b an die Knoten $\{X, SinkB\}$. Der Knoten X kombiniert die eingehenden Pakete a und b zu $a+b$ und sendet diese an die beiden Empfänger.

Durch die Kombination der Pakete a und b muss X nur ein Paket ($a+b$) und nicht zwei Pakete (a, b) weitersenden. Die Empfänger haben nun jeweils ein Klartext-Datenpaket und ein codiertes Datenpaket erhalten. Mittels des Klartext-Datenpakets kann das codierte Datenpaket decodiert werden. Beide Empfänger haben alle Informationen erhalten.

Die Berechnung der codierten Pakete basiert auf mathematisch endlichen Körpern, auch Galois-Körper genannt. Hierdurch muss bei der Decodierung lediglich ein lineares Gleichungssystem gelöst werden. [30, Einleitung+Kapitel 8]

2.3.1 Linear Network Coding

Das Linear Network Coding ist die einfachste Variante des Network Codings. Hierbei werden die Informationen in der Reihenfolge, in der sie bei einem Knoten im Netzwerk eingehen, verknüpft. Der Netzwerkfluss ist hierbei wie in Abbildung 2.1 gezeigt.

Seien i und j Paket-Identifikatoren, D_i ein eingehendes Paket (Klartextpaket) und E_j ein codiertes ausgehendes Paket, so kann mittels c_j ein Koeffizientvektor bestimmt werden, der die Abbildung mehrerer D_i zu einem E_j bestimmt. Der Koeffizientenvektor beinhaltet für jeden Paket-Identifikator i einen Eintrag, der entweder 1 oder 0 sein kann. Für jedes Klartextpaket, dass in das entsprechende codierte Paket eingeht, wird eine 1 im Koeffizientenvektor eingetragen. Die Summe aller eingehenden Pakete D_i stellt somit das codierte Paket E_j dar. Die Formel (2.1) stellt diese Abhängigkeit dar. Hierbei ist c_j^i der Koeffizient, der das Eingehen des Klartextpakets D_i in E_j beschreibt.

$$E_j = \sum_{i=1}^{n} c_j^i * D_i \tag{2.1}$$

Die Empfängerknoten kennen in der Regel nur E_j und c_j. Das heißt, dass jeder Empfängerknoten das entsprechende lineare Gleichungssystem lösen muss, um die einzelnen Klartextpakete (D_i) zu erhalten. Hierzu ist es notwendig, dass jeder Empfänger mindestens die Anzahl an Paketen m erhält, die der Sender gesendet hat. Es gilt also $m \geq n$, mit n : Anzahl der gesendeten Pakete.

Network Coding kann durch lineare Kombination der Informationen auch in Netzwerken mit mehreren Multicast-Sendern erfolgreich eingesetzt werden. [45]

2.3.2 Random Network Coding

Eine andere Variante des Network Coding ist das Random Network Coding (auch Random Linear Network Coding genannt). Der Unterschied zum Linear Network Coding besteht im Wesentlichen darin, dass die Wahl der Informationen / Koeffizienten, die in ein codiertes Paket einfließen, zufällig erfolgt. Das heißt auch, dass nicht alle eingehenden Pakete in ein ausgehendes Paket einfließen müssen. Ein Vorteil ist, dass die zufällige Codierung eine effektive Komprimierung von korrelierenden Daten aufweist. [25, 26]

Wird die Formel (2.1) angenommen und der Koeffizientenvektor c_j durch einen zufälligen Koeffizientenvektor β_j ausgetauscht, kann mittels der folgenden Formel eine zufällige lineare Kombination von Klartextpaketen erstellt werden.

$$E_j = \sum_{i=1}^{n} \beta_j^i * D_i, \quad Pr(\beta_j = \beta) = \frac{1}{p} \quad \forall \beta \in \mathbb{F}_q \tag{2.2}$$

Hierbei ist $\mathbb{F}_q$ ein endlicher Körper, der sämtliche codierte Nachrichten umfasst. Diese Vorgehensweise kann völlig ohne Koordination unter den Knoten eines Netzwerks eingesetzt werden. Das Resultat kann eine beschleunigte Verteilung von Informationen im gesamten Netzwerk sein. [1, 12]

Auch in P2P-Netzwerken können Random Network Coding Verfahren erfolgreich eingesetzt werden. So finden sie zum Beispiel beim Austausch großer Daten, wie auch bei der Verteilung von Multimedia-Live-Streams, Einsatz. Die Basis stellt hierbei das Random Gossiping, auch der Random Phone Call genannt, dar. Das Prinzip ist hierbei, dass eine Information an n Empfänger verteilt werden soll. Die Frage ist, wie viele Runden es benötigt, um die Information an alle n Empfänger zu verteilen, wenn jeder Knoten, der die Information kennt, genau einen weiteren Knoten pro Runde involvieren darf. Es hat sich gezeigt, dass die Anzahl der Runden durch $\log_2 n + \log n + O(1)$ zu berechnen ist. In P2P-Netzwerken ist es jedoch meistens nicht nur eine Information, sondern es sind häufig viele, die verteilt werden müssen. Die Anzahl der Runden kann dann auf $k + \log_2 n$ für k Informationen gesetzt werden. [32]

2.4 Aggregation

Unter dem Begriff „Aggregation“ wird das Zusammenfassen von Informationen verstanden. Das Ziel ist es, weniger Daten senden zu müssen. Empfängt ein Knoten zum Beispiel zwei Temperaturwerte aus seinem geographischen Umfeld, könnte er diese zu einer Durchschnittstemperatur zusammenfassen. Bei der Aggregation wird zwischen einer verlustbehafteten und einer verlustfreien Aggregation unterschieden. Bei der verlustbehafteten Variante können die Ausgangsdaten nicht wiederhergestellt werden. In diesem Beispiel existiert nur

noch die Durchschnittstemperatur. Die verlustfreie Aggregation hingegen garantiert eine vollständige Wiederherstellung der Ausgangsdaten. Auf diese Weise bleibt das Abrufen der zwei Temperaturwerte möglich. [18]

In dieser Arbeit wird die Art der Grunddatenmenge (Temperatur, etc.) nicht beachtet. Stattdessen werden die Daten mittels der XOR-Operation verknüpft. Ein ähnliches Verfahren wird auch bei LT Codes angewendet [37]. Beim XOR-Verfahren werden die einzelnen Werte auf Bitebene mittels der XOR-Operation verknüpft. Das Resultat hat dieselbe Länge (in Bit) wie der längste eingehende Wert. Weiterhin müssen die eingegangenen Werte identifizierbar sein, um den Decodierprozess anstoßen zu können. Hierzu wird in einem Koeffizientenvektor festgehalten, welche Ausgangsinformation in einen codierten Wert eingeflossen sind.

Der Decodierprozess ist wie der Codierprozess aufgebaut. Es wird jedoch der codierte Wert als Basis genommen. Bereits bekannte Informationen werden mit dem codierten Wert mittels der XOR-Operation verknüpft und so die Klartextwerte extrahiert. Hierbei werden die Eigenschaften der XOR-Operation ausgenutzt, wie das folgende Beispiel zeigen soll. Die XOR-Operation wird durch $\oplus$ repräsentiert.

Für XOR gilt

$\oplus$	0	1
0	0	1
1	1	0

Sollen die Informationen a, b und c aggregiert werden und ist dec der aggregierte Wert, so ergibt sich

$$\begin{aligned} a &= 00110011 \\ \oplus b &= 10011001 \\ \oplus c &= 10001000 \\ \hline dec &= 00100010 \end{aligned}$$

Nimmt man an, dass ein Empfänger die Informationen a und c, sowie den aggregierten Wert dec erhalten hat, kann er b wie folgt berechnen.

$$\begin{aligned} a &= 00110011 \\ \oplus dec &= 00100010 \\ \oplus c &= 10001000 \\ \hline b &= 10011001 \end{aligned}$$

2.4.1 Decodierprozess

Beim Auslesen der Daten des Netzwerks erhält ein Administrator viele aggregierte Werte. Diese Werte müssen nun untereinander kombiniert werden, um die Klartext-Informationen

zu erhalten. Das Prinzip ist hierbei dasselbe wie vorangehend oben beschrieben. Jedoch kann es sein, dass während des Decodierprozesses nicht alle Informationen, die in den aggregierten Wert eingegangen sind, vorliegen. Diese nicht vollständig decodierten Pakete werden in einem extra Puffer gehalten und fließen immer wieder in den Decodierprozess mit ein. Sämtliche decodierten Informationen werden zur Decodierung verwendet. Die Abbildung 2.2 veranschaulicht das Vorgehen. [44, Kapitel 3.2.2]

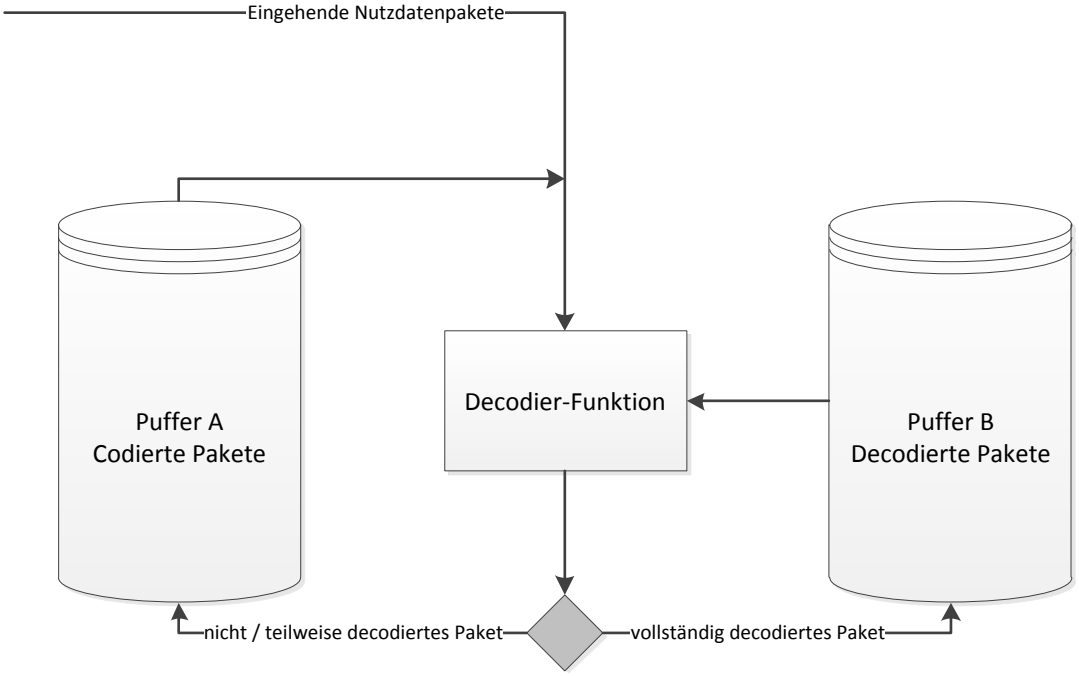

Abbildung 2.2: Darstellung des Decodierprozesses von aggregierten Werten

2.5 Konfidenzintervalle

In der Statistik werden Konfidenzintervalle zur Schätzung von Parametern bei Stichproben eingesetzt. Sie dienen dabei der Eingrenzung des gesuchten Wertes auf ein stark begrenztes Intervall. Konfidenzintervalle sind zudem genauer als eine Punktschätzung. Grundlegend unterscheidet man bei Konfidenzintervallen zwischen der Kenntnis und Unkenntnis über die Standardabweichung.

Im Rahmen dieser Arbeit wird es nicht möglich sein, die Standardabweichung der ermittelten Werte zu bestimmen. Im Folgenden beschränkt sich daher die Einführung in Konfidenzintervalle auf die Herangehensweise bei unbekannter Standardabweichung. Weitere Informationen zu Konfidenzintervallen können in [52] und [56] nachgelesen werden.

Um ein Konfidenzintervall zu bestimmen, wird zunächst das Konfidenzniveau $1-\alpha$ festgelegt. Das Konfidenzniveau gibt die Wahrscheinlichkeit an, dass das Konfidenzintervall den gesuchten Parameter überdeckt. Somit ist α die Irrtumswahrscheinlichkeit. Nun muss entschieden werden, welche Art der Verteilung vorliegt. Unterschieden wird zwischen einer Normalverteilung und einer beliebigen Verteilung. [56, S.329f]

2.5.1 Konfidenzintervall bei unbekannter Standardabweichung für eine Normalverteilung

Bei einer Normalverteilung der ermittelten Werte und einer unbekannten Standardabweichung wird zur Bestimmung des Konfidenzintervalls die erwartungstreue Stichprobenvarianz eingesetzt. Zunächst muss die Anzahl der Stichproben n festgelegt werden. Nachfolgend ist das arithmetische Mittel $\overline{X}$ aus den Stichprobenwerten $\{X_1,...,X_n\}$ zu bestimmen. Die erwartungstreue Stichprobenvarianz ist definiert als [56, S.334]:

$$S_V^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i-\overline{X})^2 \tag{2.3}$$

Weiterhin muss das zugehörige Quantil $t_{n-1;1-\alpha/2}$ bestimmt werden, da es sich bei dieser Stichprobenfunktion um eine t-verteilte Funktion mit $n-1$ Freiheitsgraden handelt. Eine Tabelle zur Bestimmung des Quantils befindet sich in [56, S.362]. Das Konfidenzintervall ist nun definiert als [56, S.335]:

$$[\overline{X}-t_{n-1;1-\alpha/2}\frac{S_V}{\sqrt{n}},\overline{X}+t_{n-1;1-\alpha/2}\frac{S_V}{\sqrt{n}}] \tag{2.4}$$

2.5.2 Konfidenzintervall bei unbekannter Standardabweichung für eine beliebige Verteilung

Ist bei vorliegenden Stichprobenwerten keine Normalverteilung festzustellen, kann das Konfidenzintervall durch einen größeren Stichprobenumfang dennoch bestimmt werden. Eine allgemeine Faustregel besagt, dass der Stichprobenumfang ≥ 30 sein sollte [56, S.335]. Anstelle des Quantils der t-Verteilung wird nun das Quantil der Standardnormalverteilung $z_{1-\alpha/2}$ eingesetzt. Das Konfidenzintervall ist somit definiert als [56, S.335]:

$$[\overline{X}-z_{1-\alpha/2}\frac{S_V}{\sqrt{n}},\overline{X}+z_{1-\alpha/2}\frac{S_V}{\sqrt{n}}] \tag{2.5}$$

Der Wert für das Quantil $z_{1-\alpha/2}$ kann aus der Tabelle in [56, S.360] abgelesen werden.

2.5.3 Auswahl des Verfahrens zur Bestimmung der Konfidenzintervalle in dieser Arbeit

Zur Auswahl eines Verfahrens zur Bestimmung der Konfidenzintervalle sind die Messwerte betrachtet worden. Diese unterliegen teilweise einer starken Streuung. Im Verlauf dieser Arbeit kann deshalb nicht von einer Normalverteilung der Stichprobenwerte ausgegangen werden. Stattdessen liefert die Simulation einen großen Stichprobenumfang. Zur Bestimmung der Konfidenzintervalle wird der Ansatz für eine beliebige Verteilung der Stichprobenwerte, wie in Abschnitt 2.5.2 dargestellt, angewendet.

