Aus dem Institut für Telematik
der Universität zu Lübeck

Direktor:
Prof. Dr. rer. nat. Stefan Fischer

Erkennung und Nutzung von Struktur in drahtlosen Sensornetzen

Inauguraldissertation
zur
Erlangung der Doktorwürde
der Universität zu Lübeck
– Aus der Sektion für Informatik/Technik und Naturwissenschaften –

vorgelegt von
Frau Dipl.-Inf. Daniela Krüger
aus Hamburg

Lübeck, im Oktober 2010

Erster Berichterstatter:	Prof. Dr. rer. nat. Stefan Fischer
Zweiter Berichterstatter:	Prof. Dr. rer. nat. Volker Linnemann

Tag der mündlichen Prüfung: 21. Dezember 2010

Zum Druck genehmigt. Lübeck, den 22. Dezember 2010

Vorwort

Bereits Marie Freifrau von Ebner-Eschenbach (1830 - 1916) wusste, dass Wissen ein Gut ist, welches sich vermehrt, wenn man es teilt. Richtig bewusst wird einem der Sinn dieser Weisheit erst während der Forschungsarbeit, wenn Diskussionen völlig neue Aspekte oder Visionen hervortreten lassen. Die beiden Projekte FleGSens und FRONTS, an denen ich während meiner Doktorandenzeit beteiligt war, aber auch der Gedankenaustausch mit den Kollegen, haben mir gezeigt, dass das Arbeiten im Team außerordentlich fruchtbar und bereichernd sein kann.

Bedanken möchte ich mich primär bei meinem Doktorvater Prof. Stefan Fischer, der diese Arbeit überhaupt erst ermöglicht und mich immer motiviert und unterstützt hat. Des Weiteren gilt großer Dank allen meinen Diskussionspartnern Dr. Carsten Buschmann, Dr. Dennis Pfisterer und Peter Rothenpieler, die auch Koautoren der wissenschaftlichen Veröffentlichungen sind.

Am Projekt FleGSens waren neben den bereits genannten Lübecker Kollegen Denise Dudek, Christian Haas, Andreas Kuntz und Prof. Martina Zitterbart von der Universität Karlsruhe sowie Christian Wieschebrink, Markus Ullmann und Frank Gehring vom Bundesamt für Sicherheit in der Informationstechnik beteiligt. In FRONTS entstand eine besonders enge Zusammenarbeit mit Dr. Alexander Kröller, Dr. Tom Kamphans und Prof. Sándor P. Fekete von der Technischen Universität Braunschweig.

Für Übernahme des Korrefererats möchte ich mich ganz herzlich bei Prof. Dr. rer. nat. Volker Linnemann bedanken, für das Korrekturlesen dieser Dissertationsschrift bei Birgit Schneider, Dr. Carsten Buschmann und meinen Eltern. Sie haben mir mit zahlreichen Kommentaren sehr geholfen, die Worte dieser Schrift in einen sinnvollen Zusammenhang zu bringen.

Allen Kollegen und ehemaligen Kollegen Sebastian Ebers, Nils Glombitza, Daniel Bimschas, Oliver Kleine, Kinan Hakim, Maick Danckwardt, Claudia Becker, Dirk Schmidt, David Gregorczyk, Winfried Schöch, Timm Bußhaus, Dr. Martin Lipphardt, Dr. Christian Werner, Dr. Axel Wegener, Dr. Stefan Ransom und Prof. Horst Hellbrück danke ich für die angenehme Atmosphäre, für die sie am Institut für Telematik beständig gesorgt haben.

Zusammenfassung

Drahtlose Sensornetze sind heterogene verteilte Systeme, bestehend aus vielen kleinen Sensorknoten, die Umweltdaten, wie z. B. Temperatur, Feuchtigkeit oder Bewegung erfassen. Die Daten können sie über ihre Funkschnittstelle untereinander austauschen und zu höherwertigen Aussagen zusammenfügen. Erst durch die Erweiterung der Daten um Kontextinformationen wie Zeit und Ort erhalten diese eine aussagekräftige Bedeutung. Kontextinformationen bezüglich der Position lassen sich aus der räumlichen Anordnung von Knoten ableiten, sofern diese einer besonderen Struktur – z.B. ähnlich einem Straßennetz – unterliegt. Neben der räumlichen Anordnung können die Hardwareausstattung sowie die Rollenverteilgung von Knoten zu einer besonderen Struktur des Netzes führen. Diese Arbeit beschäftigt sich mit der Erkennung und Nutzung von solchen strukturellen Gegebenheiten in unterschiedlichen Themengebieten.

Zunächst wird ein dezentrales Verfahren vorgestellt, welches symbolische Positionen von Knoten innerhalb einer räumlichen, graphenförmigen Struktur ermittelt. Das Verfahren eruiert weiterhin die Anordnungsreihenfolgen von direkten Nachbarn von Knoten. Wie die Reihenfolgen zur Verfolgung von Objekten innerhalb der Graphstruktur verwendet werden können, wird ebenfalls gezeigt. Hierzu wird ein neues, dezentrales und adaptiv arbeitendes Verfahren beschrieben, welches die Bewegungen mehrerer Objekte gleichzeitig erkennt.

Da Sensornetze trotz des geringen Energievorrats von Sensorknoten auch Langzeitanwendungen realisieren sollen, ist der Einsatz verschiedener Energiesparmodi unabdingbar. Es wird deshalb ein Duty-Cycle-Verfahren vorgestellt, welches an unterschiedliche Anforderungen bezüglich der Nachrichtenverzögerung und des Nachrichtenaufkommens sowie des gewünschten Duty-Cycles angepasst werden kann. Da benachbarte Knoten zu ähnlichen Zeitpunkten aufwachen, erfahren Nachrichten, die weitergeleitet werden sollen, nur eine kurze Verzögerung. Zusätzlich wird die Kollisionswahrscheinlichkeit im Vergleich zum wahlfreien Medienzugriff reduziert. Besonders effizient ist das Verfahren bei linearen Netzstrukturen, bei denen die Knoten in einer (gewundenen) Linie angeordnet sind. Es wird aber allgemein analysiert, für welche Netzstrukturen das Verfahren geeignet ist. Außerdem wird gezeigt, welche Kommunikationsmuster bevorzugt unterstützt werden und wie sich die verschiedenen Parameter gegenseitig bedingen.

Verteilte Anwendungen zu realisieren, ist allerdings heute immer noch besonders bei eingebetteten Systemen schwierig, da sie kaum Möglichkeiten zur experimentellen und iterativen Softwareentwicklung bieten. Simulatoren bilden die Unzuverlässigkeit der drahtlosen Kommunikation, zeitliche Aspekte sowie die Parallelität des Systems aller-

dings nur annäherungsweise ab. Diese Eigenschaften sind jedoch von entscheidender Wirkung und somit für die Implementierung effektiver und effizienter Protokolle extrem wichtig. Netze in Testumgebungen sind daher ein sinnvolles und komfortables Mittel zur Verifikation von Software für Sensornetze. Sie erfordern aber auch, dass die Programme auf den Sensorknoten ohne großen Kostenaufwand austauschbar sind. Eine verbesserte Verarbeitung der aufgezeichneten Daten oder Weiterentwicklung eines Protokolls sind ebenfalls nur in bestehende Netze integrierbar, wenn Knoten im Sensornetz automatisiert neu programmiert werden können. Manuelles Programmieren ist kostenintensiv und unpraktikabel.

Diese Arbeit präsentiert deshalb ein robustes Programmierverfahren, welches auch in heterogenen Multihop-Netzen eingesetzt werden kann. Die Heterogenität sowie die Menge der zu programmierenden Knoten bilden per se eine Netzwerkstruktur. Beim vorgeschlagenen Verfahren etabliert das Netz zusätzlich eine Struktur, die eine Kommunikation zwischen den zu programmierenden Knoten und dem Knoten ermöglicht, über den das Programm ins Netz injiziert wird. Es arbeitet energiesparend, dadurch dass unbeteiligte Knoten während des Programmiervorgangs im Schlafmodus sein können und fehlende Programmteile primär bei nah gelegenen Knoten nachgefordert werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Eingebettete Systeme finden sich mittlerweile in nahezu allen unseren Lebensbereichen. Meist sind sie – wie z. B. in Haushaltsgeräten oder Heizungsanlagen – unsichtbar und ausschließlich für die Steuerung eines Gerätes zuständig. Auch kommunizierten sie bislang meist weder mit uns noch untereinander.

Die Erweiterung von Messsystemen um ein Funkmodul hat vor einigen Jahren jedoch zu einer neuen Technologie geführt – den drahtlosen Sensornetzen. Sie bestehen aus vielen kleinen, unabhängigen Geräten, den so genannten *Sensorknoten*, die über einen Prozessor, Speicher, eine Energieversorgung, Sensorik und insbesondere über ein Modul zur drahtlosen Kommunikation verfügen. Aufgrund ihrer Ungebundenheit können sie in beinahe beliebiger Umgebung eingesetzt werden. Mit Hilfe der immer vielfältigeren und leistungsfähigeren Sensoren erheben sie in ihrer Umgebung Daten, und durch das Funkmodul sind sie in der Lage, die erhobenen und eventuell vorverarbeiteten Messwerte drahtlos untereinander auszutauschen.

Eine solche Kooperation von mehreren Geräten ermöglicht nicht nur die Aggregation von gemessenen Daten zu höherwertigen Aussagen, sondern auch eine Aufgabenteilung und Selbstkonfiguration. Steht einem Gerät nicht mehr so viel Energie zur Verfügung wie seinen Nachbarn, können diese beispielsweise seine Aufgaben übernehmen. Auch können leicht neue Sensorknoten in das so entstandene drahtlose Netz integriert werden.

1.1 Motivation und Problemstellung

Die Hardwareentwicklung der letzten Jahre hat gezeigt, dass die Sensorknoten tendenziell immer kompakter gebaut werden können. Visionäre sprechen auch vom „intelligenten Staub“ [118], das heißt die Geräte werden für den Menschen unsichtbar, während die Umwelt zur intelligenten Umgebung wird. Sensornetze könnten dann sogar in die Blutbahnen von Lebewesen eingebracht oder in Baumaterial integriert werden. Die intelligente Brücke und das intelligente Gebäude würden dann selbständig Materialermüdung und Erneuerungsbedarf erkennen und melden.

Die Sensornetze, die bereits im Einsatz sind, bestehen allerdings meist noch aus einer kleineren Anzahl an Geräten und die verfügbaren Sensorknoten sind insbesondere durch

große Energiespeicher noch mehrere Kubikzentimeter groß. Ziel dieser Arbeit ist es, einen wissenschaftlichen Beitrag zu leisten, um die Miniaturisierung der Energiespeicher zu ermöglichen sowie der Realisierbarkeit, der Handhabbarkeit und der effizienten Gestaltung auch größerer Netze einen Schritt näher zu kommen.

Gerade um größere Sensornetze realisieren zu können, müssen aber noch konkrete Probleme gelöst werden. Diese Arbeit beschäftigt sich mit drei verschiedenen Problemen:

Problematisch ist z. B. die Entwicklung effizienter, und insbesondere korrekter Software für große Netze. Fest installierte Sensornetze in Testumgebungen helfen den Entwicklern, Design- und Programmierfehler frühzeitig zu finden. Das experimentelle Testen verschiedener Protokolle und Anwendungen in einer Testumgebung erfordert jedoch, dass immer wieder neue Programme auf den Geräten installiert werden müssen. Aber auch in einem Sensornetz im „richtigen Einsatz“ unterliegt die Software einem Lebenszyklus, so dass sie nicht nur entwickelt und installiert, sondern auch gewartet und weiterentwickelt werden muss. Soll ein Sensornetz zusätzliche, ursprünglich nicht geplante Aufgaben übernehmen oder verfeinerte Berechnungen durchführen, muss das laufende Programm ausgetauscht werden.

Ein manuelles Programmieren über Kabel würde – insbesondere wenn Geräte schwer zugänglich sind und sich in einem geschlossenen Gehäuse befinden – einerseits die Kosten immens erhöhen und andererseits ein entferntes Programmieren sinnlos machen, da vor Ort ein Kabel angeschlossen werden muss. Damit jedoch möglichst wenig zusätzliche Kosten entstehen und die Geräte durch den Programmiervorgang nur wenig Energie verbrauchen, muss der Programmaustausch möglichst schnell und weitgehend automatisiert durchgeführt werden. Existierende Lösungen eignen sich entweder nicht für große Netze oder basieren darauf, dass alle Geräte dasselbe Programm erhalten. Da Heterogenität bei Sensornetzen aber durch unterschiedliche Hardwarekomponenten (z. B. verschiedene Energieversorgungen oder einzelne Geräte mit GPS-Empfänger) sowie durch unterschiedliche Rollen der Geräte (z. B. Daten erheben, weiterleiten, speichern oder in eine Datenbank schreiben) durchaus die Regel und nicht die Ausnahme ist, besteht Bedarf nach einem selektiven Programmierprotokoll.

Ein weiteres großes Problem ist der bereits erwähnte kleine Energievorrat, der dazu führt, dass Geräte üblicherweise nur zeitweise aktiv sein können. Durch inaktive Phasen der Knoten können aber große Nachrichtenverzögerungen entstehen. Bei vielen Anwendungen, z.B. bei denen Alarmmeldungen eine Rolle spielen, ist es jedoch essentiell, dass der Anwender umgehend informiert wird. Durch den Trend der Miniaturisierung wächst das Problem noch weiter, da insbesondere die Energiespeicher kleiner werden müssen.

Derzeitige Lösungen realisieren häufig keinen konstant niedrigen Energieverbrauch, sondern beruhen darauf, dass dieser für eine begrenzte Zeit höher sein kann als üblich. Das hängt jedoch von der Kapazität des Energiespeichers ab und kann nicht immer ermöglicht werden. Außerdem eröffnet der bedarfsorientierte Energieverbrauch eine Sicherheitslücke,

die Angreifer mittels Vortäuschen von Bedarf ausnutzen können, um Sensorknoten zum Ausfallen zu bringen.

Andere existierende Verfahren unterstützen ausschließlich die Kommunikation zu genau einem Zielempfänger und nicht das Senden einer Nachricht an alle Sensorknoten, die sich in Funkreichweite befinden. Viele Anwendungen benötigen jedoch diese Art von Kommunikation (z. B. zur Datenaggregation) und ein mehrmaliges Senden der Nachricht (an jeden Nachbarknoten einzeln) wäre hinsichtlich Energie- und Zeitaufwand extrem kostenintensiv und ineffektiv.

Einige Autoren schlagen ein Vorgehen vor, welches in seiner Grundidee der in der vorliegenden Arbeit präsentierten Lösung ähnelt. Sie gehen aber von unrealistischen Bedingungen, wie der zuverlässigen Kommunikation aus, zeigen weder Grenzen noch Implikationen auf und erprobten ihre Verfahren bisher nicht in der Praxis.

Schließlich besteht das Problem, dass Sensorknoten meist nicht über Positionsinformationen verfügen, ohne die die erhobenen Daten jedoch an Bedeutung verlieren. Sensornetze unterliegen aber aufgrund der räumlichen Anordnung, der Heterogenität oder der Kontexte der einzelnen Geräte häufig einer räumlichen oder kontextuellen Struktur. Es stellt sich daher die Frage, wie Informationen über eine gegebene Struktur gewonnen und wie aus diesen wiederum Positionsinformationen abgeleitet werden können. Mit dieser Fragestellung beschäftigt sich eine Arbeit, bei der von Netzen extrem hoher Dichte ausgegangen wird. Die vorgeschlagene Lösung eignet sich jedoch nicht bei Netzen niedriger Dichte.

Die Frage, inwieweit spezielle Strukturen zum Lösen eines Problems oder zur Effizienzsteigerung explizit eingesetzt werden können, stellt sich aber nicht nur bei der Gewinnung von Positionsinformationen, sondern auch bei den anderen genannten Problemen. Die vorliegende Arbeit thematisiert die unterschiedliche Themengebiete deshalb vor dem Hintergrund der Erkennung und Nutzung besonderer struktureller Gegebenheiten.

1.2 Wissenschaftliche Beiträge und Struktur der Arbeit

Die vorliegende Arbeit liefert Beiträge zur Erkennung und Nutzung von Struktur in Sensornetzen in unterschiedlichen Bereichen. Kapitel 2 führt den Leser in die Thematik der Sensornetze grundlegend ein. Es beschreibt den Aufbau von Sensornetzen sowie deren Anwendungen, Eigenschaften und Beschränkungen. Die nachfolgenden drei Kapitel schließen die eigenen wissenschaftlichen Beiträge der Autorin ein:

1. Ein verteiltes Protokoll zur Erkennung von Geräteanordnungen und darauf aufbauend ein Verfahren zur Objektverfolgung in Graphstrukturen,
2. analytische, simulative und experimentelle Untersuchungen zu einem Duty-Cycle-Verfahren sowie

3. ein Protokoll zur Reprogrammierung von heterogenen Netzen über die Funkschnittstelle.

Kapitel 3 umfasst zum einen ein neues Verfahren zur Erkennung der eindimensionalen Anordnung von benachbarten Sensorknoten. Mit Hilfe von geschätzten Distanzen werden die Nachbarknoten in Reihenfolgen gebracht sowie Ortsinformationen über die eigene Position im Netz abgeleitet. Zum anderen befasst sich das Kapitel mit einer Kategorie von Sensornetzanwendungen, der Objektverfolgung. Es wird ein neues Verfahren vorgestellt, diskutiert und evaluiert, welches die Kenntnis über die Nachbarschaftsstruktur zur Objektverfolgung nutzt. Das Verfahren basiert auf der Annahme, dass ein passierendes Objekt von den Sensorknoten in der Reihenfolge der vorher berechneten Anordnungen detektiert wird. Es wird gezeigt, dass auch mehrere Objekte erfolgreich verfolgt werden können. Das Wissen über Nachbarschaftsreihenfolgen und die eigene Position kann generell in Szenarien wie Fluren, Straßennetzen oder Kanalsystemen hilfreich sein.

Kapitel 4 betrachtet das Problem der Organisation der Aktivphasen von Sensorknoten, um Energie zu sparen. Es wägt die sich prinzipiell widersprechenden, aber für den praktischen Einsatz häufig notwendigen Forderungen nach geringer Latenz und geringem Duty-Cycle, gegeneinander ab und untersucht ein Verfahren, bei dem die aktiven Phasen insgesamt eine Art Welle bilden. Gruppen von Sensorknoten wechseln dabei der Reihe nach in den aktiven und nach einer gewissen Zeit wieder in den inaktiven Zustand. Diese Methode erlaubt es, einen Kompromiss etablieren, ohne dabei Sicherheitsaspekte zu vernachlässigen. Anders als bei bereits existierenden Arbeiten werden diejenigen Aspekte beleuchtet, die einen praktischen Einsatz ermöglichen. Außerdem wird bei den Untersuchungen von unzuverlässiger Kommunikation ausgegangen und es werden verschiedene räumliche Anordnungen der Sensorknoten (lineare, rechteckförmige und komplexe Netzwerkstrukturen) betrachtet. Weiterhin wird das Verfahren zum ersten Mal mittels Hardware getestet.

Kapitel 5 beschäftigt sich mit der Reprogrammierung von Sensornetzen mittels Nachrichtenaustausch. Es analysiert detailliert die für ein Reprogrammieren von einer beliebigen Teilmenge der Knoten eines Sensornetzes notwendigen Schritte und optimiert jeden einzelnen der Schritte. Bestehende Verfahren gehen meist ausschließlich auf die Verteilung des neuen Programms ein oder sind nur in homogenen Netzen anwendbar. Das in der vorliegenden Arbeit vorgestellte Verfahren ist im Gegensatz zu existierenden Lösungen nicht nur robust gegen Nachrichtenverlust, sondern funktioniert insbesondere auch in sehr großen, heterogenen Netzen. Heterogenität führt (neben der räumlichen Anordnung von Knoten) zur einer speziellen Netzstruktur. Das Verfahren nutzt die Tatsache, dass Geräte, die ein neues Programm erhalten, fehlende Programmteile untereinander ergänzen können. Nur wenn lokal keine Programmteile mehr verfügbar sind, dient eine zuvor aus dem gesamten Netz konstruierte Infrastruktur der robusten Kommunikation mit dem Knoten, die das Programm ausliefert. Auch das effiziente Sammeln von Daten aus dem Netz wird in diesem Zusammenhang untersucht.

Kapitel 2

Grundlagen

Die vorliegende Arbeit beschäftigt sich mit der Erkennung und Nutzung von besonderen Strukturen in Sensornetzen. Dieses Kapitel führt den Leser zunächst in grundlegende Aspekte des behandelten Themengebiets ein. Die ersten beiden Abschnitte des Kapitels beschreiben den Aufbau eines einzelnen Sensorknotens sowie die für diese Arbeit relevante Sensorik. Abschnitt 2.3 erläutert anschließend den Aufbau von Sensornetzen, während Abschnitt 2.4 ihren Nutzen motiviert. Die beiden darauf folgenden Abschnitte zeigen besondere Anforderungen und Eigenschaften von Sensornetzen auf, während sich Abschnitt 2.7 mit der Modellierung dieser Eigenschaften in der Simulation beschäftigt. Abschnitt 2.8 beschreibt die zum Einsatz gekommene Testumgebung des Projektes FRONTS. Schließlich führen Abschnitt 2.9 und Abschnitt 2.10 in die Grundlagen einiger Protokolle zum Duty-Cycling und Routing ein.

2.1 Aufbau eines Sensorknotens

Ein Sensorknoten (kurz: Knoten) umfasst einen Prozessor, einen Transceiver (zusammengesetzt aus „transmitter“ und „receiver“) zum Senden und Empfangen von Nachrichten, eine Energieversorgung, Speicher und Sensorik.

Typische Energielieferanten sind Batterien und Akkumulatoren, seltener steht eine Netzstromversorgung zur Verfügung. Akkumulatoren können z. B. mit Hilfe von Solarzellen oder Vibrationsenergieumwandlern geladen werden, so dass die Sensorknoten im Idealfall immerwährend mit ausreichend Energie versorgt sind. Abbildung 2.1 zeigt einen Sensorknoten, der über einen Akkumulator mit Energie versorgt wird und zusätzlich über Ein- und Ausgabeschnittstellen (einen Erweiterungsstecker und eine LED) verfügt.

Der Speicher teilt sich meist in flüchtigen Speicher (RAM – Random Access Memory) und nicht-flüchtigen Speicher (EEPROM – Electrically Erasable Programmable Read-Only Memory) auf. Der flüchtige Speicher dient dem temporären Speichern von Daten, wie z. B. Messwerten, die zu einem späteren Zeitpunkt weitergeleitet werden sollen. Der nicht-flüchtige Speicher ist kostengünstiger, aber langsamer les- und schreibbar. Er besteht bei den meisten Architekturen aus einem Flash-EEPROM und dient dem Speichern von großen Datenmengen wie Programm, Schlüsselmaterial und größeren Mengen an

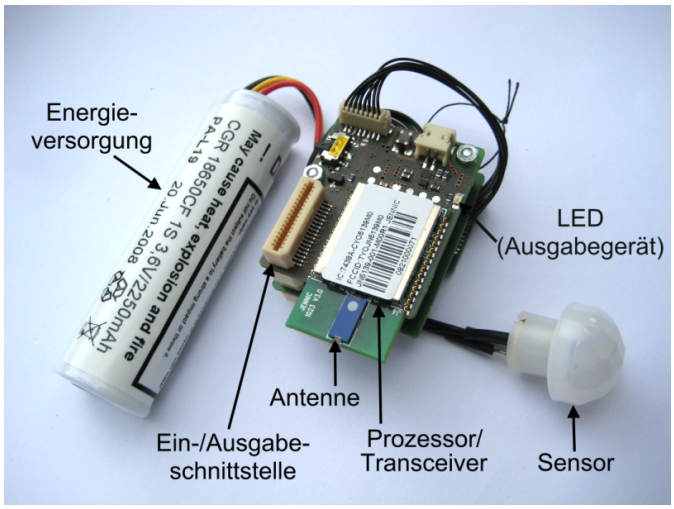


Abbildung 2.1: Sensorknoten

Messwerten. Flash-Speicher bestehen typischerweise aus wenigen Sektoren, z. B. der Größe 32 KByte, die zwar byteweise beschreibbar, aber nur als Ganzes löschbar sind. Dadurch ist es nicht möglich, beliebig kleine Teile im Speicher zu ersetzen, ohne andere Daten zu löschen. Ältere Hardwareplattformen verfügen noch über einen herkömmlichen EEPROM. Diese können zwar byteweise gelöscht werden, sind aber beim Schreib-/Lesezugriff noch langsamer.

(a) Mica2 (b) TelosB (c) Scatternode

Abbildung 2.2: Verschiedene Sensorknoten

Während in Abbildung 2.1 ein iSense-Knoten von der Firma coalesenses [13] dargestellt ist – ähnlich der Knoten, die im Rahmen dieser Arbeit zum Einsatz kamen –, zeigt Abbildung 2.2 Knoten anderer Hersteller: den Mica2 von Crossbow [20], den TelosB von der FU Berlin [89] und den Scatternode von ScatterWeb [21, 99]. Tabelle 2.1 gibt die zugehörigen Hardwaredaten an. Sowohl der Prozessor als auch der Transceiver können üblicherweise in verschiedene Zustände versetzt werden, in denen sie unterschiedlich viel

Energie verbrauchen. Die Stromverbräuche der jeweiligen Zustände sind in der Tabelle mit I_x gekennzeichnet.

	Mica2	MicaZ	TelosB	iSense 39x	Scatternode
Jahr	2002	2003	2004	2006	2006
CPU	8 Bit 1-8 MHz	8 Bit, 1-8 MHz	16 Bit 8 MHz	32 Bit 16 MHz	16 Bit 8 MHz
$I_{Wachzustand}$	8 mA	8 mA	2,4 mA	9 mA	225 mA
$I_{Schlafzustand}$	15 µA	15 µA	6 µA	10 µA	25 µA
RAM	4 KByte	4 KByte	10 KByte	96 KByte	5 KByte
Flash	128+512 KByte	128+512 KByte	48 KByte	128 KByte	55+512 KByte
Funkfrequenz	868/916 MHz	2,4 GHz	2,4 GHz	2,4 GHz	868 MHz
Max. Übertragungsrate	19,2 kBit/s	250 kBit/s	250 kBit/s	250 kBit/s	19,2 kBit/s
Funkreichweite	<150 m	20 - 100 m	50 - 125 m	30 - 100 m	1 km
$I_{Empfangs-bereitschaft}$	10 mA	17 mA	17 mA	30 mA	25 mA
$I_{Empfangen}$	10 mA	17 mA	19 mA	30 mA	25 mA
I_{Senden}	27 mA	20 mA	19 mA	30 mA	70 mA
Größe in mm^3 (ohne Energieversorgung)	58 x 32 x 7	58 x 32 x 7	66 x 33 x 7	45 x 30 x 9	67 x 38 x 12

Tabelle 2.1: Eigenschaften ausgewählter Hardware-Plattformen

Der Prozessor kann aktiv (auch: wach) oder inaktiv (auch: schlafend) sein, wobei er das Programm nur im aktiven Zustand abarbeiten kann. Während des inaktiven Zustands laufen nur Oszillatoren, über die der Prozessor beispielsweise wieder aktiviert werden kann. Der Stromverbrauch des iSense-Prozessors beträgt etwa 9 mA im aktiven und etwa $10\mu A$ im inaktiven Zustand.

Ein Transceiver kann *senden*, *empfangen*, *(empfangsbereit) warten* und *schlafen*, wobei nur der Schlafzustand mit seinem extrem niedrigen Energieverbrauch (üblicherweise weniger als 1 mA) einen Energiesparmodus darstellt. Jedoch kann der Sensorknoten dann weder Nachrichten versenden noch empfangen.

iSense-Knoten bestehen aus einem mit 16 MHz getakteten 32 Bit RISC Prozessor (JN5139) und haben 128 KByte Flash-Speicher, wobei drei Viertel davon mit Programmcode belegt werden können. Das Programm wird vor dem Booten in den 96 KByte großen RAM

kopiert. Die Funkschnittstelle ist IEEE 802.15.4 kompatibel und ermöglicht eine maximale Übertragungsrate von 250 kBit/s im 2,4 GHz Band bei einer Maximallänge einer Nachricht von 116 Byte (MTU – Maximum Transfer Unit).

Geräte, die niedrigere Funkfrequenzen verwenden, erreichen teilweise nur eine maximale Datenrate von 19,2 kBit/s. Je geringer die Frequenz ist, desto größer ist dafür die Wellenlänge und somit auch die Reichweite. Üblicherweise können die Geräte über unikale (das heißt nur einmal vorkommende), 16 Bit lange Adressen identifiziert werden.

Senden und Empfangen von Nachrichten kostet auf der iSense-Plattform, verglichen mit dem empfangsbereiten Warten, keinen zusätzlichen Strom. Allerdings gibt es Hardwareplattformen, wie die Mica-Serie, bei denen das Senden von Nachrichten mehr Energie verbraucht als die reine Empfangsbereitschaft und als das Empfangen von Nachrichten. Auch deshalb ist es von essenzieller Bedeutung, dass Protokolle ein möglichst geringes Nachrichtenaufkommen haben.

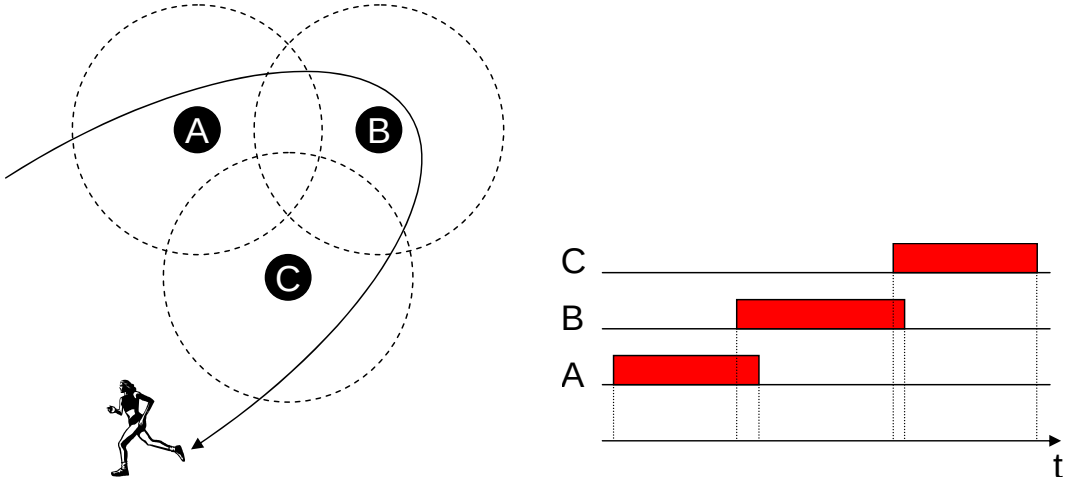
Die Größenangaben der Tabelle beziehen sich nur auf die Platine, auf der Prozessor und Funkchip aufgebracht sind. Insbesondere schließen sie keine Energieversorgung ein. Allein ihre Größe beträgt derzeit bei den meisten kommerziellen Geräten noch wenige Kubikzentimeter. Arbeiten von Forschern führen aber zur Verkleinerung der Chips. Der Nachfolger des Mica Sensorknotens ist der so genannte SPEC Sensorknoten [42], der ohne Energieversorgung nur noch $5x5x2mm^3$ groß ist. Miniaturisierung und Kostensenkung sind wichtige Faktoren für die Marktfähigkeit von Sensornetzen und um der Vision vom intelligenten Staub näher zu kommen. Die Prioritäten der Hersteller liegen deshalb weniger in der Vergrößerung des Speichers oder der Erhöhung der Prozessorgeschwindigkeit, sondern mehr in der Optimierung der Energieversorgung und Verkleinerung aller Komponenten.

2.2 Sensorik zur Objektdetektion

Ein *Sensor* ist eine Einheit, die ein physikalisches Phänomen wie Licht, Wärme oder Schall in elektrische oder andere Signale umwandelt (vgl. [128]). Mit *Detektion* ist der Entdeckungsprozess der Existenz des Phänomens gemeint. Unterere und oberere Grenzwerte bestimmen dabei, ob bei einer Messung von der Existenz des Phänomens ausgegangen werden kann.

Generell ist die Spanne der eingesetzten Sensoren breit. Sie reicht von Temperatur- und Feuchtigkeitssensoren über Magnetsensorik sowie Gyrometern, Beschleunigungs- und Vibrationssensoren bis hin zu Mikrofonen und Kameras. Zur Objektdetektion werden am häufigsten – wie auch in dieser Arbeit – Sensoren eingesetzt, die ein binäres Ergebnis liefern. Umgangssprachlich werden sie meist als *Näherungssensor* (englisch: proximity sensor) bezeichnet, obwohl *Präsenzsensor* eine präzisere Beschreibung darstellt. Er liefert „true“, wenn sich mindestens ein Objekt im vom Sensor überwachten Bereich befindet,

ansonsten „false“. Abbildung 2.3(a) stellt beispielsweise drei Sensoren (A, B und C) mit abstrahierten Sensorbereichen (begrenzt durch Kreise) und den zurückgelegten Weg (Pfeil) einer Person dar. Abbildung 2.3(b) zeigt die über die Zeit entstandenen Auslösesignale der Sensoren.



(a) Sensorbereiche und Weg eines Objekts

(b) Auslösesignale über die Zeit

Abbildung 2.3: Detektion eines Objekts

Eine Lichtschranke ist z. B. so ein Präsenzsensor. Ein Passiv-Infrarot (PIR)-Sensor reagiert dagegen auf Temperaturveränderungen in seinem Detektionsbereich. Er liefert zwar auch ein binäres Signal, aber erkennt nicht die Anwesenheit eines Objektes wie die Lichtschranke, sondern detektiert es nur für den Fall, dass es sich bewegt.

Einige Arbeiten zur Objektdetektion gehen auch von Schallsensoren aus. Sie messen mit Hilfe eines Mikrofons die Amplitude der Schallwellen und leiten daraus die Entfernung des Objektes ab. Werden viele Mikrofone nebeneinander in einem Sensor eingesetzt, kann aus der Einstrahlrichtung der Schallwellen die Richtung berechnet werden, in der sich das Objekt befindet.

2.3 Aufbau eines Sensornetzes

Ein drahtloses Sensornetzwerk (englisch: WSN – Wireless Sensor Network, kurz: Sensornetz) ist schematisch in Abbildung 2.4 dargestellt. Es setzt sich aus vielen Sensorknoten zusammen (links), von denen ein oder mehrere Knoten üblicherweise über eine Kommunikationsmöglichkeit mit einem Backendsystem (rechts) verfügen. Diese werden als *Gatewayknoten* oder *Senken* bezeichnet, da sie aufgrund ihrer Verbindung zum Backendsystem meist als Datensenken, das heißt als Sammelstelle und Zwischenspeicher der zu

übertragenden Daten, dienen. Das Backendsystem dient der weiteren Verarbeitung, Speicherung und Anzeige der Daten aus dem Netz. Die Verbindung zwischen Gatewayknoten und Backendsystem kann sowohl drahtlos (z. B. über Bluetooth) als auch kabelgebunden (z.B. über eine serielle Schnittstelle) realisiert sein. Wird zusätzlich ein anderes Netzwerk wie das Internet, ein GSM-Netz (Global System for Mobile Communications), ein UMTS-Netz (Universal Mobile Telecommunications System) oder LAN (Local Area Network) verwendet, müssen Gatewayknoten und Backendsystem nicht in räumlicher Nähe sein.

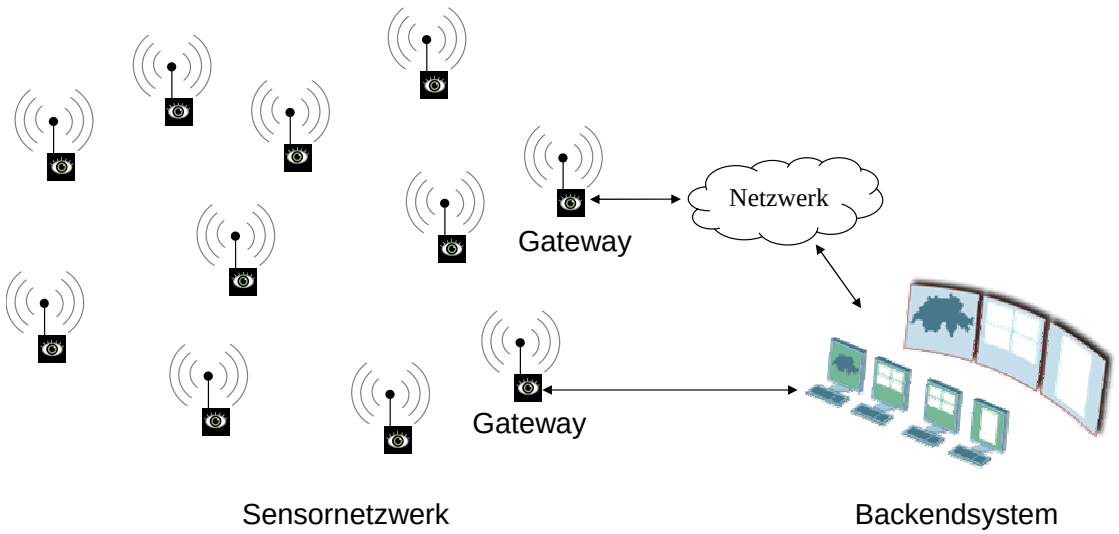


Abbildung 2.4: Sensornetzwerk (links) mit Kommunikationsmöglichkeit zu einem Backendsystem (rechts)

Die Sensorknoten kommunizieren drahtlos über ihre Funkschnittstelle. Eine *Broadcast*-Nachricht ist hierbei eine adressfreie Nachricht und ist für alle Knoten bestimmt, die sie empfangen. Ihr Empfang wird nicht bestätigt. Eine *Unicast*-Nachricht enthält eine konkrete Zieladresse und gegebenenfalls die Aufforderung zum Senden einer Bestätigung. Empfängt der Zielknoten eine solche Nachricht, sendet er eine Bestätigung, falls diese gefordert wurde.

Im Weiteren beschreibt die Formulierung „Knoten A ist Nachbar von Knoten B“ die Tatsache, dass Knoten B zumindest zeitweilig Nachrichten von A direkt empfangen kann. Eine solche direkte Kommunikationsverbindung wird auch als *Link* bezeichnet. Die *Netzdichte* bezeichnet die mittlere Anzahl an Nachbarn über alle Knoten sowie über die Zeit.

Der Begriff *Hop* bezieht sich immer auf die Anzahl an Weiterleitungen, die notwendig ist, um eine Kommunikation zwischen zwei beliebigen Knoten zu erreichen. Direkte Nachbarn kommunizieren über einen Hop. Gibt mindestens zwei Knoten, die nicht (direkt oder über mehrere Hops hinweg) miteinander kommunizieren können, so ist das Netz

nicht zusammenhängend und man spricht von zwei oder mehr *Netzpartitionen* (kurz: Partitionen).

Der *Netzwerkdurchmesser* ist die maximale Anzahl an Hops, die eine Nachricht von einem beliebigen Knoten im Netz zu einem beliebigen anderen Knoten über den kürzesten Kommunikationspfad benötigt.

Mit *Ende-zu-Ende-Verzögerung* wird im Weiteren die Zeit bezeichnet, die zwischen der Erzeugung einer Nachricht auf einem Knoten und dem Empfang der Nachricht auf einem anderen Knoten vergeht.

Einige Anwendungen und Protokolle erfordern, dass Daten aus dem gesamten Netz an einer oder mehreren Senken gesammelt werden. Damit die hierzu notwendigen Nachrichten das Kommunikationsnetz nicht überlasten und der Energieverbrauch möglichst gering gehalten wird, sollten die erhobenen Daten bereits von den Knoten vorverarbeitet und komprimiert werden, sofern dies möglich ist.