2.6 TinyOS

TinyOS ist ein speziell auf drahtlose Sensornetzwerke abgestimmtes Betriebssystem. Es ist komplett Open-Source. Es ist komponentenbasiert und arbeitet mit ereignisbasierten Aufrufen. Die Komponenten werden zu einem Programm verbunden und über die definierten Schnittstellen angesprochen. Durch das ereignisbasierte Model kann TinyOS sämtliche Hardware in einen Schlafmodus versetzen und so Energie sparen. Tritt ein Ereignis auf, wird die Hardware aktiviert, der Programmteil ausgeführt und die Hardware wieder in den Schlafmodus geschickt. TinyOS wurde in der eigens für TinyOS entwickelten C Erweiterung nesC[5] implementiert.

Weiterhin bietet TinyOS verschiedene Schnittstellen an, um von einem PC mit dem Sensornetzwerk zu kommunizieren. Hierzu wird ein Sensorknoten an den PC angeschlossen und dieser als Basisstation zur Weiterleitung von Paketen eingesetzt. [57, 58]

[5] network embedded system C

Kapitel 3

Simulation des Sensornetzwerks in JAVA

3.1 Motivation

Ein Sensornetzwerk kann aus Tausenden von Sensorknoten bestehen. Da im Rahmen dieser Arbeit kein Sensornetzwerk in diesem Ausmaß vorhanden ist, sollen die empirischen Ergebnisse durch eine Simulation gesichert werden. Zudem bietet eine Simulation den großen Vorteil, dass sie anpassungsfähig ist und flexibel eingesetzt werden kann. Die Simulation ist somit fähig, sich jeder erdenkbaren Gegebenheit zu fügen.

In diesem Kapitel wird eine Übersicht über die während der Arbeit entwickelte Simulationsapplikation gegeben. Die wichtigsten Klassen werden in einer Übersicht dargestellt. Eine Beschreibung der simulierten Netzwerktopologie und der Durchführung der Simulation folgen im Anschluss.

3.2 Aufbau

Die Simulation wurde bewusst modular aufgebaut, um möglichst einfach verschiedene Szenarien testen zu können. In Abbildung 3.1 werden die einzelnen Komponenten und ihre prinzipiellen Verbindungen dargestellt. Folgend werden die wichtigsten Klassen kurz erläutert.

Simulator Der Dreh- und Angelpunkt der Simulationsapplikation ist die *Simulator*-Klasse. Die *Simulator*-Klasse initialisiert die Sensorknoten und den Administrator, definiert die rauschbehafteten Bereiche, veranlasst die Generierung der Daten und steuert den Sendevorgang der Daten. Weiterhin berechnet sie die Anzahl an überlebenden Sensorknoten und veranlasst das zufällige Ausfallen der Sensorknoten.

DataGenerator Sämtliche Daten, die der Quell-Sensorknoten versendet, werden in der *DataGenerator*-Klasse definiert. Die Daten werden vor dem eigentlichen Simulationsstart

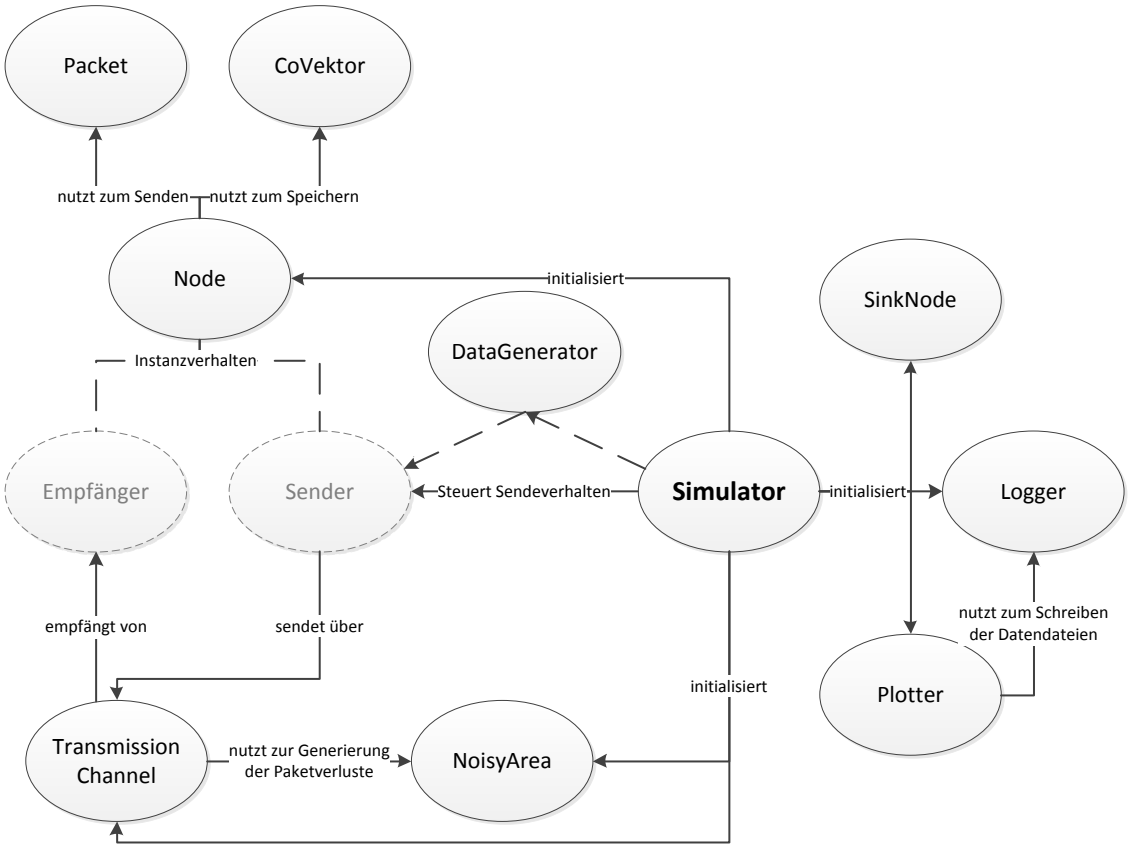

Abbildung 3.1: Genereller Zusammenhang der Simulationskomponenten

erzeugt und an die jeweiligen Simulationsrunden übergeben. So wird auch über eine Vielzahl von Simulationsrunden für ein einheitliches Quellbild gesorgt.

Node Sensorknoten werden über die Klasse *Node* definiert. Jede *Node* kann Pakete senden und empfangen und hat einen internen Speicher. Während der Initialisierung wird die *Node* entweder zu einem Sender oder zu einem Empfänger. Der Sender sendet die Daten, verpackt in einzelne Pakete (*Packet*-Klasse) an alle Empfänger in seiner Sendereichweite. Der Übertragungskanal ist rauschbehaftet und wird durch die *TransmissionChannel*-Klasse repräsentiert. Entscheidet sich der Empfänger, das eingehende Paket zu speichern, kann er es zusätzlich aggregieren und im Speicher festschreiben. Das Aggregationsverhalten wird hierbei durch die jeweilige Implementierung der *Node*-Klasse definiert. Zur Identifikation der in einen Aggregationswert eingegangenen Pakete werden die Paketnummern im Koeffizientenvektor (*CoVektor*-Klasse) festgehalten.

SinkNode Der Administrator des Netzwerks wird durch die *SinkNode*-Klasse dargestellt. Die *SinkNode* gibt am Ende der Simulation einen Befehl an alle überlebenden Empfänger, sodass diese ihm ihre Daten zu senden. Diese Daten der überlebenden Sensorknoten versucht die SinkNode zu decodieren.

Logger und Plotter Mittels der Utility-Klassen *Logger* und *Plotter* können Informationen über den Simulationshergang für den Anwender sichtbar gemacht werden. Der *Logger* schreibt hierbei die Informationen als Textdatei aus. Der *Plotter* kann diese Daten ggf. nutzen, um sie mit einem geeigneten Programm (hier GnuPlot, siehe auch [63]) grafisch aufzuarbeiten.

3.2.1 Topologie des simulierten Sensornetzwerks

Das simulierte Sensornetzwerk basiert auf einem Sender- und vielen Empfänger-Sensorknoten. Der Sender-Sensorknoten dient als Datenquelle und versendet sein Wissen an alle Empfänger-Sensorknoten. Der dabei genutzte Übertragungskanal ist rauschbehaftet. Jedem Empfänger-Sensorknoten hängt eine Wahrscheinlichkeit an, dass das eingehende Paket auf dem Übertragungskanal verloren gegangen ist. Weitere Details zur Topologie und des Versuchsaufbaus folgen im Abschnitt 4.5.2.1.

Die empirischen Studien dieser Arbeit basieren auf einem Ein-Hop-Netzwerk. Das heißt, dass zwischen dem Sender und dem Empfänger keine weiteren Sensorknoten vorhanden sind. Um die Sicherheit weiter auszubauen, um zum Beispiel die Daten über den Desaster Radius hinaus zu verteilen, können die Ergebnisse dieser Arbeit mit einigen Anpassungen auf ein Multi-Hop-Netzwerk übertragen werden. Hierzu müssten die Pakete mit der ID des Quell-Sensorknotens und einem Hop-Zähler versehen werden. Die Empfänger-Sensorknoten müssten nun für jedes Paket anhand des Hop-Zählers entscheiden, ob sie es

weitersenden, abspeichern oder verwerfen. Die Ergebnisse dieser Arbeit bleiben erhalten, solange der Administrator die entsprechende Anzahl an überlebenden Sensorknoten abfragt, die die Erkenntnisse hervorgebracht haben. Entscheidend ist, dass jeder Empfänger-Sensorknoten gegenüber den anderen Empfänger-Sensorknoten autonom über das Speicherverhalten entscheidet.

3.3 Durchführung einer Simulation

Die Simulation erfolgt prinzipiell rundenbasiert, das heißt, dass jeder Versuchsaufbau mehrere Male ausgeführt wird. Die Versuchsaufbauten sind durch eine Konfigurationsklasse im Vorfeld eingeschränkt. Die Konfiguration umfasst unter anderem die Anzahl der Simulationsrunden pro Versuchsaufbau, die Anzahl der Empfänger, die Anzahl der zu sendenden Pakete und die Anzahl der überlebenden Sensorknoten. Die Einstellung der Anzahl an überlebenden Sensorknoten erfolgt durch die Angabe eines Bereichs[6]. Während der Simulation werden dann, entsprechend pro Wert aus diesem Bereich, die Simulationsrunden absolviert.

Tabelle 3.1: Allgemeine Konfiguration des Simulationsaufbaus

Bezeichnung	Wert	Beschreibung
Zu sendende Pakete	100	Die Anzahl der Pakete, die pro Simulationsrunde gesendet werden
Empfängerknoten	20	Anzahl der Empfängerknoten, bevor ein Desaster eintritt
Anzahl überlebender Knoten	4-10	Anzahl der Empfängerknoten, die nach einem Desaster noch funktionsfähig sind

Sämtlichen Simulationen in dieser Arbeit liegt die folgende Grundkonfiguration aus Tabelle 3.1 zu Grunde. Zur Begrenzung der Simulationsdauer beträgt die Anzahl der zusendenden Pakete nur 100. Die Abbildungen 3.2 und 3.3 zeigen, dass die Simulationsergebnisse auf reale Zahlen übertragbar sind. Die Anzahl der überlebenden Knoten wurde nach ersten Versuchen auf 4-10 beschränkt. Bei weniger als vier Knoten war keine Decodierung mehr möglich. Der Decodierprozess konnte für die meisten Versuche mit weniger als zehn Knoten erfolgreich abgeschlossen werden.

3.4 Simulationsausgabe

Während der Simulation werden grundlegende Daten über die Konfiguration der Simulation, die Quelldaten, den Zustand der Knoten und die Decodierungsergebnisse in Textdateien

[6] zum Beispiel: 4-10 Empfängerknoten sollen überleben

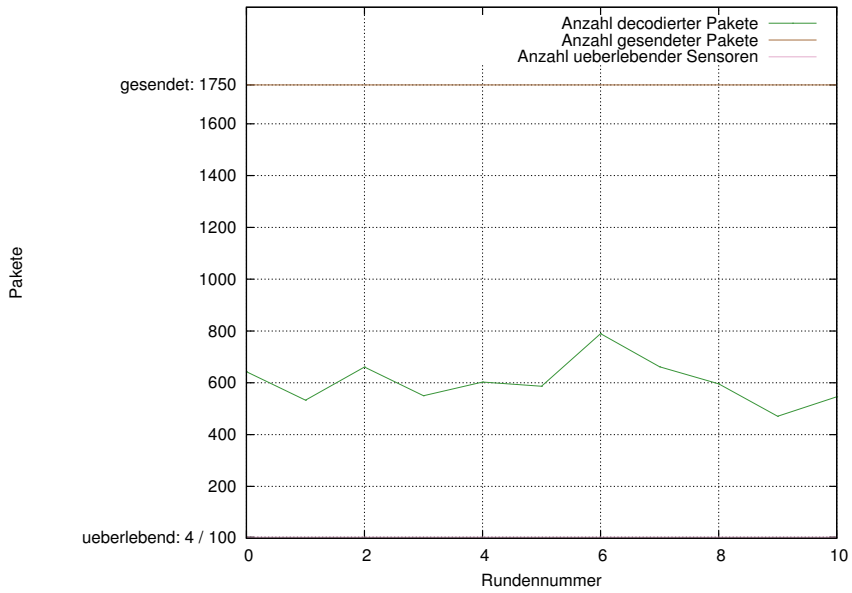

Abbildung 3.2: Simulation mit realen Werten nach Ansatz 3; 1750 gesendete Pakete bei vier überlebenden Sensorknoten aus 100 Empfängern, $R = 0.5$

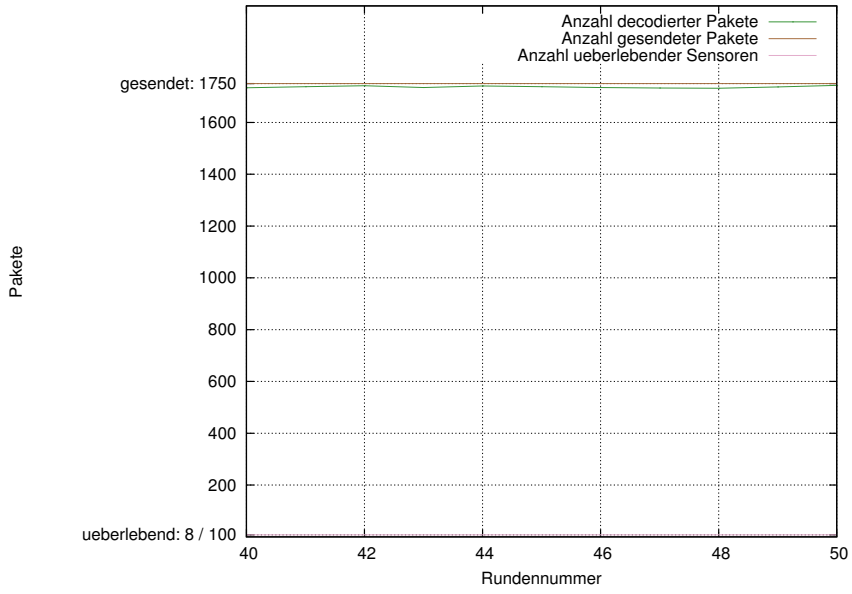

Abbildung 3.3: Simulation mit realen Werten nach Ansatz 3; 1750 gesendete Pakete bei acht überlebenden Sensorknoten aus 100 Empfängern, $R = 0.5$

geschrieben. Diese werden pro Versuchsaufbau in Ordnern gruppiert abgelegt. Zur besseren Visualisierung werden einige der Daten mittels GnuPlot graphisch aufbereitet und als Bilddateien abgelegt. Hierüber kann auf einfache Weise die Auswertung der Simulation vorgenommen werden.

Während der Simulation sind enorme Datenmengen entstanden. Diese konnten nicht alle in gedruckter Form in dieser Arbeit wiedergegeben werden. Sämtliche Simulationsergebnisse (Log-Dateien und Bilder), die zur Auswertung herangezogen wurden, befinden sich auf der beiliegenden DVD. Beispielhaft sind einige an entsprechender Stelle in dieser Arbeit eingefügt und verwiesen worden.

Kapitel 4

Verteilter Datenspeicher mittels Network Coding

4.1 Zielsetzung

In einem Sensornetzwerk soll ein verteilter Datenspeicher realisiert werden. Hierzu werden die Quell-Sensorknoten[7] die aufgezeichneten Nutzdaten per Broadcast versenden. Jeder Empfänger-Sensorknoten entscheidet für jedes empfangene Paket zufällig, ob er das Paket speichert und ob er das Paket mit bestehenden Daten aggregiert oder nicht. Es wird angenommen, dass nach einer bestimmten Zeit ein Desaster[8] eintritt, bei dem eine bestimmte Anzahl an Sensorknoten ausfallen. Ziel ist es, nach diesem Desaster und nach dem Auslesen der Daten der überlebenden Sensorknoten, die Quelldatenmenge möglichst präzise wiederherzustellen. Da es sich bei den Quelldaten meist um eine Funktion über die Zeit handelt, müssen nicht alle Daten wiederhergestellt werden, um einen Eindruck über den Datenverlauf zu erhalten. Beispielhaft werden in den Abbildungen 4.1 und 4.2 Temperaturverläufe dargestellt. Hierbei ist jeweils die Quelldatenmenge (grün) gegenüber der Datenmenge nach erfolgter Decodierung der Daten der überlebenden Sensorknoten nach einem Desaster (braun) visualisiert.

Nach allgemeinen Informationen zum Aufbau des Sensornetzwerks und der persistenten Datenspeicherung folgt nun die Ausarbeitung des Autors dieser Arbeit zum Sende- und Speicherverhalten der Sensorknoten. Es werden die grundlegenden Forschungsergebnisse dieser Arbeit aufgearbeitet. Die Erkenntnisgewinnung basiert auf der im vorangehenden Kapitel beschriebenen Simulationsapplikation.

[7] Sensoren, die Nutzdaten aus Überwachungstätigkeiten gespeichert haben und diese nun verteilen.

[8] Ereignis, bei dem der Großteil der Sensorknoten ausfällt; Der Ursprung des Desasters ist nicht von Bedeutung

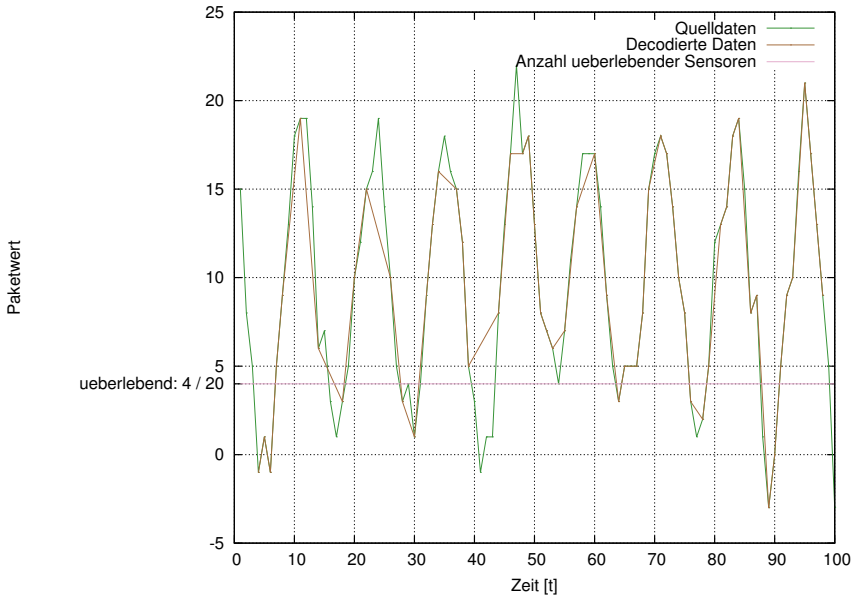

Abbildung 4.1: Gegenüberstellung der Quelldaten und der decodierten Daten bei vier überlebenden Sensorknoten

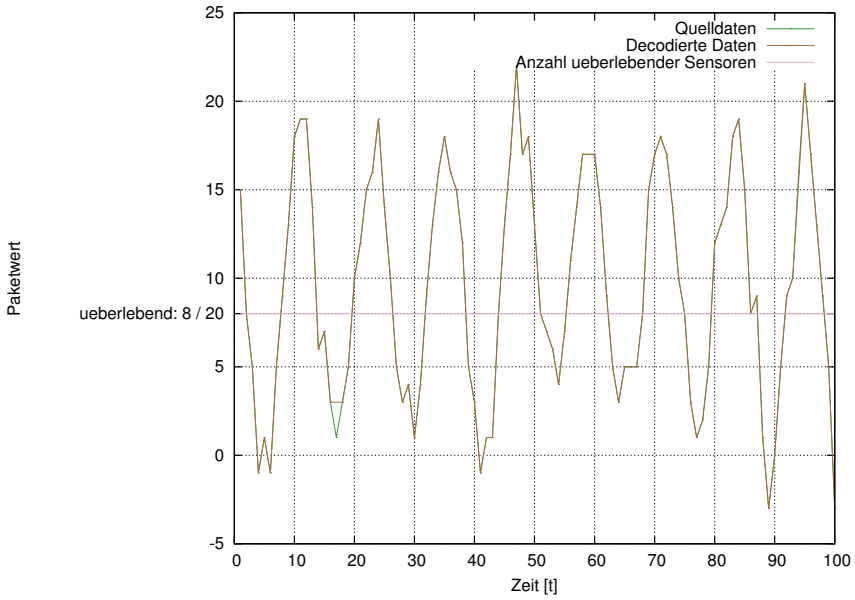

Abbildung 4.2: Gegenüberstellung der Quelldaten und der decodierten Daten bei acht überlebenden Sensorknoten

4.2 Initialisierungsphase des Sensornetzwerks

Während der Initialisierungsphase erhält jeder Sensorknoten automatisch eine eindeutige ID[9]. Hierüber können Pakete von verschiedenen Sendern logisch getrennt werden. Weiterhin initialisiert jeder Sensorknoten einen Zufallszahlengenerator, mit dem der Sensorknoten für jedes Paket zufällig entscheiden kann, ob er das Paket speichert und ob er es aggregiert. Beim Hochfahren eines jeden Sensorknotens wird ebenfalls entschieden, ob der Sensorknoten als Quell-Sensorknoten initialisiert werden soll oder ob er nur als Empfänger agiert.

4.3 Verteiltes Speichern der Informationen

Der Quell-Sensorknoten verteilt die aufgezeichneten Informationen (im Folgenden: Nutzdaten), indem er sie per Broadcast versendet. Der Aufwand des Aufbaus einer direkten Verbindung zum Kommunikationspartner ist hierdurch nicht notwendig. Der Quell-Sensorknoten kennzeichnet jedes Nutzdatenpaket mit seiner ID. Jedes Paket hat zudem eine Paketnummer. Diese Paketnummer vergibt der Quell-Sensorknoten. Auf der Empfängerseite wird diese Paketnummer in den Koeffizientenvektor des Aggregationswertes eingetragen.

4.3.1 Allgemeine Vorgaben für persistente Datenspeicherung

Sei ein Sensorknoten definiert, der als Quell-Sensorknoten seine Daten per Broadcast an alle Sensorknoten im Senderadius verteilt. Dieser Quell-Sensorknoten hat nach einer bestimmten Zeit n Pakete versendet. Es muss garantiert werden, dass nach dieser Zeit, bei einer Mindestanzahl von m Empfänger-Sensorknoten, jedes der n Pakete auf mindestens einem Empfänger-Sensorknoten gespeichert wurde. Es sei definiert, dass W_h die Wahrscheinlichkeit beschreibt, dass ein Empfänger ein Paket speichert, sodass jedes Paket auf genau einem Empfänger gespeichert wird.

$$W_h = \frac{1}{m},\ m \in \mathbb{N} \tag{4.1}$$

Sei weiterhin W_d die Standardabweichung von W_h.

$$W_d = \frac{1}{d},\ d \in \mathbb{N} \tag{4.2}$$

Dann ist die Wahrscheinlichkeit, dass ein Empfänger ein Paket speichert, im ungünstigsten Fall W_{hd}.

$$W_{hd} = \frac{1}{m} * \frac{1}{d},\ m,d \in \mathbb{N} \tag{4.3}$$

[9] *„By default, node ID and AM address are the same.“* [7]

Unter dieser Annahme werden jedoch nicht alle Pakete gespeichert. Um zu garantieren, dass jedes Paket gespeichert wird, muss die Wahrscheinlichkeit, dass ein Empfänger das Paket speichert, entsprechend um d nach oben korrigiert werden. Die Wahrscheinlichkeit, dass ein Empfänger ein Paket speichert, sei somit als W definiert.

$$W = d * \frac{1}{m},\ d,m \in \mathbb{N} \tag{4.4}$$

Beispiel: Wenn jedes Paket auf genau einem Sensorknoten gespeichert wird und die Standardabweichung $\frac{1}{2}$ beträgt, würde unter den obigen Bedingungen jedes zweite Paket verloren gehen. Somit müsste die Wahrscheinlichkeit, dass ein Paket gespeichert wird, verdoppelt werden. Dies ist in (4.4) festgehalten.

Zur Vereinfachung des Versuchsaufbaus wird dieser Ansatz weiter generalisiert. Sei R der Redundanzfaktor eines Netzwerks und x die Anzahl an Kopien eines Pakets, die auf m Empfängern verteilt gespeichert werden:

$$R = \frac{x}{m},\ x,m \in \mathbb{N}, x \leq m \tag{4.5}$$

Der Redundanzfaktor beschreibt gleichfalls die Wahrscheinlichkeit, dass ein Sensorknoten ein eingehendes Paket speichert. Die Standardabweichung sei im Redundanzfaktor bereits enthalten.

Ein weiterer Fall, der das Ergebnis beeinträchtigen kann, ist der Extremfall eines Desasters. Ein Desaster beschreibt den Fall, dass durch extreme äußere Einflüsse eine Großzahl an Sensorknoten in einem bestimmten begrenzten Bereich ausfallen. Der hierbei entstehende Desaster Radius, also der Umkreis, indem die Zerstörung gewirkt hat, sollte in der Betrachtung der Verteilung der Daten miteinfließen.

Um den Verlust von Daten zu verhindern, kann eine Weiterverteilung der Daten mittels Hops oder die direkte Speicherung auf Sensorknoten, die geografisch weiter voneinander entfernt sind, erfolgen. [21]

Der Desaster Radius fand in dieser Arbeit keine weitere Betrachtung.

4.4 Vorgehen am Quell-Sensorknoten

Nachdem der Quell-Sensorknoten neue Informationen aufgezeichnet hat, versendet er diese per Broadcast an alle Sensoren in Sendereichweite. Hierzu werden Pakete der Form $\{ID, PaketNr, Nutzdaten\}$ aufgebaut. Die ID identifiziert den Quell-Sensorknoten. Die $PaketNr$ ist eine fortlaufende Zahl. Diese identifiziert jedes Paket pro Quell-Sensorknoten eindeutig. Folglich identifiziert das Paar (ID, $PaketNr$) jedes Paket im Netzwerk eindeutig. Die $Nutzdaten$ beinhalten die Informationen, die der Quell-Sensorknoten zuvor aufgezeichnet hat.

4.5 Vorgehen am Empfänger-Sensorknoten

Der Empfänger-Sensorknoten wartet auf das Eintreffen eines neuen Pakets. Sobald ein neues Paket eintrifft, wird mittels des während der Initialisierungsphase erstellten Zufallszahlengenerators entschieden, ob das Paket gespeichert werden soll. Speichert der Empfänger-Sensorknoten das Paket, entscheidet er erneut mittels des Zufallszahlengenerators, ob und mit welchem Paket im Speicher das empfangene Paket aggregiert werden soll. Voraussetzung hierfür ist, dass bereits Pakete im Speicher liegen. Ist diese Voraussetzung nicht erfüllt, müssen Ausnahmeregelungen definiert werden. In den folgenden Ansätzen sind hierzu verschiedene Regelungen definiert.

Im Abschnitt 4.5.2 werden drei unterschiedliche Ansätze aufgezeigt, die verschiedene Arten von Aggregationsverhalten behandeln.

4.5.1 Speicherverhalten

Jeder Sensorknoten entscheidet mit Hilfe eines Zufallszahlengenerators, ob er ein eingehendes Paket speichert oder verwirft. Das Speicherverhalten wird von der Wahrscheinlichkeitsrate beeinflusst. Diese Wahrscheinlichkeitsrate wird vor der Initialisierung des Netzwerks festgelegt und ist ab diesem Zeitpunkt fest in jedem Sensorknoten verankert. Der Wert der Festlegung der Wahrscheinlichkeitsrate wird im Laufe dieses Kapitels ausgearbeitet. Eine Kommunikation zwischen den Sensorknoten, zur Abstimmung dieser Werte, ist nicht erforderlich.

4.5.1.1 Wahrscheinlichkeitsrate

Zur Festlegung der Wahrscheinlichkeitsrate sind einige Kenntnisse über das Sensornetzwerk nötig, in dem die Sensorknoten installiert werden sollen. Da kein Sensorknoten alle Pakete eines Senders speichert, sollte die Wahrscheinlichkeitsrate so gewählt werden, dass die Redundanz der Daten so groß wie nötig ist. Das heißt, dass wenn die Mehrzahl an Sensorknoten ausfällt, gerade ausreichend viele Daten aus dem Netzwerk entnommen werden können, um die Quelldaten zu reproduzieren.

In Abschnitt 4.3.1 wurde die Wahrscheinlichkeitsverteilung analysiert. In (4.5) wurde der Redundanzfaktor der Daten festgelegt. Diese Formel soll hier weiter verwendet werden.

4.5.1.2 Zufallszahlengenerator

Der Zufallszahlengenerator wird unter zwei unterschiedlichen Voraussetzungen eingesetzt. Im ersten Szenario soll entschieden werden, ob ein Paket gespeichert und ob es aggregiert wird. Es werden also nur zwei unterschiedliche Zufallswerte benötigt. Ein Wert repräsentiert *wahr* und einer *falsch*. Das zweite Szenario beschränkt sich auf die Anzahl der bereits

gespeicherten Aggregationswerte. Um zufällig einen Aggregationspartner zu wählen, muss eine Zahl im Bereich von $[0;$ Anzahl der Aggregationswerte$[$ erzeugt werden.

Beide Szenarien können mittels der Modulo-Operation[10] abgedeckt werden. Im ersten Fall wird die Zufallszahl *mod* zwei genommen. Im zweiten Fall wird die Zufallszahl *mod* der Anzahl an gespeicherten Aggregationswerte genommen.