Zur Effizienzsteigerung können aber auch Daten von mehreren Knoten, z. B. über die Bildung von Minimal-, Maximal- oder Mittelwerten, zusammengefasst werden. Ist dies nicht möglich, sondern jeder einzelne Wert von Bedeutung, können trotzdem „Gesamtnachrichten“, die die Werte mehrerer Knoten enthalten, weitergeleitet werden, statt jeden Wert separat zur Senke zu senden. Beide Vorgehensweisen reduzieren die Anzahl benötigter Nachrichten und werden als *Aggregationsverfahren* bezeichnet.

2.4 Anwendungsgebiete

Das klassische Einsatzgebiet von Sensornetzen besteht in der Erhebung von Daten in der Umgebung – sowohl in Gebäuden als auch im Freien. Die Messfrequenz bestimmt die zeitliche Auflösung der Daten. Diese werden gegebenenfalls mit Kontextinformationen angereichert oder mit Daten von Nachbarknoten korreliert. Anschließend werden sie an eine zentrale Einheit zur weiteren Auswertung oder Speicherung transportiert.

Einige Beispielanwendungen werden in [121] aufgelistet. In Gebäuden können mit Hilfe von Sensornetzen klimatische Daten wie Temperatur und Feuchtigkeit, aber auch Staub- oder Rauchentwicklung ermittelt werden. Ein Museum kann so beispielsweise nachweisen, dass vorgegebene Grenzwerte nicht überschritten werden und die in ihm ausgestellten Kunstwerke durch die in den Räumen herrschenden Bedingungen keinen Schaden nehmen.

Im Freien bieten die Beobachtung von Tieren, die Unterstützung der Landwirtschaft (z. B. Überwachung der Feuchtigkeit der Erde und Erkennung von Schädlingen beim Weinanbau) und der Katastrophenschutz Anwendungsmöglichkeiten. Mittels Überwachung von Deichen können z. B. Gefahrensituationen früher erkannt und Gegenmaßnahmen ergriffen werden. Auch die Lokalisierung von Waldbränden kann zur schnelleren Bekämpfung beitragen. Eine weitere Möglichkeit besteht in der Überwachung von Grenzen. Ein Sensor-

netz zur Grenzüberwachung wurde beispielsweise im Projekt *FleGSens* [93] entwickelt und prototypisch realisiert. Ziel des Projektes war es unter anderem an einer „grünen Grenze“ einen Eindringling inklusive seiner Bewegungsrichtung zu detektieren und möglichst schnell bei einer zentralen Stelle zu melden. Die Objektverfolgung ist generell ein großes Anwendungsgebiet für Sensornetze.

Abbildung 2.5: Sensornetz im Carlebachpark in Lübeck

Ein anderes Anwendungsfeld von Sensornetzen wird durch die Anheftung von Knoten an Fahrzeuge erschlossen: die Vehicular Ad-hoc Networks (VANETS). Mittels Kommunikation mit entgegenkommenden Fahrzeugen kann der Fahrer schon frühzeitig über eine Stausituation informiert werden; langfristiges Ziel ist allerdings die Stauvermeidung. Im Zusammenhang mit dem Straßenverkehr sind auch Parkleitsysteme zu nennen, die den Autofahrer bei Bedarf zu einem freien Parkplatz führen. Auch in der Logistik können intelligente Sensoren helfen, die Kosten zu senken [46, 111].

Werden Sensorknoten nicht am Auto angebracht, sondern am Körper getragen, spricht man von einem Body Area Network (BAN). Meist werden hierbei Vitaldaten wie Puls, Herzfrequenz, Körpertemperatur oder Sauerstoffgehalt im Blut gemessen. Auch hier sind die Anwendungsfälle vielfältig. Sie reichen von der Beobachtung des Gesundheitszu-

standes von Katastrophen- und Polizeieinsatzkräften über die pre- und postoperative Überwachung von Patienten bis hin zur Unterstützung von älteren Menschen im Ambient Assisted Living (AAL)-Bereich. Beispiele dieser Kategorie sind in [29, 81, 88, 101, 107] zu finden.

Eine Vision besteht in den Smart Materials [16], bei denen das verwendete Baumaterial (z. B. in Brücken oder Gebäuden) oder auch Farbe Sensorik enthält und Erneuerungsbedarf automatisch meldet.

2.5 Anforderungen an Sensornetze

Aus den im vorigen Abschnitt beschriebenen unterschiedlichen Einsatzgebieten resultieren ebenso diverse Anforderungen an die Hard- und Software von Sensornetzen. Dieser Abschnitt erläutert die speziell an Sensornetze gerichteten Anforderungen im Detail.

2.5.1 Geringe Kosten

Damit Sensornetze zum Einsatz kommen, müssen sie entweder preiswerter sein oder andere Vorteile gegenüber alternativen (z. B. kabelgebundenen) Lösungen haben. Aufgrund der avisierten großen Anzahl von Knoten in einem Sensornetz darf folglich auch jedes einzelne Gerät nur geringe Kosten verursachen. Dies bedeutet, dass die Hardwareentwickler der Sensorknoten unter einem hohen Kostendruck stehen. Die Hardwarebausteine müssen daher möglichst kostengünstig sein. Weiterhin dürfen weder die Entwicklung der Software noch die Installation oder die Instandhaltung des Netzes den Preis unverhältnismäßig in die Höhe treiben.

2.5.2 Selbstorganisation

Die Forderung nach einem geringen Installationsaufwand beim Ausbringen des Sensornetzes bedingt, dass die auf den Knoten ausgeführte Software weder geräte- noch ortsabhängige Parameter enthält. Ist ersteres der Fall, müsste jedes Gerät separat programmiert oder aktualisiert werden. Im zweiten Fall könnten die Geräte nicht an beliebigen Orten platziert werden. Jedes einzelne Gerät an einen speziellen Ort zu platzieren, ist jedoch besonders bei einer großen Anzahl von Geräten aufwändig. Die Sensorknoten müssen deshalb notwendige Kontext- und Routeninformationen selbst akquirieren. Man spricht deshalb von *Selbstorganisation*.

2.5.3 Geringe Baugröße

Häufig besteht die Forderung nach einer unauffälligen Integrierbarkeit der Geräte in die Umgebung. Dies setzt eine geringe Baugröße der Knoten voraus. Die Einsparung von Material kann außerdem zu einer Preissenkung führen. Kleinere Geräte können zudem flexibler eingesetzt werden, wenn – wie z. B. in Baumaterialien oder im menschlichen Körper – nur sehr wenig Platz zur Verfügung steht. Zum Ausbringen des Sensornetzes könnten die Knoten auch aus einem Flugzeug abgeworfen werden. Sie müssten dann entweder extrem leicht oder mit einem Fallschirm ausgestattet sein, damit sie beim Auftreffen auf den Boden nicht zerstört werden.

2.5.4 Skalierbarkeit

Die große Anzahl an Sensorknoten, aus der ein Netz bestehen kann, erfordert auch, dass die Software, die auf den Geräten läuft, skaliert. Algorithmen und Protokolle müssen auch funktionieren, wenn die Gesamtzahl an Knoten oder die Knotendichte sehr groß sind. Weder der Rechenaufwand noch der Bedarf an Speicherplatz dürfen im gleichen Maße steigen wie eine wachsende Anzahl der am Netz beteiligten Geräte. Konkret impliziert dies, dass ein Knoten gegebenenfalls nicht von allen Knoten im Netz Daten oder Routeninformationen gleichzeitig im Speicher vorhalten kann.

Weiterhin darf das Nachrichtenaufkommen pro Gerät nicht nennenswert mit einer wachsenden Geräteanzahl steigen. Dazu ist es erforderlich, dass die Protokolle und Algorithmen ausschließlich lokal arbeiten. Zentrale Verfahren haben neben dem hohen Datenaufkommen auch ein hohes Kommunikationsaufkommen und somit hohe Energiekosten zur Folge.

2.5.5 Kontext

Die grundlegende Aufgabe eines Sensornetzes, Daten zu erheben, kann bereits durch einfaches Auslesen von Sensoren erfüllt werden. Ohne weitere Informationen über Ort und Zeit sind Temperatur- oder Feuchtigkeitswerte jedoch wenig aussagekräftig. Damit aus gemessenen Daten Informationen werden, müssen diese deshalb mit Kontextinformationen versehen werden. Diese beziehen sich – wie beispielsweise Ort und Zeit – meist auf die eigene Situation eines Knotens; sie können aber auch aus dem Zustand von Nachbarknoten entstehen. In dieser Arbeit spielen Zeit- und Positionsbewusstsein eine Rolle, weshalb auf diese beiden Kontexte kurz eingegangen wird.

Ein *konsistentes Zeitbewusstsein* liegt dann im Netz vor, wenn die Uhren aller Knoten nahezu die gleiche Zeit anzeigen, das heißt weniger als eine bekannte Differenz voneinander abweichen. Für ein fehlendes konsistentes Zeitbewusstsein gibt es mehrere Ursachen: Zum einen haben die Knoten meist nach der Ausbringung verschiedene Startzeiten gespeichert

oder wurden zu unterschiedlichen Zeitpunkten angeschaltet; zum anderen laufen ihre Uhren unterschiedlich schnell. Zur Herstellung eines konsistenten Zeitbewusstseins wurden daher zahlreiche *Zeitsynchronisationsverfahren* vorgeschlagen (siehe [3, 19, 32, 35, 62, 66, 73, 77, 78, 82, 104]).

Bei vielen dieser Verfahren teilen Knoten ihre lokale Uhrzeit anderen Knoten mit, indem sie sie in einer Nachricht versenden. Allerdings kommt es hierbei durch die Erzeugung der Nachricht, den Medienzugriff, die Übertragung und die Verarbeitung auf Empfängerseite zu einem zeitlichen Versatz zwischen dem Generieren auf Senderseite und dem Verarbeiten der Nachricht auf Empfängerseite. Ein einfaches Setzen der Uhr auf die in der Nachricht enthaltene Zeit kann daher zu Abweichungen im Bereich von mehreren Dutzend Millisekunden führen. Die Knoten können versuchen, den entstehenden Zeitversatz herauszufinden und herauszurechnen. Alternativ können auch zwei Nachrichten mit einem definierten Abstand gesendet werden. Li und Rus schlagen in [70] vor, die von allen Nachbarn eingegangenen Zeiten zu mitteln und als eigene Zeit zu setzen. Durch mehrmaliges Senden und Mitteln konvergieren die Uhrzeiten der Knoten auf einen Wert. Übliche Synchronisationsgenauigkeiten liegen im Mikro- bis Millisekundenbereich.

Positionsbewusstsein ist die Kenntnis der eigenen Position in numerischer Form in Bezug auf ein Koordinatensystem oder in logischer Form bezüglich einer symbolischen Referenz.

Symbolische Positionen wie beispielsweise „Vitrine A in Raum X“ verwenden semantische Informationen, die allerdings häufig nicht selbstorganisierend erzeugt werden können. [34] stellt ein Verfahren zur Erkennung von Randknoten vor, welches in [52] in einem Straßennetz angewendet wird, um zwischen Lage an einer Kreuzung oder in einer Straße zu unterscheiden.

Um ein Positionsbewusstsein in Koordinatenform bei den Knoten zu etablieren, werden Knoten benötigt, die bereits ihre Position kennen, so genannte *Anker*. Mit Hilfe eines Distanzschätzverfahrens ermitteln die übrigen Knoten Distanzen zu mehreren Ankern, aus denen sie die eigene Position ableiten (vgl. [98]). In dieser Arbeit wird jedoch nicht von der Existenz solcher Anker ausgegangen, denn relative Positionen können auch ohne Anker, nur mit Distanzschätzverfahren erzielt werden.

Die in der Literatur zu findenden Distanzschätzverfahren lassen sich wie folgt kategorisieren: Einerseits gibt es Verfahren, die zusätzliche oder spezielle Hardware verwenden, um Signallaufzeiten (ToA – Time of Arrival und TDoA – Time Difference of Arrival) oder die Einstrahldauer zu messen. Neben der zusätzlichen, teuren Hardware ist hier teilweise auch Sichtkontakt zwischen den Geräten notwendig.

Andererseits gibt es Verfahren, die auf der Basis der Funkschnittstelle zu einer Distanzschätzung gelangen. Viele Autoren (vgl. [10, 15, 96]) versuchen aus der Signalabschwächung (RSS – Received Signal Strength) eine Distanz abzuleiten. Gerade in Gebäuden variiert die Signalabschwächung jedoch sehr stark in Abhängigkeit von örtlichen

und aktuellen Gegebenheiten, so dass die Schätzfehler zwischen 10 % und 100 % der Kommunikationsreichweite liegen (siehe auch [11, 31, 79, 97]).

Alternativ kann die Tatsache, ob Kommunikation möglich ist oder nicht, Rückschlüsse auf die Entfernung ziehen lassen. Diese Methode wird in [12] untersucht und wurde in dieser Arbeit verwendet. Sie beruht auf der Annahme, dass weiter entfernte Knoten weniger Nachbarn gemein haben als solche, die nah beieinander sind. Auf Basis von Vergleichen von Nachbarschaftsmengen werden Distanzen geschätzt. Dieses Vorgehen hat den Vorteil, dass die Annahme üblicherweise auch in Gebäuden gilt und die Tatsache, ob Kommunikation möglich ist oder nicht, nicht graduell beeinflusst werden kann.

2.5.6 Struktur

Ein weit verbreiteter Begriff im Bereich der Sensornetze ist die *Netzwerktopologie*. Sie bezieht sich allerdings ausschließlich auf die im Netz vorherrschenden Nachbarschaftsbeziehungen. Eine *Struktur* (auch: Gliederung) kann dagegen auch durch Kontextinformationen entstehen. Mit *Struktur* wird „die Anordnung der Teile eines Ganzen zueinander“, „innere Gliederung“ oder auch (ggf. „erhabene“) „Musterung“ bezeichnet (siehe [6]).

Eine Struktur kann zum einen eine spezielle räumliche Anordnung der Knoten sein, z. B. vorgegeben durch die Platzierung von Knoten in Gebäuden oder entlang einer langen Grenze. Zum anderen können die Kontexte der Knoten eine innere Gliederung herbeiführen. Beispiele für solche Kontexte sind:

- Alle Knoten, bei denen ein Messwert eine bestimmte Schwelle überschritten hat, bilden eine Gruppe.
- Alle Knoten, die weniger als x Volt Spannung in ihrem Akku haben, bilden eine Gruppe.
- Alle Knoten mit der gleichen Hardware bilden eine Gruppe.

So können die Heterogenität eines Netzes, die unterschiedliche Ausstattung von Knoten mit Sensorik und verschiedene Aufgaben oder Zustände der Knoten zu unterschiedlichen Rollen führen. Diese Arbeit beschäftigt sich daher mit der Frage, wie Gliederungen erfasst und effektiv in Protokollen genutzt werden können.

2.6 Eigenschaften von Sensornetzen

Aus den im vorigen Abschnitt beschriebenen Anforderungen an Sensornetze ergeben sich spezielle Eigenschaften, die beim Design von Sensornetzprotokollen und -anwendungen bedacht werden müssen. Zum Verständnis der weiteren Kapitel stellt dieser Abschnitt die Besonderheiten vor, die bei Sensornetzen zu berücksichtigen sind.

2.6.1 Geringer Energievorrat

Die Forderung nach geringer Größe und Kosten erfordert den Einsatz von möglichst kleinen und kostengünstigen Energiespeichern. Hierdurch ergibt sich jedoch, dass ein Sensorknoten nur einen begrenzten Energievorrat besitzt und nur wenig Energie auf einmal speichern kann.

Selbst wenn ein Knoten über eine Solarzelle zur Stromversorgung verfügt, begrenzt die Größe des Energiespeichers die Aufnahmefähigkeit von Energie. Die Solarzellen dürfen außerdem weder große Ausmaße haben, noch werden sie andauernd von der Sonne beschienen (z. B. im Schatten und während der Nacht). [57] zeigt, von welchen Parametern die Energieentwicklung abhängt, und schlägt ein Modell vor, wie diese vorhergesagt werden kann.

Auch in Gebäuden ist nicht immer eine Netzstromversorgung verfügbar. Während in einer Indoor-Testumgebung eine Netzstromversorgung noch für jeden Knoten realisierbar sein mag, kann es für den kommerziellen Einsatz bereits zu teuer sein, nachträglich zu einem oder zu allen Knoten Stromkabel zu verlegen.

Aus diesem Grund ist neben der Verwendung von Hardwarekomponenten, die einen geringen Energieverbrauch haben, das Energiemanagement eine zentrale Herausforderung in Sensornetzen (vgl. [108]). Als Lösung eignet sich die Reduzierung der Aktivität über die Zeit. Der Einsatz unterschiedlicher, sich abwechselnder Aktivitätszustände wird *Duty-Cycling* genannt. Hierbei gibt es typischerweise zwei Modi, den Schlaf- und den Wachmodus. Im Schlafmodus sind – bis auf einen Timer und gegebenenfalls Sensoren zur beständigen Phänomendetektion[1] – alle Hardwarekomponenten ausgeschaltet. Läuft der Timer ab und aktiviert dadurch den Knoten, wechselt dieser in den Wachzustand, in dem alle Komponenten eingeschaltet sind. Nach einer gewissen Zeit wechselt der Knoten dann wieder in den Schlafmodus. Aus Tabelle 2.1 lässt sich ableiten, dass die Funkschnittstelle einen Großteil des Energieverbrauchs verursacht, so dass es zum Stromsparen insbesondere wichtig ist, das Funkmodul auszuschalten.

Mit *Duty-Cycle* wird der Anteil der Zeit bezeichnet, den ein Knoten wach ist, bezogen auf die gesamte Zeit.

Da während der Schlafphase aufgrund des inaktiven Funkmoduls und Prozessors kein Nachrichtenaustausch möglich ist, müssen die Wachphasen benachbarter Knoten, die Nachrichten miteinander austauschen sollen, koordiniert werden. Auf mögliche Lösungen zu diesem Problem wird in Abschnitt 2.9 detailliert eingegangen.

Zu beachten ist hierbei, dass häufig ein möglichst gleichmäßiger Stromverbrauch von allen Sensorknoten erwünscht ist, damit nicht einige Knoten früher ausfallen als andere und

[1] Einige Sensoren sind in der Lage, den Prozessor durch einen Interrupt zu aktivieren. Ist dies nicht der Fall, muss eine kontinuierliche sensorische Abdeckung durch abwechselnde Wachphasen mehrerer Sensorknoten gewährleistet werden.

weder die sensorische Abdeckung lückenhaft ist, noch das Netz in mehrere Partitionen zerfällt.

Zu bedenken gilt es auch, dass es trotz aller Energiesparmaßnahmen passieren kann, dass die verbleibende Energie nicht mehr zum Betreiben des Gerätes ausreicht, so dass es zum Knotenausfall kommt. Protokolle und Anwendungen dürfen sich durch vereinzelte Knotenausfälle nicht beeinträchtigen lassen.

2.6.2 Geringe Rechenleistung und Speicherkapazität

Die Forderungen nach einer geringen Größe und geringen Kosten der Sensorknoten wirken sich jedoch nicht nur auf die Komponenten zur Energieversorgung, sondern auch auf die CPU und den Speicher der Knoten aus. Kostengünstigere Prozessoren haben meist eine geringere Taktung, so dass sie für Rechenoperationen mehr Zeit benötigen. Eine geringe Rechenleistung impliziert beispielsweise, dass keine komplizierten Verschlüsselungsverfahren möglich sind oder das Ver- und Entschlüsseln einige Sekunden dauert.

Der Speicher umfasst meist weniger als 100 KByte. Ein kleiner Speicher hat zur Folge, dass keine großen Datenmengen an Messwerten (z. B. Videos) oder Schlüsselmaterial gespeichert werden können. Ferner können die eingesetzten Protokolle nicht unbegrenzt Informationen über Nachbarn, Linkqualitäten oder Routen vorhalten. Dies muss beim Design der Protokolle berücksichtigt werden.

2.6.3 Zuverlässigkeitsdefizite

Naturgemäß bestehen Netze aus Einzelgeräten, die nicht über einen gemeinsamen Speicher verfügen, sondern über Nachrichtenaustausch kommunizieren. Das besondere Problem der drahtlosen Kommunikation ist jedoch, dass sie nicht zuverlässig ist, da sie von vielen verschiedenen Faktoren abhängt.

Zunächst ist die Ausbreitung von Signalen nicht in alle Richtungen gleich stark und endet nicht abrupt bei einer bestimmten Distanz, wovon häufig in der Literatur ausgegangen wird. Sie ist jedoch richtungsabhängig und die Signalstärke nimmt quadratisch mit der Distanz ab. Dadurch sinkt die Wahrscheinlichkeit einer erfolgreichen Nachrichtenübertragung mit wachsender Distanz zwischen Sender und Empfänger. Reflexionen und Mehrwegeausbreitung des Signals führen dazu, dass sich ein Signal auch selbst auslöschen kann. Eine Folge ist, dass Kommunikationsverbindungen unidirektional sein und auch über die Zeit variieren können. Ganesan et al. [36] untersuchten das Verhalten von einfachen und kostengünstigen Funkschnittstellen in einer experimentellen Studie und kamen zu diesen Ergebnissen.

Überlagerte Signale können von einem Empfänger nicht mehr korrekt demoduliert werden, so dass ein Paket nicht empfangen wird, wenn das Signal-zu-Rauschverhältnis zu gering ist. Da Signale jedoch auch nur bis zu einer begrenzten Distanz registriert werden können, kann es passieren, dass zwei Sender die Signale des jeweils anderen nicht detektieren (Knoten A und C in Abbildung 2.6) und deshalb gleichzeitig eine Nachricht versenden, von denen aber keine vom in der Mitte befindlichen potentiellen Empfänger (Knoten B) demoduliert werden kann. Dieses Szenario wird als *Hidden-Terminal-Problem* bezeichnet.

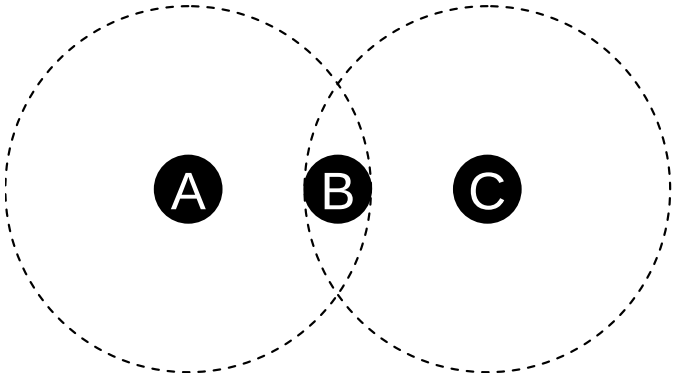


Abbildung 2.6: Hidden-Terminal-Problem: B empfängt keine der Nachrichten von A und C, wenn diese gleichzeitig senden.

Nicht nur in der Kommunikation gibt es Zuverlässigkeitsdefizite. Auch preiswerte Sensorik führt zu einer Fehlerquote in der Detektion von Phänomenen sowie zu Messfehlern. Dies muss beispielsweise durch Redundanz oder Fehlerrobustheit ausgeglichen werden.

Ein weiteres Problem entsteht durch die Eigenschaft von Oszillatoren, nicht mit der gewünschten Taktgeschwindigkeit, sondern zu schnell oder zu langsam zu laufen. Die Uhren zweier Knoten werden deshalb hardwarebedingt nicht exakt gleich laufen und auseinander driften. Wie stark eine Uhr maximal driftet, ist durch die Hardwarespezifikation beschrieben, so dass in einem begrenzten Zeitraum nur eine begrenzte Diskrepanz zwischen zwei Uhren entstehen kann. Dennoch muss auch diesem Umstand bei allen eingesetzten Protokollen Rechnung getragen werden.

2.7 Simulation von Sensornetzen

Die Entwicklung funktionierender und effizienter Software für eingebettete, verteilte Systeme ist außerordentlich schwierig. Das größte Problem besteht darin, dass die Einzelsysteme nur einfachste Ausgabeschnittstellen wie z. B. LEDs besitzen. Diagnoseinformationen stehen nicht von vornherein zentral zur Verfügung, sondern müssen erst zu einer Senke

transportiert werden. Heterogenität, die Größe von Sensornetzwerken und variierende Umwelteinflüsse erschweren die Softwareentwicklung zusätzlich.

Da in Sensornetzen somit auch nicht die Möglichkeit besteht, eine Ablaufverfolgung des Programms durchzuführen, wurden Netzwerksimulatoren entwickelt. Diese bilden das gesamte Sensornetz an zentraler Stelle nach. So stehen Informationen von allen Knoten zur Verfügung, ohne dass das untersuchte Protokoll durch zusätzlichen Nachrichtenverkehr beeinträchtigt wird. Weiterhin ermöglichen sie, definierte Szenarien zu erzeugen, in denen die Protokolle erprobt und verifiziert werden. Tests werden somit reproduzierbar. Über die Variation von einzelnen Parametern wird das Finden von Ursachen für das Verhalten von Protokollen stark vereinfacht. Eine unerwartet niedrige Ankunftsrate beispielsweise kann diverse Gründe haben. Sie kann sowohl durch eine zu hohe Netzlast oder Engpässe bei einem Knoten als auch durch veränderte Umgebungseigenschaften verursacht werden. Simulatoren ermöglichen aber insbesondere auch, Erkenntnisse über das asymptotische Verhalten von Protokollen zu gewinnen, das heißt die Entwicklung der Leistungsfähigkeit für eine wachsende Dichte oder Anzahl von Knoten.

Die Qualität und Aussagekraft von Simulationen hängt jedoch maßgeblich von der Qualität der verwendeten Modelle, die die Realität nachbilden, und deren Parametrisierung ab.

In dieser Arbeit wurde der Simulator *Shawn* [33] eingesetzt, da er im Gegensatz zu anderen Simulatoren wie *NS-2* [115] oder *TOSSIM* [68] auch größere Netzwerke mit akzeptablem Zeitaufwand simuliert und die Realität mit einstellbarer Abstraktion abbildet. Ein weiterer Vorteil von Shawn ist, dass eine Shawn-Treiberimplementierung für das Betriebssystem existiert, das in dieser Arbeit bei Experimenten verwendet wurde. So kann nahezu derselbe Code, der auf den Geräten zum Einsatz kommt, simuliert werden.

In den beiden folgenden Abschnitten wird erläutert, wie die Realität in Shawn abgebildet wurde.

2.7.1 Modellierung von Prozessorgeschwindigkeit

Die Ausführung desselben Befehls kann abhängig von der Hardwareplattform unterschiedlich lange dauern. Die Ausführungsdauer von Befehlen beeinflusst die Gesamtablaufdauer eines Protokolls jedoch nachhaltig. Basierend auf Messungen auf der iSense-Hardwareplattform wurden kurze Arbeitsschritte, wie z. B. das Verarbeiten von Nachrichten, mit einer geringen mittleren Dauer von 1,2 ms simuliert.

Für Arbeitsschritte, die überdurchschnittlich viel Zeit in Anspruch nehmen, wie z. B. das Lesen und Schreiben des Flash-Speichers, wurde die jeweilige Dauer simuliert. Das Schreiben von 32 Byte dauerte beispielsweise 145000 Takte, was bei einem 16 MHz Prozessor etwa 9 ms entspricht.

2.7.2 Modellierung der Kommunikation

Geht man davon aus, dass sich Funksignale in alle Richtungen gleichmäßig ausbreiten, entsteht um einen sendenden Knoten ein kugel- bzw. kreisförmiger Bereich, in dem das Signal empfangen werden kann. In der Literatur wird der *Kommunikationsbereich* eines Knoten sehr häufig mittels solch einer Scheibe mit dem Radius r abstrahiert. Dieses Modell wird als *Unit disk graph*-Modell bezeichnet. r ist der so genannte *Kommunikationsradius* und normalerweise invariant über die Zeit. Ist r zusätzlich für alle Knoten gleich groß, so gibt es ausschließlich bidirektionale, dauerhafte Links. Befindet sich ein Knoten innerhalb des Kommunikationsbereichs, kann er Pakete empfangen, außerhalb nicht.

Experimente mit unterschiedlicher Hardware haben jedoch gezeigt, dass sich Signale nicht richtungsunabhängig ausbreiten [36]. Zhou et al. entwickelten vor diesem Hintergrund das *Radio Irregularity Model* (RIM) [131], bei dem die Kommunikationsreichweite zusätzlich zur Distanz winkelabhängig ist, so dass sie von der Lage des potentiellen Empfängers zum Sender abhängt. Der Kommunikationsbereich ist dann keine einfache Scheibe mehr, sondern eine „ausgefranste“ Scheibe oder eine Art ungleichmäßiger Stern. Der Grad der Ungleichmäßigkeit lässt sich über die maximale (prozentuale) Veränderung der Reichweite pro Winkelgrad (DOI – Degree of irregularity) einstellen. Basierend auf [132] wurde für alle Simulationen, in denen das RI-Modell zum Einsatz kam, $DOI = 0,01$ als realistischer Wert gesetzt.

Das RI-Modell führt zwar zu einem gewissen Anteil an unidirektionalen Kommunikationsverbindungen, aber es berücksichtigt andere Charakteristiken von Funkübertragung nicht. Einerseits nimmt die Empfangswahrscheinlichkeit einer Nachricht aufgrund der sinkenden Signalstärke mit wachsender Distanz zwischen Sender und Empfänger ab; andererseits variiert die Empfangswahrscheinlichkeit aufgrund von schwankenden Umgebungsparameterwerten auch über die Zeit.

Diese Tatsachen werden vom *stochastischen Kommunikationsmodell* [133] nachgebildet, welches von Zuniga und Krishnamachari vorgeschlagen wurde. Bei diesem Modell gibt es zwei spezielle Radien r_1 und r_2. Für alle Distanzen $d < r_1$ vom Sender beträgt die Empfangswahrscheinlichkeit konstant p_{max}, wobei für p_{max} von den beiden Autoren 100% vorgesehen ist. Für $r_1 < d < r_2$ sinkt die Wahrscheinlichkeit linear von p_{max} auf 0. Für alle Distanzen $d > r_2$ bleibt die Wahrscheinlichkeit 0.

Um herauszufinden, welche Werte die Eigenschaften der in dieser Arbeit verwendeten Sensorknoten am besten abbilden, wurden am Institut Experimente durchgeführt. Dabei wurde die Ankunfsrate in Abhängigkeit von der Distanz zwischen Knoten ermittelt. Abbildung 2.7 zeigt die im Freifeld gemessenen Empfangsraten über eine variierende Distanz sowie das stochastische Kommunikationsmodell mit den Werten $p_{max} = 98\,\%$, $r_1 = 28\,\mathrm{m}$ und $r_2 = 37{,}5\,\mathrm{m}$. Diese Werte stellen eine recht gute Approximation der Realität dar, weshalb sie beim Einsatz des stochastischen Modells verwendet wurden.

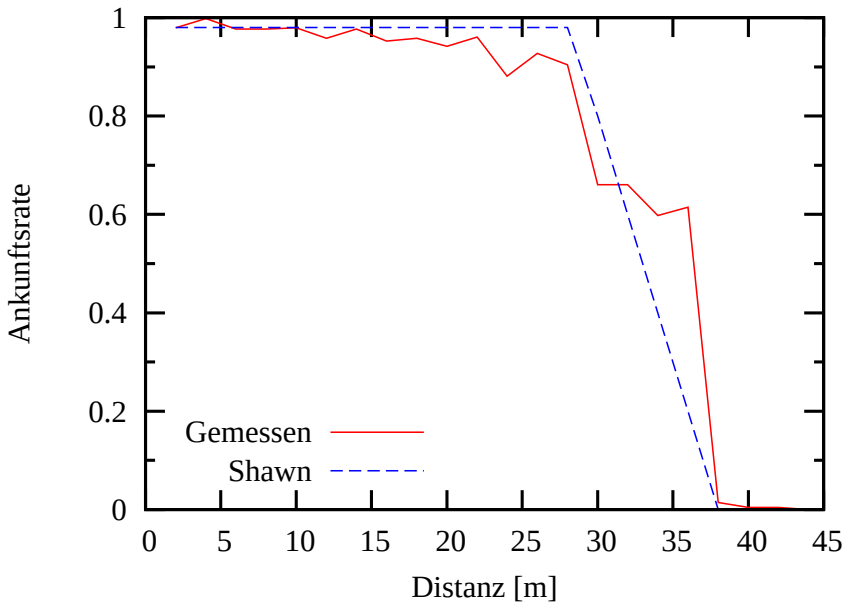


Abbildung 2.7: Paketankunftsrate beim 802.15.4 CSMA/CA Radio und dem stochastischen Simulationsmodell

Um nutzbringende Simulationen durchführen zu können, stellt sich die Frage, welche Netzdichten sinnvoll sind. Generell ist es erstrebenswert, die Dichte möglichst gering zu halten, um mit dem Netz eine große Fläche abzudecken und die Kosten niedrig zu halten. Hellbrück et al. [41] konnten jedoch zeigen, dass mobile Ad-hoc Netze mit großer Wahrscheinlichkeit dazu neigen, in mehrere Partitionen zu zerfallen, wenn die Dichte weniger als 10 Knoten beträgt. Xue et al. beweisen in [123], dass die benötigte Dichte, um ein voll verbundenes Netz zu erhalten, bei wachsender Größe des Netzes logarithmisch ansteigt. Bei 300 zufällig verteilten Knoten reichen im Mittel 13 Nachbarn, bei 100 Knoten bereits 11. Um von noch pessimistischeren Voraussetzungen auszugehen, wurden im Weiteren Dichten ab 8 Nachbarn als sinnvoll erachtet.

Umgekehrt ist häufig auch die Reichweite der Sensoren deutlich geringer als die Kommunikationsreichweite, so dass bei einer möglichst hohen Sensorabdeckung automatisch eine hohe Dichte entsteht.

2.8 FRONTS-Testumgebung

Simulationen helfen dem Entwickler, Programmierfehler zu finden und die prinzipielle Funktionsfähigkeit und Skalierbarkeit eines Protokolls zu überprüfen. Leider können Simulatoren jedoch nicht alle Aspekte des Systems und der realen Welt abbilden. Be-

sonders die zeitlichen und parallelen Abläufe sowie die Signalausbreitung sind schwer zu modellieren.

Aus diesem Grund werden Protokolle typischerweise nach dem Test im Simulator mit einer sehr kleinen Anzahl von Geräten und anschließend in einer festinstallierten Testumgebung oder mittels Prototyp erprobt, sofern die Möglichkeiten dazu gegeben sind.

Experimente dienen im ersten Schritt der Validierung der Algorithmen. Die in den Experimenten gewonnenen Daten helfen aber auch, das ursprüngliche Design weiter zu verbessern und zu optimieren. Testumgebungen müssen dazu sinnvoll aufgebaut und möglichst auch simuliert werden können. Im Folgenden wird die im Rahmen dieser Arbeit eingesetzte FRONTS-Testumgebung beschrieben.

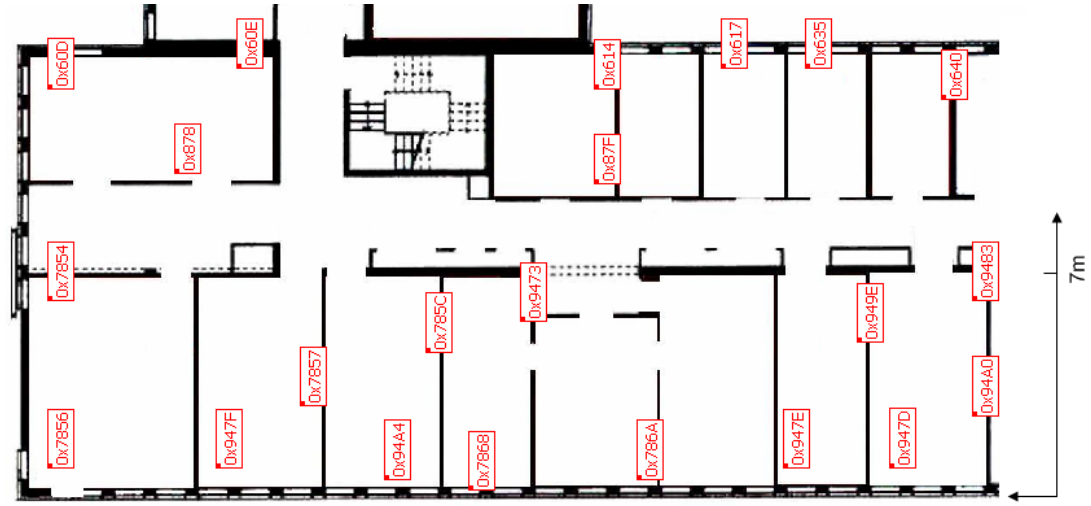


Abbildung 2.8: Knotenposition im Institut für Telematik

Die Testumgebung des Projektes FRONTS befindet sich im Institut für Telematik und besteht aus 22 netzbetriebenen iSense Sensorknoten. Abbildung 2.8 zeigt die Anordnung der Knoten im Institut. Knoten sind durch ein Rechteck, welches ihre ID enthält, dargestellt. Knoten 0x94a0 ist über einen seriellen Anschluss mit einem PC verbunden, so dass der Anwender oder Entwickler mit Hilfe des Verwaltungs- und Visualisierungstools *iShell* [18] von einem beliebigen Ort auf sie zugreifen kann. Dieser Knoten dient deshalb als Senke. Die Knoten wurden im gesamten Institut verteilt, um einen möglichst großen Netzwerkdurchmesser zu erzielen.

Um eine Idee davon zu bekommen, inwiefern die Empfangsrate der eingesetzten Sensorknoten durch die Wände beeinträchtigt wird, wurde diese über mehrere Tage hinweg zwischen allen Knotenpaaren aufgezeichnet. Abbildung 2.9 zeigt die gemittelten Ergebnisse. Die Kommunikationsreichweite ist nur etwa halb so groß wie im freien Feld, aber die Netzwerkdichte ist mit 12 bis 16 Nachbarn trotzdem recht hoch.

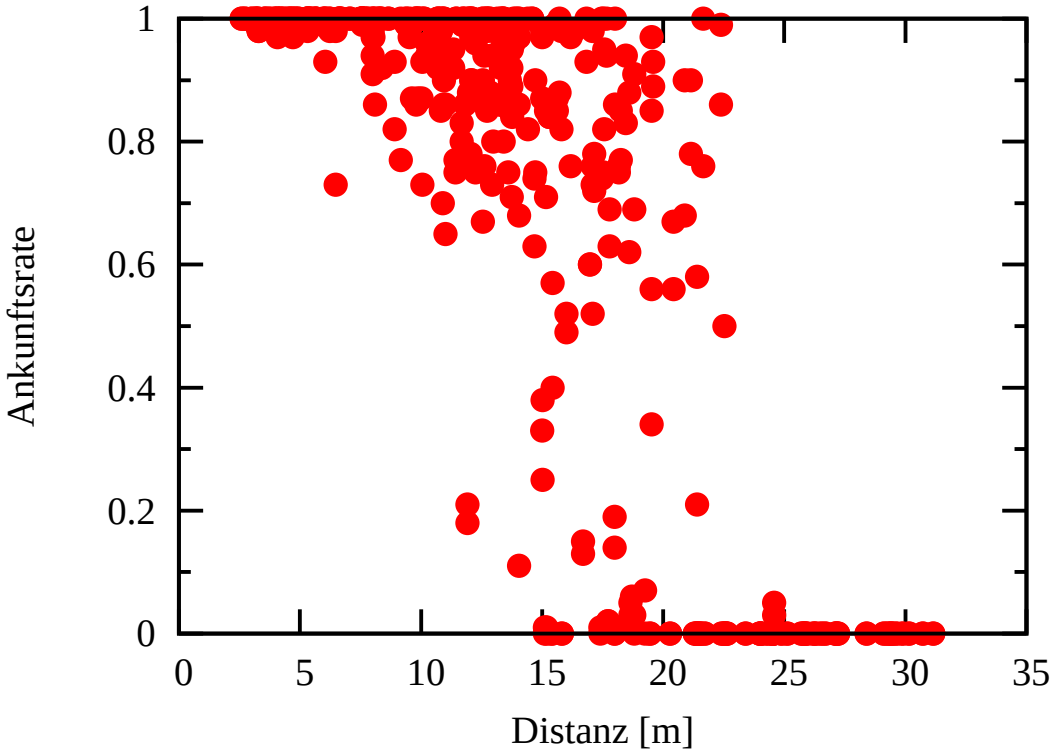


Abbildung 2.9: Paketankunftsrate in der FRONTS-Testumgebung

In Abbildung 2.10 ist für alle Raten größer als 80% in 5%-Schritten dargestellt, wie häufig eine bestimmte Rate vorkam. Beim Zählen wurde zusätzlich nach der Anzahl der Wände, die sich jeweils zwischen den Knoten befanden, gruppiert. Es ist zu erkennen, dass die Ankunftsraten bei mindestens 95 % liegen, falls höchstens eine Wand zwischen Sender und Empfänger steht. Auch bei zwei Wänden gibt es viele Kommunikationsverbindungen mit einer ähnlich hohen Paketankunftsrate. Befinden sich drei oder mehr Wände zwischen zwei Knoten, variieren die mittleren Raten sehr stark. Insgesamt lassen sich die Werte dementsprechend nur ungenau mit dem stochastischen Kommunikationsmodell approximieren. Wegen der vielen Wände, Türen und variierenden Umgebungseigenschaften (z. B. durch WLAN-Aktivität oder sich bewegende Personen) hängen die Ankunftsraten nicht nur von der Distanz ab.

2.9 Duty-Cycle-Verfahren

Wie bereits in Abschnitt 2.6 beschrieben, haben Sensorknoten aufgrund ihrer geringen Größe nur einen kleinen Energievorrat, weshalb es nötig ist, zwischen verschiedenen

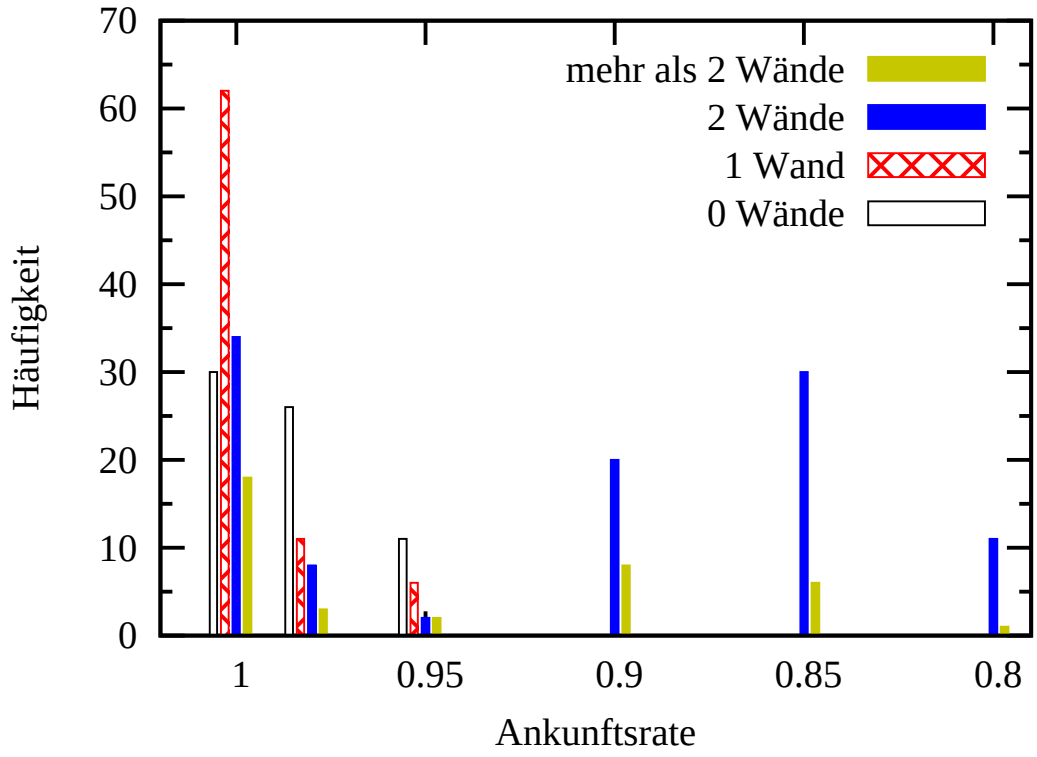


Abbildung 2.10: Häufigkeit der Paketankunftsraten

Energiesparmodi zu wechseln, also ein Duty-Cycle-Verfahren einzusetzen. Aufgrund der Tatsache, dass die empfangsbereite Radiokomponente und der Betrieb des Prozessors auf allen Hardwareplattformen einen Großteil zum Energieverbrauch beitragen, ist es sinnvoll, beide Komponenten gleichzeitig in den Energiesparmodus zu versetzen.

Die folgenden beiden Abschnitte stellen verschiedene Möglichkeiten des Duty-Cyclings vor. Abschnitt 2.9.1 geht dabei auf Verfahren ohne Zeitsynchronisation, Abschnitt 2.9.2 auf Verfahren mit Zeitsynchronisation ein.

2.9.1 Verfahren ohne Synchronisation

Long/Short Preamble Sampling Einer der ersten Vorschläge, wie trotz einer inaktiven Phase von Knoten Nachrichten übertragen werden können, war das so genannte *LPL* (Low Power Listening [48]). Hierbei wechseln sich bei jedem Knoten eine Phase der Dauer D_{aktiv}, in der die Funkschnittstelle aktiv ist, mit einer inaktiven Phase der Dauer $D_{inaktiv}$ ab. Da kein Synchronisationsmechanismus zum Einsatz kommt, weiß ein Sender nicht, wann der gewünschte Empfänger sein Funkmodul aktivieren wird. Die maximale

Wartezeit beträgt $D_{inaktiv}$. Um anzuzeigen, dass ein Knoten ein Paket senden möchte, sendet er für die Dauer $D_{inaktiv}$ eine (lange) Präambel. Während dieser Zeitspanne aktivieren alle Knoten, die die Präambel hören, ihr Funkmodul und lassen es aktiv. Nach dem Senden der Präambel folgt das eigentliche Paket. Nachteil dieser Methode ist, dass Sender und Empfänger in jedem Fall für die Dauer von $D_{inaktiv}$ Energie verbrauchen, selbst wenn der Empfänger bereits zu Beginn des Sendens der Präambel aktiv war. Weiterhin verschwenden die Nachbarn Energie, da die Präambel keine Zieladresse enthält und sie somit auch aktiv bleiben, obwohl sie nicht adressiert sind. Im IEEE 802.15.4 Standard wurde zudem definiert, dass nur komplette Pakete empfangen werden, was die Verwendung einer solchen Präambel unmöglich macht, da die Knoten hierbei in der Lage sein müssen, Bitströme zu empfangen.

Dies führte zu den Verfahren *X-MAC* [9] und *WiseMAC* [30], bei denen mehrere extrem kurze Präambeln versendet werden, die die Zieladresse enthalten. Bei X-MAC wartet der Sender zwischen dem Senden zweier Präambeln so lange, dass der Empfänger genau ein Acknowledgement senden kann, um anzuzeigen, dass er empfangsbereit ist. Die kurzen Präambeln mit Adressen senken den Energieverbrauch drastisch, da nur der Zielknoten aktiv bleibt und es nur wenig länger dauert als das Minimum, bis das eigentliche Paket versendet werden kann. Außerdem können kurze Präambeln durch paketorientierte Radios nachgeahmt werden, indem sie als reguläre Nachricht versendet werden. Allerdings führt der Acknowledgementmechanismus zu Problemen bei der Broadcast-Kommunikation, wenn die Anzahl der Empfänger hoch oder sogar unbekannt ist.

Diese Verfahren, bei denen Geräte auf Anfrage (z.B. mittels Präambel) wach bleiben, sind anfällig für so genannte *Schlafentzugsangriffe*. Hierbei versucht ein Angreifer seitens der Geräte einen großen Energieverbrauch zu erzwingen, so dass die Geräte außer Betrieb geraten, ihre Aufgabe nicht mehr durchführen können und den Angreifer nicht bemerken. Aus diesem Grund eignen sich solche Protokolle nicht für sicherheitskritische Szenarien. Bedarfsorientierte Verfahren erfordern zudem einen größeren Energiespeicher, weil die maximalen Längen von Schlaf- und Wachphasen größer sind. Dies wirkt sich negativ auf die Baugröße des Gerätes aus.

Zufallsbasierte Aufwachzeiten Zheng et al. beschreiben in [129] ein Verfahren, bei dem die Knoten zu Zeitpunkten aktiv werden, die auf Zufallsvariablen basieren. Eine solche Vorgehensweise erfordert zwar keine synchronen Uhren, aber die Ende-zu-Ende-Verzögerung von Nachrichten ist hier gerade bei großen Distanzen extrem hoch. Zufällige Aufwachzeiten eignen sich bestenfalls für mobile Netze und die meisten Autoren nehmen deshalb die Verwendung eines Zeitsynchronisationsprotokolls an.

Wake-up Radios Bei *On-demand Wake-up* Radios [39, 100] verfügen die Geräte über zwei Radiokomponenten. Ein Radiointerface ist hierbei fähig, ein zweites Radiointerface für den Nachrichtenempfang zu aktivieren, sobald es ein Signal empfängt. Ein Problem

bei der Frequenz von 2,4 GHz ist jedoch, dass sie auch von anderen Funktechnologien wie WLAN verwendet wird, was dazu führt, dass das erste Radiointerface bereits sehr viel Strom verbraucht. Die erreichbaren Energieersparnisse legitimieren deshalb nicht die zusätzlichen Kosten und negativen Auswirkungen auf die Baugröße.

2.9.2 Verfahren mit Synchronisation

Gleichzeitiges Wachen Zu den Duty-Cycle-Verfahren, bei denen Knoten synchron wach sind und ihre Funkschnittstelle aktiviert haben, gehören *S-MAC* [125], *T-MAC* [22] und *P-MAC* [130].

Beim S-MAC (Sensor-MAC) wachen und schlafen Geräte möglichst synchron jeweils für eine feste Zeitdauer. Der Medienzugriff erfolgt während der Wachphase, koordiniert nach dem RTS/CTS (Ready To Send/Clear To Send [44]) Mechanismus. Er reduziert die Kollisionswahrscheinlichkeit, indem der Quellknoten zunächst eine RTS-Nachricht sendet. Der Zielknoten antwortet bei Empfang des RTS mit einer CTS-Nachricht. Nach Empfang des CTS sendet der Quellknoten schließlich die Nachricht. Andere Knoten, die die CTS-Nachricht gehört haben, werden für eine gewisse Zeit nichts senden. Dieser Mechanismus ermöglicht allerdings nur Unicast-Kommunikation, das Senden einer Nachricht an genau einen Knoten.

T-MAC (Timeout-MAC) verwendet den gleichen Mechanismus, reduziert aber den Energieverbrauch durch vorzeitiges Beenden der aktiven Phase für den Fall, dass innerhalb einer bestimmten Zeit kein RTS empfangen wird.

In P-MAC (Pattern-MAC) sendet ein Knoten zunächst (während einer so genannten *Musteraustauschphase*) Informationen über sein nachfolgendes Sendemuster. In der sich anschließenden Musterwiederholungsphase sendet er dann Anwendungsdaten. Insbesondere bei regelmäßigem Datenverkehr ist der Overhead des Musteraustauschs allerdings unnötig, da im Vorwege bekannt ist, wann gesendet werden soll.

Bei diesen Verfahren kann es zu großen Multihop-Verzögerungen kommen, da jeder Knoten warten muss, bis der nächste Zielknoten empfangsbereit ist. Des Weiteren besteht auch hier der Nachteil, dass die Protokolle anfällig für Schlafentzugsangriffe sind, weil Knoten auf Anfrage wach bleiben.

Gestaffeltes Wachen Viele Autoren, die sich mit Duty-Cycle-Verfahren befassen, gehen davon aus, dass Knoten- und Sensorenaktivität nicht getrennt voneinander gesteuert werden können und die Sensoren des Knotens somit nur aktiv sind, wenn der Knoten aktiv ist (z. B. [110]). Ihre Verfahren werden als *k-coverage Schemes* bezeichnet. Die vorliegende Arbeit betrachtet diese Art von Verfahren allerdings nicht, da angenommen wird, dass Sensoren den Prozessor durch einen Interrupt aufwecken können.

Die Quellen, die ebenfalls von getrennt steuerbarer Hardware ausgehen und mit gestaffeltem Aufwachen arbeiten, werden detailliert in Abschnitt 4.2 behandelt. Der nun folgende Abschnitt führt in die Thematik der Weiterleitung von Nachrichten ein und beschreibt relevante Routingprotokolle im Detail.

2.10 Routingprotokolle

Routingprotokolle bestimmen in Rechnernetzen die Wege von Daten bei der Nachrichtenübermittlung. Ziel dabei ist es, einen Mechanismus zum Übertragen einer Nachricht von einem beliebigen (Quell-) Knoten zu einem beliebigen (Ziel-) Knoten zur Verfügung zu stellen. Das Routing verwendet dazu Datenstrukturen (z. B. Tabellen), die Informationen enthalten, wie ein Zielknoten erreicht werden kann. Die Informationen können entweder an zentraler Stelle vorhanden und in der zu übertragenden Nachricht enthalten sein oder jeder Knoten hält (einen Teil der) Routinginformationen vor. Auf diesen Informationen basiert dann die Entscheidung, an welchen nächsten Knoten die Nachricht weitergeleitet wird.

Ein häufig auftretendes Kommunikationsparadigma besteht in Sensornetzen in der Datenverteilung, bei der (von der Senke ausgehend) jeder Knoten eine Information bekommen soll, z. B. neue Werte von Konfigurationsparametern. Bei einer verlustfreien Kommunikation würde es ausreichen, ein CDS (Connected Dominating Set) zu bestimmen, um Nachrichten von der Senke zu allen Knoten zu senden. Ein CDS besteht aus einer Menge von Knoten, bei der jeder Knoten mit jedem anderen (gegebenenfalls über mehrere Hops) kommunizieren kann. Die Berechnung des minimalen CDS ist NP-hart. Liu et al. [74] schlagen hierfür einen Algorithmus vor, der auf logischen Koordinaten basiert. Auch das Verfahren *Starburst SSD* [5] berechnet ein CDS und basiert auf logischen Koordinaten, während andere Autoren nur Distanzen zwischen Knoten verwenden.

Präzise Positions- oder Distanzinformationen sind jedoch selten verfügbar. Zudem kann die Dynamik in den Kommunikationsverbindungen der realen Welt dazu führen, dass es keine dauerhaften Kommunikationsverbindungen gibt. Eine redundante Übertragung, wie es das *Fluten* realisiert, kompensiert die Dynamik und Unzuverlässigkeit von Kommunikationsverbindungen.

2.10.1 Fluten

Fluten ist ein Routing, welches ohne Information über die Wegewahl auskommt. Nachrichten werden dabei immer per Broadcast an alle Nachbarn gesendet. Diese leiten die Nachricht genau einmal weiter. Da sie die Nachricht potentiell mehrmals empfangen, müssen sie speichern, welche Nachrichten sie schon weitergeleitet haben. Beispielsweise können Nachrichten über eine vom Quellknoten vergebene Sequenznummer in Kombina-

tion mit der ID des Quellknoten identifiziert werden. Damit wird auch eine Erkennung von Duplikaten möglich.

Da benachbarte Knoten eine Nachricht meist gleichzeitig empfangen, kann eine zufällige Wartezeit bis zum Weiterleiten die Kollisionswahrscheinlichkeit der weitergeleiteten Nachrichten verringern. Die Wartezeit wird als *Backoff* bezeichnet.

Das Fluten ist einsetzbar, ohne dass je eine Nachricht gesendet wurde. Andere Routingprotokolle müssen hingegen zunächst Nachrichten austauschen, um Routeninformationen zu akquirieren. Ist das Netz nicht partitioniert, empfängt beim Fluten mit großer Wahrscheinlichkeit jeder Knoten im Netz die Nachricht. Details zu der Auslieferungsrate in Abhängigkeit von der Nachrichtenverlustrate finden sich in Abschnitt 5.3.6.

Allerdings wird beim Fluten ein vergleichsweise hohes Nachrichtenaufkommen erzeugt. Je dichter das Netz ist, desto länger dauert es, bis alle Knoten die Nachricht senden konnten. Lu et al. schlagen daher das *Flash-Flooding* [76] vor, bei dem die Knoten beim Medienzugriff die Hörphase auslassen. Knoten, die dieselbe Nachricht empfangen, leiten diese dadurch nahezu gleichzeitig weiter. Die Idee dabei ist, dass es trotz des gleichzeitigen Sendens immer Knoten (in nicht überlappenden Kommunikationsbereichen) gibt, die die Nachricht empfangen. In einer simulativen Evaluation wird gezeigt, dass immerhin 90 % der Knoten eine geflutete Nachricht erhalten. Allerdings bedeutet das Auslassen der Hörphase die Aufgabe der Konformität zu gängigen MAC-Standards.

Um die Anzahl der versendeten Nachrichten zu verringern, gibt es andere, standardkonforme Erweiterungen des Verfahrens. Eine Möglichkeit besteht im Dekrementieren eines Zählers bei jeder Weiterleitung. Ist der Zähler 0, wird die Nachricht nicht mehr weitergeleitet. So kann die zurückgelegte Strecke in Hops begrenzt werden. Eine Alternative ist das selektive Fluten, bei dem nicht alle Empfänger die Nachricht weiterleiten, sondern nur einige. Ob eine Nachricht weitergeleitet wird, wird dabei durch Regeln definiert und kann zufallsbasiert sowie über die Zeit variierend sein.

Ein bekanntes, selektives Verfahren ist *k-flooding*, auch *counter-based-flooding* genannt (siehe [84]). Hierbei wartet ein Knoten nach einem Nachrichtenempfang eine zufällige Zeit und leitet die Nachricht nach Ablauf dieser Zeit nur weiter, falls er sie bis dahin weniger als k-mal empfangen hat. Im Vergleich zum einfachen Fluten muss hier für jede Nachricht ein Zähler vorgehalten werden.

Zufallsbasiertes Fluten kommt dagegen ohne Zähler aus. Bei diesem Ansatz wird jede Nachricht mit einer gewissen Wahrscheinlichkeit weitergeleitet, die für alle Knoten gleich sein oder vom Zustand des Knoten abhängen kann, z. B. von der lokalen Dichte. Ist die Wahrscheinlichkeit immer gleich, ist dies das Verfahren mit dem kleinsten Code und dem geringsten Speicherverbrauch. Genaueres findet sich in [94].

2.10.2 Baum-Routing

Soll eine Nachricht im gesamten Netz verbreitet, also von jedem Knoten empfangen werden, eignet sich meist das Flutenprotokoll. Jedoch wäre es unsinnig, eine Nachricht zu fluten, die nur einen Knoten (z. B. eine Senke) adressiert. Soll zusätzlich jeder Knoten im Netz Nachrichten an diese Senke senden, ist es sinnvoll, einen Routing-Baum aufzubauen. Dabei wird für jeden Knoten der bestmögliche Weg zur Senke ermittelt. Jeder Knoten speichert mindestens die ID des Nachbarn, an den er Nachrichten weiterleitet.

Die Bewertung der Wege erfolgt über eine *Metrik*. Eine Metrik ist eine Abbildung von Eigenschaften des zu bewertenden Objekts (z. B. Knoten oder Kommunikationsverbindung) auf Kennzahlen. Verschiedene Objekte mit dieser Eigenschaft können über die entsprechenden Kennzahlen miteinander verglichen werden. Auch Multihop-Pfade lassen sich vergleichen, indem die aus der Metrik gewonnenen Kennzahlen der Einzelverbindungen, aus denen sich der Pfad zusammensetzt, summiert werden. Beispiele für Metriken sind die Anzahl an Hops [24, 119], die Paketankunftsrate [38], die empfangene Signalstärke und die Anzahl der notwendigen Wiederholungen für eine Verbindung [25] sowie der Energieladezustand [95] oder die Auslastung von Knoten.

Bevor Nachrichten an den Wurzelknoten des Baumes gesendet werden können, muss ein Baum aufgebaut werden. Der Zeitpunkt, zu dem ein Baum aufgebaut wird, bestimmt, ob das Verfahren *proaktiv* oder *reaktiv* ist. Ein proaktives Routing versucht, Routinginformationen immer aktuell zu halten, auch wenn diese noch nicht benötigt werden. Ein reaktives Verfahren baut Routen erst auf, wenn diese benötigt werden.

Das in Abschnitt 4.4 und Kapitel 5 eingesetzte Baum-Routing ist Teil der verwendeten Softwareplattform und wird nun im Detail beschrieben. Metriken beziehen sich bei diesem Verfahren immer auf Kommunikationsverbindungen. Da der Baum dazu dient, Nachrichten zur Senke zu transportieren, ist es wichtig, dass die in den kürzesten Pfaden enthaltenen Kommunikationsverbindungen in Richtung der Senke eine gute Qualität haben. Da Knoten aber nur die eingehenden Verbindungen bewerten können, müssen sie die Bewertungen ihrer eingehenden Verbindungen ihren Nachbarn mitteilen. Die Empfänger können unter verschiedenen Nachbarn den nächsten besten Knoten, ihren *Vaterknoten*, bestimmen.

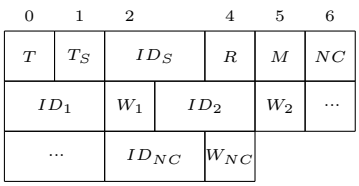


Abbildung 2.11: Baumaufbaunachricht als Byte-Array

Daher sortiert die Senke beim Baumaufbau zunächst ihre Nachbarn bezüglich der eingesetzten Metrik und sendet per Broadcast eine Nachricht, die für ihre k besten Nachbarn die IDs und die Metrikwerte W_{ID} enthält. k wird so beschränkt, dass nicht mehrere Nachrichten gesendet werden müssen. Abbildung 2.11 zeigt den Aufbau der Nachricht. Die Nachricht besteht aus folgenden Feldern:

- T: Nachrichtentyp (Typ des Baum-Routings)
- T_S: Subtyp (Typ „Baumaufbaunachricht“)
- ID_S: ID der Senke (Quelle der Nachricht)
- R: Runde
- M: Verwendete Metrik
- NC: Anzahl an eingetragenen Nachbarn
- ID_i: ID des Nachbarknotens
- W_i: Bewertung des Nachbarknotens

Da die IDs aller Nachbarn eines Knotens die maximale Länge einer Nachricht überschreiten kann, muss die ID-Liste in ihrer Länge beschränkt werden. Alle Empfänger dieser Nachricht aktualisieren ihre Routingtabelle, indem sie als Zieladresse die ID der Senke, als nächste Adresse die ID des Senders und als Bewertung den Wert W_{ID} in die Tabelle eintragen. Falls ihre ID nicht in der Nachricht vorhanden ist, tragen sie einen Standardwert ein, der eine weniger gute Bewertung repräsentiert. W_{ID} beschreibt die Bewertung des Gesamtpfades zur Senke.

Anschließend senden die Empfänger eine Nachricht, die wiederum die Bewertungen ihrer Nachbarn enthält, allerdings erhöht um den Betrag W_{ID}, so dass wieder die Bewertungen der Multihop-Pfade bekannt sind.

Erhält ein Knoten eine Nachricht mit einem Wert kleiner als W_{ID}, aktualisiert er den Eintrag in der Tabelle und sendet nochmals eine Nachricht. Auf diese Weise finden sich nach endlicher Zeit kürzeste Pfade. Jeder Knoten kennt dann seinen Vaterknoten. Die Qualität des so entstandenen Baumes hängt maßgeblich von der verwendeten Metrik ab. Bildet sie die Empfangswahrscheinlichkeit einer Verbindung gut ab, so ist auch die Wahrscheinlichkeit hoch, dass die Nachrichten bei der Senke ankommen.

Hängt die Senke eine Nachricht als Payload (auch: Nutzdaten) an die Baumaufbaunachricht an, den alle anderen Knoten ebenfalls weiterleiten, kann gleichzeitig zum Baumaufbau eine Nachricht im Netz verbreitet werden. Die maximale Anzahl k der enthaltenen IDs und Bewertungen muss dann so verringert werden, dass der Payload noch hinzugefügt werden kann.

Weitere Arbeiten, in denen Varianten des Baum-Routings beschrieben sind, finden sich in [37, 61, 67]. Beim Ad-hoc Multicast Routing Protocol (AMROUTE [120]) sendet jeder Knoten beim Baumaufbau beispielsweise nur eine Nachricht. Vaterknoten wird immer der

Knoten, von dem ein Knoten die Baumaufbaunachricht als erstes bekommen hat. Wird nun Payload an diese Nachricht gehängt, wird der Baumaufbau zum Flutenprotokoll. Der Wert M für die Metrik kann auch anzeigen, dass der Vaterknoten nicht aktualisiert werden soll. So zerstört eine geflutete Nachricht nicht den zuvor aufgebauten Baum mit kürzesten Pfaden.

Werden nun beide Protokolle (Fluten und Baum-Routing) benötigt, sollten diese in einem Protokoll implementiert sein, um Code zu sparen. Der Wert M für die Metrik kann anzeigen, ob geflutet werden soll.

2.11 Zusammenfassung

Im ersten Teil dieses Kapitels wurden der grundlegende Aufbau von Sensorknoten und -netzen sowie ihre Einsatzmöglichkeiten beschrieben. Besondere Anforderungen, wie z. B. die kleine Baugröße und der geringe Stückpreis, wurden anschließend behandelt. Neben diesen beiden Anforderungen müssen auch der Installations- und der Wartungsaufwand gering sein und die eingesetzten Protokolle und Methoden müssen skalieren. Kontextbewusstsein kann dazu genutzt werden, aus gemessenen Daten schon im Netz Informationen zu gewinnen und zu aggregieren, um die Effizienz der Verfahren zu steigern.

Aus diesen Anforderungen resultieren spezielle Eigenschaften von Sensornetzen, die beim Design von Protokollen und Verfahren beachtet werden müssen. Zu nennen sind hier der geringe Energievorrat jedes Knotens und die Ressourcenbeschränkungen, die Speicher und Rechenleistung betreffen, sowie die Zuverlässigkeitsdefizite in der drahtlosen Kommunikation, Sensorik und Uhrengenauigkeit. Wie Zuverlässigkeitsdefizite im Simulator realitätsnah modelliert werden können, wurde ebenfalls erläutert. Algorithmen sollten jedoch nicht nur in Simulatoren, sondern auch in der Realität erprobt werden. Deshalb wurde im Institut für Telematik eine Testumgebung installiert, die in Abschnitt 2.8 beschrieben wurde.

Anschließend beschäftigte sich dieses Kapitel mit verschiedenen Arten von Duty-Cycle-Verfahren. Ein Duty-Cycle-Verfahren ist für ein Sensornetz unverzichtbar, wenn es längere Zeit funktionieren soll, ohne dass die Geräte an eine Netzstromversorgung angeschlossen sind. Vor- und Nachteile verschiedener Verfahrensklassen wurden diskutiert und anhand von einigen Beispielen erläutert. Darauf aufbauend untersucht Kapitel 4 die Anwendungsmöglichkeiten und Vorteile von gestaffeltem Aufwachen in unterschiedlichen Strukturen.

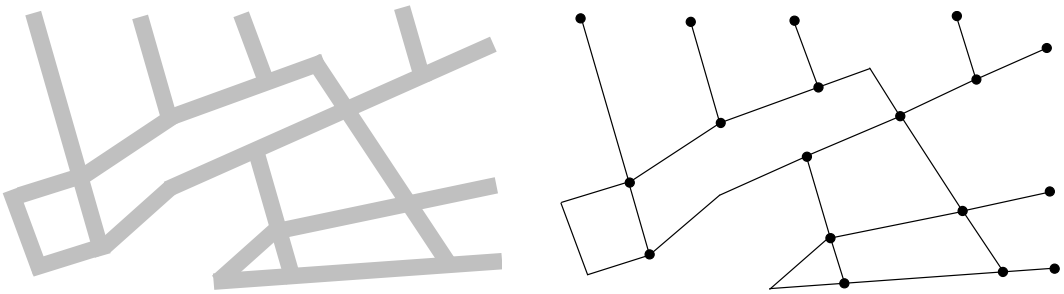
Schließlich wurden die für diese Arbeit relevanten Routingprotokolle (Fluten und Baum-Routing) erläutert. Es wurde auch gezeigt, wie beide Protokolle vereint werden können. Den Anforderungen aus Abschnitt 2.5.6 nachkommend, beschreibt das folgende Kapitel zunächst einen dezentralen Algorithmus zur lokalen Strukturerkennung und anschließend ein Objektverfolgungsverfahren, welches das Wissen über die Struktur verwendet.

Kapitel 3

Verfolgen von Objekten in Graphstrukturen

Ein Sensornetz mit der Aufgabe, Objekte zu verfolgen, wird als Object Tracking Sensor Network (OTSN) bezeichnet. Die Sensorknoten müssen hierbei ein oder mehrere Objekte lokalisieren sowie deren Bewegungen erkennen. Aufgrund ihrer kleinen Größe und Unabhängigkeit von Kabeln sind sie flexibel einsetzbar und für eine solche Aufgabe besonders gut geeignet.

Einige Forschungsarbeiten im Bereich der Objektverfolgung existieren bereits, beispielsweise zur Eindringlingserkennung oder auch für Rettungssysteme. Allerdings wurde bisher die Anordnung des Sensornetzes außer Acht gelassen, da die meisten Forscher von einer rechteckförmigen Struktur ausgehen, in der ein Sensornetz verteilt ist. Ist ein Sensornetz jedoch in einem Kanalsystem, auf Straßen oder in den Fluren von Gebäuden ausgebracht, entsteht eine straßenähnliche Struktur wie sie beispielhaft in Abbildung 3.1(a) zu sehen ist. Abbildung 3.1(b) zeigt die Struktur abstrahiert durch einen Graphen.



(a) Ausbringungsbereich des Sensornetzes (b) Graph

Abbildung 3.1: Abstraktion der Struktur durch einen Graphen

Wie die Anordnung der Knoten nun auf dezentrale Weise in eine Graphstruktur überführt werden kann, zeigt dieses Kapitel. Die grundlegende Idee besteht darin, dass jeder Sensorknoten nur mit Hilfe von Informationen von Nachbarknoten die Anordnung seiner direkten Nachbarn ermittelt. Anhand dieser Anordnung entscheidet der Sensorknoten anschließend, ob er sich in einer Kante oder in einem Knoten des Graphen befindet.

Das Kapitel stellt zudem einen Algorithmus zur Verfolgung von Objekten in solchen Graphstrukturen vor. In der vorliegenden Arbeit wird davon ausgegangen, dass jeder Sensorknoten über einen Bewegungssensor verfügt, welcher die Anwesenheit sich bewegender Objekte erkennt, und dass mehrere Objekte verfolgt werden sollen. Die Verfolgung geschieht eindimensional in jeder Kante und von Knoten zu Knoten des Graphen. Die dezentral ermittelten Anordnungen von Sensorknoten dienen der Unterscheidung zwischen mehreren Objekten.

Der nächste Abschnitt behandelt Anforderungen an Objektverfolgungsverfahren. Anschließend wird auf verwandte Arbeiten aus der Literatur eingegangen. Abschnitt 3.3 stellt ein neues Objektverfolgungsverfahren vor. Dieses wird in Abschnitt 3.4 evaluiert, wobei der Einfluss von Nachrichtenverlust ebenso betrachtet wird wie der von unzuverlässiger Sensorik. Abschnitt 3.6 schließt das Kapitel mit einer Zusammenfassung ab.

Die hier vorgestellten Ergebnisse sind in gemeinsamer Arbeit mit Carsten Buschmann und Stefan Fischer entstanden. In [54–56] wurden sie veröffentlicht.

3.1 Anforderungen an Objektverfolgungsverfahren

Generelle Anforderungen an Sensornetzanwendungen sind der geringe Ressourcenbedarf (Energie, Speicher und Rechenleistung), die geringen Kosten, die Robustheit gegen Kommunikationsfehler und die Skalierbarkeit. Die Forderung nach geringen Kosten impliziert direkt einen geringen Installationsaufwand, der bei wachsender Anzahl an Geräten nicht überproportional ansteigen darf. Zudem dürfen nur so wenige Sensorknoten wie nötig verteilt werden, damit die Kosten so niedrig wie möglich bleiben. Dies hat zur Folge, dass nur geringe Dichten entstehen.

Aus Abschnitt 2.5 und Abschnitt 2.6 lassen sich folgende weitere Anforderungen an Objektverfolgungsverfahren ableiten:

Robustheit gegen Messfehler: Kostengünstige Sensorik impliziert meist nur eine begrenzte Zuverlässigkeit, so dass Objektverfolgungsverfahren Messfehler tolerieren müssen. Für den Fall, dass die zum Einsatz kommenden Sensoren ein binäres Signal liefern, bestehen Messfehler in nicht detektierten Objekten sowie Auslöseereignissen, obwohl kein Objekt in der Nähe ist.

Adaptivität an Netzveränderungen: Bei der Objektverfolgung ist meist ein längerfristiger Einsatz des Sensornetzes gefragt, so dass ein Austauschen von Knoten vorkommen kann. Solche Veränderungen im Netz dürfen das Verfahren nicht beeinflussen.

3.2 Verwandte Arbeiten

Die Voraussetzungen und Ziele der in der Literatur zu findenden Arbeiten sind vielfältig. Ursprünglich kam die Idee, Sensornetze zur Objektverfolgung einzusetzen, aus dem militärischen Bereich; mittlerweile haben sich aber zahlreiche andere Anwendungsgebiete mit unterschiedlichen Voraussetzungen und Zielen, wie z. B. die Roboterverfolgung, entwickelt. Objektverfolgungssensornetze lassen sich nach den folgenden Kriterien klassifizieren:

- Objektanzahl: Viele Autoren nehmen an, dass genau ein Objekt verfolgt werden soll, während andere untersuchen, wie mehrere Objekte verfolgt werden können.
- Hardwareausstattung des Objekts: Das Objekt verfügt selbst über einen Sensorknoten und ist somit Teil des Netzes oder es kann nur von den Sensoren detektiert werden und nicht mit den Knoten Nachrichten austauschen.
- Sensorik: Die Spanne der eingesetzten Sensorik reicht von binären Sensoren über solche, die die Bewegungsrichtung erkennen können bis hin zu Kameras, bei denen die Bildverarbeitung eine große Rolle bei der Objekterkennung spielt.
- Ortsinformationen: Während unter freiem Himmel meist GPS-Module sowie auf freien Flächen Lokalisierungsverfahren, die auf Signalstärkemessungen basieren, Posititionsinformationen liefern können, kann in anderen Szenarien wie beispielsweise in Gebäuden oder bei Hindernissen, die die sich ausbreitenden Signale abschirmen und/oder reflektieren, nicht von einem zweidimensionalen Positionsbewusstsein auf der Seite der Knoten ausgegangen werden.
- Mobilität: Während stationäre Knoten nur Informationen über die Bewegung der zu verfolgenden Objekte erheben und weiterleiten, können mobile Sensorknoten den Objekten tatsächlich folgen.
- Verteiltheit des Verfahrens: In der Literatur finden sich sowohl Vorschläge, die Berechnungen zur Verfolgung dezentral als auch an einer zentralen Stelle durchzuführen.
- Prädiktion: Einige Autoren versuchen, den zukünftigen Weg des Objektes vorherzusagen (z. B. Tseng et al. [114]). Sie treffen allerdings meist Annahmen über die Bewegungsmuster des Objekts.

Ein kurzer Überblick über Objektverfolgungsverfahren findet sich in [7]. Zahlreiche Autoren befassen sich mit dem Problem, genau ein Objekt in einer zweidimensionalen Ebene mit Hilfe einer hohen Dichte an binären Näherungs- oder Präsenzsensoren zu lokalisieren. Zwei Aufgaben müssen von den Sensorknoten hier gelöst werden: die Lokalisierung des Objekts und die Berichterstattung an eine Senke. Mit Hilfe von Ankerknoten und einem Distanzschätzverfahren wird meist zunächst Positionsbewusstsein auf der Seite der Sensorknoten hergestellt. Anders als in Gebäuden führen die klassischen Ansätze zur Etablierung von Positionsbewusstsein in einer solchen Ebene zu recht guten Ergebnissen (vgl. [11]). Die Lokalisierung des Objektes erfolgt dann häufig mit einem so genannten

Partikelfilter [91], bei denen ein stochastisches Modell den möglichen Aufenthaltsort des Objektes beschreibt. Einen Überblick über den Einsatz von Partikelfiltern in der Robotik gibt [28].

Von Präsenzsensoren gehen unter Anderem die Autoren in [50, 63, 72, 103, 105] aus. Sie verwenden Partikelfilter und beschäftigen sich alle mit der Frage, wie man genau ein Objekt möglichst präzise verfolgt. Andere Arbeiten, die von genau einem zu verfolgendem Objekt ausgehen, beschäftigen sich mit cluster-basierten Verfahren [64, 112, 122, 124]. Xu et al. [122] schlagen beispielsweise vor, alle Sensorknoten die Position des Objekts schätzen zu lassen und die geschätzte Position innerhalb eines Clusters solange untereinander auszutauschen und aneinander anzupassen, bis sich die Schätzungen nicht mehr unterscheiden. Jeder Cluster sendet dann sein Ergebnis an eine Senke, so dass der Kommunikationsaufwand gegenüber anderen Verfahren deutlich reduziert ist.

Die Autoren von [4] nehmen an, dass der Sensor erkennt, ob sich das Objekt dem Knoten nähert oder sich vom Knoten entfernt. Akustische Sensoren und/oder Videokameras werden in [17, 40, 86] zur Objektverfolgung eingesetzt. Die Varianten sind also vielfältig, aber alle bisher genannten Arbeiten betrachten genau ein Objekt und gehen somit von anderen Voraussetzungen als in der vorliegenden Arbeit aus. Einige Autoren nehmen zusätzlich Netzwerke von sehr hoher Dichte [50] oder Sensoren ohne Messfehler [4] an.

Tsai et al. [113] untersuchen, wie eine Gruppe von Objekten lokalisiert und verfolgt werden kann. Allerdings bewegen sich die Objekte hier nicht unabhängig voneinander, wie in der vorliegenden Arbeit angenommen. Die in [2] präsentierte Arbeit kommt aus dem militärischen Forschungsbereich und betrachtet ebenfalls mehrere Objekte. Hier wird mit Hilfe eines sehr dichten Netzes von Knoten mit vielfältiger und komplexer Sensorik (Radar-, Magnet- und Schallsensoren) versucht, die Objekte in drei verschiedene Kategorien zu klassifizieren und anschließend mit Hilfe von GPS-Informationen effizient zu verfolgen. Die vorliegende Arbeit geht jedoch zum einen davon aus, dass die Sensorknoten mit einem kostengünstigen Sensor ausgestattet sind, der (nach Vorverarbeitung) ein binäres Signal liefert. Zum anderen wird angenommen, dass keine Ortsinformationen zur Verfügung stehen.

Singh et al. [105] verwenden Partikelfilter um mehrere Objekte zu verfolgen. Sie schlagen – ähnlich wie in dieser Arbeit – ein eindimensionales Verfolgen vor. Allerdings konzentrieren sie sich allein auf die Effektivität ihres Verfahrens zur Objektlokalisierung und gehen deshalb davon aus, dass alle Sensorwerte an einer zentralen Station mit genügend Rechenleistung und Speicherkapazität vorhanden sind, die die Objektpfade schätzen kann. Dieser zentrale Ansatz eignet sich aufgrund des hohen Nachrichtenaufkommens und des resultierenden Energieverbrauchs jedoch nicht für Sensornetze. Die Autoren von [27] und [85] beschränken die Objektverfolgung ebenfalls auf einen Graphen und versuchen die Bewegung von Objekten innerhalb eines Gebäudes zu verfolgen. Allerdings sind die Objekte hier mit RFID-Chips oder Sensorknoten ausgestattet, welche sie identifizieren, so dass ihr Ansatz einem Verfahren zum Verfolgen eines einzelnen Objekts entspricht.