4.5.2 Aggregationsverhalten

4.5.2.1 Versuchsaufbau

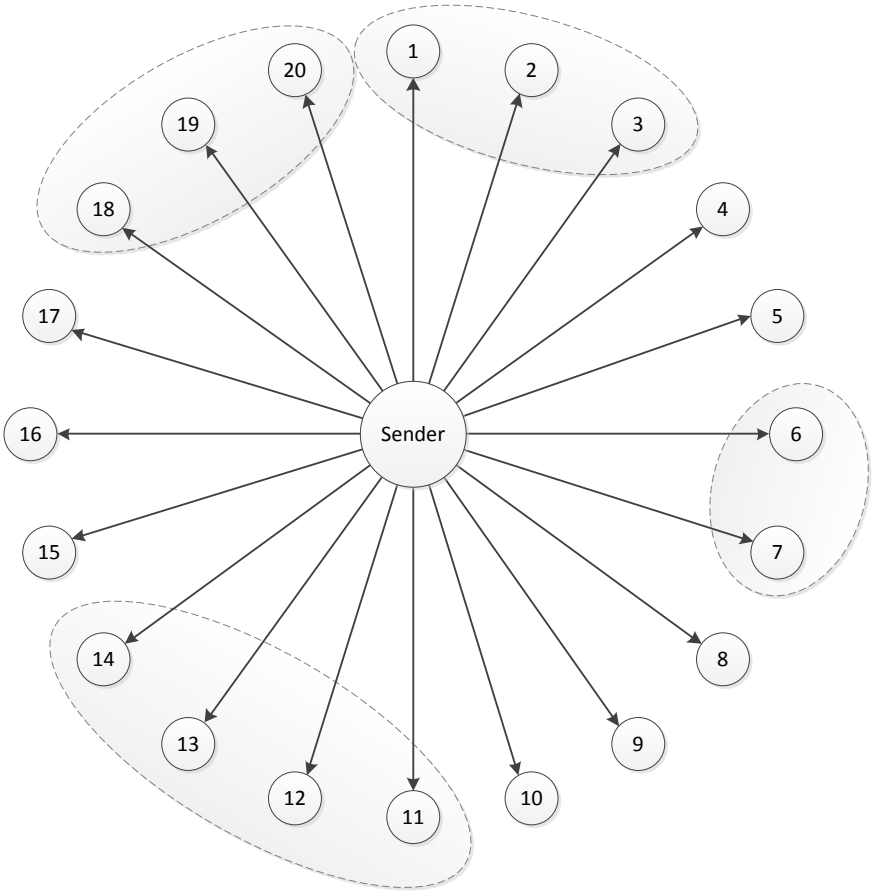

Abbildung 4.3: Visualisierung der rauschbehafteten Bereiche im Versuchsaufbau

[10] Division mit Rest, Schreibweise: $a \mod b =$ Rest der Division [39, Kapitel 13.2]

Der Versuchsaufbau wurde bei allen Ansätzen gleich vorgenommen. Die zugrunde liegenden Daten sind Durchschnittstemperaturen pro Monat der letzten Jahre. Abbildung 4.3 visualisiert die Knoten, die in einem rauschbehafteten Bereich liegen und hierdurch nicht alle Pakete empfangen. Der Verlustfaktor pro rauschbehaftetem Bereich wurde fix auf 30% Paketverlust eingestellt. Es wurden immer 1 Sender und 20 Empfänger eingesetzt. Durch die Annahme eines einzigen Senders pro Empfänger wird der Versuchsaufbau und die Auswertung wesentlich vereinfacht. Die Anzahl der Empfänger wurde der Übersichtlichkeit halber auf 20 reduziert. Es wird später gezeigt werden, dass die Ergebnisse auf eine beliebige Anzahl an Empfängern übertragbar sind.

4.5.2.2 Ansatz 1 - Fixes Aggregationsverhalten

4.5.2.2.1 Aufbau

Im ersten Ansatz wird von einem festen Aggregationsverhalten ausgegangen. Eingehende Pakete werden beim Empfänger mit einer zuvor festgelegten Wahrscheinlichkeit gespeichert. Die Wahrscheinlichkeit berechnet sich mittels des Redundanzfaktors aus der Formel (4.5). Ebenfalls anhand einer fixen Wahrscheinlichkeit entscheidet der Empfänger ob das Paket aggregiert werden soll. Hierzu wählt der Empfänger aus der Menge der bereits gespeicherten Aggregationswerte zufällig einen und aggregiert das eingehende Paket auf diesen.

4.5.2.2.2 Abhängigkeiten

Pro Sensorknoten wird ein Gesamtspeicherplatz in Höhe von S benötigt.

$$S = n * R = n * \frac{x}{m},\ n,m,x \in \mathbb{N} \tag{4.6}$$

Dieser ergibt sich aus der Anzahl der zusendenden Pakete n und dem Redundanzfaktor R aus (4.5).

Zudem sei ein fixer Wahrscheinlichkeitswert A definiert. Dieser gibt die Anzahl der parallel zu haltenden Aggregationswerte an. Die Werte für a_1 und a_2 sind frei wählbar und werden im Verlauf der Arbeit mittels der Simulation ermittelt.

$$A = \frac{a_1}{a_2},\ a_1,a_2 \in \mathbb{N}, a_1 \leq a_2 \tag{4.7}$$

Die Anzahl der parellel zu haltenden Aggregationswerte, die ein Knoten speichern muss, ergibt sich aus (4.5) und (4.7) und sei durch A_P definiert.

$$A_P = R * A * n \tag{4.8}$$

Beispiel: In einem Netzwerk werden ein Sender und 20 Empfänger installiert. Der Redundanzfaktor ist auf $R = \frac{8}{20}$ gesetzt. Die Aggregationswahrscheinlichkeit beträgt $A = \frac{1}{2}$. Der Sender sendet $n = 10$ Pakete. Somit gilt für einen Knoten, dass er vier der zehn Pakete

speichert $\left(R = \frac{8}{20} = \frac{4}{10}\right)$. Hiervon wird er zwei Pakete aggregieren $\left(4 * A = 4 * \frac{1}{2} = 2\right)$, womit zwei Aggregationswerte übrig bleiben. Ebendies erhalten wir auch über (4.8):

$$A_P = \frac{8}{20} * \frac{1}{2} * 10 = \frac{1}{5} * 10 = 2$$

Unter diesen Vorgaben wird der Speicher jedoch volllaufen und eine Decodierung der Daten nicht mehr möglich sein, da es keine Klartextpakete mehr geben wird, um den Decodierprozess der Aggregationswerte anzustoßen. Der Wert für A_P muss also so angepasst werden, sodass nach dem Senden von n Paketen noch hinreichend viele Koeffizientenvektoren mit einem Grad = 1, also Pakete als Klartext, im Speicher des Sensorknotens liegen. Die Anpassung des Werts A_P erfolgt erst später aus den Simulationserkenntnissen heraus.

4.5.2.2.3 Simulationserkenntnisse

Es hat sich gezeigt, dass sich durch die Wahl der Wahrscheinlichkeitswerte zur Speicherung und Aggregation erhebliche Unterschiede beim Decodierungserfolg einstellen. Die Tabelle 4.1 soll einen kurzen Überblick über einige spezifische Konfigurationen und die hieraus gewonnen Erkenntnisse geben. Zu jedem Versuchsaufbau wird ein Konfidenzintervall der Anzahl der erfolgreich decodierten Pakete mit der zugehörigen Anzahl der überlebenden Sensorknoten angegeben. Sämtliche Messergebnisse sind in einer Excel-Datei auf der DVD zusammengefasst. Diese Intervalle zeigen, wie viele der 100 gesendeten Pakete erfolgreich decodiert werden konnten. Die Werte sind hier auf vier Stellen nach dem Komma gerundet.

Tabelle 4.1: Darstellung der Erkenntnisse aus der Simulation zu Ansatz 1

Redundanz-faktor R	Aggregationswahr-scheinlichkeit A	Erkenntnis
0.25	0.25	viele Koeffizientenvektoren mit Grad = 1 Koeffizientenvektor-Grad-Obergrenze: 2-3 Erfolgreich decodierte Pakete: $[86,4409; 87,1591]$ bei 10 überlebende Knoten
0.25	0.5	stark schwankende Ergebnisse durch die erhöhte Aggregation vermehrt Koeffizientenvektor-Grade > 1 Koeffizienten-Grad-Obergrenze: 4-6 Erfolgreich decodierte Pakete: $[69,8185; 71,5215]$ bei 10 überlebenden Knoten
0.25	0.75	sehr wenige Koeffizientenvektoren mit Grad = 1 Koeffizienten-Grad-Obergrenze: 10-14 Erfolgreich decodierte Pakete: $[19,7915; 20,7885]$ bei 10 überlebenden Knoten
0.5	0.25	sehr viele Koeffizientenvektoren mit Grad = 1 Koeffizienten-Grad-Obergrenze: 3-4 Erfolgreich decodierte Pakete: $[97,0035; 97,3565]$ bei 7 überlebenden Knoten
0.5	0.5	Koeffizientenvektoren mit Grad = 1 ausgewogen Koeffizienten-Grad-Obergrenze: 5-7 Erfolgreich decodierte Pakete: $[98,5005; 98,7594]$ bei 9 überlebenden Knoten
0.5	0.75	sehr wenige Koeffizientenvektoren mit Grad = 1 Koeffizienten-Grad-Obergrenze: 9-11 (Max.: 25) Erfolgreich decodierte Pakete: $[47,0453, 49,5747]$ bei 10 überlebenden Knoten
0.75	0.25	unverhältnismäßig großer Speicherbedarf sehr viele Koeffizientenvektoren mit Grad = 1 Koeffizienten-Grad-Obergrenze: 3-4 Erfolgreich decodierte Pakete: $[99,0352; 99,2247]$ bei 5 überlebenden Knoten
0.75	0.5	unverhältnismäßig großer Speicherbedarf viele Koeffizientenvektoren mit Grad = 1 Koeffizienten-Grad-Obergrenze: 5-7 Erfolgreich decodierte Pakete: $[99,1175; 99,3825]$ bei 6 überlebenden Knoten
0.75	0.75	schwankende Ergebnisse durch hohe Aggregation extrem hoher Speicherbedarf Koeffizienten-Grad-Obergrenze: 10-13 (Max.: 34) Erfolgreich decodierte Pakete: $[90,3249; 92,5950]$ bei 10 überlebenden Knoten

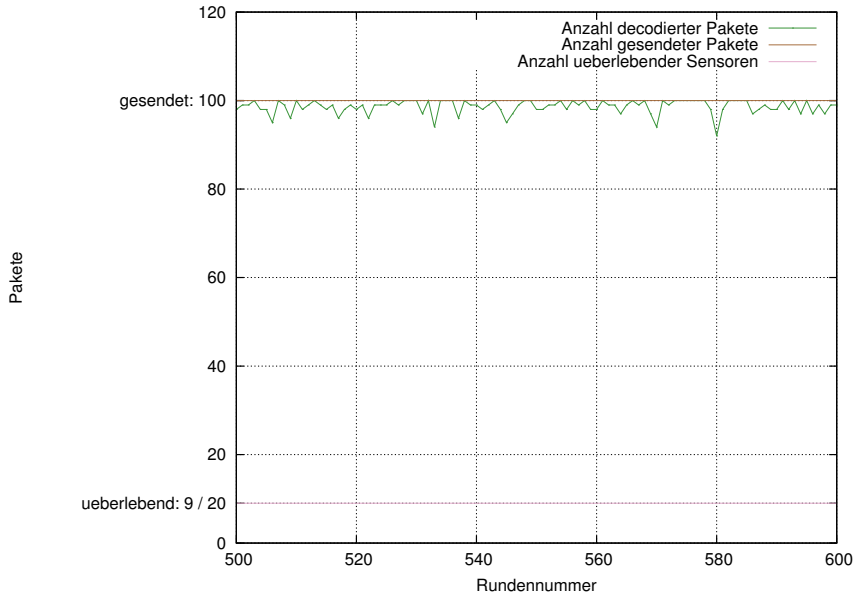

Abbildung 4.4: Anzahl der erfolgreich decodierten Werte zu der Gesamtanzahl an gesendeten Werten bei 9 überlebenden Sensorknoten aus 20 Empfängern, $R = 0.5, A = 0.5$

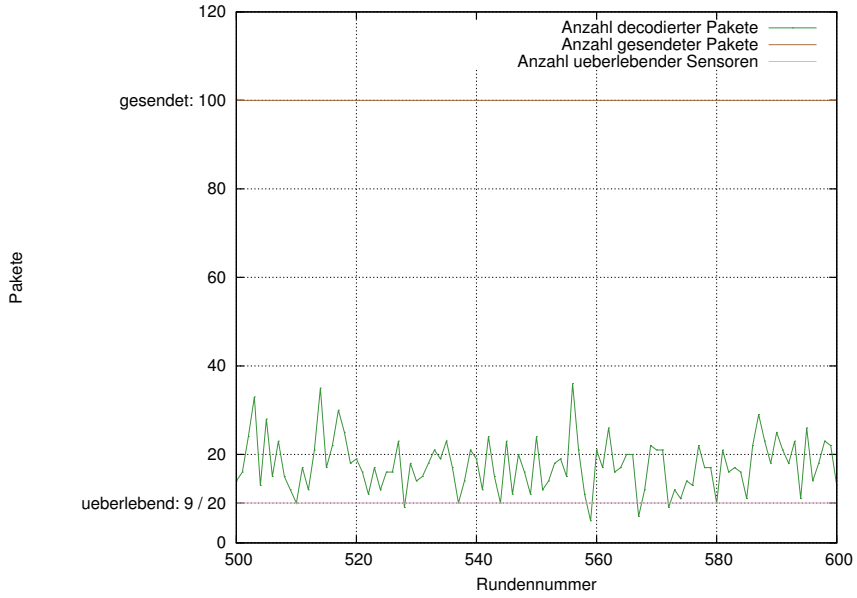

Abbildung 4.5: Anzahl der erfolgreich decodierten Werte zu der Gesamtanzahl an gesendeten Werten bei 9 überlebenden Sensorknoten aus 20 Empfängern, $R = 0.25, A = 0.75$

Es hat sich gezeigt, dass die besten Ergebnisse bei einer Konfiguration von $R = 0.5$ und $A = 0.5$ liegen. Eine Gegenüberstellung eines guten und eines schlechten Simulationsverlaufs wurde beispielhaft in den Abbildungen 4.4 und 4.5 visualisiert. Weitere Bilder befinden sich auf der beiliegenden DVD.

Es gilt außerdem, dass nur die Anzahl der überlebenden Sensorknoten relevant ist. In den Simulationsergebnissen hat sich gezeigt, dass eine Variierung der Anzahl der Empfänger keine relevante Auswirkung auf die Ergebnisse hat, solange dieselbe Anzahl an überlebenden Sensorknoten angenommen wird. Vergleiche hierzu die Abbildungen 3.2 und 3.3.

4.5.2.2.4 Vorteile

Bei guter Konfiguration liefert dieser Ansatz einen verteilten Datenspeicher mit guten bis sehr guten Decodierungseigenschaften. Trotz starken Aufkommens rauschbehafteter Bereiche bleibt die benötigte Anzahl an überlebenden Sensorknoten gering.

4.5.2.2.5 Nachteile

Wird die Aggregationswahrscheinlichkeit zu hoch angesetzt, ist die erfolgreiche Decodierung stark gefährdet. Setzt man die Aggregationswahrscheinlichkeit zu gering an, hat man keinen Vorteil gegenüber einem Speicher ohne Aggregation. Ein großes Problem stellt die Aggregation dar. Sensorknoten könnten sich bereits beim ersten Paket für eine Aggregation entscheiden. In diesem Fall ist jedoch kein Aggregationspartner vorhanden und es müsste eine Ausnahme definiert werden.

4.5.2.3 Ansatz 2 - logarithmische Steigerung des Aggregationsverhaltens

Der zweite Ansatz verfolgt das Ziel, die Aggregation erst dann zu steigern, wenn ausreichend Aggregationspartner, also Klartextpakete, vorhanden sind. Um die Aggregation im weiteren Verlauf zu begrenzen, wird eine logarithmische Steigerung als Grundlage herangezogen.

4.5.2.3.1 Aufbau

Das Speicherverhalten bleibt gegenüber Ansatz 1 unangetastet und entspricht somit weiterhin der Formel (4.5).

Die Steigerung des Aggregationsverhaltens erfolgt stufenweise. Die Stufenhöhen liegen dabei auf einer logarithmischen Kurve und seien folgend definiert:

$$r_i = \frac{d_i}{d_i + 1},\ d_i \in \mathbb{N} \tag{4.9}$$

d_i sei dabei der Divisor pro Paket i. Dieser Devisor wird in (4.11) noch genauer definiert. Durch r_i wird eine Steigerung des Aggregationsverhaltens bei ausreichend Aggregationspartnern erzielt.

Zur Aggregation wird ein zufälliger Aggregationswert aus der Menge der bisher gespeicherten Aggregationswerte als Aggregationspartner gewählt.

Begründung der logarithmischen Steigerung der Aggregationswahrscheinlichkeit
Bei der Aggregation wird stets ein bereits gespeicherter Wert als Aggregationspartner benötigt. Da diese zunächst angelegt werden müssen, sollte die Aggregation zu Beginn klein gehalten werden, bzw. gänzlich untersagt sein. Im späteren Verlauf sollte durch eine erhöhte Aggregation die Belastung des Netzwerks beim Auslesen minimiert werden. Der Verlauf der logarithmischen Kurve bietet sich an, da sie zu Beginn relativ schnell wächst und sich im weiteren Verlauf einem Maximalwert annähert. Zur weiteren Anpassung an die Problematik wird die logarithmische Kurve durch Stufen approximiert.

4.5.2.3.2 Abhängigkeiten
Die Anzahl der Stufen legt die Stufengröße fest. Durch die Stufengröße und die aktuelle Anzahl an parallel gehaltenen Aggregationswerten wird die Aggregationswahrscheinlichkeit bestimmt. Die Stufengröße sei durch s definiert.

$$s = \frac{n}{s_a},\ n, s_a \in \mathbb{N} \tag{4.10}$$

Hierbei ist s_a die Anzahl der Stufen und n die Anzahl der gesendeten Pakete. Ein optimaler Wert für s_a soll durch die Simulation ermittelt werden.

Zur Berechnung der Aggregationswahrscheinlichkeit wird ein Divisor benötigt, der die aktuelle Stufe bestimmt. Sei d_i dieser Divisor und M_i die Anzahl der zum Zeitpunkt des Eintreffens von Paket i parallel gespeicherten Aggregationswerte.

$$d_i = \left\lfloor \frac{M_i}{s} \right\rfloor,\ d_i, M_i, a \in \mathbb{N} \tag{4.11}$$

Mittels 4.9 kann nun die Aggregationswahrscheinlichkeit für das Paket i bestimmt werden.

4.5.2.3.3 Simulationserkenntnisse
Die Tabelle 4.2 stellt die Erkenntnisse aus der Simulation zu Ansatz 2 dar. Die Stufenanzahl s_a wurde zunächst festgesetzt, der Redundanzfaktor R hingegen variiert, um die Auswirkungen des Speicherverhaltens zu untersuchen. Die Ergebnissen hieraus lieferten einen festen Wert für R, mit dem anschließend unterschiedliche Konfigurationen von s_a untersucht werden konnten. Der Decodiererfolg ist als Konfidenzintervall mit zugehöriger Anzahl an überlebenden Sensorknoten dargestellt.

Tabelle 4.2: Darstellung der Erkenntnisse aus der Simulation zu Ansatz 2

Redundanzfaktor R	Stufenanzahl s_a	Erkenntnis
0.25	8	viele Koeffizientenvektoren mit Grad > 1 Koeffizienten-Grad-Obergrenze: 3-5 Erfolgreich decodierte Pakete: $[78,3977;79,3022]$ bei 10 überlebenden Knoten
0.5	8	sehr viele Koeffizientenvektoren mit Grad > 1 Koeffizienten-Grad-Obergrenze: stark variierend, 5-10 Erfolgreich decodierte Pakete: $[0,3167;0,4033]$ bei 10 überlebenden Knoten
0.75	8	extrem viele Koeffizientenvektoren mit Grad » 1 Koeffizienten-Grad-Obergrenze: 7-10 Erfolgreich decodierte Pakete: $[3,2177;3,5623]$ bei 10 überlebenden Knoten
0.5	4	viele Koeffizientenvektoren mit Grad > 1 Koeffizientengrad-Obergrenze: 4-5 Erfolgreich decodierte Pakete: $[96,6137;97,0063]$ bei 7 überlebenden Knoten
0.5	16	Koeffizienten-Grad-Obergrenze: 9-11 Erfolgreich decodierte Pakete: $[0,2007;0,2793]$ bei 10 überlebenden Knoten
0.5	6	hinreichend viele Koeffizientenvektoren mit Grad > 1 Koeffizienten-Grad-Obergrenze: 5-6 Erfolgreich decodierte Pakete: $[97,6649;98,2350]$ bei 9 überlebenden Knoten

Es hat sich gezeigt, dass sich die besten Ergebnisse bei einem Redundanzfaktor von $R = 0.5$ und einer Stufenanzahl von $s_a = 6$ eingestellt haben.

4.5.2.3.4 Vorteile Die Steigerung des Aggregationsverhaltens stellt, bei richtiger Konfiguration, eine gute Möglichkeit dar, um die Aggregation sowie den Decodierungsprozess zu sichern. Hierbei regelt sich das System durch die Angabe der Anzahl an Stufen selbst und könnte somit auf eine beliebige Anzahl an gesendeten Paketen reagieren.

4.5.2.3.5 Nachteile Die Abhängigkeiten beschränken das System auf die Kenntnis der insgesamt zusendenden Pakete n eines Quell-Sensorknotens. Diese Information ist jedoch zumeist nicht als gegeben anzusehen. Fernerhin unterliegt der Decodierungserfolg einer extrem starken Schwankung, wie in Abbildung 4.6 zu sehen ist.

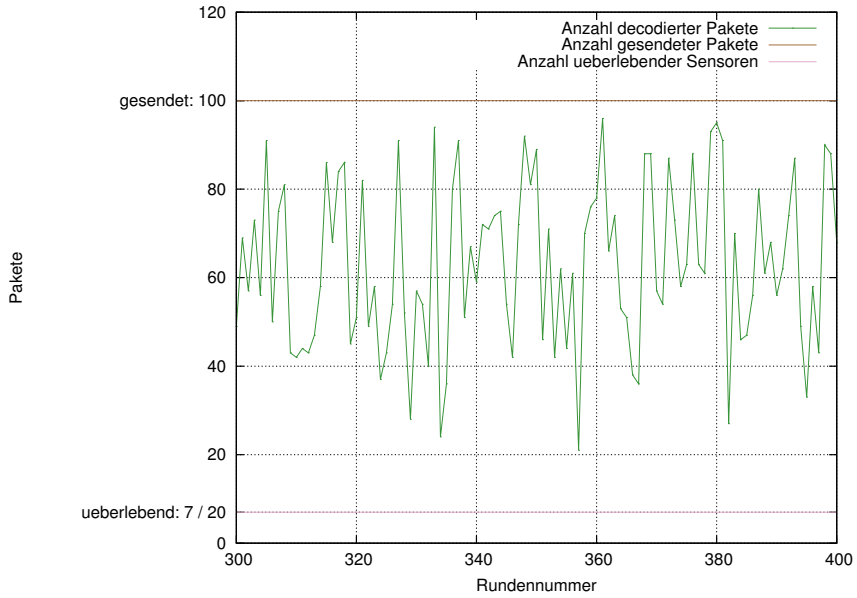

Abbildung 4.6: Anzahl der erfolgreich decodierten Werte zu der Gesamtanzahl an gesendeten Werten bei 7 überlebenden Sensorknoten aus 20 Empfänger, $R = 0.5$, $s_a = 6$

4.5.2.4 Ansatz 3 - Erst die Speicherung, dann die Aggregation

Unter der Betrachtung der Hardware eines TmoteSky Sensorknotens[19] stößt man auf die Problematik des stark begrenzten Speichers. Der Speicher des Sensorknotens sollte demnach optimal ausgenutzt werden. Durch die Erkenntnisse aus Ansatz 1 und Ansatz 2 soll dieser Ansatz nun eine hardwarenahe Konfiguration bieten, die zudem ohne große Kenntnisse über den Verlauf des Lebenszyklus einzelner Sensorknoten auskommt.

4.5.2.4.1 Aufbau

Es werden während der Initialisierung des Sensorknotens ein fester Wert, der die Anzahl an parallel gehaltenen Aggregationswerten einschränkt, und ein fester Wert zur Begrenzung des maximalen Grads je Koeffizientenvektor definiert. Über die Verteilung der Grade der Koeffizientenvektoren kann eine Aussage über die Decodierungseigenschaften des Systems getroffen werden [37, 44, 66]. Die Festlegung dieser Werte soll aus den Simulationen gewonnen werden. Es gilt weiterhin die Einschränkung, dass der Empfängerknoten mittels des

Redundanzfaktors R aus (4.5) zufällig entscheidet, ob er ein eingehendes Paket speichert. Entschließt sich der Empfängerknoten hierzu, wird er zunächst jedes Paket als Klartextpaket speichern. Sobald die Anzahl an gespeicherten Klartextpaketen gleich der zuvor definierten Anzahl an Aggregationswerten ist, schaltet der Sensorknoten um und aggregiert fortan jedes zu speichernde Paket. Dabei wählt er zufällig einen Aggregationswert[11] aus der Menge der bisher gespeicherten Aggregationswerte als Aggregationspartner.

Kritischer Speicherzustand

Um die erfolgreiche Decodierung zu sichern, überwacht jeder Sensorknoten seinen Speicher. Haben eine bestimmte Anzahl an Koeffizientenvektoren ihren maximalen Grad erreicht, sollte der Sensorknoten diese Speichergrenze an den Systemadministrator senden und sich selbst in einen Schlafmodus versetzen, um Ressourcen zu sparen. Sensorknoten, die sich selbst in den Schlafmodus versetzt haben, werden keine Pakete mehr speichern, reagieren aber weiterhin auf Abfragenachrichten[12].

4.5.2.4.2 Abhängigkeiten

Da dieser Ansatz auf Erkenntnissen aus Ansatz 1 und Ansatz 2 basiert, stellt er lediglich eine Konfigurationsempfehlung zum Einsatz des Systems dar. Die Abhängigkeiten konnten den vorangehenden Simulationen entnommen werden.

Zur Simulation wurde die Kenntnis über die Anzahl der zu sendenden Daten ausgenutzt. Hierdurch konnte in Abhängigkeit der Anzahl zu sendender Daten die Menge an parallel zuhaltenden Aggregationswerten beschränkt werden. Sei A_P die Anzahl an parallel zuhaltenden Aggregationswerten, wobei P die Anzahl der zu sendenden Pakete ist. x_P ist ein frei wählbarer Divisor, der durch die Simulation genauer eingeschränkt werden soll. Dann ergibt sich die Formel (4.12):

$$A_P = \frac{P}{x_P},\ P, x_P \in \mathbb{N} \tag{4.12}$$

Aus der Speicherauslastung und dem Decodierungserfolg kann hierüber später rückwirkend eine Aussage über die maximale Anzahl an speicherbaren Paketen gemacht werden.

Zur Begrenzung des Speichers und gleichzeitiger Sicherung des Decodierprozesses werden ein maximaler Grad je Koeffizientenvektor d_C und eine maximale Anzahl an Koeffizienten mit maximalem Grad d_{Max} definiert. Für die Festlegung von d_C und d_{Max} wird auf Grundlage der Simulationserkenntnissen eine Empfehlung ausgesprochen.

4.5.2.4.3 Simulationserkenntnisse

Während der Simulation hat sich gezeigt, dass sich für $x_P = 5$ sehr gute Decodierungs-

[11] Als Aggregationswert gelten auch die zuvor gespeicherten Klartextpakete.

[12] Nachrichten, die einen Sensorknoten veranlassen, sein aktuelles Speicherabbild an den Nachrichtensteller zu senden.

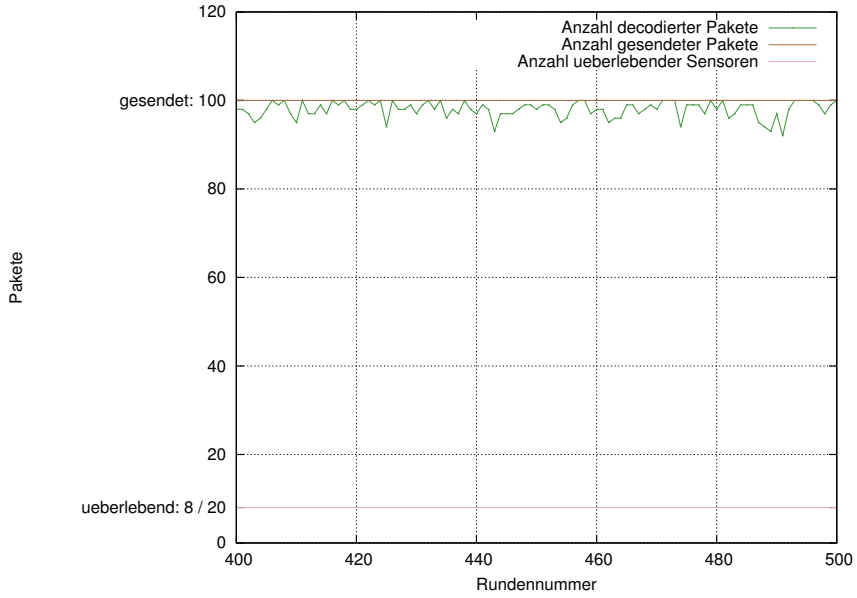

Abbildung 4.7: Menge der decodierten Daten bei 8 überlebenden Sensorknoten aus 20 Empfängern, bei einer Konfiguration nach Ansatz 3 mit $x_P = 5$, $d_C = 5$ und $d_{Max} = \frac{1}{10} * A_P$