Zusammenfassend lässt sich feststellen, dass andere Autoren sich mit der Lokalisierung eines einzelnen Objekts beschäftigen oder von anderen Bedingungen ausgehen, wie sehr komplexe Sensorik, eine hohe sensorische Abdeckung oder die Verfügbarkeit von Sensordaten an zentraler Stelle. Dieses Kapitel stellt daher ein Objektverfolgungsverfahren vor, welches mehrere Objekte mittels einfacher Sensorik in Graphstrukturen verfolgen kann, in denen keine Ortinformationen verfügbar sind. Aufgrund der unterschiedlichen Ziele lässt sich das vorgeschlagene kaum mit bestehenden Verfahren vergleichen, sondern erweitert viel mehr das Anwendungsgebiet der Sensornetze.

Die Idee ein Netzwerk, welches Löcher umfasst, in Sektionen oder Cluster einzuteilen und diese in einem Graph zu repräsentieren, wurde als erstes in [11] vorgeschlagen. In [52] stellen die Autoren vor, wie man die Graphendarstellung ermitteln kann. Allerdings legen sie den Fokus auf die Erkennung von Kreuzungen und den Rand des Netzes, insbesondere bei extrem hohen Netzwerkdichten, so dass dieser Ansatz hier nicht angewendet werden kann.

3.3 Dezentrales Verfolgen von Objekten in Graphstrukturen

In diesem Abschnitt wird ein dezentrales Verfahren zum Verfolgen von Objekten in Graphstrukturen vorgestellt. Eine Beispielsituation ist in Abbildung 3.2 dargestellt:

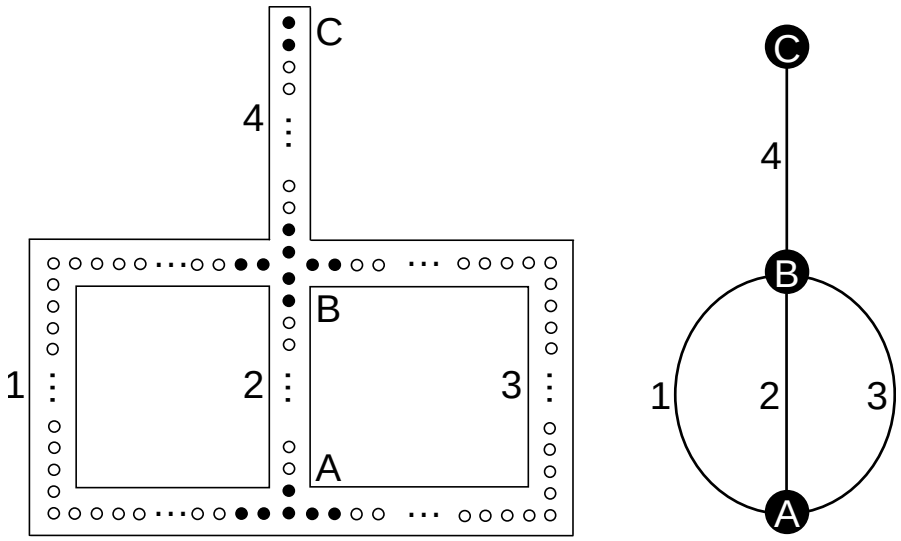


Abbildung 3.2: Szenario

Das Straßen- oder Gangsystem im linken Teil der Abbildung kann in den planaren Graphen (rechts) überführt werden. Dabei werden Kreuzungen und Sackgassenendpunkte als Knoten (im Beispiel A, B und C) und Straßen, Gänge oder Flure als Kanten (1 bis

4) repräsentiert. Die Sensorknoten, die sich im Bereich eines Knoten befinden, sind als ausgefüllte Kreise gekennzeichnet; die Sensorknoten, die zu einer Kante gehören, sind als nicht ausgefüllte Kreise dargestellt.

Die Objektverfolgung wird nun eindimensional pro Kante – von Knoten zu Knoten des Graphen – durchgeführt. An den Knoten wird die Aufgabe an die Sensorknoten der nächsten Kante übergeben. Im Weiteren werden zur besseren Lesbarkeit auch die Begriffe „Kreuzung“ für Knoten und „Flur“ für Kante verwendet. Sensorknoten, die sich in einem Knoten des Graphen befinden, werden auch als „Kreuzungsknoten“ bezeichnet.

Das hier vorgeschlagene Verfahren besteht aus zwei Phasen: der Initialisierungsphase (Abschnitt 3.3.2), die direkt nach der Ausbringung der Sensorknoten erfolgt, und der sich anschließenden Betriebsphase (Abschnitt 3.3.3), in der Objekte verfolgt werden. Abschnitt 3.3.1 beschreibt zunächst die grundlegende Funktionsweise des Verfahrens.

3.3.1 Grundlegende Funktionsweise

Ein Objektdetektionsereignis eines Sensors wird im Weiteren als *Auslöseereignis* bezeichnet. Passiert ein Objekt während der Betriebsphase den gesamten Kommunikationsbereich eines Sensorknotens, lösen im Idealfall die Sensoren seiner Nachbarn sowie sein eigener der Reihe nach aus. Tauschen die benachbarten Knoten diese Auslöseereignisse untereinander aus, können sie die aktuell aufgetretenen Auslöseereignisse in eine Reihenfolge bringen.

Die grundlegende Idee des Verfahrens besteht nun darin, dass jeder Knoten die zu erwartenden Auslösereihenfolgen bereits während der Initialisierungsphase bestimmt, indem er die eindimensionalen Anordnung seiner direkten Nachbarn ermittelt. Aktuell aufgetretene Ereignisreihenfolgen können dann mit den zuvor ermittelten abgeglichen werden.

Netzveränderungen (z. B. Knotenausfälle oder ein Knotenaustausch) verursachen neue Reihenfolgen. Um diese im Laufe der Zeit bzw. mit passierenden Objekten erlernen zu können, werden initial berechnete sowie in der Betriebsphase aufgetretene Reihenfolgen von jedem Knoten protokolliert.

Damit Objektdetektionsereignisse, die bei unterschiedlichen Knoten auftreten, zeitlich in eine korrekte Beziehung gesetzt werden können, müssen die Uhren der Knoten zu einem gewissen Grad zeitsynchronisiert sein. Die erforderliche Synchronisationsgenauigkeit hängt hierbei von der maximalen Geschwindigkeit der Objekte und dem Abstand der Sensoren ab. Meist benötigen andere Protokolle der Anwendung aber einen höheren Grad an Synchronizität, wie z. B. ein Duty-Cycling-Protokoll. Zeitsynchronisationsprotokolle und die dazugehörigen Literaturverweise wurden in Abschnitt 2.5 beschrieben.

Anstatt von festen Positionen oder GPS-Empfängern, die in Gebäuden meist keine Positionsdaten liefern können, auszugehen, wird das in [12] beschriebene Distanzschätzverfahren NIDES verwendet. Es basiert nicht wie viele andere Verfahren darauf, Signal-

abschwächung, -laufzeit oder Einstrahldauer zu messen, sondern Nachbarschaftslisten miteinander zu vergleichen.

3.3.2 Initialisierungsphase

Die Initialisierungsphase schließt sich an die Knotenausbringung direkt an. Sie kann bei Bedarf, z.B. bei großen Veränderungen im Netz, wiederholt werden, aber im Normalfall ist dies nicht notwendig, da der Algorithmus während der Betriebsphase adaptiv arbeitet.

In der Initialisierungsphase werden die folgenden Schritte durchgeführt:

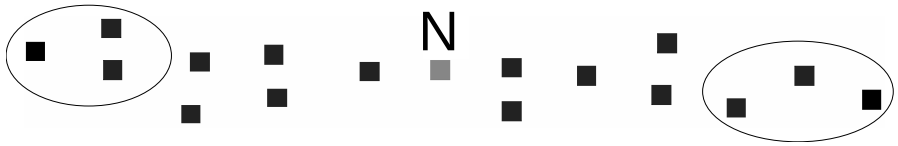
1. Jeder Sensorknoten berechnet die eindimensionale Anordnung seiner Nachbarn pro Richtung zu einem Knoten des Graphen.
2. Jeder Sensorknoten entscheidet, ob er sich an einem Knoten des Graphen, das heißt Kreuzung oder Sackgassenendpunkt, befindet oder nicht.
3. Jeder Repräsentant eines Knoten des Graphen informiert seine Nachbarn über deren Kantenzugehörigkeit.

Die Initialisierungsphase wird von allen Knoten durchgeführt, im Folgenden aber zur besseren Illustration aus Sicht eines Knoten N erläutert. Im ersten Schritt detektiert N die eindimensionale Anordnung seiner Nachbarn (vgl. Abbildung 3.3). N ist in der Abbildung als dunkelgraues Quadrat gekennzeichnet. Als erstes broadcastet N eine Hallo-Nachricht, um seine Nachbarn über seine Anwesenheit zu informieren, sammelt im Gegenzug die IDs der eingehenden Nachrichten und bildet hieraus eine Nachbarschaftsliste.

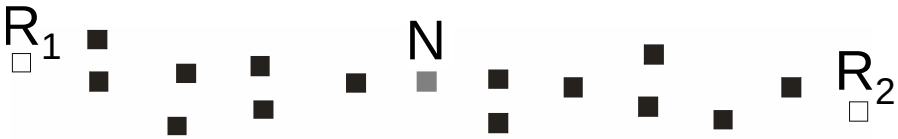
Die entstandene Nachbarschaftsliste wird von jedem Knoten wiederum per Broadcast gesendet. Empfängt ein Knoten solch eine Liste, vergleicht er sie mit seiner eigenen Nachbarschaftsliste und leitet aus dem Anteil der gemeinsamen Nachbarn bezüglich der Gesamtmenge eine (geschätzte) Distanz ab. Auch wenn die Distanz aufgrund der hohen Komplexität der Berechnungsformel nicht exakt bestimmt werden kann, so kann sie doch mittels einer Wertetabelle approximiert werden. Genaueres zur näherungsweisen Berechnung der Distanz kann in [11, 12] nachgelesen werden. Als Resultat kann N seine Nachbarschaftsliste um geschätzte Distanzen zu den Nachbarn erweitern.

Diese erweiterte Liste wird wieder per Broadcast gesendet, so dass alle Knoten die Distanzschätzungen ihrer Nachbarn untereinander kennen. Dies ist notwendig, da die Knoten nicht nur Distanzen zwischen sich und ihren Nachbarn, sondern auch die ihrer Nachbarn untereinander kennen müssen.

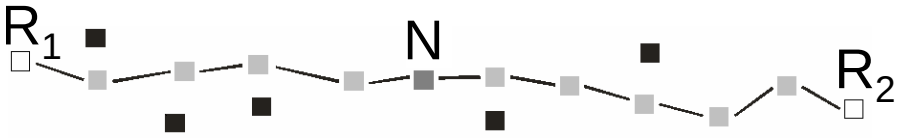
Anschließend sortiert N seine Nachbarn aufsteigend nach ihrer Distanz und wählt die letzten $r\,\%$ der Knoten (also die Knoten, die am weitesten von ihm entfernt sind) als Randknoten aus. Simulationen mit verschiedenen Dichten haben ergeben, dass $r = 40\,\%$ ein sinnvoller Wert ist. Danach unterteilt der Knoten N diese Randknoten in Abhängigkeit von ihren Distanzen zueinander in Gruppen ein. Knoten, die dicht beieinander liegen,



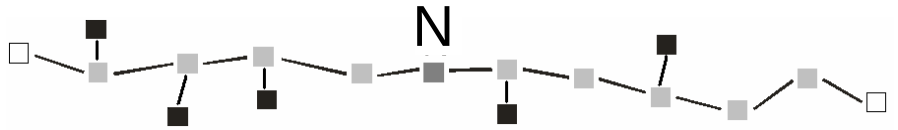
(a) Knoten N, seine Nachbarn und Gruppen



(b) Repräsentanten



(c) Berechnung der Ketten



(d) Zuordnen der verbleibenden Knoten zu Knoten einer Kette

Abbildung 3.3: Erkennung der lokalen Topologie

gehören zur selben Gruppe, wohingegen Knoten, die weit voneinander entfernt sind, verschiedenen Gruppen zugeordnet werden. Beim Einsortieren eines Nachbarn in eine Gruppe muss N über alle Nachbarknoten aller Gruppen iterieren, um herauszufinden, ob der Nachbar anderen Nachbarn aus zwei oder mehr Gruppen nah ist und diese somit vereint. Abbildung 3.3(a) zeigt die zwei entstehenden Gruppen als Kreise.

Aus jeder Gruppe wird danach der von N am weitesten entfernte Knoten als Repräsentant gekennzeichnet (weiße Quadrate R_1 und R_2 in Abbildung 3.3(b)).

Von jedem Repräsentanten aus berechnet N im dritten Schritt eine so genannte *Kette* (vgl. Abbildung 3.3(c)), indem N alle Knoten, die sich zwischen dem Repräsentanten und N befinden, nach ihren Distanzen zu N sortiert. Haben mehrere Knoten den gleichen Anteil gemeinsamer Nachbarn mit N, so ist auch ihre geschätzte Distanz zu N gleich. Von diesen Knoten wird jeweils nur einer der Kette hinzugefügt, da ihre Reihenfolge noch nicht bekannt ist.

Knoten, die in keiner der entstandenen Ketten enthalten sind, werden jeweils ihrem dichtesten Nachbarn zugeordnet, der in einer Kette enthalten ist (siehe Abbildung 3.3(d)).

Sie werden auf den dichtesten Nachbarn *gemappt*. Detektiert einer der gemappten Knoten ein sich bewegendes Objekt, betrachtet N dieses Ereignis als eines des entsprechenden Kettenknotens.

Die Qualität der Ketten hängt maßgeblich von der Genauigkeit der Distanzschätzung ab, da die Knotenreihenfolgen bei fehlerhaften Distanzen nicht korrekt berechnet werden können. Wichtig ist jedoch nicht, dass die absoluten Werte der Distanzen korrekt sind, sondern dass die Relationen erhalten bleiben, das heißt, dass die geschätzte Distanz zu einem näheren Nachbarn kleiner ist als diejenige zu einem weiter entfernten Nachbarn. Aus diesem Grund wurde NIDES anstatt einem Distanzschätzverfahren, das auf ungenauen Signalstärkemessungen beruht, verwendet.

Basierend auf der Anzahl der entstandenen Ketten kann jeder Sensorknoten nun Schritt 2 der Initialisierungsphase durchführen und entscheiden, ob er sich in einer Kante des Graphen oder in einem Knoten des Graphen befindet. Die Sensorknoten, die sich in einer Kante befinden, berechnen genau zwei Ketten. Die Knoten in der Nähe einer Kreuzung oder Sackgassenendpunkt werden ein, drei oder mehr Ketten berechnen.

Allerdings führen Distanzschätzfehler gegebenenfalls zu Fehlern in der Anzahl der Ketten, da sowohl die Auswahl von Randknoten wie auch die Sortierung der Randknoten in Gruppen auf den Distanzschätzungen basiert. Werden Distanzen unterschätzt, werden potentiell zu wenig Ketten berechnet. Distanzüberschätzungen führen zu mehr Ketten als vorhanden sind.

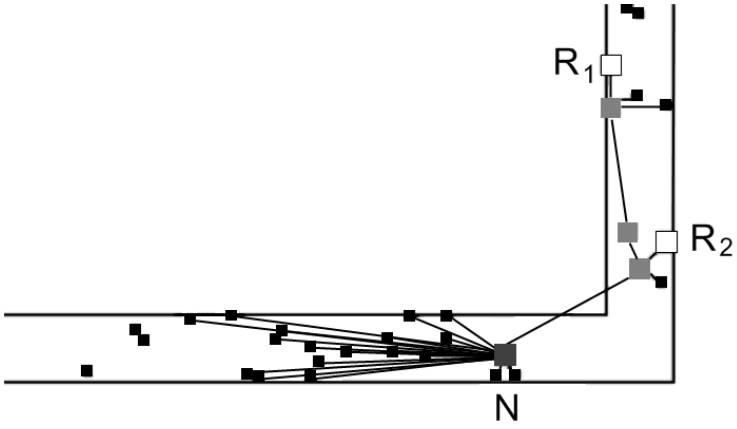


Abbildung 3.4: Fehlschlagende Erkennung von Randknoten

Ein Nachteil von Nachbarschaftsschnittmengen ist jedoch, dass diese Methode eine gleichmäßige Knotenverteilung voraussetzt. Eine überdurchschnittlich hohe Dichte führt zur Unterschätzung von Distanzen, so dass die obersten 40 % der nach Entfernung sortierten Knotenliste sich alle auf einer Seite von N befinden, wie es in Abbildung 3.4

dargestellt ist. Die Nachbarn, bei denen die Distanz unterschätzt wurde, ordnet N sich selbst zu, da er ihr dichtester Nachbar ist.

Lokale Schwankungen in der Knotendichte müssen deshalb bei der Zuordnung zu Kante oder Knoten kompensiert werden. Um die Fehlerquote hierbei möglichst gering zu halten, wird deshalb eine Abstimmung unter Nachbarknoten durchgeführt, wie sie Algorithmus 1 beschreibt.

Algorithmus 1 Abstimmung

```
number_chains := chain_list.size();
if number_chains != 2 then
  vertex_node := true;
else
  vertex_node := false;
end if
msg_list := 0;
start timeout t_vote;
message := (ID, number_chains);
broadcast message;
while t_vote not expired do
  collect incoming messages in msg_list;
end while
count_equal := 0;
for all m ∈ msg_list do
  if m.number_chains = number_chains then
    count_equal := count_equal + 1;
  end if
end for
if count_equal/msg_list.size() < 2/3 then
  vertex_node := false;
end if
```

Jeder Knoten nimmt zunächst an, dass er sich in einer Kreuzung befindet, wenn er eine von zwei abweichende Anzahl an Ketten ermittelt hat ($vertex\_node = true$). Anschließend startet er einen Timer t_{vote} und sendet eine Broadcast-Nachricht ($message$), die seine ID und seine Kettenanzahl ($number\_chains$) enthält. Bis der Timer abläuft sammelt er eingehende Nachrichten. So haben alle Knoten Kenntnis über die Kettenanzahlen ihrer Nachbarn. Nach Ablauf des Timers zählt ein Knoten, wie viele Nachbarn genauso viele Ketten ermittelt haben wie er selbst ($count\_equal$). Er betrachtet sich nur dann als Kreuzungsknoten, wenn mindestens $E\%$ seiner Nachbarn ebenfalls $number\_chains$ Ketten berechnet hat. Simulationen haben gezeigt, dass $E = \frac{2}{3}$ ein sinnvoller Wert ist.

Die Abstimmung korrigiert allerdings nur die Lokalisierungsentscheidung bezüglich Kante oder Knoten im Graphen, wirkt sich aber nicht auf zuvor berechnete Ketten aus, da sie nicht die Fehler in den Distanzschätzungen ausgleichen kann. Dies ist auch nicht notwendig, da in der Betriebsphase korrekte Ketten erstellt werden, die mit jedem Objekt eine bessere Bewertung bekommen.

Um den im Zentrum der Kreuzung gelegenen Knoten herauszufinden, broadcasten alle Knoten ihr Abstimmungsergebnis und anschließend berechnen alle Kreuzungsknoten ihre durchschnittliche Distanz zu den anderen Kreuzungsknoten. Derjenige mit der kleinsten durchschnittlichen Distanz wird als der optimale angesehen und wird Repräsentant dieser Kreuzung. In einem Sackgassenendpunkt wird der Sensorknoten mit den wenigsten Nachbarn Repräsentant des Knotenpunktes des Graphen.

Nach Erkennung der Repräsentanten bestimmen die Knoten noch ihre Kantenzugehörigkeit, indem sie zunächst annehmen, alle Nachbarn wären in derselben Kante. Nun ordnet jeder Repräsentant allen von ihm berechneten Ketten eine eindeutige Kantennummer $K_{ID} = R_i,F_j$ zu, wobei R_i seine eigene ID und F_j eine fortlaufende Nummer ist. Beispielsweise ergibt sich für den Repräsentanten mit der ID 5 und Kettennummer 1 die Kantennummer 5,1 (vgl. Abbildung 3.5). Jeder Repräsentant broadcastet die Ketten (einschließlich der gemappten Knoten) sowie ihre Kantennummern.

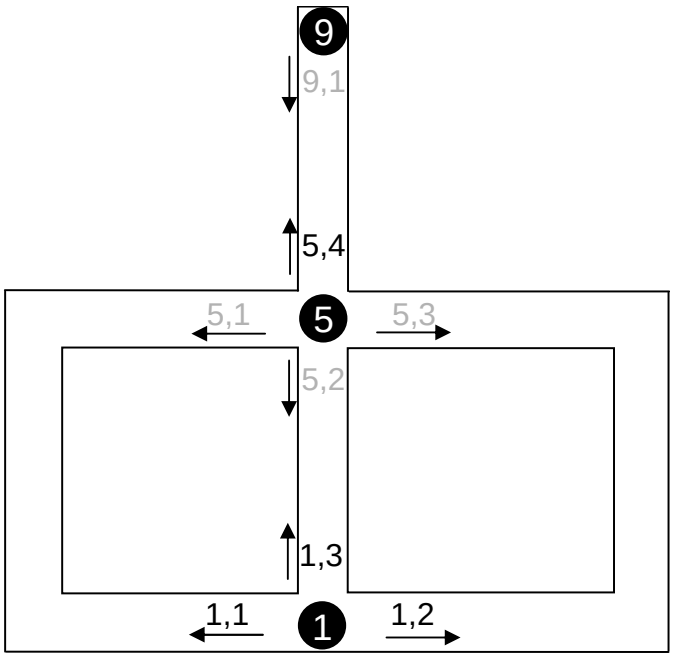


Abbildung 3.5: Von Repräsentanten vergebene Kantennummern

Die Empfänger der Nachricht, die zu einer dieser Ketten gehören, speichern die entsprechende Kantennummer K_{ID} als ihre eigene ab und markieren alle Nachbarn, die in einer anderen Kette enthalten sind, als nicht aus ihrer Kante. Sei M beispielsweise ein solcher Empfänger. Sofern M selbst zwei Ketten ermittelt und Nachbarn hat, die nicht in den empfangenen Ketten enthalten sind, broadcastet M eine Nachricht, die die Kette mit diesen Nachbarn sowie seine eigene Kantennummer K_{ID} enthält. Die Empfänger dieser Nachricht gehen genauso vor und broadcasten wiederum eine Nachricht, so dass die Kantennummer durch die gesamte Kante bis zum nächsten Knoten weitergeleitet wird. Sensorknoten, die sich nicht in einer Kante befinden, senden keine Nachricht.

Auf diese Weise erfahren alle Sensorknoten, die sich in einer Kante befinden, zwei Kantennummer. Diese beschreiben, welche Knoten des Graphen die Kante miteinander verbindet. Zur Beschreibung und Identifizierung der Kante wird ausschließlich die kleinere Nummer verwendet, so dass die Nummerierung der Kanten für alle Sensorknoten gleich ist. In Abbildung 3.5 sind die verwendeten, kleineren Nummern in schwarzer und die nicht verwendeten Nummern in grauer Schrift dargestellt.

3.3.3 Betriebsphase

Nach der Initialisierungsphase nimmt das Netz den normalen Betrieb auf. Der Übergang in die Betriebsphase kann durch eine geflutete Nachricht, aber auch durch einen Timer von jedem Knoten selbst gesteuert sein. Die grundlegende Idee ist nun, aktuell vorkommende Auslöseereignisreihenfolgen mit den im Vorhinein berechneten Ereignisketten zu vergleichen, um so entscheiden zu können, welche Auslöseereignisse zu einem Objekt gehören. Da sich die Knoten allerdings automatisch an Veränderungen im Netz anpassen sollen, bilden die berechneten Ketten die Ausgangslage für eine *Historie*, die während der Betriebsphase ergänzt und angepasst wird.

Anhand eines Beispielknotens N_{15} soll die Funktionsweise der Betriebsphase veranschaulicht werden. Nach der Initialisierungsphase beinhaltet die Historie genau alle Ketten, die bei Knoten N_{15} wie folgt aussehen könnten:

Übereinstimmungs-zähler	Letzter Übereinstimmungszeitpunkt	Kette
0	0	12 13 14 15
0	0	18 17 16 15

Tabelle 3.1: Historie von Knoten N_{15} (zu Beginn)

Wann immer der Sensor eines Knotens auslöst, sendet der Knoten per Broadcast eine so genannte *Objektnachricht*, die die folgenden Werte enthält:

- Startposition S_{Ort}: ID des Repräsentanten der zuletzt passierten Kreuzung,
- Startzeit S_{Zeit}: Zeitpunkt, zu dem der Repräsentant der zuletzt passierten Kreuzung aktiviert wurde, das heißt das Objekt in die aktuelle Kante eintrat,
- Kantennummer K_{ID}: Kantennummer der aktuellen Kante
- Aktivierungszähler Z: Anzahl an Auslöseereignissen seitdem sich das Objekt in der aktuellen Kante befindet,
- Geschwindigkeit G: durchschnittliche Zeitdauer zwischen zwei Auslöseereignissen,
- die (eigene) ID des aktivierten Knotens und
- der aktuelle (eigene) Auslösezeitpunkt

Startposition und -zeit identifizieren ein Objekt unter der Annahme, dass nicht zwei Objekte zum selben Zeitpunkt denselben Sensor auslösen können, eindeutig. Sie bilden deshalb die *Objekt-ID*.

Die Knoten speichern alle eingehenden Objektnachrichten in einem *Nachrichtenpuffer*, in dem die Nachrichten nach der enthaltenen Objekt-ID geordnet werden. Nachrichten, die sich auf dasselbe Objekt beziehen, werden nach ihrem Auslösezeitpunkt sortiert. Der Nachrichtenpuffer hat eine feste Größe, so dass neue Nachrichten alte ersetzen müssen, wenn dieser voll ist. Die Puffergröße sollte so gewählt sein, dass Nachrichten von mehreren Objekten dort gespeichert werden können, bis diese Objekte den Kommunikationsbereich eines Knotens passiert haben.

Es sei beispielhaft angenommen, dass Knoten N_{15} kürzlich 4 Objektnachrichten empfing, zwei von N_{12}, eine von N_{13} und eine von N_{14}. In diesem Fall wird der Nachrichtenpuffer zwei Einträge für Objekte enthalten, wie es in Tabelle 3.2 dargestellt ist. Eine Zeile entspricht einem Objekt. Ein Tupel ((ID_{Ort}; Zeitpunkt), auch: geordnetes Paar) setzt sich aus einer ID, die den Ort des Objektes bezeichnet, und einem Zeitpunkt, zu dem das Objekt detektiert wurde, zusammen.

Objekt-ID O_{id} =(S_{Ort}; S_{Zeit})	Kante K_{ID}	Zähler Z	Geschw. G	Auslöseereignisse von Nachbarknoten (ID; Zeitpunkt in Sek.)
(1; 8,4)	1,1	14	1,6	(12; 27,6) (13; 29,1) (14; 30,8)
(1; 9,3)	1,1	12	1,8	(12; 30,6)

Tabelle 3.2: Nachrichtenpuffer von Knoten N_{15} (zum Zeitpunkt des Auslöseereignisses)

Löst der Sensor eines Knotens aus, versucht der Knoten eine Sequenz von Auslöseereignissen in dem Puffer zu finden, welche zu einem Eintrag in der Historie passt. Dazu konstruiert er nacheinander für jeden Eintrag im Puffer, also für jedes Objekt, die entsprechende Ereigniskette, indem er die Knoten-IDs aus den sortierten Tupeln extrahiert

und seine eigene ID anhängt. Für das Beispiel bedeutet dies, dass bei Aktivierung des Knotens N_{15} zwei Ketten entstehen: $12, 13, 14, 15$ und $12, 15$.

Alle resultierenden Ketten werden mit allen Einträgen in der Historie verglichen, wobei das Paar, welches am besten zueinander passt, gesucht wird. Der Operator zum Vergleich von Ereignisketten wurde hierbei vom so genannten *Local Alignment* [102] abgeleitet. Es liefert ein Ähnlichkeitsmaß, das die Permutation und das Fehlen von Einträgen toleriert. Wie es im Detail eingesetzt wurde, wird im Anhang A beschrieben.

Da keine extrem hohen Netzwerkdichten betrachtet werden, ist die für die Berechnung benötigte quadratische Laufzeit kein Problem für den Protokollablauf. Um den quadratischen Speicherplatzverbrauch zu vermeiden, kann die Berechnung in zwei Vektoren der Größe n (= Anzahl der Nachbarn von N) durchgeführt werden, da es ausreicht, die vorherige Zeile sowie den Maximalwert zu speichern.

Zusätzlich zum reinen Vergleich der Zeichenketten aus dem Nachrichtenpuffer mit denen aus der Historie mittels Local Alignment wird geprüft, ob die Zeit, die seit dem letzten Auslösezeitpunkt vergangen ist, zu der im Nachrichtenpuffer gespeicherten Geschwindigkeit des Objekts passt. Weiterhin muss sich der zuletzt aktivierte Knoten in derselben Kante wie der aktuell aktivierte Knoten befinden. Passt eines von beiden nicht, wird das Ergebnis des Vergleichs abgewertet, damit immer eine Unterscheidung möglich ist, auch wenn beispielsweise zwei Objekte dieselben Nachbarknoten aktiviert haben.

Nach dem Vergleich der Ketten mit den Einträgen in der Historie werden sowohl Nachrichtenpuffer wie auch die Historie aktualisiert. Der Nachrichtenpuffer wird wie folgt aktualisiert: Kann aus dem Nachrichtenpuffer keine passende Kette gefunden werden, das heißt alle Ergebnisse des Vergleichsoperators liegen unterhalb eines Schwellwertes, wird eine neue Kette begonnen. Die Kette aus dem Nachrichtenpuffer oder die neu erzeugte Kette wird im Folgenden als *aktuelle Kette* bezeichnet. Sie wird – basierend auf der Geschwindigkeit des Objekts –, nachdem das Objekt den Kommunikationsbereich des Knotens verlassen hat, aus dem Nachrichtenpuffer gelöscht, um Speicherplatz zu sparen.

Die Historie wird wie folgt aktualisiert: Existiert in der Historie bereits eine exakt gleiche Kette wie die aktuelle Kette, wird ein zu diesem Eintrag gehörender Übereinstimmungszähler erhöht. Andernfalls wird die aktuelle Kette der Historie hinzugefügt, um Adaptivität an neue Knoten und neue Knotenanordnungen zu gewährleisten.

Übereinstimmungszähler	Letzter Übereinstimmungszeitpunkt	Kette
~~0~~ 1	~~0~~ 31,8	12 13 14 15
0	0	18 17 16 15

Tabelle 3.3: Historie von Knoten N_{15} (nach einem Auslöseereignis)

Sei der Auslösezeitpunkt von Knoten N_{15} beispielsweise 31,8. N_{15} findet den höchsten Grad an Übereinstimmung, indem er das erste Objekt (1; 8,4) aus dem Nachrichtenpuffer (siehe Tabelle 3.2) mit dem ersten Eintrag in der Historie korreliert. N_{15} wird deshalb, wie in Tabelle 3.3 dargestellt ist, den Übereinstimmungszähler dieses Eintrags erhöhen und den Zeitpunkt der letzten Übereinstimmung auf den eigenen Auslösezeitpunkt aktualisieren (31,8).

Schließlich konstruiert der aktivierte Knoten eine Objektnachricht, die er per Broadcast an seine Nachbarn sendet. N_{15} versendet die Objektnachricht: O_{id}=(1; 8,4), Kantennummer $K_{ID} = 1{,}1$, Zähler $Z = 15$, Geschwindigkeit $G = 1{,}6$, neues Auslöseereignis (ID= 15; Zeitpunkt= 31,8).

Zusammenfassend führt ein aktivierter Knoten folgende Schritte aus:

1. Bilden von Ketten aus den im Nachrichtenpuffer vorhandenen Objektnachrichten erweitert um die eigene ID,
2. Vergleichen der gebildeten Ketten mit den Einträgen in der Historie und Finden der besten Kombination,
3. Aktualisieren von Nachrichtenpuffer und Historie,
4. Senden einer Objektnachricht.

3.4 Simulative Evaluation

Für die simulative Evaluation wurde das in Abbildung 3.2 dargestellte Szenario mit einer Größe von 50 x 50 m verwendet. 140 Sensorknoten wurden im Abstand von 1,5 m in allen Kanten platziert. Jeder Sensorknoten konnte sich bewegende Objekte in einem Umkreis mit etwa 1 m Radius detektieren.

Das Szenario bestand aus vier Kanten (Fluren) und drei Knoten (zwei Kreuzungen und ein Sackgassenendpunkt) mit den Graden 3, 4 und 1. Es deckt somit alle möglichen Fälle für Graphstrukturen ab, da Knoten höherer Grade zumindest bei der vorgeschlagenen Strategie nur äquivalente Fälle darstellen. Zusätzlich hatten zwei der Flure zwei rechtwinklige Ecken. Diese Regionen sind besonders problematisch, weil das Distanzschätzverfahren hier zu fehlerhaften Ergebnissen tendiert. Neben falschen Sensorknotenreihenfolgen kann dies auch zu fehlerhaften Ortsinformationen bezüglich Knoten und Kanten des Graphen führen.

Alle Ergebnisse dieses Kapitels sind mit dem Simulator Shawn und dem RI-Modell als Kommunikationsmodell entstanden (vgl. Abschnitt 2.7). Um eine für Sensornetze sinnvolle Dichte (vgl. Abschnitt 2.7) zu erreichen, wurde der Kommunikationsradius hier auf 9 m festgesetzt. Die meisten Knoten hatten dadurch 12 Nachbarn (6 auf jeder Seite).

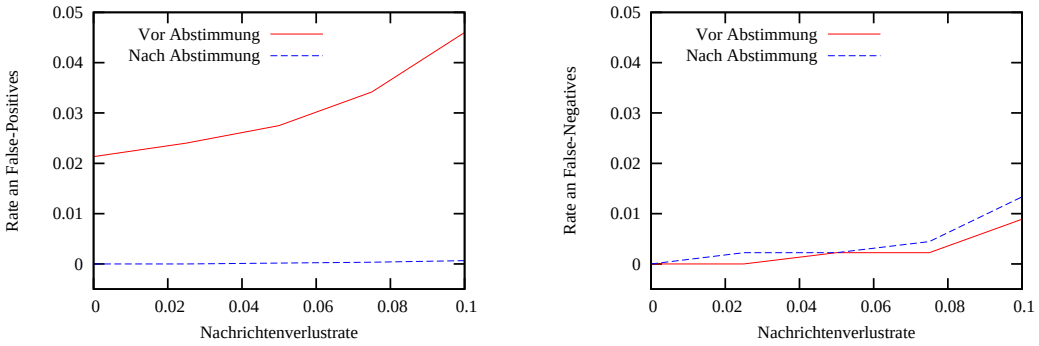
Knoten, die sich an einer Kreuzung befanden, hatten sogar 24 Nachbarn, Knoten im Sackgassenendpunkt dagegen nur 6.

Um eine statistisch aussagekräftige Evaluation zu erzielen, wurden die Ergebnisse von 100 Simulationsläufen mit unterschiedlicher Zufallszahleninitialisierung gemittelt.

Die beiden Phasen Installations- und Betriebsphase wurden getrennt analysiert. Der folgende Abschnitt behandelt die Qualität der Initialisierungsphase, also die Erkennung der lokalen Topologie, während Abschnitt 3.4.2 die Evaluationsergebnisse der eigentlichen Objektverfolgung diskutiert.

3.4.1 Evaluation der Topologieerkennung

Als erstes wurde die Erkennung von Knoten des Graphen evaluiert. Für den korrekten Ablauf der sich anschließenden Betriebsphase ist die Abwesenheit von fälschlicherweise als Kreuzungsknoten identifizierten Sensorknoten in Fluren ebenso wichtig wie die Erkennung von mindestens einem Sensorknoten in jeder Kreuzung als Kreuzungsknoten.



(a) Falsche Kreuzungsknoten

(b) Fehlende Kreuzungserkennung

Abbildung 3.6: Erkennung von Knoten des Graphen

Abbildung 3.6(a) zeigt die Rate an Sensorknoten, die sich fälschlicherweise als in einer Kreuzung befindlich klassifizierten, für verschiedene, definierte Nachrichtenverlustraten. Während vor dem Abstimmungsprozess die Fehlerrate noch von 2 % ohne Nachrichtenverluste auf 4,5 % bei immerhin 10 % Nachrichtenverlusten steigt, ist die Fehlerkennungsrate nach der Abstimmung 0 % und steigt nahezu überhaupt nicht. Dies verdeutlicht die Effektivität des Abstimmungsprozesses.

Verhindert die Abstimmung erfolgreich die fälschliche Erkennung von nicht vorhandenen Kreuzungen, indem sie die Anzahl an Kreuzungsknoten reduziert, muss trotzdem gewährleistet sein, dass alle tatsächlichen Kreuzungen erkannt werden. Abbildung 3.6(b) zeigt den Anteil an Knoten, der sich fälschlicherweise nicht als Kreuzungsknoten erkannt

hat, für Nachrichtenverlustraten zwischen 0 % und 10 %. Während sich bei weniger als 7 % Nachrichtenverlusten nahezu alle Kreuzungsknoten richtig erkannt haben, steigt die Fehlerrate leicht für höhere Nachrichtenverlustraten. Der Abstimmungsprozess hat hierauf jedoch fast keinen zusätzlichen negativen Einfluss. Sind hohe Nachrichtenverlustraten zu erwarten, sollten Nachrichten mehrmals gesendet werden, um ein korrektes Kontextbewusstsein zu etablieren.

Als nächstes wurde die Qualität der berechneten Nachbarschaftsreihenfolgen evaluiert. Um zu analysieren, welchen Einfluss die Auswahl des Distanzschätzverfahrens hierbei hat, wurden neben NIDES auch die exakten Distanzen sowie Distanzen, die mit einem zufälligen Fehler behaftet waren, verwendet und deren Ergebnisse als Vergleichswerte herangezogen. Weiterhin wurden zwei verschiedene Knotenverteilungen eingesetzt: eine zufällige Verteilung der 140 Knoten sowie die Ausrichtung der Knoten an einem Gitter wie in Abbildung 3.2 dargestellt, um die Auswirkung von Schwankungen in der Dichte zu untersuchen. Abbildung 3.7 zeigt für die drei Distanzschätzverfahren sowie für beide Verteilungen die Qualität der Ketten. Dazu wurden die berechneten Ketten mit den korrekten Reihenfolgen mittels Local Alignment verglichen. Die Summe aller Vergleichsergebnisse wurde dann auf die maximal mögliche Summe normiert.

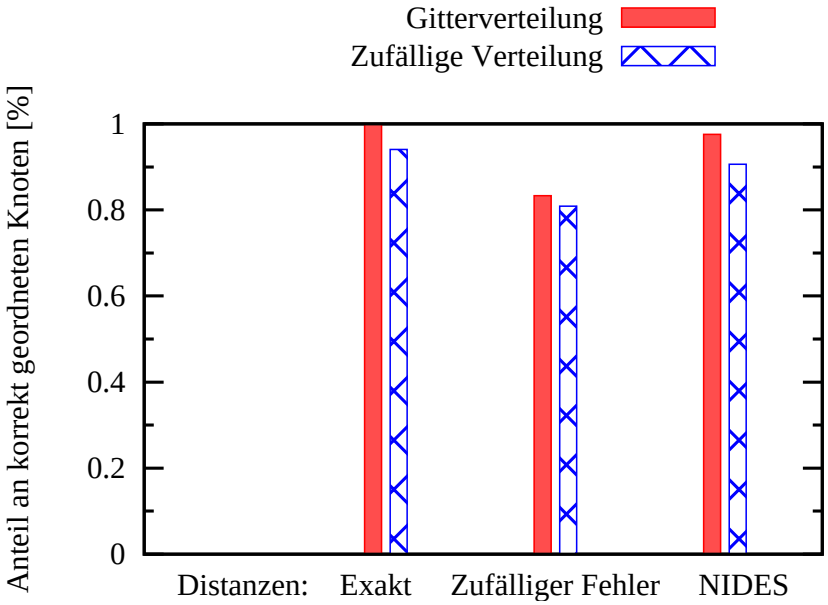


Abbildung 3.7: Qualität der Nachbarschaftsreihenfolgen bzgl. Ordnung

Die Abbildung zeigt, dass die Ergebnisse mit NIDES (rechts) nahezu genauso gut sind wie die bei der Verwendung der exakten Distanzen (links), obwohl es in einem so unregelmäßigen Szenario wie hier zu sehr großen Schätzfehlern von 12 % bis 20 % kommt.