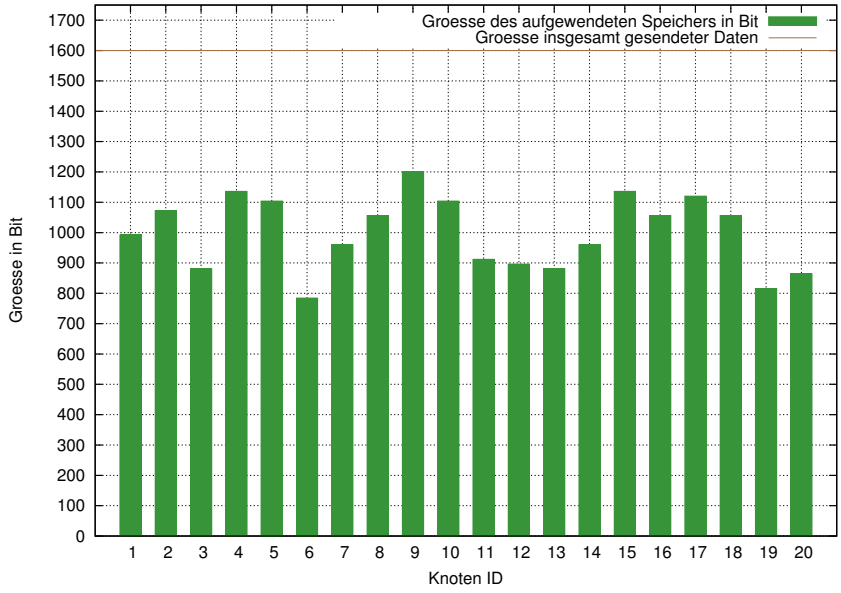

Abbildung 4.8: Speicherausnutzung der Sensorknoten in Bit

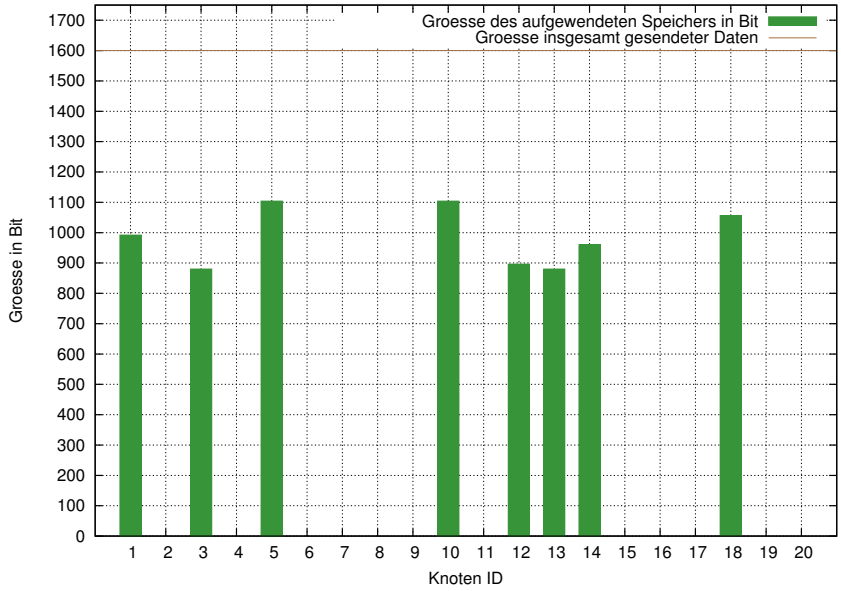

Abbildung 4.9: Speicherausnutzung der überlebenden Sensorknoten in Bit

ergebnisse eingestellt haben. Hierbei wurden überwiegend Koeffizientenvektoren mit einem maximalen Grad von 5 produziert. Die Speicherauslastung betrug hierbei ca. 65-70% der Menge an insgesamt gesendeten Daten (Vergleiche dazu Abbildung 4.8 und Abbildung 4.9) Somit können Empfehlungen für die Werte wie folgt ausgesprochen werden.

$$d_C = 5$$
$$d_{Max} = \frac{1}{10} * A_P$$

4.5.2.4.4 Vorteile

Da dieser Ansatz klar auf Hardwareeinschränkungen ausgelegt wurde, kann er direkt auf ein reales Sensornetzwerk übertragen werden. Zudem zeigte er, dass sehr gute Decodierungseigenschaften mit einer relativ hohen Speicherausnutzung vereinbar sind.

4.5.2.4.5 Nachteile

Auch dieser Ansatz ist prinzipiell abhängig von der Anzahl zusendender Daten. Jedoch kann diese Abhängigkeit umgekehrt werden und eine fixe Größe für A_P implementiert werden, aus der die maximale Anzahl an zusendenden Daten abgeleitet werden kann.

4.5.2.4.6 Alternatives Verhalten der Sensorknoten bei kritischem Speicherzustand

Sensorknoten, deren Speicher einen kritischen Speicherzustand erreichen, könnten anfan-

gen, die ältesten Daten zu überschreiben. Durch die Aggregation der Daten ist eine Ermittlung des Datums eines Aggregationswertes jedoch problematisch, da Werte aus verschiedenen Zeitpunkten eingeflossen sind. Es müssten Verfahren zur Bestimmung des Alters der Aggregationswerte entwickelt werden.

Kapitel 5

Implementierung in TinyOS

5.1 Motivation

Als de facto Standard Betriebssystem im Bereich der Sensorknoten hat sich TinyOS herausgestellt. Durch die spezielle Ausrichtung auf die Anforderungen der Sensorknoten, in Betrachtung der Ressourcen und Lebensdauer sowie der unterschiedlichen Hardwareplattformen, bietet TinyOS eine optimale Grundlage zur Entwicklung von Software für drahtlose Sensornetzwerke. Zudem wurde die C-Erweiterung nesC [58] extra für TinyOS geschrieben um den Anforderungen noch mehr gerecht zu werden. TinyOS bietet die besten Voraussetzungen, um eigene Implementierungen in Sensornetzwerken zu realisieren.

Die folgende Implementationsbeschreibung umfasst den generellen Aufbau der TinyOS Applikation, die vom Autor dieser Arbeit entwickelt wurde. Beispielhaft sollen nun einige Quellcode-Abschnitte aufgezeigt und erläutert werden. Ein abschließender Abschnitt geht auf die Einschränkungen der Implementierung ein.

5.2 Aufbau

Der Aufbau der Implementierung richtet sich stark nach den Vorgaben, die durch die Entwicklungsumgebung von TinyOS gegeben sind. Durch das Verbinden von angebotenen Interfaces mit der eigenen Anwendung und der Implementierung der benötigten Events, konnten einige vorhandene Module genutzt werden. Die verwendete Zielplattform ist Telos-B [19]. Die Abbildungen 5.1 und 5.2 zeigen den allgemeinen Aufbau und die Abhängigkeiten der Anwendung.

Die Anwendung unterteilt sich in zwei unterschiedliche Implementierungen. Die primäre Implementierung ist **TinyDSC**. Diese stellt die Logik der Sensorknoten inklusive des Speicherungsalgorithmus und der Kommunikation zwischen Sender und Empfänger bereit. **TinyDSSinkC** fungiert als eine Basisstation für den Administrator. Der Sensorknoten, der

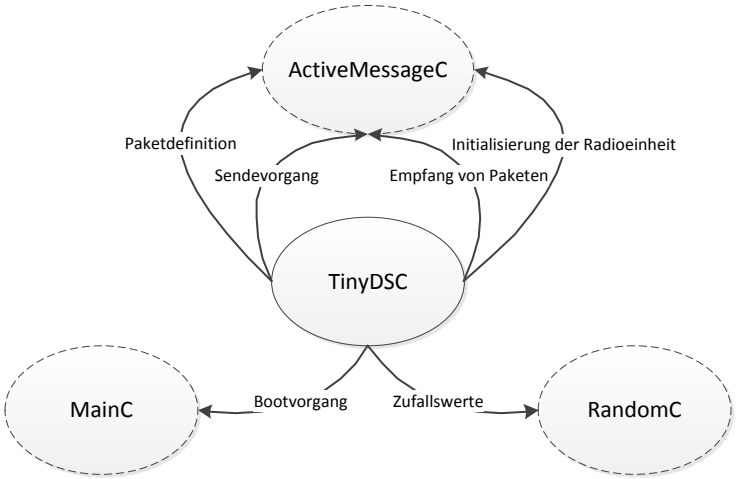

Abbildung 5.1: Abhängigkeiten des TinyDSC Modules

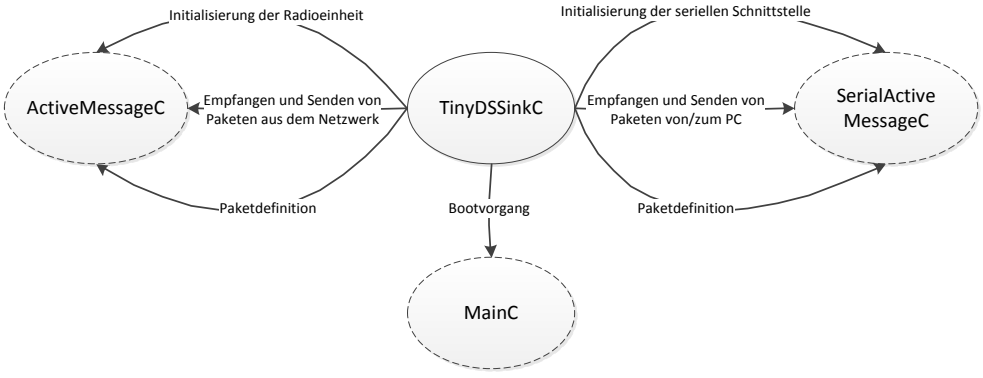

Abbildung 5.2: Abhängigkeiten des TinyDSSinkC Modules

diese Implementierung erhält kann über eine Java-Applikation angesprochen werden. Abbildung 5.3 zeigt das prinzipielle Zusammenspiel zwischen der Java-Applikation und der Basisstation.

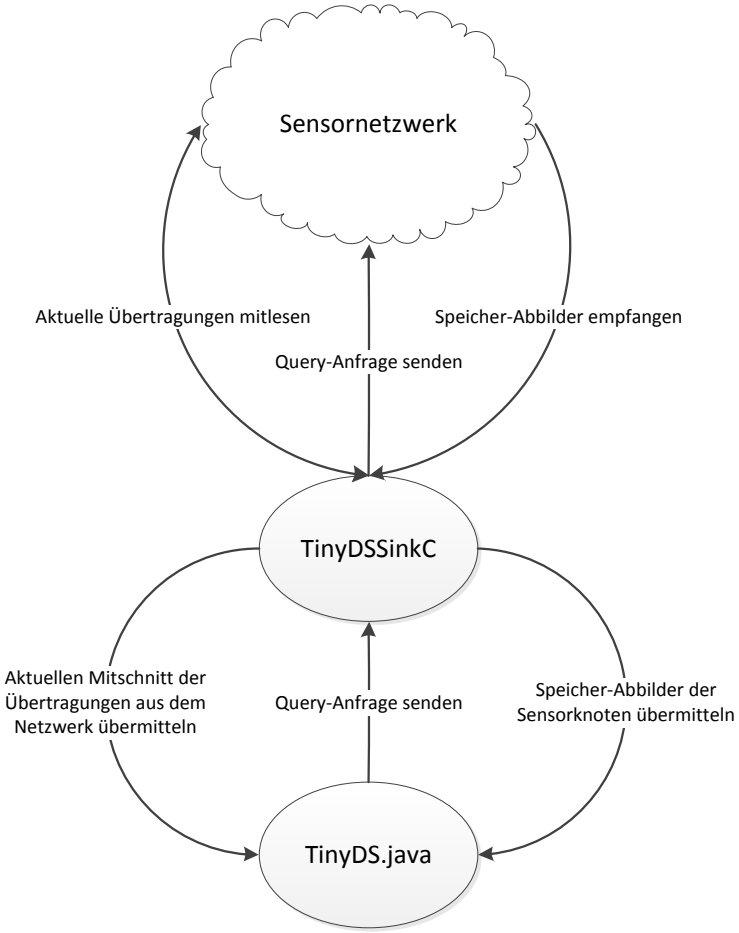

Abbildung 5.3: Verbindungen zwischen der Java-Applikation und der Basisstation sowie dem Sensornetzwerk.

5.3 Implementierung

Jeder Sensorknoten, der in einem Sensornetzwerk eingesetzt werden soll, wird mit der TinyDSC Applikation programmiert. Mittels einer Variablen kann im Vorfelde bestimmt werden, ob der Sensorknoten als Sender oder als Empfänger aggieren soll. Das folgende Listing zeigt die Implementierung des Sendevorgangs in nesC / TinyOS.

```
event void Read.readDone(error_t result, uint16_t data) {
    // Ist noch ein Sendevorgang aktiv,
    // wird nichts unternommen
    if(!sendBusy) {
        tinyds_msg_t* dspacket = (tinyds_msg_t*)
            (call Packet.getPayload(&packet, sizeof (tinyds_msg_t)));
        // LED einschalten um senden zu symbolisieren
        call Leds.led2On();
        // Nachricht zusammensetzen
        dspacket->packetId = nextPacketId;
        dspacket->nodeId = TOS_NODE_ID;
        dspacket->value = data;
        // Nachricht senden
        if (call AMSend.send(AM_BROADCAST_ADDR,
            &packet, sizeof (tinyds_msg_t)) == SUCCESS) {
            sendBusy = TRUE;
        }
        // Paket ID erhoehen
        nextPacketId++;
    }
}

event void AMSend.sendDone(message_t* msg, error_t error) {
    // Pruefen, ob es unser Paket war,
    // das gesendet wurde
    if (&packet == msg) {
        sendBusy = FALSE;
        // Senden abgeschlossen -> LED aus
        call Leds.led2Off();
    }
}
```

Das Paket wird mit dem eigens definierten Typ *tinyds_msg_t* erstellt und über das TinyOS Interface ActiveMessageC (hier *AMSend*) versendet. Die Paketnummer (*packetId*) wird nach dem Absenden erhöht. Status-LEDs signalisieren den Sendevorgang. Das folgende Listing zeigt die Definition des Typs *tinyds_msg_t*.

```
typedef nx_struct tinyds_msg {
    // Aktuelle Paketnummer (vergeben vom Sender)
    nx_uint16_t packetId;
    // ID des Senders
```

```
    nx_uint16_t nodeId;
    // Nutzdaten
    nx_uint16_t value;
} tinyds_msg_t;
```

Der Empfangsprozess wurde, wie im Ansatz 3 in 4.5.2.4 simuliert, 1:1 in TinyOS übertragen. In den folgenden Listings sind Auszüge der wesentlichen Punkte der Implementierung dargestellt. Diese werden jeweils nachfolgend erläutert.

```
// Zufaelliges Entscheiden, ob eingehendes Paket
// gepeichert werden soll
if (call Random.rand32() % 2 == 0) { ... }
```

Zunächst wird entschieden, ob das eingehende Paket gespeichert werden soll. Hierzu wird die in TinyOS integrierte Schnittstelle RandomC (hier *Random*) verwendet.

```
// Entscheiden, ob eingehendes Paket aggregiert werden soll
if (aggregationCount >= TINYDS_MAX_AGGR_VALUES) { ... }
```

Wurde entschieden, dass das eingehende Paket gespeichert wird, prüft der Sensorknoten, wie viele Pakete er bereits parallel im Speicher hat. Übersteigt diese Anzahl eine definierte Grenze, werden alle nachfolgenden Pakete auf die bestehenden Pakete aggregiert. Die Grenze der Anzahl der parallel gehaltenen Pakete kann, wie in Abschnitt 4.5.2.4 beschrieben, mit Hilfe der Simulationsapplikation bestimmt werden.

```
// Zufaelliges Paket waehlen
uint16_t aggrTo = call Random.rand32() % aggregationCount;
```

Im Falle der Aggregation wird zufällig ein Aggregationspartner aus den bereits gespeicherten Paketen ausgewählt.

```
// Aggregation
tinyDSMemory[aggrTo].coeff[tinyDSMemory[aggrTo].coeffIndex]
    = dspacket->packetId;
tinyDSMemory[aggrTo].coeffIndex
    = tinyDSMemory[aggrTo].coeffIndex + 1;
tinyDSMemory[aggrTo].value
    = tinyDSMemory[aggrTo].value ^ dspacket->value;
```

Auf dieses Paket wird nun das eingehende Paket aggregiert und die Paketnummer dem Koeffizientvektor hinzugefügt.

```
// Paket fuer sich speichern
tinyDSMemory[aggregationCount].coeff[0]
    = dspacket->packetId;
tinyDSMemory[aggregationCount].coeffIndex = 0;
```

```
    tinyDSMemory[aggregationCount].value = dspacket->value;
    aggregationCount++;
```

Entscheidet der Zufall, dass ein eingehendes Paket gespeichert werden soll und befinden sich weniger, als durch die Grenze angegeben, parallel gehaltene Pakete im Speicher des Sensorknotens, so trägt der Sensorknoten das eingehende Paket als neuen Eintrag mit neuem Koeffizientenvektor in den Speicher ein.

Die SinkNode kommuniziert zum Auslesen der Daten aus dem Netzwerk mit einer JavaApplikation. Die JavaApplikation steuert den Sensorknoten, auf dem die TinyDSSinkC Applikation installiert wurde. Die TinyDSSinkC Applikation leitet die Pakete aus dem Netzwerk an die JavaApplikation und die Pakete der JavaApplikation an das Netzwerk weiter. Die JavaApplikation kann, nach erfolgtem Auslesen der Daten aus dem Netzwerk, diese decodieren. Das folgende Listing zeigt den Decodierprozess in Java. Dieser basiert auf der Linearkombination der einzelnen Pakete.

```
public Map<Integer, Integer> decodeData(
    Set<Map<CoVektor, Integer>> encodedData) {
    // Bisher gefundene kodierte und dekodierte Pakete löschen
    encodedPackets = new HashMap<CoVektor, Integer>();
    decodedPackets = new HashMap<Integer, Integer>();

    // Nach und nach versuchen die Pakete zu dekodieren
    for(Map<CoVektor, Integer> memoryDump : encodedData) {
        for(Map.Entry<CoVektor, Integer> packet
            : memoryDump.entrySet()) {
            // akuteller Koeffizientenvektor und kodierter Wert
            CoVektor coefficients = new CoVektor(packet.getKey());
            Integer encodedValue = packet.getValue();

            // Der Koeffizientenvektor hat mehr als einen Eintrag,
            // Eintrag muss dekodiert werden
            if(coefficients.size() > 1) {
                decodePacket(coefficients, encodedValue);
            }
            // Koeffizient hat nur einen Eintrag,
            // das Paket ist also nicht codiert
            else if(coefficients.size() == 1){
                decodedPackets.put(coefficients.first(), encodedValue);
                // Prüfe den kodierten Puffer auf dekodierte Pakete
                // und entferne sie
                Map<CoVektor, Integer> currentEncodedPackets
```

```
                = new HashMap<CoVektor, Integer>(encodedPackets);
            for(Map.Entry<CoVektor, Integer> encPacket
                : currentEncodedPackets.entrySet()) {
                if(encPacket.getKey().size() == 1) {
                    if(!decodedPackets.containsKey(
                        encPacket.getKey())) {
                        decodedPackets.put(encPacket.getKey().first(),
                            encPacket.getValue());
                    }
                    encodedPackets.remove(encPacket.getKey());
                }
            }
            // Versuchen alle noch nicht decodierten
            // Paket im Speicher zu dekodieren
            currentEncodedPackets
                = new HashMap<CoVektor, Integer>(encodedPackets);
            for(Map.Entry<CoVektor, Integer> encPacket :
                currentEncodedPackets.entrySet()) {
                decodePacket(new CoVektor(encPacket.getKey()),
                    encPacket.getValue());
            }
        }
    }
    return new HashMap<Integer, Integer>(decodedPackets);
}
```

5.4 Einschränkungen der Implementierung

Gegenüber der Simulationsumgebung musste die Implementierung mit einigen Einschränkungen vollzogen werden. Durch die knappen Speicherressourcen und das Fehlen einer dynamischen Allokierung von Speicher auf den Sensorknoten muss während der Initialisierungsphase ein fester Speicherbereich für die Aggregationswerte und die Koeffizientenvektoren angelegt werden. Bei der Festlegung dieser Werte kann auf die Erkenntnisse aus dem Ansatz 3 aus 4.5.2.4 zurückgegriffen werden.

Sämtlicher Quellcode befindet sich auf der beiliegenden DVD.

Kapitel 6

Fazit

In dieser Arbeit wurde ein verteilter Datenspeicher in Sensornetzwerken mittels Network Coding realisiert. Dabei wurden die wenigen Ressourcen, wie der eingeschränkte Speicherplatz und die begrenzte Energieversorgung, ebenso wie die hohe Ausfallwahrscheinlichkeit von Sensorknoten betrachtet. Der stark Ressourcen beanspruchende, drahtlose Übertragungskanal hatte einen großen Einfluss auf die betrachteten Ansätze in dieser Arbeit. Als Grundlage kam eine ausgewählte Menge der Grundpfeiler zur Realisierung von Software für drahtlose Sensornetzwerke des WSAN4CIP zum Einsatz. Diese waren *Persistenz*, *Verteilung über die drahtlose Verbindung*, *Leistungsfähigkeit von Abfragen* und *Netzwerktopologie*. Die restlichen Grundpfeiler (*Sicherheit*, *Einschränkung beim Speichern* und *Desaster Radius*) erlagen einer klaren Ausgrenzung.

Aus diesen Grundpfeilern sind drei Ansätze zur verteilten Speicherung und Aggregation mittels Network Coding in einem drahtlosen Sensornetzwerk entstanden. Der erste Ansatz lässt zuvor festgelegte Wahrscheinlichkeiten über die Speicherung und die Aggregation entscheiden. Die Ergebnisse waren bereits gut, erschienen jedoch nicht flexibel genug. Im zweiten Ansatz wurde eine Optimierung des Speicherverhaltens erwägt. Über die Anzahl der bereits vorhandenen Aggregationswerte pro Sensorknoten konnte der Sensorknoten die Aggregationswahrscheinlichkeit anpassen. So steigerte sich die Aggregationwahrscheinlichkeit mit der Anzahl an bereits angelegten Aggregationswerten. Dieses Verfahren wies jedoch, trotz einer Beschränkung der Steigerung der Aggregationswahrscheinlichkeit, keine guten Decodierungseigenschaften auf. Aus den Erkenntnissen der ersten zwei Ansätze ist ein dritter Ansatz entstanden, der als Empfehlung zur Konfiguration eines verteilten Datenspeichers in drahtlosen Sensornetzwerken anzusehen ist. Bei diesem letzten Ansatz stand eine Fokusierung auf die Hardwareeinschränkungen, insbesondere des Speichers, im Vordergrund. Dabei konnte eine gute Verteilung der Daten mit sehr guten Decodierungseigenschaften verbunden werden. Als besonderen Punkt hat sich gezeigt, dass die Anzahl der Empfänger nicht entscheidend ist, sofern die Anzahl an überlebenden Sensorknoten dieselbe ist.

Zur Erkenntnisgewinnung wurde eine Simulationsapplikation entwickelt. Diese wurde in

der Programmiersprache JAVA geschrieben und kann leicht auf jede denkbare Konfiguration dieser Arbeit angepasst werden. Die Ergebnisse der Simulationen speichert die Applikation als Text- und Bilddateien. Hierüber kann eine Auswertung der Simulation erfolgen.

Eine weitere Implementierung wurde in TinyOS erstellt. Diese basiert auf den Erkenntnissen des dritten Ansatzes. Sie zeigt, dass die theoretischen Annahmen auf reale Hardwarekomponenten übertragbar sind.

Kapitel 7

Ausblick

In der vorliegenden Arbeit wurden viele Aspekte zur Realisierung eines verteilten Datenspeichers in Sensornetzwerken beachtet. Es konnten jedoch nicht alle Problematiken und Verfahren genauer betrachtet und behandelt werden. Es besteht somit weiterer Forschungsbedarf. Die folgenden Themen sollen einen kurzen Überblick über mögliche Fortführungen dieser Arbeit geben.

- Aufbauend auf den Erkenntnissen dieser Arbeit sollten die ausgelassenen Aspekte des WSAN4CIP Anforderungskatalogs weiter betrachtet und eingearbeitet werden. Die Punkte *Sicherheit*, *Desaster Radius* und *Einschränkungen beim Speichern* sind von großer Bedeutung und dürfen auf Dauer nicht vernachlässigt werden.

- Es könnte außerdem untersucht werden, ob die Replikation den Vorteil gegenüber Erasure Coding erbringt. In Peer-to-Peer-Systemen kann gezeigt werden, dass Erasure Coding weniger Redundanz benötigt als die einfache Replikation [55, Kapitel 11.7.2]. Diese Erkenntnis könnte auf diese Arbeit übertragen werden, um weitere Gründe zur Nutzung eines verteilten Datenspeichers in drahtlosen Sensornetzwerken mittels Network Coding zu liefern.

- Weiterhin sollte eine parallele Speicherung von Paketen, inklusive der Aggregation, bei mehreren Sendern in einem auf diese Arbeit aufbauenden Projekt analysiert werden.

- Mit dem Abschnitt 4.5.2.4.6 wurde bereits ein weiteres Thema für eine aufbauende Arbeit aufgeführt. Hierbei müsste ein alternatives Verhalten der Sensorknoten bei einem kritischen Speicherzustand untersucht werden. Dies ist ebenfalls durch den Punkt *Einschränkungen beim Speicher* aus den Grundpfeilern des WSAN4CIP Anforderungskatalogs abgedeckt.

- Zur intensiven Betrachtung der decodierten Daten der überlebenden Sensorknoten nach einem Desaster wäre der Einsatz einer Kurvendiskussion möglich. Hierzu müsste

ein Algorithmus gefunden werden, der eine Formel aus den decodierten Daten generiert. Die Durchführung der Kurvendiskussion könnte nun auf der entstehenden Formel erfolgen. Eine Softwareapplikation wäre auf diese Weise in der Lage, extreme Veränderungen im Verlauf der Daten zu finden. Durch das Zusammenspiel zwischen dem Sensornetzwerk, einigen SinkNodes und einer Software-Applikation zur Durchführung der Kurvendiskussion könnte ein automatisiertes Warnsystem entstehen.

- Die empirischen Studien dieser Arbeit basieren auf einem Ein-Hop-Netzwerk. Um die Sicherheit weiter auszubauen, beispielsweise um die Daten über den Desaster Radius hinaus zu verteilen, können die Ergebnisse dieser Arbeit mit einigen Anpassungen auf ein Multi-Hop-Netzwerk übertragen werden. Hierzu müssten die Pakete mit der ID des Quell-Sensorknotens und einem Hop-Zähler versehen werden. Die Empfänger-Sensorknoten könnten nun für jedes Paket anhand des Hop-Zählers entscheiden, ob sie es weitersenden, abspeichern oder verwerfen. Die Ergebnisse dieser Arbeit bleiben erhalten, solange der Administrator die entsprechende Anzahl an überlebenden Sensorknoten abfragt, die die Erkenntnisse hervorgebracht haben. Entscheidend ist, dass jeder Empfänger-Sensorknoten gegenüber den anderen Empfänger-Sensorknoten autonom über das Speicherverhalten entscheidet.

Anhang A

DVD

A.1 Quellcode

Sämtliche Quellcodes zur Simulation und zur Implementierung in TinyOS befinden sich auf der beiliegenden DVD.

A.2 Simulationsausgabe

Während der Simulationen wurden mehr als 99.000 Bilder erzeugt. Diese konnten leider nicht alle im gedruckten Format als Quelle der Erkenntnisse in diese Arbeit eingebunden werden. Sämtliche Simulationsausgaben, sowohl die Log-Texte wie auch die Bilddateien, befinden sich auf der beiliegenden DVD. Ebenfalls ist die Berechnung der Konfidenzintervalle in einer Excel-Datei enthalten.

A.3 Strukturierung des Inhalts der DVD

Die DVD ist durch Ordner strukturiert. Es befindet sich eine *README*-Datei im Wurzelverzeichnis, in der zusätzlich zu dieser Auflistung sämtliche Verzeichnisse noch einmal beschrieben sind.

```
DVD/
 |_ README.txt            Informationen zur DVD
 |_ /Bilder               selbst erstellte Grafiken als PDFs
 |_ /Quellcode            Quellcodes der entwickelten Applikationen
     |_ /NodeSimu
         |_ /doc          Dokumentation der Simulationsapplikation
         |_ /src          Quellcodes der Simulationsapplikation
     |_ /TinyDS
         |_ /doc          Dokumentation der Simulationsapplikation
         |_ /TinyDS       Quellcodes der TinyOS-Applikation der
                          Sensorknoten im Netzwerk
         |_ /TinyDSSink   Quellcodes der TinyOS-Applikation
                          der SinkNode
 |_ /Simulationsergebnisse
    |_ /Auswertungen - Konfidenzintervalle.xlsx
                          Excel 2010 Datei mit den Berechnungen der
                          Konfidenzintervalle
    |_ /images           erzeugte Grafiken, zur Auswertung der
                          Messergebnisse der Simulation
    |_ /logs             erzeugte Log-Dateien der Messwerte
                          der Simulation
```

Glossar

Aggregation Eine Aggregation ist die Verknüpfung von zwei oder mehr Daten zu einem Datum. In der Informatik wird die Verknüpfung häufig mittels XOR vorgenommen.

bps Kurzform für bit per second (Bits pro Sekunde), Größenangabe der Menge an Bits, die pro Sekunde durch eine Datenleitung fließen können.

Broadcast Art eines Sendevorgangs, bei dem der Sender seine Daten ohne direktes Ziel an alle Empfänger im Senderadius sendet.

Byte Größeneinheit für Datenspeicher, 1 Byte = 8 Bit, 1 KB (KiloByte) = 1024 Byte

Cluster Ansammlung von vielen gleichartigen Objekten, die miteinander vernetzt sind; Cluster-Computer haben ähnliche Hardware aber das gleiche Betriebssystem und sind mit einem Hochgeschwindigkeitsnetzwerk vernetzt. [55, S.34]

et al. lateinisch: et alii = und andere; Abkürzungsform bei der Angabe von mehr als zwei Autoren [40, S.219]

ID Kennung zur eindeutigen Identifikation eines Objektes

Kontamination Kontamination beschreibt eine Verseuchung durch Radioaktivität, chemische Stoffe oder Bakterien. [28, S. 507]

MHz Megahertz, von der Einheit Hertz, Taktrate/Vorgänge pro Sekunde und Mega, griechisch: mégas, hier für den Faktor 10^6

Netzwerktopologie Topologie beschreibt den Aufbau des Netzwerks

OpenSource engl. für quelloffen, eine Software, deren Quellcode öffentlich zugänglich ist

PC Kurzform für Personal Computer, häufig auch nur Computer

Peer Ein Teilnehmer in einem Peer-to-Peer Netzwerk

Quota Angabe des nutzbaren Speicherplatzes auf Systemen mit mehreren Benutzern

Redundanz Redundanz beschreibt das mehrfach Vorhandensein derselben Einheit in einem System.

Uptime Die Zeit, die ein Computer im Netzwerk erreichbar ist.