Die trotzdem gute Qualität lässt sich auf die bewahrten Abstandsrelationen zwischen Knoten zurückführen.

Die Ergebnisse der Distanzen mit einem zufälligen Fehler von bis zu 20 % (Mitte) fallen dagegen deutlich schlechter aus. Die ungleichmäßigen Schätzfehler ziehen nicht nur fehlerhafte Reihenfolgen, sondern auch falsche Mappings nach sich. Weiterhin zeigen die Simulationen, dass die zufällige Verteilung die Ergebnisse für alle Verfahren im Vergleich zur Gitterverteilung nicht wesentlich verschlechtert. Fehler in den initialen Ketten werden in der Betriebsphase nach und nach behoben.

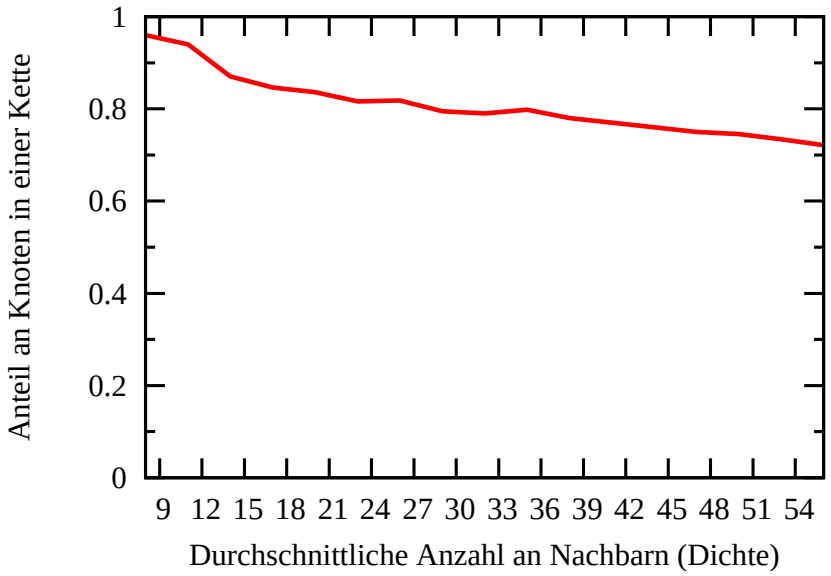


Abbildung 3.8: Länge der Nachbarschaftsreihenfolgen in Abhängigkeit von der Dichte

Neben der Wirkung der unterschiedlichen Verteilungen und Distanzschätzverfahren wurde der Einfluss der Knotendichte auf die Länge der berechneten Ketten simulativ untersucht. Kurze Reihenfolgen erschweren die richtige Zuordnung des eigenen Auslöseereignisses während der Betriebsphase maßgeblich. Deshalb ist es wichtig, dass so wenig Knoten wie möglich gemappt werden. Das Ergebnis dieser Simulationen ist in Abbildung 3.8 dargestellt. Es ist zu erkennen, dass mit steigender Dichte der Anteil an Knoten, die in einer Reihenfolge enthalten sind, linear sinkt. Für eine mittlere Dichte von 12 Nachbarn liegt der Anteil bei fast 95 %. Bei 50 Nachbarn sind es nur noch 75 %, was allerdings kein Problem darstellt, da die absolute Länge der Ketten sogar länger ist als bei niedriger Dichte, nämlich 18,75 statt 5,7.

3.4.2 Evaluation der Objektverfolgung

Der folgende Abschnitt beschreibt die Evaluationsergebnisse des zweiten Teils des Objektverfolgungsverfahrens, der Betriebsphase. Das Verfahren wurde auf mehrere Bewegungsmuster, zusammengesetzt aus unterschiedlich vielen Objekten, die sich mit verschiedenen Geschwindigkeiten in dem in Abbildung 3.2 dargestellten Szenario bewegten, angewendet. Die Bewegungsmuster beinhalteten einander entgegenkommende, nachfolgende und überholende Objekte:

1. *1after2*: Ein Objekt bewegt sich von C nach A durch die Flure 4 und 1, dann nimmt ein zweites Objekt denselben Weg und schließlich bewegt sich ein drittes Objekt in genau umgekehrter Richtung.
2. *Succeeding*: Zwei Objekte bewegen sich im Abstand von 2 m mit gleicher Geschwindigkeit von A nach B durch 1 and 4.
3. *Overtaking*: Zwei Objekte bewegen sich von A nach C durch 1 und 4, wobei das zweite Objekt das erste überholt.
4. *1towards4*: Vier Objekte bewegen sich mit einem Abstand von 5 m von A nach C, während sich ein fünftes Objekt von C nach A durch dieselben Flure bewegt, den vier sich folgenden Objekten also entgegenkommt.
5. *3Ways*: Drei Objekte bewegen sich von A nach C, ein Objekt durch Flur 1, eins durch Flur 2 und das dritte durch Flur 3. Die Objekte starten zur gleichen Zeit, bewegen sich aber mit unterschiedlicher Geschwindigkeit, so dass sie in etwa gleichzeitig ankommen.

Die Bewegungsmuster stellen eine repräsentative Auswahl aus möglichen Szenarien dar und decken sowohl einfache als auch schwierige Fälle ab. Bewegen sich sehr viele Objekte in dem Graphen, so dass die Sensoren aller Knoten ständig Objektdetektionsereignisse melden, ist eine Zuordnung und Verfolgung nicht mehr möglich.

Geprüft wurde bei allen Simulationen nicht nur, ob der Weg jedes einzelnen Objekts korrekt verfolgt, sondern auch, ob die Reihenfolge der Objekte korrekt erkannt wurde.

Einfluss von zeitlicher Variation bei der Auslösung und von Nachrichtenverlusten

Es muss davon ausgegangen werden, dass nicht alle Sensoren perfekt zum Flur ausgerichtet sind und ein Sensor abhängig von der Geschwindigkeit der Objekte das Objekt auch nicht immer am selben Ort detektiert. Je schneller sich ein Objekt bewegt desto früher wird es beispielsweise meist detektiert. Um diese Effekte im Simulator abzubilden, wurde auf den perfekten Auslösezeitpunkt ein zufälliger (positiver oder negativer) Wert als Fehler aufaddiert. Im Simulationsmodell ist der perfekte Auslösezeitpunkt genau dann, wenn das Objekt den dichtesten Punkt zum Sensor erreicht, unter der Voraussetzung, dass es sich

überhaupt soweit nähert, dass es vom Sensor erfasst wird. Zu diesem Zeitpunkt wurde der zufällige Fehler hinzuaddiert, was auch die in der Realität meist nicht realisierbare, perfekte Gitteranordnung der Knoten kompensiert.

Vorzeitige und verspätete Auslöseereignisse haben einen gewissen Anteil an vertauschten Ereignissen zur Folge, was die Korrelation mit bekannten Reihenfolgen nachhaltig erschwert. Außerdem kann die Geschwindigkeit nicht mehr so leicht festgestellt werden, da sich diese für das Objekt von Knoten zu Knoten verändert. Dieses Vorgehen simuliert aber nicht nur eine variierende Geschwindigkeit pro Objekt, sondern behindert auch die Differenzierung zwischen sich überholenden und dicht folgenden Objekten.

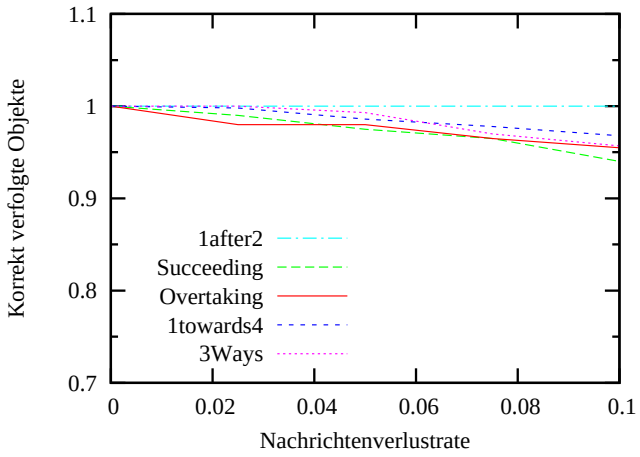
Die Fehler wurden mittels einer Gaußschen Zufallsverteilung erzeugt, wobei die Standardabweichung $\sigma = 0{,}45$ betrug. Dies führte zum einen zu den nicht perfekten Auslösezeitpunkten, die für alle Detektionsereignisse unterschiedlich und unabhängig von der Geschwindigkeit der Objekte waren. Zum anderen führte dies sogar zu 2 % vertauschter Auslöseereignisse, die in der Realität bei einem Abstand von $1{,}5$ m zwischen den Sensoren sehr unwahrscheinlich sind. Abbildung 3.9(a) und Abbildung 3.9(b) zeigen die Ergebnisse für alle Bewegungsmuster und verschiedene Nachrichtenverlustraten.

Bei den Simulationen, deren Ergebnisse in Abbildung 3.9(a) dargestellt sind, wurde kein Fehler auf die Auslösezeitpunkte addiert. Hier ist zu erkennen, dass alle Bewegungsmuster korrekt erkannt und somit alle Objekte korrekt verfolgt wurden, falls alle Nachrichten ankommen. Mit zunehmender Nachrichtenverlustrate sinkt die Rate der korrekt verfolgten Objekte auf etwa 95 %, außer für das einfachste Bewegungsmuster *1after2*. Bei diesem einfachen Ablauf werden trotz Nachrichtenverlust alle Objekte richtig verfolgt. Das Bewegungsmuster *3Ways* ist aufgrund der nahezu gleichzeitig an derselben Kreuzung ankommenden und sich gleichzeitig weiterbewegenden Objekte besonders schwierig. Wenn drei Objekte aus verschiedenen Richtungen bei derselben Kreuzung ankommen, können die Knoten das Ereignis noch zu den bisherigen Ereignissen in ihrem Nachrichtenpuffer zuordnen. Bewegen sich die Objekte dann aber auf demselben Weg weiter, können alle verantwortlich für die folgenden Auslöseereignisse sein und die Knoten in diesem Flur haben keine Möglichkeit zwischen ihnen zu unterscheiden. Deshalb werden die Objekte zwar in Bezug auf ihren Weg richtig verfolgt, nicht aber unbedingt in ihrer Ankunftsreihenfolge.

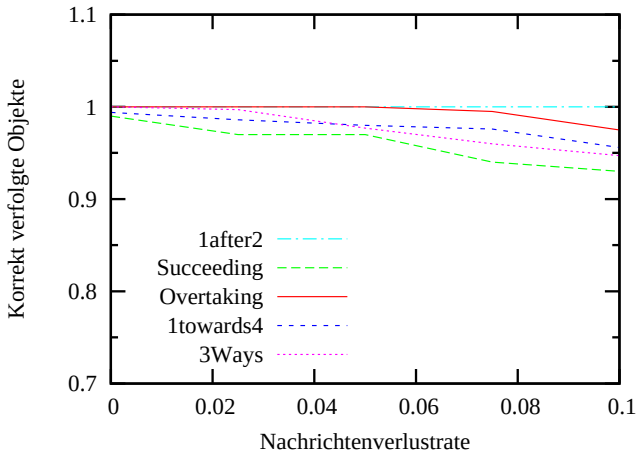
Wie in Abbildung 3.9(b) zu sehen ist, sind die Ergebnisse mit variierenden Auslösezeitpunkten nahezu genauso gut wie diejenigen, bei denen die Zeitpunkte nicht fehlerbehaftet waren, was die Robustheit des Verfahrens verdeutlicht.

Einfluss fehlender Auslöseereignisse

Da Hardwarefehler dazu führen können, dass sich bewegende Objekte von einem Sensor übersehen werden, wurde auch der Einfluss fehlender Auslöseereignisse untersucht. Fehlerraten von bis zu 10 % wurden simuliert. Während Abbildung 3.10(a) die Ergebnisse



(a) Standardabweichung $\sigma = 0$

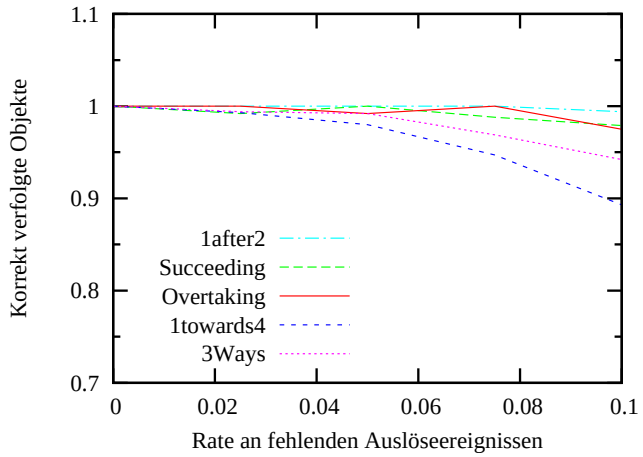


(b) Standardabweichung $\sigma = 0{,}45$

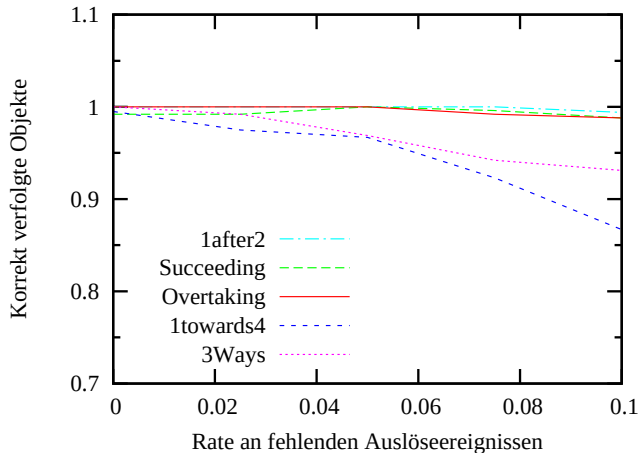
Abbildung 3.9: Einfluss von zeitlicher Variation

für den Fall zeigt, dass die Auslösezeitpunkte exakt sind, stellt Abbildung 3.10(b) die Ergebnisse für den Fall, dass sie variieren, dar. Die Nachrichtenverlustrate war bei diesen Simulationen 0 %, so dass der Einfluss der fehlenden Auslöseereignisse analysiert werden konnte.

Für Raten von bis zu 5 % an fehlenden Auslöseereignissen ähneln die Ergebnisse denen aus dem vorhergehenden Abschnitt ohne Auslösefehler, aber stattdessen mit Nachrichtenverlusten, sehr. Dies ist nicht überraschend, da es aus Sicht der Nachbarn eines Knoten keinen Unterschied macht, ob dessen Sensor fälschlicherweise nicht ausgelöst hat oder seine resultierende Nachricht verloren gegangen ist. Nur für den direkt betroffenen Knoten



(a) Standardabweichung $\sigma = 0$



(b) Standardabweichung $\sigma = 0{,}45$

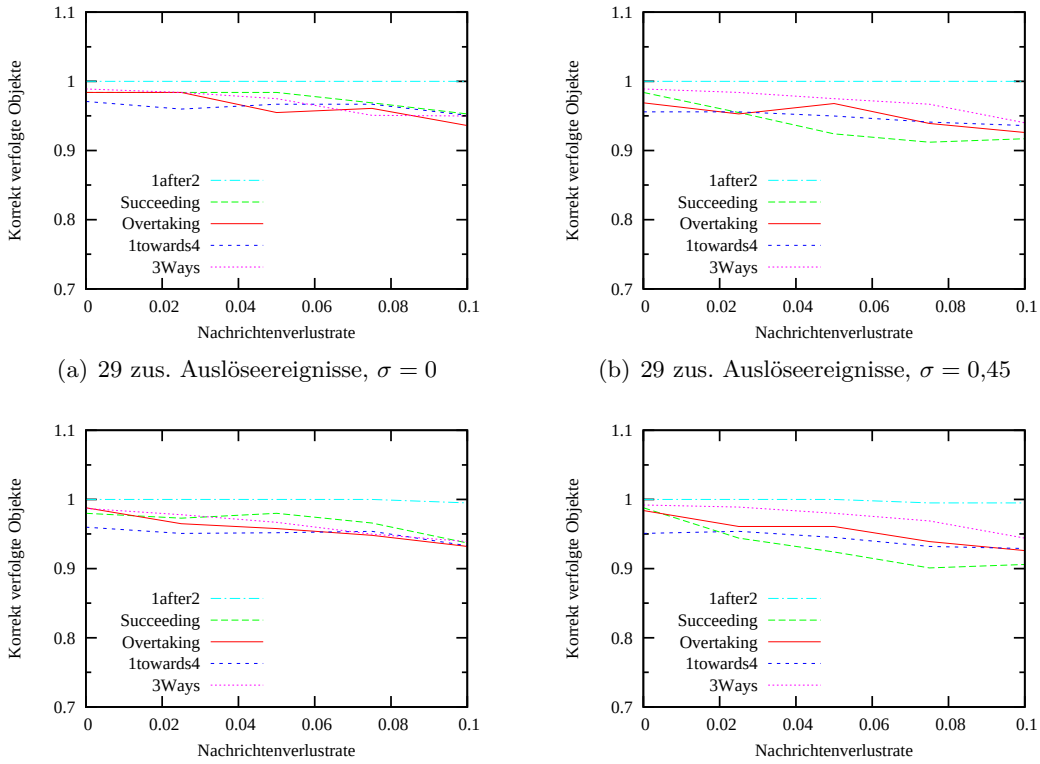
Abbildung 3.10: Einfluss fehlender Auslöseereignisse

gibt es einen Unterschied, da er einer Ereigniskette in seinem Nachrichtenpuffer kein eigenes Ereignis anfügen kann. Folgen beispielsweise zwei Objekte einander sehr dicht und der Knoten übersieht das erste, kann das durch das zweite Objekt hervorgerufene Auslöseereignis beim Korrelationsschritt zur falschen Entscheidung führen, was bei einem Nachrichtenverlust nicht der Fall wäre. Dieser Unterschied ist verantwortlich für die leicht geringere Rate an korrekt verfolgten Objekten im Vergleich zu den Ergebnissen aus dem vorigen Abschnitt.

Experimente mit Passiv-Infrarotsensoren haben jedoch gezeigt, dass diese – hervorgerufen durch Luftzug, Temperaturschwankungen oder Hardwarefehler – eher dazu neigen zu

viele als zu wenige Ereignisse zu erkennen. Aus diesem Grund wurde der Einfluss von zusätzlichen Auslöseereignissen evaluiert.

Einfluss von zusätzlichen Auslöseereignissen



(a) 29 zus. Auslöseereignisse, $\sigma = 0$

(b) 29 zus. Auslöseereignisse, $\sigma = 0{,}45$

(c) 58 zus. Auslöseereignisse, $\sigma = 0$

(d) 58 zus. Auslöseereignisse, $\sigma = 0{,}45$

Abbildung 3.11: Erkennungsrate über verschiedene Nachrichtenverlustraten mit zusätzlichen Auslöseereignissen

Zwei Raten an zusätzlichen Auslöseereignissen wurden untersucht: 29 sowie 58 zufällige zusätzliche Ereignisse pro Knoten pro Tag. Diese Raten entsprachen 4 % und 8 % an falschen Auslösungen, wenn jeder Knoten alle 2 Minuten (etwa einmal pro Simulation) auslöst.

Abbildung 3.11 zeigt die Raten an korrekt verfolgten Objekten für verschiedene Standardabweichungen und Nachrichtenverlustraten. Es ist zu sehen, dass die zusätzlichen Ereignisse das Objektverfolgungsverfahren kaum beeinträchtigen. Während der Fehler um etwa 2 % für niedrige Nachrichtenverlustraten steigt, sinkt der Einfluss der zusätzlichen

Ereignisse für höhere Nachrichtenverlustraten noch mehr. Wenn jede zehnte Nachricht verloren geht, ist kein Einfluss mehr erkennbar.

Dennoch ist festzustellen, dass die Beeinträchtigung der Erkennung vom Bewegungsmuster abhängt. Offensichtlich haben Fehlauslösungen den größten Einfluss, wenn sich mehrere Objekte nah beieinander befinden: Die zusätzlichen Ereignisse erschweren hier die Zuordnung von Sensorereignissen zu den richtigen Objekten.

3.5 Eigenschaften des Verfahrens

Das vorgeschlagene Verfahren erfüllt sowohl während der Installations- als auch in der Betriebsphase die in Abschnitt 3.1 beschriebenen Anforderungen an Objektverfolgungsverfahren.

Geringer Ressourcenbedarf: Sowohl die Topologieerkennung als auch das Objektverfolgungsverfahren arbeiten sehr ressourcensparend. Im Rahmen der Topologieerkennung sendet jeder Knoten nur vier bis sechs Nachrichten und muss die Distanzen zwischen benachbarten Knoten lediglich temporär in einer Matrix speichern. Die bedeutet zwar quadratischen Platzverbrauch, ist aber vor dem Hintergrund der recht geringen Netzdichte und der Möglichkeit, die Distanzen in Dezimetern in einem Byte repräsentieren zu können kein Problem. Hat ein Knoten beispielsweise 19 Nachbarn, benötigt er 40 Bytes für die Adressen der Knoten als Indizes und $20 \cdot 20 = 400$ Bytes für die Distanzen, also insgesamt 440 Bytes. Während der Objektverfolgung sendet ein Knoten pro Objekt nur eine Nachricht. Der Nachrichtenpuffer enthält für jedes in der direkten Nachbarschaft aufgetretene Ereignis nur die ID des aktivierten Knotens sowie den Auslösezeitpunkt. Die Startzeit, die Startposition, die Geschwindigkeit und die Anzahl der Auslösungen werden nur pro Objekt gespeichert. Die Größe der Historie kann leicht beschränkt werden; allerdings sollte sie mindestens so viel Platz bieten, dass alle Nachbarschaftsreihenfolgen gespeichert werden können, das heißt alle IDs der Nachbarknoten.

Geringer Installationsaufwand: Da das Verfahren nicht auf positionsabhängigen Informationen basiert, kann jeder Knoten beliebig positioniert werden. Nach der Ausbringung kann sofort die Initialisierungsphase gestartet werden, die dann automatisch abläuft. Anschließend wird (ebenfalls automatisch) die Betriebsphase gestartet.

Robustheit gegen Messfehler: Die in Abschnitt 3.4.1 beschriebenen Evaluationsergebnisse zeigen, dass die Erkennung der lokalen Topologie sehr robust ist gegen die in Graphstrukturen kaum vermeidbaren Fehler in der Distanzschätzung. Abschnitt 3.4.2 belegt die Robustheit des Objektverfolgungsverfahrens gegen Fehler in der Sensorik

wie zusätzliche oder fehlende Auslöseereignisse sowie eine zeitliche Variation in den Ereignissen.

Robustheit gegen unzuverlässige Kommunikation: Die simulative Evaluation zeigt weiterhin, dass beide Verfahren (die Erkennung der lokalen Topologie wie auch die Objektverfolgung) robust gegen Nachrichtenverlustraten von bis zu 8 % sind. Die Knoten sollten deshalb nur Nachrichten von Nachbarn verarbeiten, von denen sie mindestens 95 % empfangen. Mit Hilfe einer fortlaufenden Sequenznummer in den Nachrichten können Knoten solche Nachbarn identifizieren. Experimente in der FRONTS-Testumgebung haben gezeigt, dass Knoten, die nah beieinander sind und zwischen denen sich keine Wand befindet, meist sogar mehr als 98 % der Nachrichten empfangen. Sind für alle Kommunikationsverbindungen jedoch höhere Verlustraten zu erwarten oder ermittelt worden, müssen Nachrichten zweimal oder mehrmals gesendet werden.

Skalierbarkeit: Eine große Anzahl an Knoten stellt weder für die Erkennung der lokalen Topologie noch für die Objektverfolgung ein Problem dar, da die Algorithmen lokal arbeiten und die Nachrichtenlänge ohne Auswirkung beschränkt werden kann. Allerdings wurde das Verfahren nicht für extrem hohe Dichten entworfen, da davon ausgegangen wird, dass möglichst wenig Sensorknoten verwendet werden, um den Preis insgesamt niedrig zu halten.

Adaptivität an Netzveränderungen: An neue oder ausgefallene Knoten werden die Knotenreihenfolgen automatisch angepasst, sobald wieder ein Objekt den betroffenen Bereich passiert. Die neu hinzugekommenen Knoten können wahlweise lokal die Initialisierungsphase anstoßen oder ihre Nachbarschaftsreihenfolgen während der Betriebsphase erlernen.

Grenzen der relativen Positionen: Es bleibt anzumerken, dass die Objektverfolgung an Hand der Graphstruktur durchgeführt und bisher kein Wissen über die relative Lage von Kanten im Graphen erzeugt wird. Soll die relative Lage von Kanten ermittelt werden, könnten die Sensorknoten während der Initialisierungsphase die relativen Lagen der Ketten, die sie ermittelt haben, mittels der geschätzten Distanzen zwischen Nachbarknoten bestimmen. Die Sensorknoten können jedoch keine absoluten Positionen ermitteln. Die Ausrichtung des Graphen bleibt offen und es kann nicht erkannt werden, ob die Graphstruktur in der Abstraktion spiegelverkehrt ist. Sollen die relativen Positionen der Sensorknoten realen Positionen zugeordnet werden, muss mindestens zwei Knoten im Graphen eine absolute Position zugeordnet werden oder ein Testobjekt muss die Kanten passieren, so dass aus den entstehenden Meldungen dann z.B. Linien generiert und in einer Karte oder einem Grundriss angezeigt werden können, die der Anwender verschieben kann. Es kann zudem nicht unterschieden werden, ob zwei entgegenkommende Objekte

aneinander vorbeigehen oder aufeinandertreffen und umgekehren, da die Objekte nicht identifizierbar sind.

3.6 Zusammenfassung

Dieses Kapitel stellt zunächst einen dezentralen Algorithmus zur Erkennung von Nachbarschaftsanordnungen vor. Ein Knoten in einer komplexen Netzwerkstruktur ist damit in der Lage, seine direkten Nachbarn in eindimensionale Reihenfolgen zu bringen, die mit der Realität übereinstimmen. Die Nachbarn auf jeder Seite des Knotens werden hierbei auf der Basis von geschätzten Distanzen in eine separate Reihenfolge gebracht. Dass die Distanzschätzungen perfekt sind, ist nicht notwendig. Solange die Schätzfehler die Relationen der Knotenpositionen bewahren, ist das Ergebnis korrekt.

Darauf aufbauend präsentiert das Kapitel ein dezentrales Objektverfolgungsverfahren, welches mit der Kenntnis über die Netzstruktur sowie die Knotenreihenfolgen in der Lage ist, mehrere Objekte zu verfolgen. Das Netzwerk wird hierbei in eine Graphstruktur überführt, in dessen Kanten ein eindimensionales Verfolgen durchgeführt wird. An Knoten des Graphen wird die Aufgabe an die Sensorknoten der nächsten Kante übergeben. Zum Verfolgen innerhalb eines Abschnitts nutzt das Verfahren die Tatsache, dass ein passierendes Objekt von den Sensorknoten genau in der Reihenfolge detektiert wird, wie diese angeordnet sind. Über einen Nachrichtenaustausch werden Nachbarn über Auslöseereignisse informiert und die durch ein Objekt hervorgerufenen Ereignisse miteinander korreliert.

Simulative Untersuchungen belegen die Robustheit des Verfahrens gegenüber Mess- und Kommunikationsfehlern. Nur mittels lokaler Interaktion und ohne Positionsinformationen werden verschiedene, komplexe Bewegungsmuster erkannt.

In diesem Kapitel wurde gezeigt, wie eine gegebene Netzstruktur erkannt und gezielt bei der Objektverfolgung eingesetzt werden kann. Hervorgehend aus den in Abschnitt 2.6.1 beschriebenen Eigenschaften von Sensornetzen und der zentralen Bedeutung des Energiesparens, beschäftigt sich das folgende Kapitel mit dieser Fragestellung bei der Verwendung eines Duty-Cycle-Verfahrens.

Kapitel 4

Duty-Cycle-Management

Aufgrund der kleinen Energiespeicher von Sensorknoten ist die Verwendung verschiedener Aktivitätszustände meist unvermeidbar. Batterien und Akkumulatoren mit großer Kapazität sind typischerweise selbst sehr groß, was der angestrebten Miniaturisierung der Geräte entgegensteht. Häufiges Ersetzen von Batterien würde zudem die Wartungskosten aufgrund der großen Geräteanzahl sowie der eventuell schwierigen Zugänglichkeit der Geräte in nicht akzeptabler Weise in die Höhe treiben. Eine lange Betriebsdauer kann daher aber nur erreicht werden, wenn Hardwarekomponenten, die viel Strom verbrauchen, zeitweise in einen Energiesparmodus versetzt werden. Während der Energiesparphasen können diese jedoch weniger oder gar keine Operationen ausführen.

Aus Abschnitt 2.6 lässt sich ableiten, dass das Funkmodul und der Prozessor großen Anteil am Energieverbrauch haben. Um eine lange Betriebsdauer realisieren zu können, müssen deshalb beide Komponenten zeitweilig ausgeschaltet werden. Im Gegensatz zum Prozessor, der unabhängig von anderen Geräten arbeitet, muss das Ausschalten des Funkmoduls koordiniert und sorgfältig geplant werden. Ein unkoordiniertes Duty-Cycling resultiert potentiell in langen Nachrichtenverzögerungen, wenn auf die Wachphase von Knoten in einem Multihop-Pfad gewartet werden muss. Im ungünstigen Fall können benachbarte Knoten gar nicht miteinander kommunizieren, weil sie immer zu unterschiedlichen Zeiten aktiv sind.

Dieses Kapitel beschäftigt sich deshalb mit dem Problem, trotz eines extrem niedrigen Duty-Cycles (1 % oder weniger) geringe Multihop-Verzögerungen zu erreichen. Auch wenn nur einige Kommunikationsmuster (z. B. vom Netz zur Senke) höhere Anforderungen an die Verzögerung stellen als andere, müssen dennoch alle Kommunikationsmuster (z. B. von der Senke zum Netz und Broadcast-Kommunikation) unterstützt werden, um die Ausführung anderer Protokolle wie eine Zeitsynchronisation oder Datenaggregation zu ermöglichen. Das hier untersuchte Verfahren basiert auf dem gestaffelten Aufwachen von Geräten in den bevorzugten Kommunikationsrichtungen, so dass eine Art Welle entsteht.

Zunächst ausgehend von einer linienförmigen Netzstruktur werden anschließend auch komplexere und allgemeinere Strukturen diskutiert. Zusätzlich werden der Einfluss von unzuverlässiger Kommunikation sowie die Bedingungen für den praktischen Einsatz des Verfahrens detailliert untersucht.

Entstanden ist diese Arbeit in dem Projekt *FleGSens* [93], in dem ein System zur sicheren Grenz- und Liegenschaftsüberwachung entwickelt wurde. Dabei wurden Sensorknoten entlang einer Grenze positioniert, so dass linienförmige Strukturen entstanden. Ein wichtiger Sicherheitsaspekt war, dass die eingesetzten Protokolle resistent gegen Angreifer sein sollten – insbesondere gegen Angreifer, die versuchen einen Schlafentzugsangriff durchzuführen. Hierbei wird bei den Knoten ein unnatürlich hoher Energieverbrauch erzeugt, damit diese mangels Spannungsversorgung ausfallen und weder Objekte detektieren noch Nachrichten versenden können. Dort könnte ein Eindringling die Grenze dann unentdeckt passieren. Weiterhin sollten sicherheitskritische Nachrichten, so genannte *Alarmnachrichten*, ohne große Verzögerungen zur Senke transportiert werden. Die Verzögerung sollte hier deshalb v.a. bei Nachrichten vom Netz zur Senke niedrig sein.

Im Folgenden wird zunächst auf spezielle Anforderungen an Duty-Cycle-Verfahren eingegangen. Abschnitt 4.2 stellt anschließend andere Arbeiten aus der Literatur, sowie deren Vor- und Nachteile dar. In Abschnitt 4.3 wird mit CUPID (CommUnication Pattern Informed Duty Cycling in Sensor Networks) ein Verfahren vorgestellt, das innerhalb bestimmter Grenzen flexibel an die Anforderungen der Anwendung bezüglich Nachrichtenverzögerung und Duty-Cycle angepasst werden kann. Auch theoretische Aspekte, die beim Design eines Duty-Cycle-Verfahrens beachtet werden müssen, damit es in der Praxis einsetzbar ist, werden aufgezeigt und diskutiert. Abschnitt 4.4 beinhaltet eine simulative, Abschnitt 4.5 eine experimentelle Evaluation. Abschließend werden die Ergebnisse des Kapitels in Abschnitt 4.7 zusammengefasst.

Teile der in diesem Kapitel präsentierten Ergebnisse sind in gemeinsamer Arbeit mit Dennis Pfisterer und Stefan Fischer entstanden und wurden in [59] veröffentlicht.

4.1 Anforderungen an Duty-Cycle-Verfahren

Ein Duty-Cycle-Verfahren sollte verschiedene Anforderungen erfüllen:

Permanent geringer Energieverbrauch: Ein konstant geringer Energieverbrauch ist erforderlich, um einen dauerhaften Betrieb des Sensornetzes zu gewährleisten. Ein zeitweiliger Mehrverbrauch an Energie kann zum Ausfall von Knoten und damit zu einer Partitionierung oder einem Ausfall des Netzes führen.

Geringe Ende-zu-Senke-Verzögerung: Messwerte oder (Alarm-)Meldungen müssen üblicherweise zu einer Senke transportiert werden, über die die Informationen wiederum dem Netzbetreiber bereitgestellt werden. Gerade im Falle eines Alarms ist eine geringe Verzögerung dieser Nachricht wichtig, damit der Betreiber die Möglichkeit hat, kurzfristig zu reagieren. Da auch vom entferntesten Knoten Nachrichten gemeldet werden können, ist eine geringe maximale Ende-zu-Senke-Verzögerung von essentieller Bedeutung.

Unterstützung von Broadcast-Kommunikation: Um Daten mit den Nachbarknoten austauschen, korrelieren und aggregieren zu können, erfordern nahezu alle Sensornetzanwendungen, dass jeder Knoten mit allen seinen Nachbarn kommunizieren kann. Damit jedoch die zur Verfügung stehende Bandbreite effizient genutzt und Energie gespart werden kann, ist es zwingend notwendig, dass Nachrichten, die alle Nachbarn empfangen sollen, per Broadcast gesendet werden können. Dazu müssen alle Nachbarn des Senders gleichzeitig empfangsbereit sein.

4.2 Verwandte Arbeiten

Die in Abschnitt 2.9 vorgestellten Duty-Cycle-Verfahren gehen von grundsätzlich anderen Voraussetzungen aus, wie beispielsweise der Anwesenheit von Spezialhardware bei Wakeup-Radios oder der Möglichkeit des bedarfsorientierten Wachbleibens. Während die erste Annahme zur Erhöhung der Kosten führt, widerspricht die zweite Annahme der Resistenz gegen Schlafentzugsangriffe und erfordert inhärent einen größeren Energiespeicher, was wiederum der kleinen Baugröße entgegenwirkt.

Verschiedene Arbeiten beschäftigen sich aber auch mit *gestaffelten Aufwachphasen*, bei denen Knoten in der Weiterleitungsrichtung einer Nachricht nacheinander aufwachen. Stankovic et al. [14] nennen dieses Verfahren *streamlined wakeup*, nehmen aber zuverlässige Kommunikationsverbindungen an, so dass diese nicht realistisch abgebildet werden. Vor allem aber gehen sie von vielen redundanten Sensorknoten aus, bei denen die Sensoraktivität vom Betrieb des Prozessors abhängt. Ein Schlafen von Knoten, ohne die Datenerhebung zu unterbrechen, wird so durch eine hohe Abdeckungsdichte möglich. Die Autoren legen den Fokus ihrer Untersuchung auf eine möglichst schnelle Detektion eines Ereignisses durch die Sensoren und betrachten nicht, wie lange es dauert, das Ereignis zu melden.

Li et al. nennen den Ansatz in [71] *fast path algorithm*. Sie etablieren mittels zusätzlicher Wachphasen einen einzelnen Pfad zwischen Quell- und Zielknoten, aber dies geschieht unabhängig vom globalen Duty-Cycle und unabhängig von anderen Pfaden. Sollen jedoch beispielsweise alle Knoten mit der Senke kommunizieren, ist es sinnvoller, das gesamte Netz zu betrachten, anstatt voneinander unabhängige Pfade zu konstruieren. Ausgehend von beschränkten Ressourcen kann ein Knoten nur wenige Pfade verwalten, so dass ihre Variante nicht skaliert. Sie ist vorwiegend geeignet, wenn einzelne Knoten große Datenmengen an einen anderen Knoten senden sollen. In der vorliegenden Arbeit wird dagegen eine Lösung für das gesamte Netz präsentiert, die eine effiziente Kommunikation vom Netz zur Senke und umgekehrt ermöglicht.

In [8] überführen die Autoren das Netz in einen Graphen, bei dem die Sensorknoten durch Knoten und die Kommunikationsverbindungen durch Kanten repräsentiert werden. Das Ziel besteht nun darin, jedem Knoten einen Slot zu zuweisen, in dem er empfangsbereit ist, so dass die maximale Ende-zu-Ende-Verzögerung minimal ist. Die Autoren beweisen

anhand der graphentheoretischen Abstraktion, dass die Minimierung der Ende-zu-Ende-Verzögerung NP-hart ist, wenn alle Daten an zentraler Stelle verfügbar sind. Sie leiten daraus eine optimale Lösung für eine Rechteckstruktur ab, bei der die Knoten an einem Gitter ausgerichtet sind und genau vier Nachbarn haben. Solch ein Zustand ist in der Realität allerdings schwer herzustellen und auch sie gehen von zuverlässiger Funkkommunikation aus. Weiterhin beträgt der kleinste untersuchte Duty-Cycle 5%, sofern keine Nachrichten gesendet werden. Sendet ein Knoten eine Nachricht, muss er sowohl wissen, wann der Zielknoten empfangsbereit ist, als auch zu dieser Zeit zusätzlich aktiv sein. Da nicht alle seine Nachbarn gleichzeitig empfangsbereit sind, wird bei diesem Vorschlag keine Broadcast-Kommunikation unterstützt. In der vorliegenden Arbeit werden dagegen Duty-Cycles bis 0,1% betrachtet und Broadcast-Kommunikation ermöglicht.

Keshavarzian et al. [49] schlagen zwei Methoden vor, die sie *crossed-ladders* bzw. *multi-parent scheme* nennen. Dabei werden Knoten in Abhängigkeit von ihrer hop-basierten Distanz zur Senke in Gruppen zusammengefasst, wobei alle Knoten einer Gruppe gleichzeitig wach sind. Doch auch ihre theoretischen Untersuchungen basieren auf zuverlässigen Kommunikationsverbindungen und lassen das Hidden-Terminal-Problem vollständig außer Acht, welches bei ihrem ersten Vorschlag von Relevanz ist. Tests haben jedoch gezeigt, dass die Annahme der zuverlässigen Kommunikation unrealistisch ist (vgl. [132]). Weiterhin gehen die Autoren von einem kleinen Netzdurchmesser von höchstens 4 Hops aus. Ihr zweiter Vorschlag unterstützt keine Broadcast-Kommunikation, da das Netz in mehrere Partitionen eingeteilt wird, die unabhängig voneinander arbeiten. Ihre Simulationsergebnisse beziehen sich ausschließlich auf die Erzeugung der Partitionen.

Alle genannten Ansätze betrachten ausschließlich den Fall von seltenen Ereignissen, das heißt, dass einzelne Knoten sporadisch Nachrichten zu einer Senke senden und nicht regelmäßig in kurzen Intervallen, wie es beispielsweise das Aufzeichnen von Umweltdaten erfordern kann. Einige Verfahren skalieren nicht oder unterstützen keine Broadcast-Kommunikation. Weiterhin basieren die Untersuchungen von vielen Autoren auf einem zeitlich invarianten, scheibenförmigen Kommunikationsbereich. Hier bleibt offen, welche Auswirkungen variierende Reichweiten haben.

Im Gegensatz zu diesen Ansätzen wird in der vorliegenden Arbeit die in der Realität unvermeidbare unzuverlässige Kommunikation berücksichtigt. Der vorgeschlagene Algorithmus skaliert und stellt sicher, dass alle potentiellen Empfänger eines sendenden Knotens empfangsbereit sind, so dass auch Broadcast-Kommunikation unterstützt wird und Knoten sich nicht um die Wachzeiten von Nachbar- oder Vaterknoten kümmern müssen.

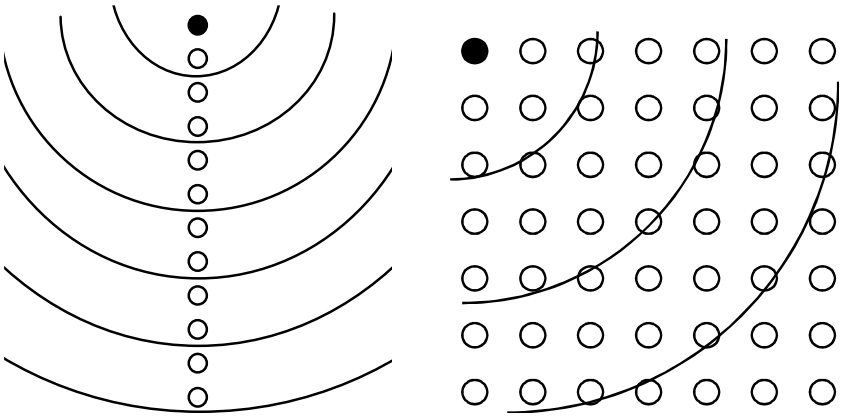
4.3 CUPID – Communication Pattern Informed Duty-Cycling

Der folgende Abschnitt beschreibt das Duty-Cycle-Verfahren CUPID, welches flexibel an das Kommunikationsaufkommen sowie die Anforderungen an Verzögerung und

Duty-Cycle angepasst werden kann. Ziele sind, die Ende-zu-Ende-Verzögerung, die Kollisionswahrscheinlichkeit und den Energieverbrauch zu minimieren und gleichzeitig Broadcast-Kommunikation zu ermöglichen.

4.3.1 Grundlegende Funktionsweise

Um eine geringe Ende-zu-Ende-Verzögerung zu erreichen, ist es notwendig, dass alle Knoten, die eine zur Senke zu transportierende Nachricht weiterleiten, nacheinander wach sind. Die grundlegende Idee besteht deshalb darin, dass Gruppen von Nachbarknoten nacheinander in den Wach- und wieder in den Schlafmodus wechseln. Alle Knoten einer Gruppe folgen demselben Schlaf-/Wachzyklus. Die Gruppen wachen nach und nach in Abhängigkeit von ihrer hop-basierten Distanz zu einem Referenzknoten (z. B. der Senke) auf, bleiben für eine bestimmte Zeit wach und gehen nacheinander wieder schlafen. Insgesamt formen die Zyklen aller Knoten eine *Welle* von der Senke zum entferntesten Knoten und wieder zurück. Dadurch können Nachrichten (wie Autos in einer grünen Welle bei Ampelschaltungen) sofort weitergeleitet werden.



(a) Linienförmig (b) Rechteckförmig

Abbildung 4.1: Hop-basierte Distanzen (Kreisausschnitte) zu einem Referenzknoten (gefüllter Kreis) in unterschiedlichen Strukturen

Beispielszenarien sind schematisch in Abbildung 4.1 dargestellt. In Abbildung 4.1(a) sind die Knoten in einer eindimensionalen Reihe angeordnet. Die Ringe begrenzen die hop-basierte Distanz zu einem Referenzknoten, der als ausgefüllter Kreis gekennzeichnet ist. Jeder andere Knoten gehört nun in Abhängigkeit von seiner Distanz zu einer Gruppe.

Neben der linienförmigen Struktur wurde im Rahmen dieser Arbeit auch der allgemeinere Fall einer rechteckförmigen Struktur betrachtet, wie sie in Abbildung 4.1(b) dargestellt

ist. Der Referenzknoten befindet sich hier beispielhaft in der linken oberen Ecke, kann aber ebenso gut in der Mitte des Netzes platziert sein.

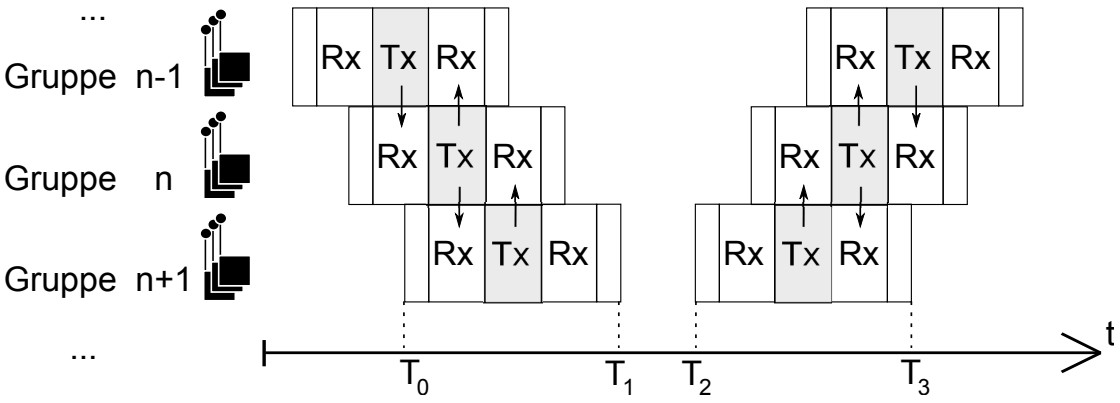


Abbildung 4.2: Überblick über die Phasen von CUPID

Abbildung 4.2 zeigt beispielhaft drei Gruppen aus der linienförmigen Netzwerkstruktur: Gruppe $n-1$, n und $n+1$, wobei Gruppe n n Hops vom Referenzknoten entfernt ist, Gruppe $n-1$ $n-1$ Hops usw. Wie bereits erwähnt, folgen alle Knoten einer Gruppe demselben Schlaf-/Wachzyklus. Die Wachphasen einer Gruppe sind durch weiße und graue Boxen dargestellt. Auf der Abszisse ist die Zeit aufgetragen. Gruppe $n+1$ wacht zu den Zeitpunkten T_0 und T_2 auf und geht zu den Zeitpunkten T_1 und T_3 wieder in den Schlafmodus. Innerhalb einer Wachphase ist ein Knoten auch immer empfangsbereit. Während der Schlafphase ist auch das Funkmodul inaktiv. Im mittleren Drittel der Wachphase darf der Knoten zusätzlich zum Empfangen von Nachrichten (gekennzeichnet mit Rx) auch Nachrichten senden (gekennzeichnet mit Tx). Die schmalen Balken am Anfang und Ende der Wachphasen dienen der Kompensation von asynchronen Uhren. Sie werden im Weiteren genauer erläutert.

Durch dieses Vorgehen ist gewährleistet, dass alle Nachbarn eines Senders wach sind. Ein sendender Knoten aus Gruppe n erreicht gleichzeitig seine Nachbarn aus Gruppe $n-1$ wie auch aus Gruppe $n+1$. In weiter entfernten Gruppen hat der Knoten keine direkten Nachbarn. Da die Welle durch das gesamte Netz fortgeführt wird, gilt dies für alle Gruppen.

Die folgenden Abschnitte beschreiben einzelne Aspekte von CUPID detailliert. Abschnitt 4.3.2 listet die Annahmen, die CUPID zu Grunde liegen, auf. Abschnitt 4.3.3 erläutert Parameterabhängigkeiten und Implikationen, Abschnitt 4.3.4 zeigt, wie CUPID zur Laufzeit auf den Knoten etabliert werden kann.

4.3.2 Annahmen

Das hier präsentierte Verfahren nimmt an, dass die Uhren der Knoten auf irgendeine Art zeitsynchronisiert werden. Wie dies genau geschieht, spielt keine Rolle. Wichtig ist aber, dass das Protokoll eine bekannte Genauigkeit garantiert, mit der CUPID arbeiten kann. Details hierzu werden in Abschnitt 4.3.3 diskutiert.

Weiterhin benötigt das Verfahren einen definierten Referenzknoten (typischerweise eine Senke), zu dem alle anderen Knoten ihre hop-basierte Distanz bestimmen.

Der Netzwerkdurchmesser muss zumindest näherungsweise bekannt sein. Eine Möglichkeit zur Ermittlung des Durchmessers wird in Abschnitt 4.3.4 angegeben.

CUPID geht von typischen Sensorknoten aus, die nicht über ein Wakeup-Radio verfügen, und basiert auf festen Knotenpositionen. Für mobile Netze ist CUPID nicht geeignet.

4.3.3 Design

Grundsätzlich wird davon ausgegangen, dass im Design ein festes Szenario vorgegeben ist, bei dem einige Parameter fix sind (wie beispielsweise der Netzwerkdurchmesser oder die Position der Senke) und einige erwünscht, aber nicht fix sind (z. B. die maximale Ende-zu-Ende-Verzögerung oder der Duty-Cycle). Im Folgenden wird gezeigt, welche Parameter für CUPID relevant sind und wie diese sich gegenseitig beschränken.

Abkürzung	Einheit / Wertebereich	Parameter
ND	$[Hops]$	Netzwerkdurchmesser
EEZ	$[s]$	Ende-zu-Ende-Verzögerung
DC	$\in [0, 1]$	Duty-Cycle
$d_{slotlen}$	$[s]$	Sendeslotlänge
d_{sil}	$[s]$	Länge der Ruhephase
d_{SFL}	$[s]$	Superframelänge
d_{tol}	$[ms]$	Toleranzlänge

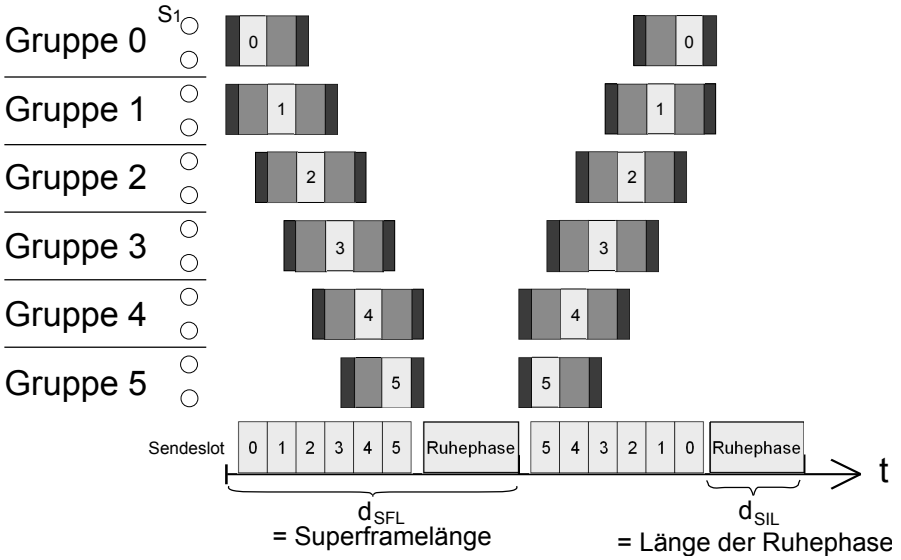
Tabelle 4.1: Für CUPID relevante Parameter

Tabelle 4.1 listet die relevanten Parameter mit ihren Einheiten oder Wertebereichen und die im Weiteren verwendeten Abkürzungen auf. Während die ersten drei Parameter bereits in den Grundlagen definiert wurden, werden die letzten vier Parameter anhand von Abbildung 4.3 erläutert.

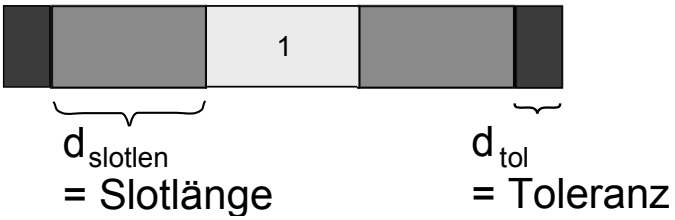
Alle Knoten einer Gruppe teilen einen so genannten *Sendeslot*, in dem sie per CSMA/-CA auf das Funkmedium zugreifen. Die Gesamtanzahl an Sendeslots entspricht dem Netzwerkdurchmesser ND, da es ND Gruppen gibt. Nach Ablauf des letzten Sendeslots

folgt eine *Ruhephase* der Länge d_{sil}, in der kein Knoten wach ist. Die Sendeslots, die Ruhephase und so genannte *Toleranzphasen* bilden zusammen einen *Superframe*.

Toleranzphasen der Länge d_{tol} ermöglichen den Einsatz des Verfahrens, auch wenn die Uhren der Knoten nicht perfekt synchronisiert sind. Dies wird im nächsten Abschnitt genauer beschrieben. Für die *Superframelänge* gilt: $d_{SFL} = ND \cdot d_{slotlen} + d_{sil} + 2 \cdot d_{tol}$.



(a) CUPID im Detail



(b) Struktur einer Wachphase

Abbildung 4.3: CUPID im Detail

Beim Anwendungsdesign muss als erstes entschieden werden, in welche Richtung die Welle innerhalb eines Superframes laufen soll (z. B. neunmal vom Netz zum Referenzknoten und einmal umgekehrt). So wird CUPID der – durch die Anwendung vorgegebenen – Datenflussrichtung optimal angepasst.

Der folgende Abschnitt diskutiert wie CUPID mit dem kaum vermeidbaren Problem nicht perfekt synchronisierter Uhren umgeht.

Umgang mit Synchronisationsungenauigkeiten

Koordinierte Schlaf-/Wachzyklen erfordern die Synchronisation von Uhren, welche – bis zu einer protokoll- und hardwareabhängigen Genauigkeit – durch ein geeignetes Protokoll erreicht werden kann. Die Hardwarespezifikation beschreibt die maximale Ungenauigkeit, die dazu führt, dass die Uhren auseinander driften. Sei die Genauigkeit eines Oszillators beispielsweise mit 20 ppm (parts per million) spezifiziert. Der ungünstigste Fall besteht dann darin, dass die Uhr eines Knoten 20 ppm zu schnell und die eines anderen 20 ppm zu langsam läuft. Binnen 25 s erhöht sich die Differenz zwischen beiden Uhren um eine Millisekunde und binnen 5 Minuten um 12 ms. Lässt man nun alle 5 Minuten das Zeitsynchronisationsprotokoll ablaufen, ergibt sich ein maximaler Zeitunterschied von 12 ms (zuzüglich der Synchronisationsungenauigkeit).

Ein Duty-Cycle-Verfahren muss den maximalen Zeitunterschied tolerieren. Dieser wird deshalb mit d_{tol} bezeichnet. Zwei unterschiedliche Herangehensweisen kamen prinzipiell in Frage, um sicherzustellen, dass alle Nachbarknoten eines Senders wach sind: Der erste Ansatz besteht darin, alle Knoten vorzeitig (d_{tol} vor dem eigentlichen Beginn der Wachphase) aufwachen zu lassen und später (d_{tol} ms nach Ende der eigentlichen Wachphase) in den Schlafmodus zu versetzen. So kompensieren die potentiellen Empfänger eine mögliche Uhrenabweichung. Für den Fall, dass die Uhren perfekt synchronisiert sind, findet während der Toleranzphasen keine Kommunikation statt und die Menge an unnötig aufgewendeter Energie E_{max_wasted} ist maximal. Es gilt:

$$E_{max\_wasted} = 2 \cdot d_{tol} ms. \tag{4.1}$$

Alternativ können aber auch, wie in Abbildung 4.4 dargestellt, die Sendeslots verkürzt werden. Alle Sender warten hierbei zu Beginn ihres Sendeslots d_{tol} ms, bis sie mit dem Senden beginnen, und beenden das Senden d_{tol} ms vor Ende des Sendeslots. Hier gilt:

$$E_{max\_wasted} = 3 \cdot 2 \cdot d_{tol} ms. \tag{4.2}$$

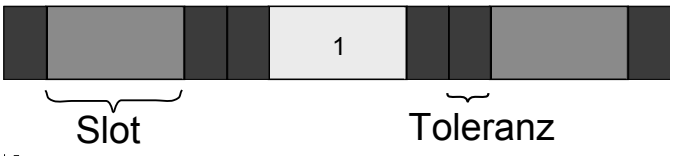


Abbildung 4.4: Kompensation von Synchronisationsungenauigkeiten durch Sender

Da bei der zweiten Herangehensweise im schlimmsten Fall das Dreifache an Energie verschwendet wird, wurde der erste Ansatz gewählt und umgesetzt, wie es Abbildung 4.3

bereits zeigt. Allerdings muss bedacht werden, dass bei der gewählten Variante Slots direkt aufeinander folgen, was den Nachteil mit sich bringt, dass asynchrone Uhren zu sich überlappenden Slots und somit zu mehr Wettbewerb beim Medienzugriff führen.

Parameterabhängigkeiten

Nicht alle Parameterwerte für Netzwerkdurchmesser, Duty-Cycle, Slotlänge und Superframelänge (bzw. Ende-zu-Ende-Verzögerung) lassen sich miteinander kombinieren. Inwiefern die Parameter sich gegenseitig bedingen und welche Wertebelegungen möglich sind, beschreibt der nun folgende Abschnitt.

Zunächst wird der Duty-Cycle betrachtet. Der Duty-Cycle von Knoten, die weder zur ersten noch zur letzten Gruppe gehören, ist am größten, da diese in jedem Superframe drei Sendeslots lang wach bleiben (ihren eigenen, den ihrer Vorgängergruppe und den ihrer Nachfolgergruppe). Für sie gilt:

$$DC = \frac{Wachzeit}{d_{SFL}} \tag{4.3}$$

$$= \frac{3 \cdot d_{slotlen} + 2 \cdot d_{tol}}{d_{SFL}}. \tag{4.4}$$

Erfordert die Anwendung beispielsweise eine maximale Ende-zu-Ende-Verzögerung von 300 ms bei einem Netzwerkdurchmesser von 30 Hops (was 30 Sendeslots entspricht), darf die Slotlänge höchstens 10 ms betragen, wenn die Länge der Ruhephase $d_{sil} = 0$ ist. $\frac{d_{EED}}{ND} = \frac{d_{SFL} - d_{slotlen} - d_{sil}}{ND}$ ist also eine obere Schranke für die Slotlänge. Die Superframelänge abzüglich der Länge eines Slots und d_{sil} entspricht genau der maximalen Ende-zu-Ende-Verzögerung, da die Senke frühestens im vorletzten Slot etwas empfangen kann. Die Slotlänge definiert somit auch die Mindestverzögerung pro Hop.

Reduziert man die Slotlänge für einen gegebenen Netzwerkdurchmesser, wird dadurch auch die Ende-zu-Ende-Verzögerung verringert. Allerdings müssen die Slots lang genug sein, um mindestens eine Nachrichtenübertragung zu ermöglichen. Die Zeit, die benötigt wird, um eine Nachricht zu senden und zu empfangen, ist also die absolute untere Schranke für die Slotlänge. Sie ist abhängig vom verwendeten Radiointerface sowie von der maximalen Nachrichtenlänge. Sollen Knoten aufgrund von einem dauerhaft höheren Datenaufkommen mehr als eine Nachricht pro Slot senden, müssen die Slots dementsprechend länger sein. Je mehr Knoten sich einen Slot teilen und je mehr Nachrichten pro Knoten gesendet werden müssen, desto größer muss die Slotlänge sein.

Der Netzwerkdurchmesser limitiert aber nicht nur die Slotlänge, sondern auch den größtmöglichen Wert für den Duty-Cycle für eine gegebene Ende-zu-Ende-Verzögerung. Um das Maximum des Duty-Cycles zu erreichen, wird keine Ruhephase durchgeführt

($d_{sil} = 0$). Für den Fall, dass die Uhren perfekt synchronisiert sind und nicht driften ($d_{tol} = 0$), gilt unter Verwendung von Gleichung 4.4:

$$DC \leqslant \frac{3 \cdot d_{slotlen} + 2 \cdot d_{tol}}{ND \cdot d_{slotlen} + d_{sil}} \tag{4.5}$$

$$\leqslant \frac{3 \cdot d_{slotlen} + 2 \cdot 0}{ND \cdot d_{slotlen} + 0} \tag{4.6}$$

$$\leqslant \frac{3}{ND}. \tag{4.7}$$

Je größer der Netzwerkdurchmesser ist, desto kleiner ist der maximal mögliche Duty-Cycle für eine gegebene Ende-zu-Ende-Verzögerung, sofern nur eine Welle zur Zeit durch das Netz läuft. Ist der Netzwerkdurchmesser beispielsweise 5, so beträgt der größtmögliche Duty-Cycle 60 %. Bei einem Netzwerkdurchmesser von 100 können dagegen höchstens 3 % Duty-Cycle realisiert werden.

Erfordert eine Anwendung geringere Energieverbräuche (das heißt kleinere Duty-Cycle-Werte), so kann die Slotlänge verringert und damit die Ruhephase verlängert werden. Es stellt sich aber auch die Frage, wie es möglich ist, den Duty-Cycle zu vergrößern, ohne die Superframelänge zu vergrößern. Es wäre beispielsweise vorstellbar, mehrere Wellen gleichzeitig durch das Netz laufen zu lassen, so dass Knoten während eines Superframes mehrmals wach sind. Allerdings hätte dies zur Folge, dass Wellen zu irgendeinem Zeitpunkt aufeinander zulaufen. Senden dann zwei Gruppen gleichzeitig zu der sich zwischen ihnen befindenden Empfängergruppe, kommt es systembedingt zu Kollisionen, da sich die Sender gegenseitig nicht hören. Da mehrere Wellen somit nicht sinnvoll erscheinen, wird dieser Fall nicht weiter betrachtet.

Unterstützen von beliebigen Kommunikationsmustern

Bisher wurde implizit angenommen, dass Daten vorwiegend von oder zu einem Referenzknoten fließen, also in Wellenbewegungsrichtung. CUPID ist für einen derartigen Datenfluss spezifiziert. Die Verzögerung bis zum Referenzknoten einer Nachricht, die zu einem zufälligen Zeitpunkt erzeugt und der Sendewarteschlange hinzugefügt wird, setzt sich zusammen aus der Zeitspanne zwischen dem Erzeugen und dem ersten Versenden und der Zeitspanne zwischen dem ersten Versenden und der Ankunft beim Referenzknoten. Erstere ist für alle Knoten im Mittel gleich, da die Superframes von allen Knoten gleich lang sind. Die Gesamtverzögerung hängt also nur von der Pfadlänge zum Referenzknoten ab.

Sollen jedoch zwei beliebige Knoten Nachrichten austauschen, kann es zu deutlich größeren Verzögerungen im Vergleich zu anderen Verfahren (wie z. B. dem synchronen Aufwachen) kommen. Man betrachte beispielsweise zwei benachbarte Knoten N_1 aus Gruppe 1 und

N_2 aus Gruppe 2. N_2 sendet eine Nachricht MSG_1 an N_1, welcher mit einer Nachricht MSG_2 antworten soll. Im ungünstigen Fall läuft die Welle gerade von Gruppe 1 zu Gruppe n. Dann ist der Sendeslot von N_1 beim Empfang von MSG_1 bereits vergangen und M_2 kann erst nach der Dauer von $2d_{SFL}$ gesendet werden.

CUPID ist also insbesondere für die Kommunikation in zwei Vorzugsrichtungen optimiert – zu oder von einem Referenzknoten. Multihop-Kommunikation orthogonal zur Wellenbewegungsrichtung dauert vergleichsweise lange.

Unterstützen von komplexen Netzwerkstrukturen

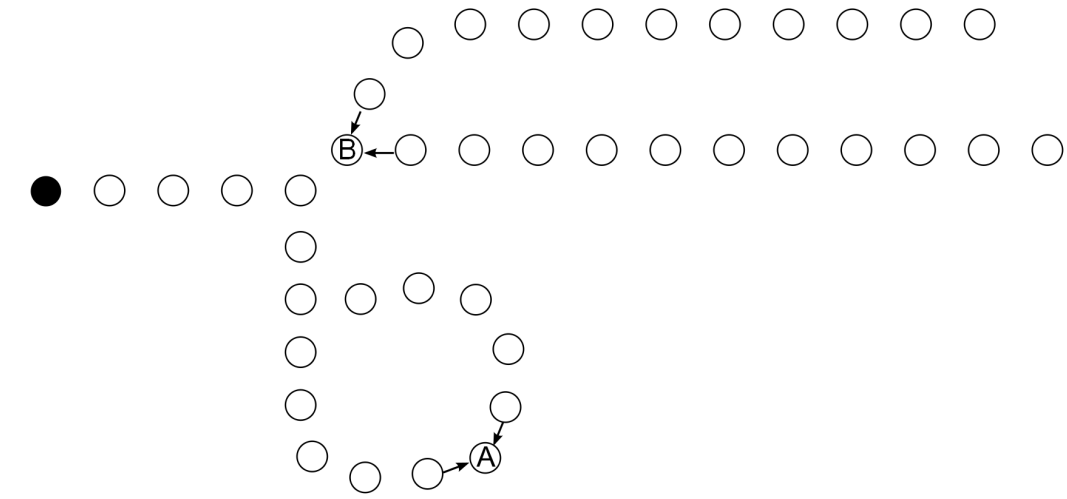


Abbildung 4.5: Komplexere Netzwerkstruktur

Komplexere Netzwerkstrukturen, wie z. B. in Abbildung 4.5 dargestellt, führen dazu, dass Wellenfronten aufeinander treffen, was das Kollisionsrisiko durch Hidden Terminals deutlich erhöht. Im Beispiel treffen zwei Wellenfronten, die sich von der Senke (links) entfernen, bei Knoten A aufeinander und führen systemimmanent zu Kollisionen. Bei Knoten B treffen nur Wellenfronten aufeinander, wenn die Welle sich in die andere Richtung bewegt, d. h. sich der Senke nähert.

In komplexen Strukturen mit Kreuzungen und Kammern lässt sich CUPID zwar auch anwenden, aber an einigen Stellen kommt es gehäuft zu Kollisionen. Betrachtet man das Netz als einen Graphen, lässt sich feststellen, dass Zyklen solche Stellen verursachen. Knotenpunkte sind problematisch, wenn sich die Welle in Richtung Senke bewegt.

4.3.4 Initialisierungsphase von CUPID

CUPID teilt sich in zwei Phasen: Nach der Ausbringung der Knoten und der Synchronisation der Uhren wird zunächst die Initialisierungsphase durchgeführt. Hier berechnen die Knoten selbstständig alle für CUPID notwendigen Parameter wie beispielsweise Aufwach- und Sendezeitpunkte, die Slotlänge und die Länge der Ruhephase, sind aber permanent aktiv.

Es wird davon ausgegangen, dass die Werte für Duty-Cycle und Toleranzdauer vom Anwendungsprogrammierer vorgegeben werden und somit zum Übersetzungszeitpunkt bekannt sind. Sie können aber natürlich auch in der Initialisierungsphase gesetzt werden. Aber auch die Gesamtanzahl der Gruppen sowie seine Gruppenzugehörigkeit muss jeder Knoten kennen. Auf welche Art ein Knoten diese beiden Parameter erhält, liegt nicht im Fokus dieser Arbeit, so dass dieser Abschnitt nur eine mögliche Variante vorstellt:

- Gruppenzugehörigkeit: Um jedem Knoten eine Gruppennummer zuzuordnen, werden die kürzesten Pfade zum Referenzknoten bestimmt. Dazu initiiert der Referenzknoten einen Baumaufbau eines Baum-Routings, wobei die (hop-basierten) Distanzen in den Nachrichten enthalten sind. Ist der Baumaufbau abgeschlossen, kennt jeder Knoten die Länge des kürzesten Pfades (das heißt die Distanz) zum Referenzknoten.

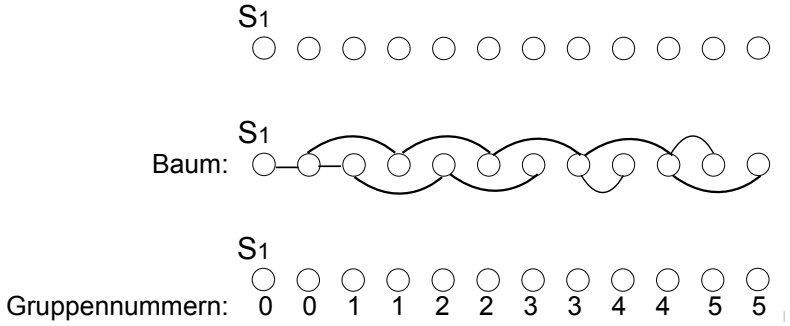


Abbildung 4.6: Erkennung der Gruppenzugehörigkeit

- Gesamtanzahl der Gruppen: Zunächst senden alle Knoten eine Nachricht an ihren Vaterknoten des Baum-Routings, der sie an den eigenen Vaterknoten weiterleitet, falls der enthaltene Werte größer ist, als die bisher empfangenen Werte. Die Nachricht enthält somit das Maximum aller Gruppennummern (der bereits empfangenen Gruppennummern und der eigenen Gruppennummer). Je weiter ein Knoten vom Referenzknoten entfernt ist, desto früher sendet er, so dass so wenig Nachrichten wie möglich gesendet werden. Der Referenzknoten kennt nun das Maximum, welches inkrementiert um eins der Gesamtzahl an Gruppen entspricht. Er flutet die Gesamtzahl ins Netz, damit sie alle Knoten kennen. Falls klar ist, welcher

Knoten am weitesten entfernt ist, kann dieser direkt seine Gruppennummer um eins inkrementiert fluten.

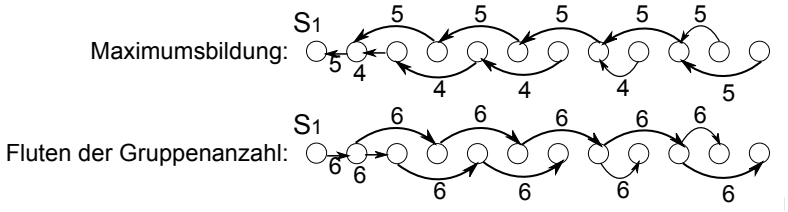


Abbildung 4.7: Erkennung der Gruppenanzahl

Algorithmus 2 Computation of CUPID's parameters

```
slot_len := (EEZ – 2*d_tol)/ND;
DC := (3*slot_len + 2*d_tol)*100/EEZ;
if DC > DC_DESIRED then
  slot_len := EEZ*DC_DESIRED/100;
  if slot_len > 2*d_tol then
    slot_len := slot_len – 2*d_tol;
    slot_len := slot_len / 3;
  else
    //Fehler: gewünschte Parameterkombination nicht möglich!
    slot_len := 1;
  end if
end if
d_sil := EEZ – slot_len*ND – 2*d_tol;
```

Algorithmus 2 zeigt, wie in der Initialisierungsphase aus gegebenen und ermittelten Parametern die Slotlänge und die Länge der Ruhephase berechnet werden. Zunächst werden die in der vorliegenden Netzwerkstruktur maximal mögliche Slotlänge *slot_len* und der sich durch diese Slotlänge ergebende Duty-Cycle *DC* bestimmt. Ist der Duty-Cycle größer als der gewünschte (*DC_DESIRED*), wird die Slotlänge aus dem gewünschten Duty-Cycle berechnet. Zuletzt wird die Länge *d_sil* der Ruhephase ermittelt.

Anschließend kann in die Betriebsphase übergegangen werden. Damit alle Knoten gleichzeitig in die Betriebsphase wechseln, flutet der Referenzknoten eine weitere Nachricht, die die Zeit T_C enthält, zu der gewechselt werden soll. T_C liegt hierbei mindestens soweit in der Zukunft, wie es dauert, eine Nachricht durch das gesamte Netz zu propagieren.

Während der Betriebsphase sind die Knoten nur zu den in Abschnitt 4.3.3 beschriebenen Zeiten wach und tauschen ausschließlich anwendungsorientierte – jedoch keine

CUPID-bezogenen – Nachrichten aus. Die Organisation von CUPID ist mit der Initialisierungsphase abgeschlossen.

Werden dem Netz neue Knoten hinzugefügt, können diese entweder auf die Nachrichten von Nachbarn hören und ihre Gruppennummer lernen oder die Initialisierungsphase kann wiederholt werden.

4.4 Simulative Evaluation

Um das vorgeschlagene Verfahren zu evaluieren, wurden zwei Vergleichsverfahren implementiert, welche die beiden am weitesten verbreiteten Verfahrensklassen repräsentieren und für die die gleichen Annahmen gelten, so dass sie auf derselben Hardware bei denselben Anwendungen einsetzbar sind. Die drei Verfahren wurden dann in verschiedenen Szenarien mit unterschiedlicher Parametrisierung erprobt. U.a. wurden Duty-Cycle, Netzwerkdurchmesser und Netzwerkdichte variiert.

Das erste Benchmarkverfahren realisiert ein synchrones Wachen und Schlafen aller Knoten, die während der Wachphase mittels CSMA auf das Funkmedium zugreifen. Es wird deshalb im Folgenden *SC-CSMA* (synchronous cycling with carrier sense multiple access) genannt. Direkt nach dem Aufwachen sowie vor dem Schlafengehen gibt es wie bei CUPID eine Toleranzphase, die vor Nachrichtenverlust durch asynchrone Uhren schützt.

Das zweite Vergleichsverfahren erweitert das erste dahingehend, dass jeder Knoten innerhalb der synchronen Wachphase einen eigenen Zeitslot zugewiesen bekommt, in dem nur er senden darf, nicht aber seine Nachbarn. Es basiert also auf TDMA, weshalb es *SC-TDMA* (synchronous cycling with time division multiple access) genannt wird.

Der folgende Abschnitt beschreibt die verwendeten Simulationsparameter. Die beiden sich anschließenden Abschnitte beinhalten die Evaluationsergebnisse bei einer linienförmigen (vgl. Abschnitt 4.4.2) und einer rechteckförmigen (vgl. Abschnitt 4.4.3) Struktur.

4.4.1 Simulationsumgebung

Als Kommunikationsmodell wurde das stochastische Modell, wie in Abschnitt 2.7 beschrieben, verwendet, so dass eine Nachricht mit einer Wahrscheinlichkeit, die vom Abstand zwischen Sender und potentiellem Empfänger abhängt, empfangen wurde.

Wie oben erwähnt, gehen CUPID und die beiden Vergleichsverfahren von der regelmäßigen Durchführung eines Zeitsynchronisationsverfahrens aus. d_{tol} muss dann auf den maximalen zeitlichen Abstand zwischen zwei beliebigen Uhren gesetzt werden. d_{tol} sollte in einem angemessenen Verhältnis zur Länge der Wachphase stehen, beispielsweise weniger als 30 % der Wachphase ($d_{tol} < 0.3 \cdot (3 \cdot d_{slotlength})$). Die Wahl sollte aber auch eine möglichst seltene Synchronisation ermöglichen, damit neben den Nachrichten des

Synchronisationsprotokolls möglichst viele andere (anwendungsbezogene) Nachrichten ausgetauscht werden können und der Energieverbrauch minimiert wird. Aufgrund der Genauigkeit der Oszillatoren der iSense-Knoten von 20 ppm wurde $d_{tol} = 12$ ms gesetzt. Das bedeutet, dass alle 5 Minuten synchronisiert werden muss und der Abstand zweier Uhren im schlimmsten Fall 12 ms beträgt. Der Wert für d_{tol} hat keine Auswirkung auf die Paketauslieferungsrate oder Ende-zu-Ende-Verzögerung, einzig auf den resultierenden Duty-Cycle.

Um der Tatsache, dass die Uhren nach dem Ausbringen und Einschalten der Knoten initial nicht synchronisiert sind, im Simulator Rechnung zu tragen, wurde zu Beginn jeder Simulation für jeden Knoten ein zufälliger Wert r ermittelt, der zu der lokalen Zeit des Knotens addiert wurde. r wurde zwischen 0 und d_{tol} gewählt. Auch die Drift wurde bei jedem Knoten im Simulator auf einen Wert zwischen 0 und 20 ppm eingestellt.

Um statistisch aussagekräftige Ergebnisse zu erzielen, wurden für jedes Parameterset die Ergebnisse von 100 Simulationsläufen mit unterschiedlichen Ausgangswerten für die Zufallsreihen gemittelt.

4.4.2 Linienförmige Struktur

Für die ersten Simulationen wurde eine linienförmige Simulationsfläche erzeugt, in der die Knoten äquidistant verteilt wurden, so dass sie eine Perlenkette bildeten. Die Dichte wurde zwischen 9 und 30 Nachbarn pro Knoten variiert. Diese entsprach beim SC-TDMA-Verfahren auch der verwendeten Slotanzahl.

Die beiden Knoten an den Enden der Kette wurden als Senken S_1 und S_2 deklariert. S_1 diente als Referenzknoten, zu dem die anderen Knoten ihre hop-basierte Distanz bestimmten.

Als erstes wurde der Einfluss von verschiedenen Parametern, wie z. B. dem Netzwerkdurchmesser, auf die Verzögerung einer einzelnen Nachricht von S_1 zu S_2 untersucht. Da Fluten ein häufig verwendetes Kommunikationsparadigma darstellt, flutete S_1 während der Betriebsphase eine mittelgroße Nachricht von 60 Bytes, wobei der Zeitpunkt des Erzeugens bei S_1 sowie die Empfangszeitpunkte bei allen anderen Knoten aufgezeichnet wurden. In der Simulation wurden dabei jedoch nicht die lokalen Uhren, sondern Zeiten einer Weltzeituhr verwendet. Die Ende-zu-Ende-Verzögerung konnte so berechnet werden.

S_1 erzeugte die Nachricht immer zum bestmöglichen Zeitpunkt: zu Beginn seiner Sendephase. Werden Nachrichten zu einem zufälligen Zeitpunkt erzeugt, verlängert sich die mittlere Verzögerung jedoch um die mittlere Wartezeit bis zur nächsten Wachphase. Je länger die Superframes sind, desto größer ist diese Zeit. Da bei CUPID beide Wellenrichtungen immer abwechselnd angewendet wurden, beträgt sie hier $1{,}5 \cdot d_{SFL}$ s, bei den Vergleichsverfahren hingegen nur $0{,}5 \cdot d_{SFL}$ s.

Die Parameter wurden wie folgt festgelegt:

- Netzwerkdurchmesser $ND = 50\,\text{hops}$,
- Dichte $D = 10$ Nachbarn im Mittel und
- Duty-Cycle $DC = 1\,\%$.

Bei Untersuchungen wurde maximal einer dieser Parameter variiert. Zunächst wurde jedoch die Superframelänge verändert und für jedes Verfahren anschließend auf den für diesen Duty-Cycle optimalen Wert gesetzt. Für CUPID waren dies $d_{SFL} = 8\,\text{s}$, für die beiden anderen Verfahren dagegen $d_{SFL} = ND + 20\,\text{s}$.

Einfluss der Superframelänge

Abbildung 4.8 zeigt für alle drei Verfahren die mittlere Dauer in Sekunden, die die Nachricht benötigt, um von S_1 bis zu S_2 zu gelangen. Die Dauer ist (logarithmisch) aufgetragen über die Superframelänge, die ebenfalls in Sekunden angegeben ist. Die vertikalen Balken zeigen den minimal sowie den maximal aufgetretenen Wert aus der jeweiligen Simulationsreihe an.