WSAN4CIP Wireless Sensor and Actuator Networks for the Protection of Critical Infrastructures

WSN Drahtloses Sensornetzwerk (von engl.: Wireless Sensor Network)

XOR Bitoperator, der zwei Bits miteinander verrechnet.

xor	0	1
0	0	1
1	1	0

Literaturverzeichnis

[1] ACEDANSKI, Szymon ; DEB, Supratim ; MÉDARD, Muriel ; KOETTER, Ralf: How good is random linear coding based distributed networked storage. In: *In NetCod*, 2005

[2] AGARWAL, A. ; CHARIKAR, M.: On the advantage of network coding for improving network throughput. In: *Information Theory Workshop, 2004. IEEE*, Oktober 2004, S. 247 – 249

[3] AGARWALA, S. ; PAUL, A. ; RAMACHANDRAN, U. ; SCHWAN, K.: e-SAFE: An Extensible, Secure and Fault Tolerant Storage System. In: *Self-Adaptive and Self-Organizing Systems, 2007. SASO '07. First International Conference on*, Juli 2007, S. 257 –268

[4] AHLSWEDE, R. ; CAI, Ning ; LI, S.-Y.R. ; YEUNG, R.W.: Network information flow. In: *Information Theory, IEEE Transactions on* 46 (2000), jul, Nr. 4, S. 1204 –1216. – ISSN 0018-9448

[5] AKYILDIZ, IF ; SU, W. ; SANKARASUBRAMANIAM, Y. ; CAYIRCI, E.: Wireless sensor networks: a survey. In: *Computer networks* 38 (2002), Nr. 4, S. 393–422

[6] ALY, S.A. ; KONG, Zhenning ; SOLJANIN, E.: Fountain Codes Based Distributed Storage Algorithms for Large-Scale Wireless Sensor Networks. In: *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, April 2008, S. 171 – 182

[7] AZIM, Tahir ; LEVIS, Philip: *Porting TinyOS 1.x Code to TinyOS 2.0*. Webseite. 10 2006. – URL `http://www.tinyos.net/tinyos-2.x/doc/html/porting.html`. – Zugriffsdatum: 23.06.2011

[8] BATTEN, Christopher ; BARR, Kenneth ; SARAF, Arvind ; TREPETIN, Stanley: *pStore: A Secure Peer-to-Peer Backup System*. 2001

[9] CAO, Ning ; WANG, Qian ; REN, Kui ; LOU, Wenjing: Distributed Storage Coding for Flexible and Efficient Data Dissemination and Retrieval in Wireless Sensor Networks. In: *Communications (ICC), 2010 IEEE International Conference on*, Mai 2010, S. 1 –5. – ISSN 1550-3607

[10] COX, On P. ; MURRAY, Christopher D. ; NOBLE, Brian D.: Pastiche: making backup cheap and easy. In: *In OSDI: Symposium on Operating Systems Design and Implementation*, 2002, S. 285–298

[11] CROSSBOW TECHNOLOGY, INC.: *Crossbow TELOSB Datenblatt*. Online PDF. – URL `http://www.willow.co.uk/TelosB_Datasheet.pdf`. – Zugriffsdatum: 04.07.2011

[12] DEB, S. ; MEDARD, M. ; CHOUTE, C.: On random network coding based information dissemination. In: *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, sept. 2005, S. 278 –282

[13] DIMAKIS, A.G. ; GODFREY, P.B. ; WU, Yunnan ; WAINWRIGHT, M.J. ; RAMCHANDRAN, K.: Network Coding for Distributed Storage Systems. In: *Information Theory, IEEE Transactions on* 56 (2010), September, Nr. 9, S. 4539 –4551. – ISSN 0018-9448

[14] DIMAKIS, A.G. ; PRABHAKARAN, V. ; RAMCHANDRAN, K.: Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes. In: *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, April 2005, S. 111 – 117

[15] DIMAKIS, A.G. ; PRABHAKARAN, V. ; RAMCHANDRAN, K.: Decentralized erasure codes for distributed networked storage. In: *Information Theory, IEEE Transactions on* 52 (2006), Juni, Nr. 6, S. 2809 – 2816. – ISSN 0018-9448

[16] DIMAKIS, A.G. ; PRABHAKARAN, V. ; RAMCHANDRAN, K.: Distributed Fountain Codes for Networked Storage. In: *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on* Bd. 5, Mai 2006, S. V. – ISSN 1520-6149

[17] DIMAKIS, Alexandros G. ; RAMCHANDRAN, Kannan ; WU, Yunnan ; SUH, Changho: A Survey on Network Codes for Distributed Storage. In: *CoRR* abs/1004.4438 (2010). – URL `http://arxiv.org/abs/1004.4438`

[18] FASOLO, E. ; ROSSI, M. ; WIDMER, J. ; ZORZI, M.: In-network aggregation techniques for wireless sensor networks: a survey. In: *Wireless Communications, IEEE* 14 (2007), april, Nr. 2, S. 70 –87. – ISSN 1536-1284

[19] FATLAND, Rob: *Tmote Sky specifications*. Webseite. Oktober 2007. – URL `http://robfatland.net/seamonster/index.php?title=Mote_Specifications#Tmote_Sky_specifications`. – Zugriffsdatum: 18.06.2011

[20] GAY, David ; LEVIS, Philip ; CULLER, David ; BREWER, Eric: *Introduction nesC 1.1 Language Reference Manual*. `http://www.tinyos.net/tinyos-1.x/doc/nesc/ref.pdf`. 2003

[21] GIRAO, Joao ; WESTHOFF, Dirk ; MYKLETUN, Einar ; ARAKI, Toshinori: TinyPEDS: Tiny persistent encrypted data storage in asynchronous wireless sensor networks. In: *Ad Hoc Netw.* 5 (2007), September, S. 1073–1089. – ISSN 1570-8705

[22] GÄRNER, Sven: *Konzeption und Realisierung eines synchronisierenden Peer-to-Peer Dateisystems*, Hamburg, Hochschule für Angewandte Wissenschaften, Fak. Technik und Informatik, Dep. Informatik, Diplomarbeit, 2009

[23] HAAS, Z.J. ; LIANG, B.: Ad hoc mobility management with uniform quorum systems. In: *Networking, IEEE/ACM Transactions on* 7 (1999), April, Nr. 2, S. 228 –240. – ISSN 1063-6692

[24] HESSLER, Alban: *Concepts for providing dependable and secure elementary services in WSANs*. Deliverable D4.1. April 2010. – URL `http://www.wsan4cip.eu/fileadmin/public_documents/Deliverables/D4.1_Concepts-for-secure-services_WSAN4CIP-225186.pdf`. – Zugriffsdatum: 19.06.2011

[25] HO, Tracey ; KOETTER, Ralf ; MEDARD, Muriel ; KARGER, David R. ; EFFROS, Michelle: The benefits of coding over routing in a randomized setting. In: *In Proceedings of 2003 IEEE International Symposium on Information Theory*, 2003

[26] HO, Tracey ; MEDARD, Muriel ; SHI, Jun ; EFFROS, Michelle ; KARGER, David R.: On Randomized Network Coding. In: *In Proceedings of 41st Annual Allerton Conference on Communication, Control, and Computing*, 2003

[27] HUND, Johannes: *Applied Network Coding in Wireless Networks*, Fachhochschule Heidelberg, Diplomarbeit, 2007

[28] KNAUR VERLAG, München (Hrsg.): *Knaurs Lexikon von a bis z*. Knaur Verlag, 1999

[29] KONG, Zhenning ; ALY, S. ; SOLJANIN, E.: Decentralized Coding Algorithms for Distributed Storage in Wireless Sensor Networks. In: *Selected Areas in Communications, IEEE Journal on* 28 (2010), Februar, Nr. 2, S. 261 –267. – ISSN 0733-8716

[30] KURZWEIL, Hans: *Endliche Körper: Verstehen, Rechnen, Anwenden*. Springer, 2008

[31] LEVIS, Philip: *TinyOS Programming*. PDF Online. 2006. – URL `http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf`. – Zugriffsdatum: 18.06.2011

[32] LI, Baochun ; NIU, Di: Random Network Coding in Peer-to-Peer Networks: From Theory to Practice. In: *Proceedings of the IEEE* 99 (2011), März, Nr. 3, S. 513 –523. – ISSN 0018-9219

[33] LI, Xiaoping ; XU, Hu ; LU, Hao: A node cooperation based random linear network coding algorithm for wireless sensor networks. In: *Networked Computing and Advanced Information Management (NCM), 2010 Sixth International Conference on*, August 2010, S. 164 –168

[34] LILLIBRIDGE, Mark ; ELNIKETY, Sameh ; BIRRELL, Andrew ; BURROWS, Mike ; ISARD, Michael: A cooperative Internet backup scheme. In: *In Proceedings of the 2003 USENIX Annual Technical Conference*, 2003, S. 29–41

[35] LIN, Mu ; LUO, Chong ; LIU, Feng ; WU, Feng: Compressive Data Persistence in Large-Scale Wireless Sensor Networks. In: *GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference*, Dezember 2010, S. 1 –5. – ISSN 1930-529X

[36] LIN, Yunfeng ; LIANG, Ben ; LI, Baochun: Data Persistence in Large-Scale Sensor Networks with Decentralized Fountain Codes. In: *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, Mai 2007, S. 1658 –1666. – ISSN 0743-166X

[37] LUBY, M.: LT codes. In: *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002, S. 271 – 280. – ISSN 0272-5428

[38] MATTERN, Friedemann ; ROEMER, Kay: *Drahtlose Sensornetze*. Online PDF. – URL `http://www.vs.inf.ethz.ch/res/papers/sensornetze.pdf`. – Zugriffsdatum: 20.06.2011

[39] MEINEL, Chrisoph ; MUNDHENK, Martin: *Mathematische Grundlagen der Informatik, Mathematisches Denken und Beweisen, Eine Einführung*. 3. Auflage. Teubner, 2006

[40] MESSING, Barbara: *Das studium: Vom Start zum Ziel - Lei(d)tfaden für Studierende*. Springer, 2006

[41] MOTEIV CORPORATION: *tmote sky Datenblatt*. Online PDF. – URL `http://www.snm.ethz.ch/pub/uploads/Projects/tmote_sky_datasheet.pdf`

[42] NAGAJOTHY, M. ; RADHA, S.: Network lifetime enhancement in wireless sensor network using network coding. In: *Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on*, Juni 2009, S. 1 –4

[43] PAGEL, Heiko: *Datensicherung über Peer-to-Peer Netzwerke*, Hamburg, Hochschule für Angewandte Wissenschaften, Fak. Technik und Informatik, Dep. Informatik, Bachelorarbeit, Februar 2006

[44] PERREY, Heiner: *Untersuchung der Vertraulichkeit nach Obfuskierung durch Fountain Codes*, Hamburg, Hochschule für Angewandte Wissenschaften, Fak. Technik und Informatik, Dep. Informatik, Bachelorarbeit, August 2010

[45] ROBERT LI, Shuo yen ; YEUNG, Raymond W. ; CAI, Ning: Linear network coding. In: *IEEE Transactions on Information Theory* 49 (2003), S. 371–381

[46] ROSSI, M. ; BUI, N. ; ZANCA, G. ; STABELLINI, L. ; CREPALDI, R. ; ZORZI, M.: SYNAPSE++: Code Dissemination in Wireless Sensor Networks Using Fountain Codes. In: *Mobile Computing, IEEE Transactions on* 9 (2010), Dezember, Nr. 12, S. 1749 –1765. – ISSN 1536-1233

[47] ROSSI, M. ; ZANCA, G. ; STABELLINI, L. ; CREPALDI, R. ; HARRIS, A.F. ; ZORZI, M.: SYNAPSE: A Network Reprogramming Protocol for Wireless Sensor Networks Using Fountain Codes. In: *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on*, Juni 2008, S. 188 –196

[48] SCHMITZ, Marieluise (Hrsg.): *Pons Wörterbuch für Schule und Studium: Englisch-Deutsch / Deutsch-Englisch*. Bd. 1. PONS GmbH, 2007

[49] SHAH-MANSOURI, V. ; WONG, V.W.S.: Link Loss Inference in Wireless Sensor Networks with Randomized Network Coding. In: *GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference*, Dezember 2010, S. 1 –6. – ISSN 1930-529X

[50] STEFANOVIC, C. ; VUKOBRATOVIC, D. ; KARABENC, T. ; ROVCANIN, M. ; CRNOJEVIC, V.: On energy efficiency of rateless packet scheme for distributed data storage in wireless sensor networks. In: *Wireless On-demand Network Systems and Services (WONS), 2010 Seventh International Conference on*, Februar 2010, S. 61 –65

[51] STEINMETZ, Ralf ; WEHRLE, Klaus: *Peer-to-Peer Systems and Applications*. Springer, 2005

[52] STELAND, Prof. Dr. A.: *Basiswissen Statistik*. Springer, 2007

[53] SUBRAMANIAN, Nalin ; YANG, Chanjun ; ZHANG, Wensheng: Securing Distributed Data Storage and Retrieval in Sensor Networks. In: *Pervasive Computing and Communications, 2007. PerCom '07. Fifth Annual IEEE International Conference on*, März 2007, S. 191 –200

[54] TANENBAUM, Andrew S.: *Computer Networks*. 2. Auflage. Prentice Hall, 1988

[55] TANENBAUM, Andrew S. ; STEEN, Maarten van: *Verteilte Systeme - Prinzipien und Paradigmen*. 2. Auflage. Pearson Studium, 2008

[56] TESCHL, Gerald ; TESCHL, Susanne: *Mathematik für Informatiker*. Springer, 2006 (2)

[57] TINYOS: *TinyOS*. Webseite. – URL `http://www.tinyos.net/`. – Zugriffsdatum: 23.06.2011

[58] UC BERKELEY WEBS PROJECT: *nesC*. Webseite. – URL `http://nescc.sourceforge.net/`. – Zugriffsdatum: 23.06.2011

[59] UGUS, O. ; HESSLER, A. ; WESTHOFF, D.: Performance of Additive Homomorphic EC-ElGamal Encryption for TinyPEDS / NEC Europe Ltd. Juli 2007. – Forschungsbericht

[60] VENKAT, Kripasagar: Efficient Multiplication and Division Using MSP430 / Texas Instruments. URL `http://focus.ti.com/lit/an/slaa329/slaa329.pdf`, September 2006. – Forschungsbericht

[61] WANG, Yu ; HENNING, I.D. ; HUNTER, D.K.: Efficient Information Exchange in Wireless Sensor Networks using Network Coding. In: *Network Coding, Theory and Applications, 2008. NetCod 2008. Fourth Workshop on*, Januar 2008, S. 1 –6

[62] WEIWEI, Jiao ; CANFENG, Chen ; DONGLIANG, Xie ; JIAN, Ma: Efficient data collection in wireless sensor networks by applying network coding. In: *Broadband Network Multimedia Technology, 2009. IC-BNMT '09. 2nd IEEE International Conference on*, Oktober 2009, S. 90 –94

[63] WILLIAMS, Thomas ; KELLEY, Colin: *gnuplot 4.4 An Interactive Plotting Program*. Online PDF. März 2010. – URL `http://www.gnuplot.info/docs_4.4/gnuplot.pdf`. – Zugriffsdatum: 19.06.2011

[64] YEUNG, Raymond W. ; LI, Shou-Yen R. ; CAI, Ning ; ZHANG, Zhen: *Network Coding Theory*. now Publishers Inc., 2006

[65] ZELENKO, Dmitry: Sensornetze / Smart-Dust / Technische Universität München. 2006. – Forschungsbericht

[66] ZHU, Hongpeng ; ZHANG, Gengxin ; LI, Guangxia: A novel degree distribution algorithm of LT codes. In: *Communication Technology, 2008. ICCT 2008. 11th IEEE International Conference on*, nov. 2008, S. 221 –224

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. August 2011	
Ort, Datum	Unterschrift