Es sei an dieser Stelle daran erinnert, dass eine Vergrößerung der Superframelänge eine Verlängerung von Wachphase, Slotlänge und somit auch der Zeit, in der ein Knoten ein Paket senden oder weiterleiten kann, zur Folge hat. Die Superframelänge muss natürlicherweise größer als ein Minimalwert sein, um eine Slotlänge zu erreichen, die der darunter liegenden MAC-Schicht das Senden von mindestens einer Nachricht erlaubt.

Zunächst werden die Ergebnisse von CUPID betrachtet. Für d_{SFL} kleiner als 8 s hängt es vom Zufall ab, wie viele Superframes vergehen, bis die Nachricht bei S_2 angekommen ist. Je länger die Wachphase ist, desto wahrscheinlicher wird es, dass die Nachricht in einem einzigen Superframe durch das gesamte Netz geflutet werden kann. Eine Superframelänge von 8 s ist offensichtlich ausreichend für ein Netzwerk mit einem Durchmesser von 50 Hops. Hier beträgt die Ende-zu-Ende-Verzögerung nur 942 ms.

Ein Vergrößern der Superframelänge führt nur zu später beginnenden Slots (außer dem ersten). Da S_2 aber erst in den letzten drei Slots dieses Superframes wach ist und die Nachricht nicht vor Beginn des drittletzten Slots empfangen kann, bedeutet eine größere Slotlänge auch eine größere Ende-zu-Ende-Verzögerung. Aus diesem Grund steigt die Kurve mit steigender Superframelänge langsam an. Minimum, Maximum und Durchschnittswert unterscheiden sich hier kaum. Sie hängen nur von der gewählten Dauer bis zum Weiterleiten der Nachricht im letzten Hop ab.

Beim SC-CSMA-Verfahren verläuft die Kurve stufenförmig. Hier benötigt die Nachricht für d_{SFL} kleiner als 38 mehrere Wachphasen, bis sie bei S_2 ankommt. Je länger die Wachphase ist, desto öfter kann die Nachricht in einer Wachphase weitergeleitet und

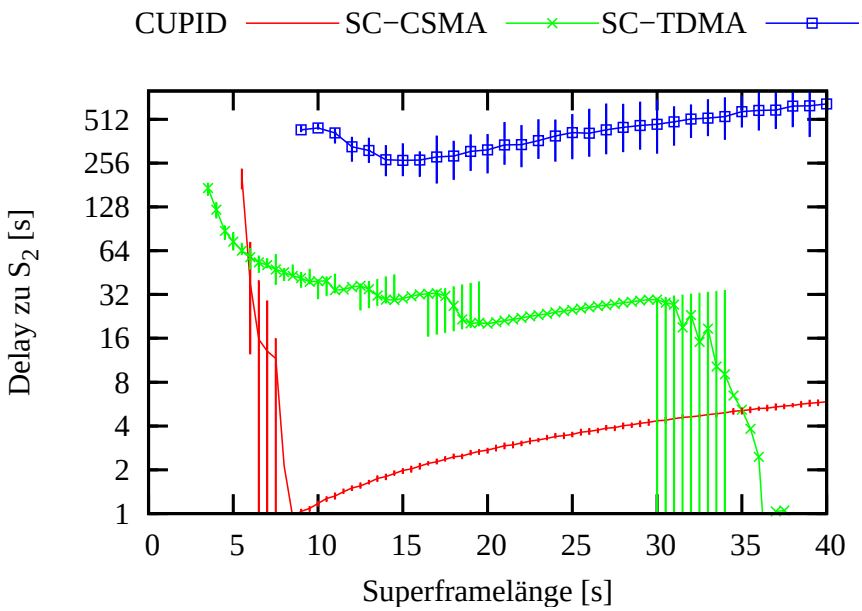


Abbildung 4.8: Mittlere Ende-zu-Ende-Verzögerung im günstigsten Fall ($ND = 50$)

so über eine größere Distanz hinweg verbreitet werden. Erst ab $d_{SFL} = 38\,\text{s}$ ist die Wachphase lang genug, um mit nur einer Wachphase für die 50 Hops auszukommen.

Für kleinere Superframelängen gibt es zum einen die Segmente der Kurve, in denen eine feste Anzahl an Wachphasen benötigt wird und die Verzögerung langsam ansteigt, weil die Zeit bis zur nächsten Wachphase immer größer wird, z. B. zwischen 22 und 33 s. Zum anderen gibt es die Segmente, in denen n oder $n-1$ Wachphasen benötigt werden, wobei mit wachsender Superframelänge die Wahrscheinlichkeit steigt, dass eine Wachphase weniger benötigt wird. Solch ein Segment ist z. B. von $d_{SFL} = 17$ bis 21 s zu sehen. Auch hier tragen längere Superframes zu einer wachsenden Verzögerung bei. Für $d_{SFL} > 40$ bleibt die Verzögerung niedrig, da eine Wachphase hier ausreicht, um die Nachricht zu fluten. Eine noch längere Wachphase hat jedoch kaum Einfluss auf die Verzögerung, da nur die (durch Zufall ermittelte) Zeitspanne, nach deren Ablauf die Nachricht tatsächlich versendet werden kann, langsam wächst.

Das SC-TDMA-Verfahren schneidet im Vergleich zu den beiden anderen Verfahren deutlich schlechter ab. Die Ende-zu-Ende-Verzögerung beträgt hier im Minimum 250 s. Dies liegt an den bezüglich der Ausbreitungsrichtung der Nachricht ungeordneten Sendeslots. Im schlimmsten Fall sind die Slots so ungünstig angeordnet, dass die Nachricht während einer Wachphase aller Knoten um nur einen Hop weitergeleitet wird, auch wenn die Wachphase theoretisch 50 Weiterleitungen zulassen würde.

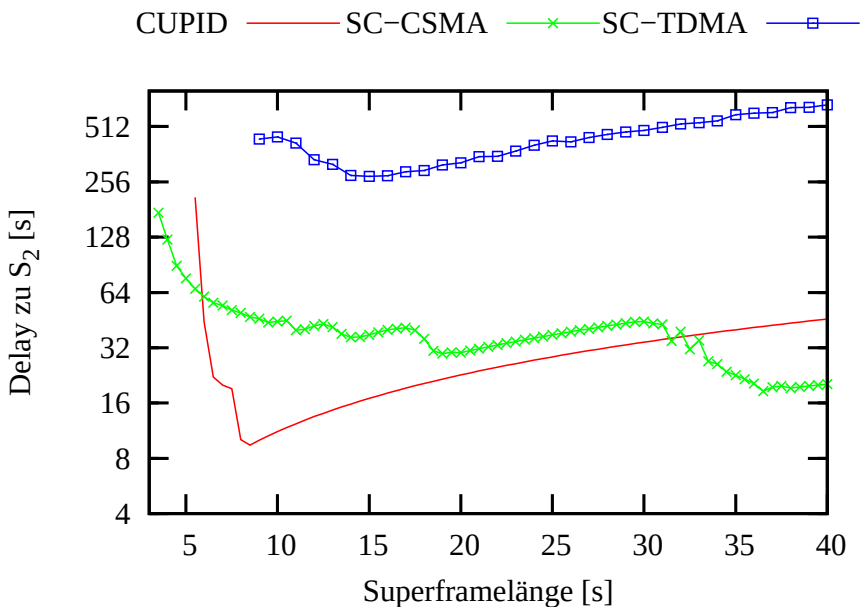


Abbildung 4.9: Ende-zu-Ende-Verzögerung im durchschnittlichen Fall ($ND = 50$)

Die in Abbildung 4.9 dargestellten Kurven ergeben sich, wenn man zu den vorherigen Ergebnissen die mittlere Wartezeit addiert. Sie repräsentieren somit die mittlere Verzögerung beim Senden zu einem beliebigen Zeitpunkt. Eine Nachricht, die wie eine Alarmnachricht zu einem zufälligen Zeitpunkt erzeugt wird, benötigt bei SC-CSMA im Mittel mindestens 20, bei CUPID nur 10 s. Gerade bei Alarmnachrichten ist jedoch eine schnelle Auslieferung wichtig.

Die Kurve von CUPID steigt insgesamt ein wenig stärker an als die der anderen Verfahren, weil es hier im schlimmsten Fall (wenn der Sendeslot von S_1 gerade vorüber ist) fast drei (statt zwei) ganze Superframes dauert, bis S_2 die Alarmnachricht erhält, da nur jeder zweite Superframe die Kommunikation in Richtung von S_2 unterstützt. Ist vor Ausbringung der Knoten bekannt, wie groß das Nachrichtenaufkommen in jeder Richtung sein wird, kann CUPID daran angepasst werden. So kann die Welle beispielsweise in 90 % der Fälle in die eine und nur in 10 % der Fälle in die andere Richtung laufen. Im Extremfall würde die Welle immer in die gleiche Richtung laufen. Dann würde die Kurve in gleichem Maße steigen wie bei den Vergleichsverfahren.

Einfluss des Netzwerkdurchmessers

Als nächstes wurde der Einfluss des Netzwerkdurchmessers untersucht, wobei für jedes Verfahren die geeignetste Superframelänge gewählt wurde ($d_{SFL} = 8$ s bei CUPID, für

die beiden anderen Verfahren $d_{SFL} = ND + 20\,\text{s}$). An dieser Stelle sei aber noch einmal darauf hingewiesen, dass kürzere Superframes bezüglich Durchsatz von Vorteil sind, da sie ein früheres Senden weiterer Nachrichten ermöglichen.

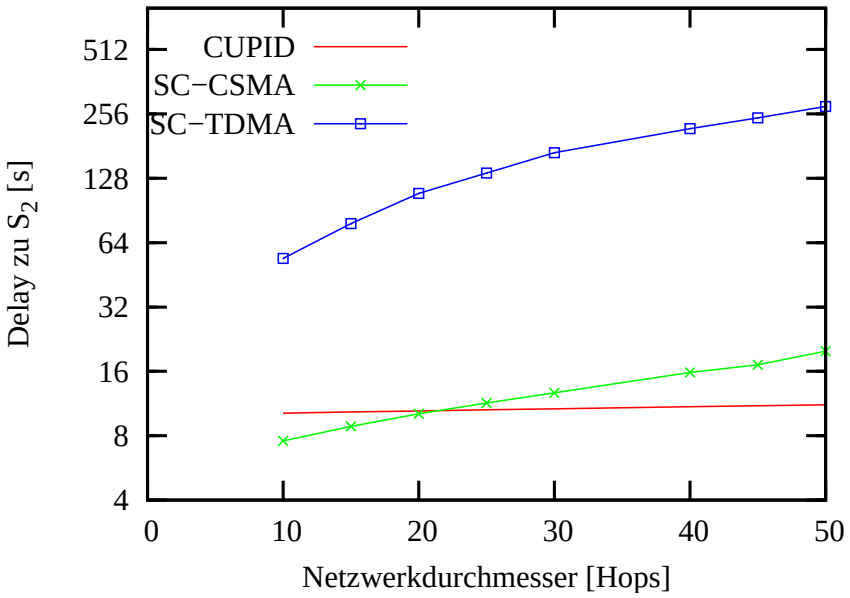


Abbildung 4.10: Mittlere Verzögerung über Netzwerkdurchmesser

Abbildung 4.10 zeigt ebenfalls logarithmisch aufgetragen die Ende-zu-Ende-Verzögerung der Nachricht zuzüglich der mittleren Wartezeit für Netzwerkdurchmesser zwischen 10 Hops und 50 Hops. Bei kleineren Netzwerken mit einem Durchmesser von weniger als 20 Hops liefern CUPID und SC-CSMA ähnliche Ergebnisse. Die Verzögerung einer Alarmnachricht beträgt hier etwa 10 s. Ab 25 Hops lohnt es sich dann, CUPID einzusetzen und je größer der Netzwerkdurchmesser ist, desto größer ist auch der erzielte Benefit. Die Werte für $ND = 50$ Hops entsprechen logischerweise den besten Werten aus Abbildung 4.9 des vorhergehenden Abschnitts.

SC-TDMA kann auch bei kleineren Netzwerkdurchmessern nicht die gleiche Effizienz erzielen. Bei 10 Hops benötigt die Nachricht hier etwa 50 s.

Einfluss der Dichte

Die Ergebnisse für verschiedene Dichten (6, 10 und 20) sind in Abbildung 4.11 dargestellt. Der Einfluss der Dichte auf die Ende-zu-Ende-Verzögerung ist offensichtlich recht gering. Die Kurven liegen sowohl bei CUPID als auch bei SC-CSMA sehr nah beieinander. Dies ist wenig überraschend, da eine höhere Dichte zwar den Wettbewerb um das Medium erhöht,

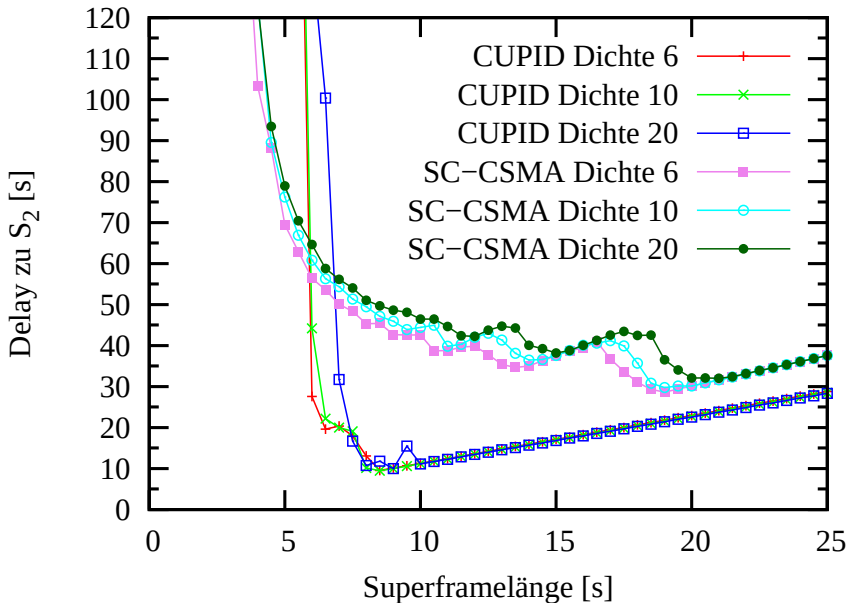


Abbildung 4.11: Mittlere Verzögerung bei verschiedenen Dichten

aber solange die Nachricht in jedem Hop von mindestens einem Knoten weitergeleitet wird, hat dies keine Auswirkung auf die Ende-zu-Ende-Verzögerung.

Einfluss des Duty-Cycles

Auch der Einfluss des Duty-Cycles wurde untersucht. Abbildung 4.12 zeigt die mittlere Verzögerung der Nachricht von S_1 für verschiedene Duty-Cycles bis 10 %. Es ist zu erkennen, dass CUPID für geringe Duty-Cycles (kleiner als 7 %) zu besseren Ergebnissen führt als SC-CSMA. Ab etwa 3 % ist es dann deutlich besser und lohnt den zusätzlichen Aufwand der Initialisierungsphase. Bei einem Duty-Cycle von 0,1 % beträgt die Verzögerung bei CUPID 80 s, bei SC-CSMA etwa 200. Je weniger Energie zur Verfügung steht und je größer die gewünschte Lebensdauer des Netzes ist, desto sinnvoller ist es, CUPID einzusetzen.

Die bisherigen Ergebnisse zeigen, dass CUPID in solchen schlauch- oder linienförmigen Szenarien, in denen ein geringer Duty-Cycle gefordert ist, eine extrem geringe Ende-zu-Ende-Verzögerung erzielt. Der folgende Abschnitt befasst sich mit dem allgemeineren Fall einer rechteckförmigen Struktur.

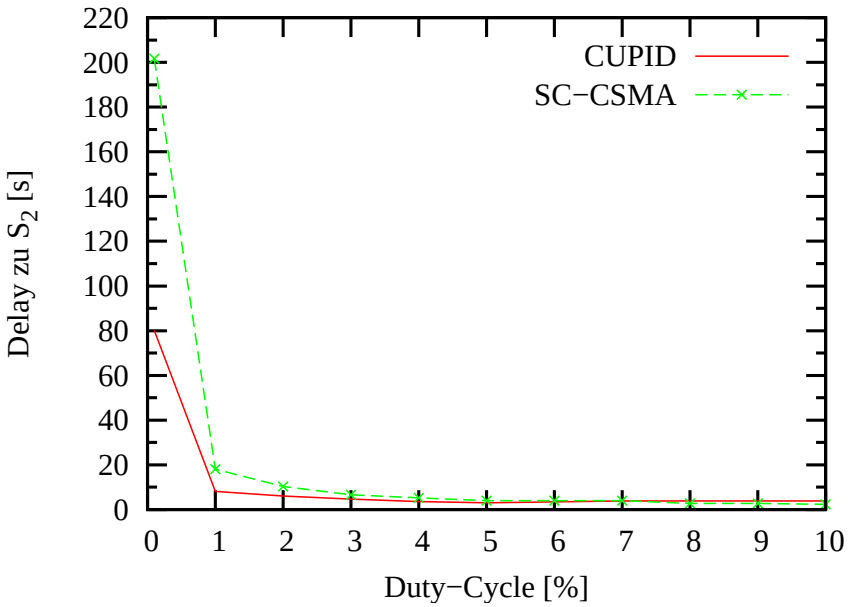


Abbildung 4.12: Mittlere Verzögerung über Duty-Cycle

4.4.3 Rechteckförmige Struktur

Der Fall, dass die Knoten des Sensornetzes in einer Art Schlauch angeordnet sind, mag ein außergewöhnlicher sein, der bei der Grenz- oder Deichüberwachung vorkommt. Deshalb wurde CUPID auch in einem allgemeineren Szenario evaluiert. Abbildung 4.13 zeigt dieses Szenario. Es ist im Großen und Ganzen rechteckförmig, bildet jedoch links einen Kreis, hat oben in der Mitte eine Sackgasse und ist nicht konvex. Auf der Simulationsfläche wurden 900 Knoten äquidistant verteilt, wobei der Knotenabstand 15 m betrug, was zu etwa 10 Hops in vertikaler und 30 Hops in horizontaler Richtung führt. Weiterhin hat jeder Knoten in jeder Richtung meist zwei Nachbarn, das heißt 24 im Maximum, wenn in allen Richtungen Knoten vorhanden sind.

Ein typisches Kommunikationsmuster in einem Sensornetz besteht im Sammeln von (gegebenenfalls vorverarbeiteten) Daten an der Senke, von der sie an ein Backendsystem weitergegeben werden. Um Daten zu einer Senke zu transportieren, eignet sich eine Baumstruktur (vgl. Abschnitt 2.10.2). Sie reduziert das Nachrichtenaufkommen massiv im Vergleich mit Protokollen beim Fluten. Wie so ein Baum aussehen kann, zeigt die Abbildung ebenfalls. Die Senke, die die Konstruktion des Baumes angestoßen hat, befindet sich in der linken unteren Ecke. Die Breite einer Linie ist dabei ein Maß für die Anzahl der in einem bestimmten Zeitraum empfangenen Nachrichten über diese Kommunikationsverbindung. Je breiter sie ist, desto mehr Nachrichten wurden hier gesendet und empfangen.

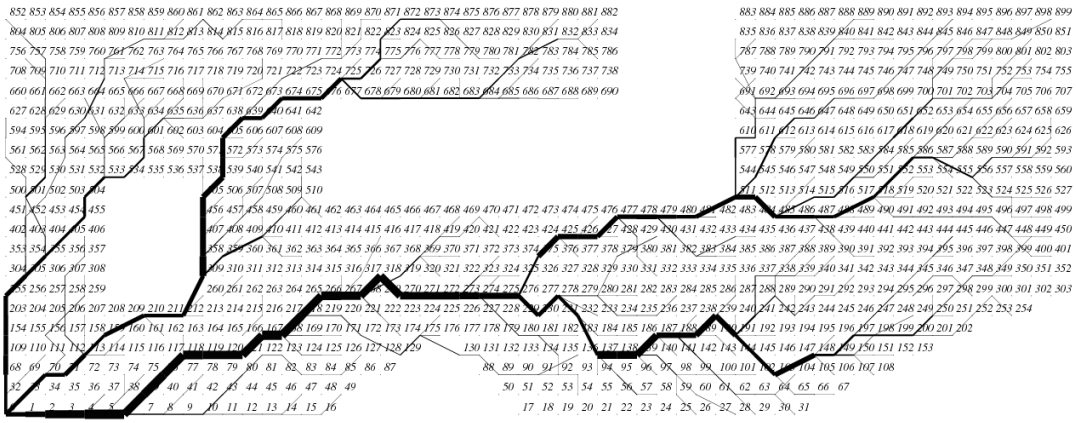


Abbildung 4.13: Rechteckszenario

Unterschiedliche Teilmengen an Knoten sendeten nach dem Baumaufbau im gleichen Superframe eine Nachricht entlang des Baumes zur Senke. 1 % bis 20 % zufällig ausgewählter Knoten sendeten eine Nachricht, so dass insgesamt 9 bis 180 Nachrichten gesendet wurden. Jeder einzelne Nachrichtenempfang wurde mit einem Acknowledgement bestätigt. Blieb die Bestätigung aus, wurde eine Nachricht höchstens dreimal gesendet. Allerdings waren die Bestätigungsnachrichten nur an direkte Nachbarn gerichtet und das Fehlen von Multihop-Acknowledgements impliziert logischerweise eine unzuverlässige Multihopkommunikation, so dass dem Quellknoten unbekannt bleibt, ob seine Nachricht bei der Senke angekommen ist.

Die folgenden beiden Abschnitte zeigen die über 100 Simulationsläufe gemittelte Ankunftsrate bei der Senke S_1 und wie lange die Nachrichten im Mittel bis zu S_1 brauchten.

Ankunftsrate

1 % bis 20 % der Knoten sendeten eine Nachricht im gleichen Superframe an S_1. Wieviel Prozent der Nachrichten bei S_1 ankamen, ist in Abbildung 4.14 dargestellt. SC-CSMA liefert für alle Prozentzahlen die schlechteren Ergebnisse. Werden 9 Nachrichten gesendet, kommen nahezu unabhängig von der Superframelänge zwischen 90 % und 95 % bei der Senke an; senden 90 bzw. 180 Knoten eine Nachricht, kommen nur 30 % bis 40 % bzw. 20 % bis 30 % an. Das Problem bei diesem Verfahren ist, dass Knoten, die sich nicht direkt hören können, aber trotzdem denselben Vaterknoten haben, häufig zu ähnlichen Zeitpunkten senden, was systembedingt zu Kollisionen führt.

CUPID transportiert dagegen 90 % bis 98 % der Nachrichten zur Senke, wenn 1 % der Knoten eine Nachricht senden, und zwischen 70 % und 80 %, wenn 90 oder 180 Knoten

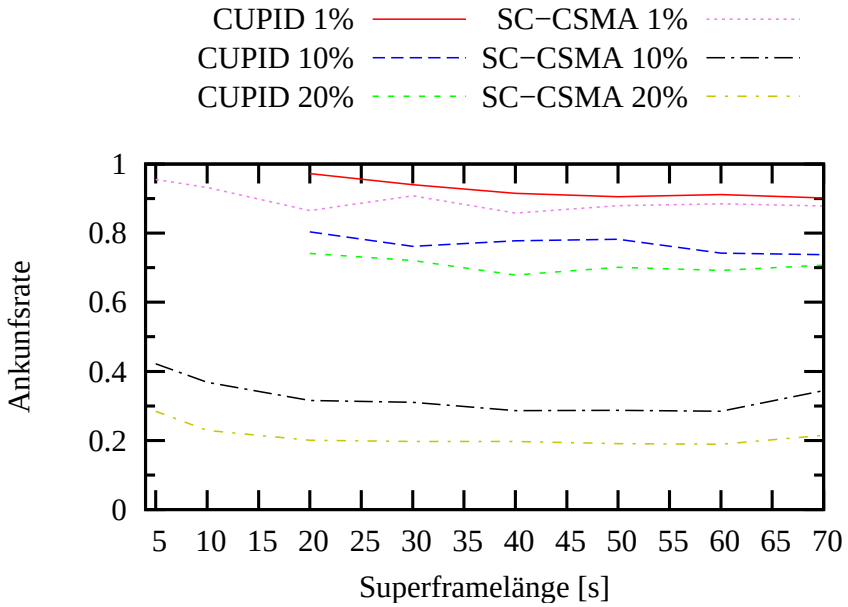


Abbildung 4.14: Ankunftsrate

eine Nachricht senden. Durch das geordnete Senden steigert CUPID die Ankunftsrate maßgeblich, besonders wenn das Nachrichtenaufkommen hoch ist.

Verzögerung

Der vorhergehende Abschnitt betrachtete die Paketankunftsraten bei der Senke. Abbildung 4.15 zeigt die dazugehörigen Verzögerungen der bei S_1 angekommenen Nachrichten, gemittelt über die Nachrichten einer Simulation sowie über alle Simulationsläufe.

Bei 9 gesendeten Nachrichten war die Ankunftsrate bei beiden Verfahren in etwa vergleichbar. CUPID liefert hier jedoch die besseren Ergebnisse hinsichtlich des Delays, wenn man bedenkt, dass beim Senden zu einem beliebigen Zeitpunkt im Mittel noch die halbe Superframelänge addiert werden muss. Daneben erlauben kürzere Superframes ein früheres Senden einer weiteren Nachricht. Bei SC-CSMA reicht offensichtlich eine Slotlänge von 700 ms aus, um die Nachrichten innerhalb einer Wachphase zur Senke zu transportieren. Sobald jedoch die Slotlänge kürzer ist, steigt das Delay von 24 auf 78 s. Bei CUPID benötigten die Nachrichten dagegen nur zwischen 9 und 28 s.

Die kleineren Werte des SC-CSMA-Verfahrens bei 90 und 180 gesendeten Nachrichten sind darauf zurückzuführen, dass nur die bei S_1 angekommenen Nachrichten in die Wertung einfließen konnten. Nach einer kurzen Phase, in der es zu sehr vielen Kollisionen kommt, werden keine Nachrichten mehr weitergeleitet und an S_1 ausgeliefert. Bei CUPID

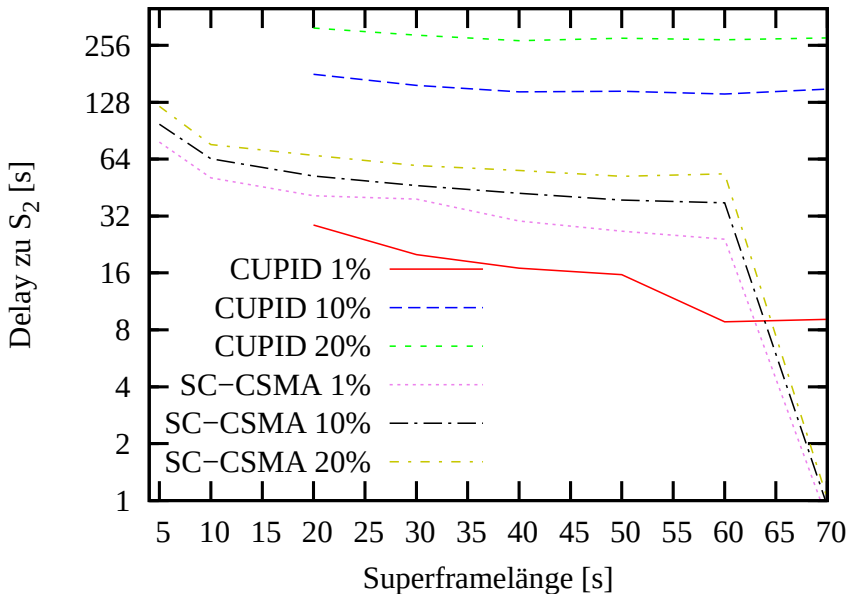


Abbildung 4.15: Durchschnittliche Verzögerung

dauert es zwischen 150 und 180 bzw. 280 und 315 s, bis keine Nachricht mehr an S_1 ausgeliefert werden. Mit wachsendem Nachrichtenaufkommen steigt logischerweise die Dauer bis die Nachrichten bei S_1 ankommen. Dadurch dass bei CUPID aber weniger Knoten gleichzeitig senden, kommt es zu weniger Kollisionen als bei SC-CSMA.

4.5 Experimentelle Evaluation

Um die Korrektheit und Einsetzbarkeit von CUPID zu demonstrieren, wurden Experimente mit etwa 160 iSense-Sensorknoten durchgeführt. Da iSense-Software sowohl für Shawn als auch für Geräte kompiliert werden kann, konnte hierfür der identische Code verwendet werden.

Als Benchmarkverfahren diente hier nur SC-CSMA aufgrund des deutlich schlechteren Leistungsverhaltens von SC-TDMA in den Simulationen. Jedes Einzelexperiment lief etwa 5 Minuten. Zu Beginn jedes Experiments wurden alle Knoten mit einer mittleren Genauigkeit von 1,5 ms synchronisiert. Zur Laufzeit speicherten die Knoten alle Evaluationsdaten lokal im RAM, um das Experiment nicht zu beeinflussen. Nach dem Ende des Experiments wurden die Daten mit Hilfe der Funkschnittstelle eingesammelt.

Die Sendeleistung der Knoten wurde auf den niedrigsten Wert (Abschwächung um 30 db) festgelegt, damit trotz begrenztem Platz ein großer Netzwerkdurchmesser entstand.

Andernfalls (bei der größtmöglichen Sendeleistung) hätte das Netz eine Längenausdehnung von mehr als 1, 5 km gehabt. Mit der niedrigen Sendeleistung und einem Abstand von 40 cm zwischen zwei Knoten war das Netz etwa 60 m lang. Die Knoten wurden an zwei unterschiedlichen Orten ausgebracht: zum einen in der Nähe eines Gebäudes, wo etwa 12 Hops gemessen wurden; zum anderen auf einer freien Ebene im Carlebachpark in Lübeck (siehe Abbildung 4.16). Dort betrug der Netzwerkdurchmesser 23 Hops. Das Gebäude bestand zu einem großen Teil aus Metall, so dass die Kommunikationsradien vermutlich durch viele Reflexionen variierten.

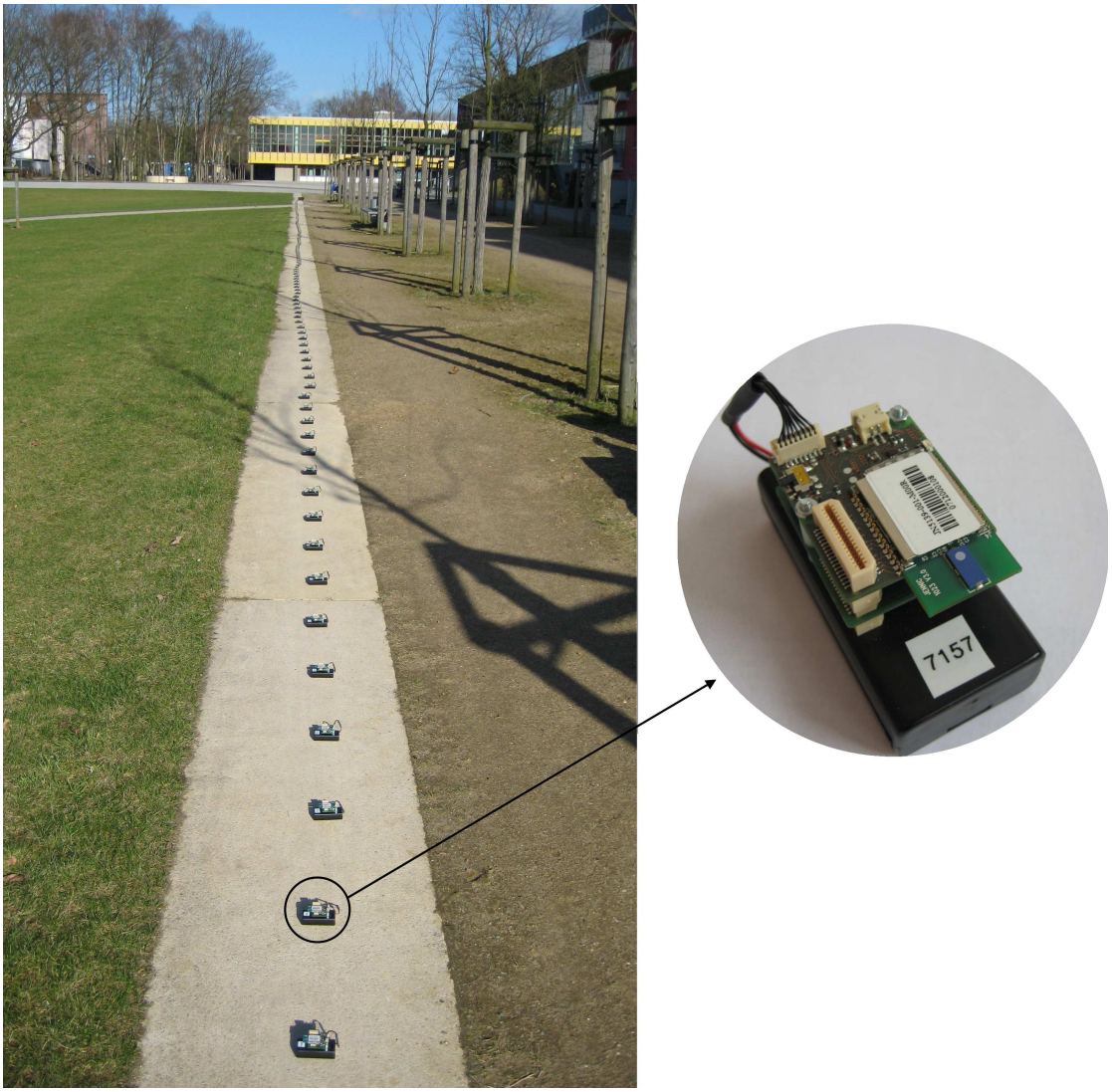


Abbildung 4.16: Experimentaufbau

Der Experimentablauf war wie folgt: Zunächst wurden Zeitsynchronisation und Initialisierungsphase durchgeführt. Anschließend wechselten alle Knoten in die Betriebsphase, in der eine Senke S_1 eine Nachricht genau zu Beginn einer Sendephase flutete, so dass es sich hier ebenfalls um den günstigsten Fall handelt. Während sich S_1 die Sendezeit merkte, speicherten alle anderen Knoten die Ankunftszeit, so dass die maximale Verzögerung berechnet werden konnte.

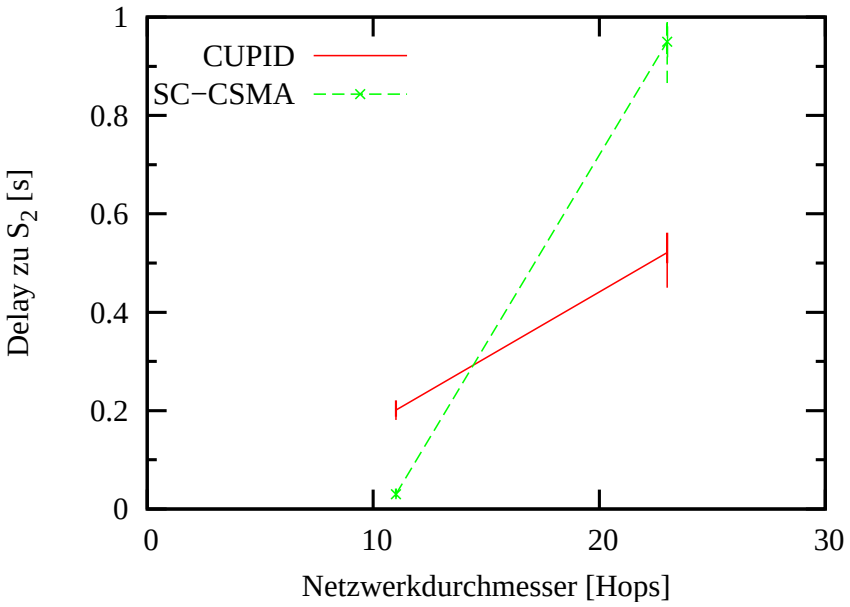


Abbildung 4.17: Verzögerung für Durchmesser 12 und 23

Das Experiment wurde zwölfmal durchgeführt, so dass Abbildung 4.17 die maximale Verzögerung zeigt, gemittelt über zwölf Nachrichten. Es ist zu erkennen, dass beim ersten Deployment in der Nähe des Gebäudes (Durchmesser 12) das SC-CSMA-Verfahren aufgrund der gleichzeitigen Wachphasen aller Knoten bessere Ergebnisse liefert. Nur wenn alle Knoten simultan wach sind, führen höhere Kommunikationsreichweiten, z. B. durch Reflexionen, zu kürzeren Ende-zu-Ende-Verzögerungen. Im größeren Szenario schneidet CUPID besser ab als SC-CSMA: In nur 500 ms wurde die Nachricht über die 23 Hops transportiert, SC-CSMA brauchte dagegen 900 ms.

Addiert man nun den mittleren Offset von 10 s zu den Ergebnissen, der vorhanden ist, wenn die Nachricht nicht genau zu Beginn der Wachphase, sondern zu einem beliebigen Zeitpunkt erzeugt wird, so stimmen die Ergebnisse mit den Simulationsergebnissen aus Abbildung 4.10 überein.

4.6 Eigenschaften von CUPID

Das gestaffelte Aufwachen, das CUPID vorsieht, erfüllt alle in Abschnitt 4.1 beschriebenen Anforderungen an Duty-Cycle-Verfahren:

Geringer Installationsaufwand: Da CUPID nicht auf Positionsinformationen basiert, können die Geräte an beliebige, von ihrer ID unabhängige Orte ausgebracht werden. Einzig muss ein Referenzknoten existieren, der die Installations- und die Betriebsphase startet, was aber kein Problem darstellt, da in einem Sensornetz üblicherweise mindestens ein Gatewayknoten vorhanden ist, der diese Aufgabe übernehmen kann. Weiterhin muss entweder ein zweiter Knoten Wissen darüber haben, dass er der vom Referenzknoten am weitesten entfernte Knoten ist, oder der Netzwerkdurchmesser muss ungefähr bekannt sein oder geschätzt werden. Während der Initialisierungsphase müssen nur zwei Nachrichten durch das gesamte Netz gesendet werden (ein Baumaufbau und eine geflutete Nachricht). Falls der Netzwerkdurchmesser ermittelt werden muss, muss von allen Knoten noch eine Nachricht (die die maximale Gruppennummer akquiriert) gesendet werden. Diese Phase kann in wenigen Sekunden durchgeführt werden und das Netz kann anschließend in den normalen Betriebszustand übergehen.

Permanent geringer Energieverbrauch: Während der Betriebsphase wird fortwährend der gewünschte Duty-Cycle umgesetzt. Die Knoten bleiben weder phasenweise noch auf Anfrage länger aktiv, so dass sie gegen den Energieentzugsangriff resistent sind.

Geringe Ende-zu-Senke-Verzögerung: Die gestaffelten Wachphasen werden in CUPID so optimiert, dass sie eine oder zwei Vorzugsrichtungen für Datentransfer effizient unterstützen. Diese Richtungen können vom Netz zur Senke, von der Senke zum Netz oder auch Ende-zu-Ende zwischen zwei beliebigen Knoten sein. CUPID ist in diesem Punkt flexibel konfigurierbar.

Unterstützung von Broadcast-Kommunikation: CUPID garantiert, dass alle Nachbarn eines Senders empfangsbereit sind, so dass eine Broadcast-Nachricht alle Nachbarn erreicht.

Geringe Kollisionswahrscheinlichkeit: Da Knoten nur in einem bestimmten Zeitfenster ihrer Gesamtwachzeit Nachrichten senden dürfen und in der übrigen Zeit nur hören, ob ihre Nachbarn etwas senden, wird die Kollisionswahrscheinlichkeit im Vergleich zum Senden zu einem beliebigen Zeitpunkt deutlich reduziert. Ist das Netz linienförmig und die Linie nicht breiter als die Kommunikationsreichweite der Knoten, können sich sogar alle Knoten hören, die gleichzeitig senden dürfen, so dass es zu fast keiner Kollision kommt.

4.7 Zusammenfassung

Dieses Kapitel stellt mit CUPID ein Duty-Cycle-Verfahren für Sensornetze vor. Das Verfahren ermöglicht kurze Nachrichtenverzögerungen in der Richtung des bevorzugten Nachrichtenflusses, indem die Knoten in dieser Richtung nacheinander in den aktiven und wieder in den inaktiven Modus versetzt werden, so dass eine Art Welle entsteht. Es wurde durch analytische, simulative und experimentelle Untersuchungen gezeigt, wie gestaffeltes Aufwachen effizient und flexibel eingesetzt und insbesondere genau auf die Bedürfnisse der Applikation zugeschnitten werden kann.

Besonders bei sehr geringen Duty-Cycles (2 % und weniger) und großen Netzwerkdurchmessern hebt sich CUPID bezüglich Nachrichtenverzögerung in einer oder zwei Vorzugsrichtungen von anderen Verfahren ab. Auch ist es resistent gegen Energieverbrauchsangriffe und garantiert, dass alle potentiellen Empfänger eines Senders empfangsbereit sind.

Aus diesen Gründen eignet sich das Verfahren auch für sicherheitsrelevante Anwendungen. Bei linienförmigen Netzwerkstrukturen führt das Verfahren sogar zur vollständigen Vermeidung des Hidden-Terminal-Problems und die dadurch hervorgerufenen Kollisionen. Aber auch in rechteckförmigen Strukturen reduziert CUPID die Verzögerung und die Kollisionswahrscheinlichkeit. In extrem komplexen Strukturen, wie sie durch Straßennetze entstehen, sollte CUPID nicht unbedingt eingesetzt werden, denn Zyklen führen systembedingt zu Kollisionen.

Während zum Netz hinzukommende oder ausgefallene Knoten sowie unzuverlässige Kommunikationsverbindungen für CUPID kein Problem darstellen, kann das Verfahren dagegen nicht in einem mobilen Netz eingesetzt werden. Bei extrem schwankenden Kommunikationsreichweiten kann es nicht den Vorteil nutzen, dass temporär auch weiter entfernte Knoten erreicht werden können und so eine größere Distanz in gleicher Zeit überwunden werden kann.

Struktur kann jedoch nicht nur durch die räumliche Anordnung der Knoten entstehen, sondern auch durch die Heterogenität bezüglich der Hardware, der Software oder des Kontextes von Knoten. Das folgende Kapitel befasst sich damit, wie beim Reprogrammieren von Knoten sinnvoll mit Heterogenität umgegangen werden kann.

Kapitel 5

Selektives Reprogrammieren

Ein laufendes Programm durch ein anderes zu ersetzen, ist immer eine schwierige und kritische Aufgabe. In Sensornetzen ist es aufgrund der Verteiltheit des Systems und der beschränkten Ressourcen besonders schwierig. Trotzdem muss es auch in Sensornetzen möglich sein, die Software auf den Geräten auszutauschen, da – beispielsweise in Langzeitmessanwendungen – neue Anforderungen oder Aufgaben entstehen können, an die die Software angepasst werden muss. Auch eine Weiterentwicklung eines Verfahrens, wie z. B. eine verfeinerte Verarbeitung der Messdaten, kann eine Aktualisierung der Software erfordern.

Aufgrund der fehlenden Verkabelung und der Unzugänglichkeit der Geräte sowie aus Kostengründen kommt ein manuelles Reprogrammieren über ein Kabel nicht in Frage. In Netzen, bei denen nicht alle Knoten mit einer Senke kommunizieren können, ist auch ein Singlehop-Mechanismus nicht ausreichend. Hier ist ein Multihop-Verfahren erforderlich, um den Aufwand und die Kosten für den Softwareaustausch niedrig zu halten.

Zu beachten ist, dass üblicherweise nicht alle Geräte neu programmiert werden sollen und unter Umständen auch nicht mit demselben Programm ausgestattet werden können. Zum einen können auf den Geräten unterschiedliche Anwendungen laufen, wie z. B. eine Mess- oder eine Weiterleitungsanwendung. Zum anderen sind in der Regel nicht alle Geräte mit derselben Hardware ausgestattet (z. B. CPU, GPS oder Sensorik), so dass sie unterschiedliche Codes benötigen. Zum Teil werden sogar Geräte verschiedener Hersteller in einem Netz kombiniert. All dies führt zu einem heterogenen Netz, in dem nur ein Teil der Knoten die neue Anwendung erhalten soll.

Die Struktur der zu programmierenden Knoten ist a priori nicht bekannt. Die Knoten können sich sowohl nah beieinander – im selben Bereich des Netzes – befinden, als auch sehr weit voneinander entfernt oder über das Netz verteilt sein. Im Mittelpunkt dieses Kapitels steht daher das Reprogrammieren einer beliebigen Teilmenge von Knoten in einem Sensornetz. Es werden theoretische und praktische Aspekte diskutiert, die beim Design eines zuverlässigen, effizienten und skalierenden Programmierverfahrens bedacht werden müssen. Das Kapitel stellt ein Verfahren vor, welches darauf beruht, dass nach dem Verteilen des Programms nah beieinander liegende Knoten fehlende Teile direkt untereinander austauschen können, anstatt diese von einer weit entfernten Senke

anzufordern. Im Vergleich zu anderen Arbeiten reduziert das vorgestellte Verfahren den Aufwand an Zeit und die Anzahl der versendeten Nachrichten erheblich.

Die Geräte, die nicht programmiert werden sollen, müssen gegebenenfalls eine Infrastruktur bereitstellen, die das Programmieren der anderen Geräte ermöglicht. Werden sie für den Programmiervorgang nicht benötigt, können sie währenddessen in den Schlafzustand wechseln, um Energie zu sparen.

Das Kapitel gliedert sich in sechs Abschnitte. Im ersten Abschnitt werden Anforderungen an Programmierverfahren vorgestellt. Anschließend wird auf verwandte Arbeiten aus der Literatur mit ihren Vor- und Nachteilen eingegangen. Abschnitt 5.3 beschreibt ein robustes und effizientes Verfahren zum Reprogrammieren einer Teilmenge eines Sensornetzes. Abschnitt 5.4 umfasst die Evaluation und diskutiert Ergebnisse aus Simulationen und Experimenten in einer Testumgebung. Schließlich werden in Abschnitt 5.5 die Eigenschaften des vorgestellten Verfahrens den Anforderungen gegenübergestellt und in Abschnitt 5.6 die Ergebnisse zusammengefasst.

Die in diesem Kapitel präsentierten Algorithmen, Untersuchungen und Ergebnisse sind in gemeinsamer Arbeit mit Carsten Buschmann und Dennis Pfisterer entstanden [58].

5.1 Anforderungen an Programmierverfahren

Folgende Anforderungen müssen an ein Programmierverfahren gestellt werden:

Zuverlässigkeit: Die wichtigste Eigenschaft eines Programmierverfahrens ist die Zuverlässigkeit. Die Knoten dürfen eine neue Anwendung nur dann starten, wenn alle Programmteile korrekt vorhanden sind. Da es typischerweise nur eine Gesamtanwendung auf einem Knoten gibt, können nicht wie bei PCs oder Großrechnern einzelne Programme gestartet werden.

Feedback über den Erfolg und Misserfolg bei jedem einzelnen Knoten: Das Programmierverfahren muss den Anwender über den Erfolg und Misserfolg des Reprogrammierens aller Knoten informieren; denn nur dann ist es möglich, ausgefallene Geräte ausfindig zu machen und manuell zu prüfen oder auszutauschen.

Robustheit gegen unzuverlässige Kommunikation: Der Erfolg des Programmierens darf weder durch unidirektionale noch durch variierende Kommunikationsverbindungen gefährdet sein.

Umgang mit ausfallenden Knoten: Steht einem Knoten nicht mehr ausreichend Energie zur Verfügung, so dass er ausfällt, darf der Programmiervorgang bei anderen Knoten trotzdem nicht beeinträchtigt werden.

Geringer Ressourcenbedarf: Sehr wichtig ist, dass der Code des Programmierverfahrens möglichst klein ist. Je kleiner der Code des Verfahrens ist, desto mehr Speicher steht für die eigentliche Sensornetzanwendung zur Verfügung. Weiterhin muss aber auch zur Laufzeit möglichst wenig Speicherplatz verbraucht werden. Alle Hardwarebeschränkungen von Sensorknoten müssen berücksichtigt werden.

Sicherheit: Die Knoten müssen gegen unerlaubtes Reprogrammieren geschützt sein.

Geringe Dauer: Das Reprogrammieren muss in möglichst kurzer Zeit durchgeführt werden, damit die Funktion des Netzes nur kurz beeinträchtigt wird. Zudem folgt die eigentliche Anwendung üblicherweise einem Duty-Cycle kleiner als 100 %, so dass die Knoten während des Programmierprozesses, bei dem der Duty-Cycle 100 % beträgt, mehr Energie verbrauchen. Diese Phase muss deshalb so kurz wie möglich gehalten werden.

Geringer Energieverbrauch: Betrachtet man das gesamte Netz, kann Energie gespart werden, indem sich so viele Knoten wie möglich während des Programmierprozesses im Schlafmodus befinden. Gegeben, dass die Knoten vor und nach dem Programmierprozess einem Duty-Cycle folgen, führt eine geringe Dauer des Programmierprozesses ebenfalls zu einem geringen Energieverbrauch. Bei Hardwareplattformen, bei denen das Senden von Nachrichten mehr Energie als die reine Empfangsbereitschaft verbraucht, hilft auch eine möglichst geringe Anzahl an Nachrichten, um den Energieverbrauch klein zu halten. Eine geringe Anzahl an Nachrichten schadet in keinem Fall, da es umgekehrt derzeit keine Plattform gibt, bei der das Senden weniger Energie kostet als die Empfangsbereitschaft. Dennoch ist bei den meisten Plattformen der Unterschied im Energieverbrauch zwischen Senden und Empfangsbereitschaft nicht so groß wie der zwischen Empfangsbereitschaft und einem ausgeschalteten Funkmodul. Das Schlafen von Knoten sowie die geringe Dauer des Programmierprozesses werden deshalb als wichtiger angesehen.

Skalierbarkeit: Das Programmierverfahren muss sowohl in großen Netzen als auch in Netzen hoher Dichte funktionieren. Um auch bei neueren Hardwareplattformen einsetzbar zu sein, sollte es außerdem mit großen Programmen (größer als 100 KByte) umgehen können.

Geringe Kosten: Der Kostenaufwand des Anwenders muss möglichst gering gehalten werden, indem das Reprogrammieren weitestgehend automatisiert abläuft.

Feedback über den Fortschritt bei allen Knoten oder insgesamt (optional): Als zusätzliche Funktion eines Programmierverfahrens könnte die Option implementiert sein, dass der Anwender über den Fortschritt des Programmierprozesses für jeden einzelnen Knoten oder auch akkumuliert über das gesamte Netz informiert wird.

Die Codegröße, der Zeit- und Energieverbrauch sowie die Kosten für den Anwender dienen als Gütekriterien, während alle anderen Anforderungen obligatorisch sind.

5.2 Verwandte Arbeiten

Ein Überblick über Reprogrammierverfahren findet sich in [117] (2006). Auf die für diese Arbeit relevanten Arbeiten wird an dieser Stelle eingegangen. Die meisten Autoren gehen davon aus, dass das gesamte Netz reprogrammiert werden soll, so dass ihre Verfahren nicht anwendbar sind, wenn nur ein Teil der Knoten betroffen ist. Dieser Abschnitt geht auf die verschiedenen Gruppen von Verfahren ein und stellt selektive Verfahren und ihre Vorgänger detaillierter vor.

Trickle [69] war eines der ersten Reprogrammierverfahren. Hier wurde ein Mechanismus, bestehend aus drei Phasen (Advertise, Request, Data), angewendet. In der ersten Phase kündigt ein Knoten per Broadcast an, dass er ein neues Programm mit der ID x hat. Knoten, die feststellen, dass ihr laufendes Programm nicht diese ID hat und somit veraltet ist, broadcasten eine Request-Nachricht, um das neue Programm anzufordern. Direkte Nachbarn, die das aktuelle Programm bereits haben, antworten, sofern sie noch nicht k Antworten von anderen Knoten empfangen haben. So verbreitet sich das Update im Vergleich zu einer gefluteten Nachricht nur sehr langsam von Knoten zu Knoten. Das Vorgehen erfordert jedoch, dass die zu programmierenden Knoten einen zusammenhängenden Graphen bilden. Die Autoren gingen sogar davon aus, dass alle Knoten das Programm bekommen sollen und dass das gesamte Programm in ein Funkpaket passt. Impala, wie die Middleware des ZebraNet-Projektes [75] genannt wird, verwendet einen ähnlichen Mechanismus. Auch hier wird zunächst die ID der Anwendung ausgetauscht. Das Programm wird anschließend allerdings per Unicast angefordert.

Melete [126] erweitert Trickle dahingehend, dass es eine selektive Verteilung des Programms ermöglicht, indem das Programm höchstens n-mal weitergeleitet wird. Die Verbreitung ist so auf einen hop-basierten Umkreis um die Senke beschränkt. Befindet sich jedoch mindestens ein zu programmierender Knoten nicht in der Nähe der Senke, sondern am anderen Ende des Netzes, funktioniert Melete wie Trickle.

Deluge [43] erweitert Trickle, so dass es möglich ist, größere Programme zu verbreiten, die in mehrere Nachrichten aufgeteilt werden müssen. Die einzelnen Programmteile werden per Pipelining verbreitet: Während weiter von der Senke entfernte Knoten noch Teil 1 anfordern, können nah bei der Senke positionierte Sensorknoten schon Teil 2 anfordern. Programmteile werden hier per Unicast angefordert. Allerdings bekommen hier wiederum

alle Sensorknoten das Programm. Aqueduct [26] erweitert deshalb Deluge, indem es ein selektives Reprogrammieren unterstützt. Hierbei wird vor dem Anfordern des Programms ein Baum aufgebaut, der die zu programmierenden Geräte als Blätter und Knoten enthält. Geräte, die nicht programmiert werden, bilden so genannte *Aquädukte*, welche die Kanten des Baumes sind. Sie stellen immer kürzeste Multihop-Pfade zwischen den Knoten dar und bestehen ausschließlich aus symetrischen Links. Anschließend greift der Mechanismus von Trickle. Unterschiedlich ist nur, dass jedes Gerät das Programm über einen Multihop-Pfad von seinem Vaterknoten anfordert und das Programm in viele Nachrichten aufgeteilt wird. Allerdings beinhaltet Aqueduct noch große Defizite, welche die Autoren beschreiben. Eine extrem geringe Geschwindigkeit und eine fehlende Robustheit gegen Schwankungen in den Kommunikationsverbindungen sind die größten Probleme. Das Verfahren ermöglicht auch nicht, zur Laufzeit eine beliebige Teilmenge des Netzes zu programmieren, da eine Anwendungs-ID, die die Knoten zur Übersetzungszeit erhalten, bestimmt, welche Knotengruppe betroffen ist.

Neben diesen so genannten *polite-gossip*-Ansätzen gibt es auch Arbeiten (siehe [92, 109, 116]), in denen eine Senke ihre direkten Nachbarn programmiert, die das Programm erst dann weiterleiten, wenn sie es vollständig erhalten haben. Dieser hopweise Mechanismus eignet sich jedoch nicht für ein selektives Reprogrammieren, da er darauf beruht, dass die Menge der zu programmierenden Knoten zusammenhängend ist.

Ein TDMA-basierter Ansatz wird in [60] beschrieben. Er ist allerdings nur in positionsbewussten Sensornetzwerken anwendbar. Andere Autoren befassen sich damit, ein mobiles Netz neu zu programmieren (siehe [23, 87]), aber auch sie gehen davon aus, dass alle Knoten aktualisiert werden sollen.

Lee et al. beschreiben in [65] zwei neue Aspekte: Einerseits schlagen die Autoren vor, anstatt des langsameren EEPROM-Speichers, den schnelleren Flash-Speicher zum Zwischenspeichern des Programms zu verwenden. Sie teilen das Programm in maximal 255 *Seiten*, wobei eine Seite aus 8 Nachrichten besteht. Knoten speichern nun für jede Seite, von der Nachrichten fehlen, zwei Bytes: die Seitennummer im ersten und die fehlenden Nachrichten (als Bitvektor) im zweiten Byte. So müssen im ungünstigsten Fall 510 Bytes aufgewendet werden, was für Sensorknoten einen recht hohen Ressourcenverbrauch bedeutet. Weiterhin ergibt sich aus den dargestellten Einschränkungen, dass das Verfahren nicht anwendbar ist, wenn das Programm mehr als 2040 Nachrichten umfasst.

Andererseits empfehlen Lee et al., in einer Vorbereitungsphase jeden Knoten entscheiden zu lassen, welche Rolle er hat. Die Rolle kann sein: *Selektiert* (der Knoten wird reprogrammiert), *weiterleitend* oder *schlafend*. Dazu senden alle Knoten, die das Programm bekommen wollen, eine Nachricht per Baum-Routing an die Senke. Alle Knoten, die eine solche Nachricht weiterleiten, werden für den gesamten Programmierprozess zu Weiterleitungsknoten. Die Übrigen gehen für die Dauer des Programmierprozesses in den Schlafzustand. Diese Methode ist allerdings weder robust gegen schwankende und unidirektionale Kommunikationsverbindungen noch gegen ausfallende Knoten. Alle drei

Aspekte können dazu führen, dass der zuvor aufgebaute Baum zerfällt und daher keine Kommunikation mehr zwischen Senke und zu programmierenden Knoten möglich ist. Da alle unbeteiligten Knoten jedoch schlafen, kann auch kein neuer, funktionierender Baum aufgebaut werden. Weiterhin ist das von Lee et al. vorgeschlagene Verfahren nicht effizient, da fehlende Programmteile immer bei der Senke angefordert werden, auch wenn Knoten, die näher sind als die Senke, die Teile nachliefern könnten.

Zusammenfassend lässt sich feststellen, dass die meisten Arbeiten, die sich mit dem Reprogrammieren von Sensornetzen beschäftigen, nur in einem homogenen Netz funktionieren, in dem alle Knoten programmiert werden sollen. Außerdem befassen sich viele Autoren ausschließlich mit dem Programmiervorgang an sich, nicht aber mit dem Informieren des Anwenders über im Netz vorhandene Knoten oder deren Status, das heißt über den Fortschritt und Erfolg des Programmierens. Auch das Problem, wann die neue Anwendung gestartet werden soll, betrachten sie nicht.

Aus diesem Grund stellt dieses Kapitel ein selektives, robustes und effizientes Programmierverfahren vor und beleuchtet alle Aspekte, die beim Design eines Programmierverfahrens wichtig sind. Die Idee von Lee et al. zum Verteilen des Programms wurde ebenfalls evaluiert. Sie wird im Weiteren mit *Senke - Baum* bezeichnet. Verfahren, die nur Teile des Codes austauschen [47, 80, 83, 90], können mit dem vorgeschlagenen Reprogrammierverfahren kombiniert werden. Allerdings kostet das hierbei notwendige dynamische Linken von Code unverhältnismäßig viel Speicherplatz, so dass solche Verfahren in Sensornetzen nicht sinnvoll erscheinen.

5.3 Selektives Reprogrammieren

Bevor das Programmierverfahren beschrieben wird, stellt der folgende Abschnitt vor, von welchen Annahmen ausgegangen wurde und somit unter welchen Bedingungen das Programmierverfahren eingesetzt werden kann. Anschließend werden die grundlegende Funktionsweise sowie die einzelnen Schritte, die zum Reprogrammieren notwendig sind, detailliert erläutert.

5.3.1 Annahmen

Als erstes wird von einem statischen, nicht partitionierten Multihop-Netz ausgegangen, in dem ein Gatewayknoten (im Weiteren *Senke* genannt) existiert, über den das neue Programm im Netz verbreitet wird und der dem Anwender Informationen aus dem Netz zur Verfügung stellt. Um ermitteln zu können, welche Knoten im Netz existieren, muss die Größe des Netzes bekannt sein. Weiterhin sollte die mittlere Netzdichte grob abgeschätzt werden können.

Abhängig von der Hardwareplattform schreiben Knoten, die reprogrammiert werden sollen, eingehende neue Programmteile direkt in den Flash- oder in den EEPROM-Speicher, von wo sie sie vor dem Neustarten in das RAM kopieren.

Entweder läuft das Programmierprotokoll parallel zu der Anwendung oder die Knoten starten (sobald sie wissen, dass sie programmiert werden sollen) einen Bootloader, der das Programmierprotokoll umfasst. Die Anwendung der Knoten, die nicht reprogrammiert werden, beinhaltet alle vom Programmierprotokoll benötigten Protokolle, z. B. ein Baum-Routing. Diese Knoten können nur einen geringen Teil des Programms im Speicher vorhalten, gegebenenfalls nur wenige Nachrichten.

Die in dieser Arbeit verwendeten Sensorknoten auf Basis des Standards IEEE 802.15.4 [45] kommunizieren, beruhen die vorgeschlagenen Senderaten und Zeitspannen auf einer theoretisch möglichen Datenrate von 250 kBit/s. Für abweichende Raten müssen die Werte des Verfahrens entsprechend angepasst werden. Verfahren, die auf speziellen MAC-Verfahren beruhen, werden nicht betrachtet.

Auch die angegebenen Absolutwerte der Codegrößen sind nicht allgemein gültig, sondern beziehen sich auf die eingesetzte Hard- und Softwareplattform.

Schließlich wird davon ausgegangen, dass jeder Knoten eindeutig identifiziert und adressiert werden kann und dass – entweder bei den Knoten selbst oder zentral – Informationen über den Hardwaretyp jedes Knotens vorliegen.

5.3.2 Grundlegende Funktionsweise

Knoten, die reprogrammiert werden sollen, werden im Folgenden als *selektierte Knoten* bezeichnet. Knoten, die Nachrichten weiterleiten müssen, so dass sie während des Reprogrammierungsprozesses nicht in den Schlafzustand gehen dürfen, werden *Weiterleitungsknoten* genannt, die Vereinigung beider Gruppen *beteiligte Knoten*.

Grundsätzlich besteht das Programmierverfahren aus den folgenden Phasen, die bezüglich ihrer Dauer möglichst optimiert werden sollen. Um dem Leser ein Gefühl dafür zu geben, wie viel Zeit für jede Phase in etwa aufgewendet werden muss, enthält die Kurzbeschreibung der Phasen auch Hinweise zum jeweiligen Zeitaufwand.

1. *Ermittlung von Knoten (Abschnitt 5.3.3):* Zunächst muss an der Senke gesammelt werden, welche Knoten im Netz existieren, damit die zu programmierenden Knoten selektiert werden können. Hierzu müssen von jedem Knoten Kenndaten zur Senke transportiert werden. Gerade in großen und tiefen Netzen ist dies aufwändig und muss mittels geeigneter Aggregation der Daten optimiert werden. Nur so kann die hierfür benötigte Dauer auf einige Sekunden begrenzt werden. Anschließend werden die selektierten Knoten darüber informiert, dass sie programmiert werden sollen, und es wird eine Infrastruktur zur Kommunikation zwischen selektierten Knoten und Senke sowie umgekehrt aufgebaut. Die Zeit, die vergeht, bis alle Knoten wach

und empfangsbereit sind, hängt von der laufenden Anwendung ab und kann ohne deren Beeinflussung nicht verkürzt werden.

2. *Verteilen des Programms (Abschnitt 5.3.4):* Aufgeteilt in mehrere Funkpakete wird das Programm im Netz verbreitet. Bei einer typischen Netzdichte von weniger als 30 Nachbarn und in Abhängigkeit vom gewählten Verfahren kann bei einer theoretischen Datenrate von 250 kBit/s etwa alle 70 ms ein Paket ins Netz geflutet werden. Wird ein Programm beispielsweise in 1000 Pakete geteilt, müssen für diesen Prozess somit ungefähr 70 s aufgewendet werden.

3. *Übertragen von fehlenden Teilen (Abschnitt 5.3.5)*: Fehlende Programmteile müssen nachgefordert und erneut gesendet werden. Die grundlegende Idee besteht darin, zweistufig vorzugehen und fehlende Teile zunächst von Nachbarknoten anzufordern. Erst wenn diese keine Teile mehr liefern können, werden diese bei der Senke angefordert. Die Dauer dieser Phase hängt stark von der Anzahl der fehlenden Programmteile und der Nachrichtenverlustrate ab. Sie kann zwischen einigen Sekunden oder wenigen Minuten dauern.

4. *Abschluss und Neustart (Abschnitt 5.3.6):* Von selektierten Knoten werden Informationen bezüglich ihres Status eingesammelt, um zu verifizieren, dass alle Knoten programmiert wurden und, um Informationen zu liefern, bei welchen Knoten der Programmierprozess nicht erfolgreich abgeschlossen werden konnte. Dies wird wie in der ersten Phase durchgeführt. Anschließend wird mittels Reboot des Mikroprozessors das neue Programm gestartet. Abschnitt 5.3.6 zeigt, warum dieser Schritt erst durchgeführt werden sollte, wenn alle selektierten Knoten das gesamte Programm erhalten haben. Da die zu verbreitende Information über den Neustart in eine Nachricht passt, ist die Dauer dieser Phase verglichen mit dem Verteilen des Programms vernachlässigbar.

Um ein unerlaubtes Reprogrammieren der Knoten zu verhindern, wird ein Sicherheitsmechanismus benötigt. Bei physisch unzugänglichen Knoten reicht es aus, alle Nachrichten mit einem geheimen Schlüssel zu codieren. Der Besitz des Schlüssels authorisiert dann das Reprogrammieren der Knoten. Andernfalls muss ein Authentifizierungsverfahren verwendet werden, wie es beispielsweise in [53] beschrieben wird. Es nutzt ausschließlich Hashfunktionen und ist für Programmierverfahren geeignet, bei denen Broadcast-Nachrichten authentifiziert werden sollen. Da Authentifizierungsverfahren nicht im Vordergrund dieser Arbeit standen, wurde das Verfahren [53] nicht implementiert oder evaluiert.

Die folgenden Abschnitte stellen die einzelnen Phasen des in dieser Arbeit vorgeschlagenen Programmierverfahrens im Detail vor.

5.3.3 Knotenermittlung und Aufbau einer Infrastruktur

Zunächst müssen die Knoten ermittelt werden, die programmiert werden sollen. Sie müssen anschließend darüber informiert werden und das Netz muss eine Infrastruktur bereitstellen, über die die Senke und die zu programmierenden Knoten Nachrichten austauschen können. Die erste Phase teilt sich in die folgenden Einzelschritte:

Auffinden von Knoten/Startzeitpunkt verbreiten: Im ersten Schritt werden die zu programmierenden Knoten ermittelt. Ein Kriterium, wie beispielsweise eine Anwendungs-ID oder ein Hardwaretyp, kann definieren, welche Knoten das neue Programm erhalten. Häufig muss aber auch der Anwender wissen, welche Knoten im Netz existieren, und die zu programmierenden Knoten auswählen. Da dem Anwender jedoch a priori nicht zwingend Informationen über die Knoten zur Verfügung stehen, anhand derer er diese Auswahl treffen könnte, müssen alle relevanten Knotendaten an der Senke gesammelt und dem Anwender zur Verfügung gestellt werden.

Ein Problem dabei ist, dass die Knoten üblicherweise einem anwendungsabhängigen Duty-Cycle folgen, so dass das Programmierprotokoll warten muss, bis ein Knoten wach und empfangsbereit ist, bevor es diesen über den Updateprozess informieren kann. Auch müssen die Wach- und Schlafphasen mehrerer Knoten nicht notwendigerweise gleichzeitig stattfinden oder gleich lang sein.

In einem stationären Netz sollte deshalb die Senke, die das Anwendungsverhalten kennt, eine *Aufwachzeit*-Nachricht fluten, sobald sie Knoten als empfangsbereit erwartet. Die Nachricht enthält eine Zeitdauer, die anzeigt in wie vielen Sekunden, Minuten oder Stunden der Programmierprozess beginnt. Empfänger dieser Nachricht berechnen daraus den Startzeitpunkt, zu dem sie aufwachen und vorerst wach bleiben sollen.

Um Code zu sparen, sollte das Baum-Routing (wie in Abschnitt 2.10.2 beschrieben) in der Lage sein, Nachrichten zu fluten.

Knoteninformationen sammeln: Zum Startzeitpunkt werden alle für den Anwender relevanten Informationen wie ID, Anwendungs-ID und Hardwaretyp von allen Knoten mittels Baum-Routing an der Senke gesammelt. Werden die Nachrichten von allen Knoten separat zur Senke gesendet, werden im ungünstigsten Fall (das heißt größtmögliche Tiefe des Baumes) $O(n^2)$ Nachrichten gesendet. Effizienter ist es, Nachrichten zu aggregieren, das heißt zusammenzufassen und nur eine Gesamtnachricht weiterzuleiten. Genauere Informationen zur Wirkung von Aggregation können in [51] nachgelesen werden. Dies erfordert aber auch, dass entweder das Baum-Routing eingehende Nachrichten vor dem Weiterleiten an das Programmierprotokoll weitergibt oder ein Baum-Routing in das Programmierprotokoll integriert ist.

In dem hier untersuchten Szenario des Reprogrammierens sind alle betroffenen Knoten wach und sollen in möglichst kurzer Zeit eine Nachricht zur Senke senden. Die meisten Autoren, die sich mit der Aggregation von Daten beschäftigen, gehen jedoch davon aus, dass auf den Knoten eine Anwendung läuft, die einem Duty-Cycle unterliegt, und Knoten regelmäßig, aber zu unterschiedlichen Zeitpunkten, Daten an eine Senke senden müssen (z. B. [106]). Einige Verfahren überlassen es auch dem darunterliegenden MAC- oder TDMA-Protokoll, Kollisionen zu vermeiden und Nachrichten wiederholt zu senden (z. B. [127]). Das bedeutet aber, dass entweder der Anwendungsprogrammierer auf dieses MAC-Protokoll festgelegt ist oder dass zusätzlicher Code und Management der verschiedenen MAC-Protokolle notwendig sind. Beides ist für ein flexibles Reprogrammierverfahren, das möglichst wenig Code umfassen soll, nicht akzeptabel. Einige Arbeiten beschäftigen sich auch mit der optimalen Sendereihenfolge der Knoten, wenn mehrere Funkkanäle zur Verfügung stehen. Das synchrone Wechseln von Kanälen mit einer Genauigkeit im Millisekundenbereich ist jedoch nicht realistisch und erfordert ebenfalls zusätzlichen Code. Im Folgenden werden nur anwendbare und praktische Lösungsansätze für die Datenaggregation betrachtet.

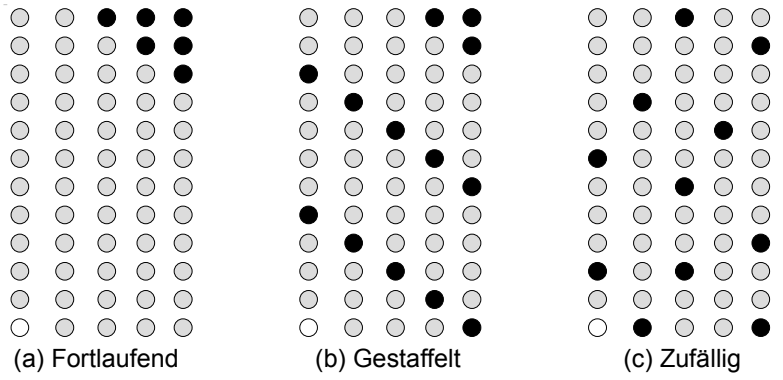


Abbildung 5.1: Aggregationsverfahren; Knoten, die mit dem Senden ihrer ID beginnen, sind als schwarz gefüllte Kreise dargestellt. Die Senke (nicht ausgefüllter Kreis) befindet sich in der linken, unteren Ecke.

Fortlaufendes Senden: Gegeben sei, dass das Baum-Routing beim Baumaufbau dafür sorgt, dass jeder Knoten seine hop-basierte Distanz zur Senke kennt. Intuitiv wäre dann ein sinnvolles Vorgehen zum Einsammeln, mit den Knoten zu beginnen, die am weitesten von der Senke entfernt liegen, wie es Annamalai et al. [1] beschreiben. Da der vorgeschlagene Ansatz jedoch ein zentrales Verfahren ist und darauf beruht, dass alle Informationen über den Baum an der Senke vorhanden sind, wurde nur die Idee übernommen. Statt der optimalen Sendereihenfolge aller Knoten für jede Gruppe von Knoten wurde ein gemeinsamer Sendeslot angewendet. Abbildung 5.1(a) zeigt dieses Vorgehen. Knoten, die als erstes zu ihren Vaterknoten senden, sind als schwarz ausgefüllte Kreise dargestellt. Die Senke befindet sich in

der linken unteren Ecke (nicht ausgefüllter Kreis). Die Gruppen ergeben sich aus der hop-basierten Distanz zur Senke. Innerhalb einer Gruppe dient ein zufälliger Backoff der Bestimmung des exakten Sendezeitpunktes. Anschließend senden die Vaterknoten ihre ID, usw. Dieser Ansatz wird im Folgenden als *Fortlaufendes Senden* bezeichnet.

Gestaffeltes Senden: In großen Netzwerken, bei 32-Bit-Adressierung oder für den Fall, dass noch mehr Informationen als die ID gesammelt werden sollen, ist es jedoch unwahrscheinlich, dass eine Nachricht ausreicht, um alle IDs zu übertragen. Die Informationen müssen dann in mehreren Teilen übertragen werden und viele Knoten müssen mehrere Nachrichten weiterleiten, sobald eine weitere Aggregation die Länge einer Nachricht überschreitet. Damit ein Knoten jedoch nicht zwei oder mehr Nachrichten auf einmal senden muss und unnötigerweise Staus entstehen, liegt die Idee nahe, mit dem Senden parallel zu beginnen. Vor dem Hintergrund, dass Knoten, die 4 Hops voneinander entfernt liegen, gleichzeitig senden können, ohne dass ihre Nachrichten kollidieren, ist es sinnvoll, dass Knoten mit den Distanzen 4, 8, 12, 16, 20, ... mit dem Senden beginnen (siehe Abbildung 5.1(b)). Anschließend senden die Knoten mit einer Distanz von 3, 7, 11, 15, 19 usw. Dieses Vorgehen wird *Gestaffeltes Senden* genannt. Weitere Vorteile dieses Vorgehens bestehen darin, dass der Netzwerkdurchmesser nicht bekannt sein muss und dass die Knoten weniger Daten zur gleichen Zeit speichern müssen.

Sowohl beim fortlaufenden als auch beim gestaffelten Senden hat jede Gruppe von Knoten, die eine bestimmte Distanz zur Senke hat, einen gemeinsamen Zeitslot, innerhalb dessen ihre Mitglieder einen zufälligen Zeitpunkt zum Senden auswählen. Beim gestaffelten Senden berechnen die Knoten ihren Sendezeitpunkt T wie folgt:

$$T = rand\,(10) \cdot d + (3 - d_{sink}\%4) \cdot 10 \cdot d \tag{5.1}$$

wobei d_{sink} die hop-basierte Distanz zur Senke und d die Dichte bezeichnet.

Zufälliges Senden: Eine noch minimalere Möglichkeit besteht darin, dass alle Knoten – unabhängig von ihrer Distanz zur Senke – zu einem zufälligen Zeitpunkt senden. Dieser Ansatz ist in Abbildung 5.1(c) dargestellt und wird als *Zufälliges Senden* bezeichnet. Auch hier senden die Knoten alle bisher empfangenen IDs zusammen mit ihrer eigenen.

Die drei Verfahren (fortlaufendes, gestaffeltes und zufälliges Senden) erfordern weder eine Organisationsphase noch zentral vorhandene Informationen und ihre Implementierungen erzeugen nur etwa 800 Byte Code, weshalb sie für das Programmierverfahren geeignet sind. In Abschnitt 5.4.1 werden sie bewertet und miteinander verglichen.

Insgesamt geschieht das Einsammeln von Daten mit folgendem ***Aggregationsalgorithmus:*** Die Senke sendet eine *Anfrage*-Nachricht, wobei sie gleichzeitig einen Baum aufbaut. In Abhängigkeit von der Phase sendet jeder oder nur jeder selektierte Knoten, der die Nachricht empfängt, seine ID zurück an die Senke. Der Zeitpunkt des Sendens richtet sich nach dem eingesetzten Verfahren. Jeder Knoten fügt dabei seine ID zu einer temporären Liste mit allen von Kindknoten bereits empfangenen IDs hinzu und sendet die IDs in so wenig Nachrichten wie möglich an seinen Vaterknoten. Diese werden auf MAC-Ebene mit Anforderung auf Acknowledgement versendet, so dass der Sender die IDs bei einem Übertragungsfehler wieder seiner temporären Liste hinzufügt und die Nachricht nach einem zufälligen Backoff von wenigen Millisekunden wiederholt. Empfängt die Senke innerhalb einer bestimmten Zeit keine ID und hat noch nicht alle IDs erhalten, so wiederholt sie die Anfrage, die wieder einen neuen Baum konstruiert. Dies ist notwendig, da Knoten ausfallen oder ihre Nachrichten zeitweilig nicht empfangen werden könnten. Sind die fehlenden IDs bekannt, können diese direkt angefordert werden, indem die Senke sie in die Anfrage-Nachricht integriert. Ein Timer sorgt dafür, dass die Senke nicht unbegrenzt auf die Antwort eines ausgefallenen Knotens wartet.

Die Anfrage-Nachrichten enthalten dabei nur den Typ „Programmierprotokoll“ und den Subtyp der Anfrage, z. B. „Präsenzanfrage“. Die Spezifikation der Antwort-Nachrichten ist in Abbildung 5.2 dargestellt,

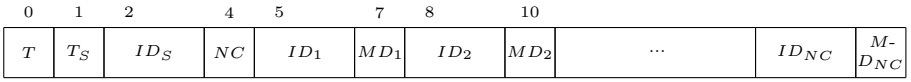


Abbildung 5.2: Antwort-Nachricht als Byte-Array

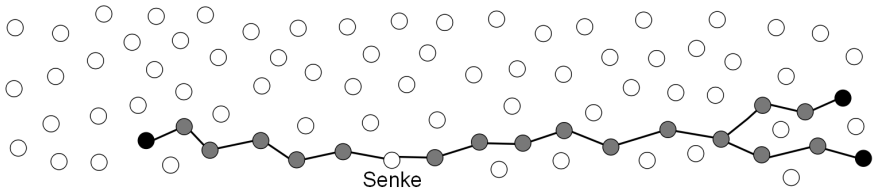
wobei die Byte-Felder der Nachricht die folgenden Bedeutungen haben:

- T: Nachrichtentyp (Typ „Programmierprotokoll“)
- T_S: Subtyp (Typ der Antwortnachricht, z. B. „Präsenzantwort“)
- ID_S: ID der Senke (Ziel der Nachricht)
- NC: Anzahl an eingetragenen IDs
- ID_i: ID des Knotens
- MD_i: Metadaten des Knoten (optional), z. B. Hardwaretyp, Anwendungs-ID, ...

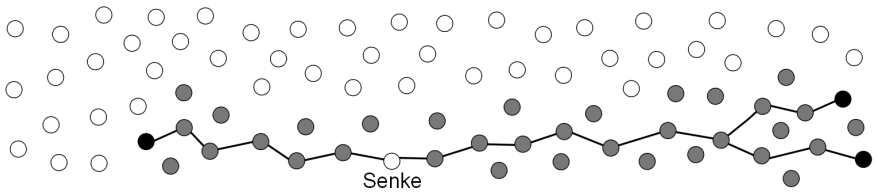
Knotenauswahl: Aus den an der Senke gesammelten IDs der Knoten im Netz bestimmt der Anwender oder ein Algorithmus die Menge der selektierten Knoten. Um diese über ihre Auswahl zu informieren, muss die Senke die IDs der selektierten Knoten ins Netz senden. Sie sendet deshalb ein oder mehrere *Auswahl*-Nachrichten, auf die die selektierten Knoten mit einer *Auswahlantwort*-Nachricht reagieren. Dies geschieht wieder mit dem beschriebenen Aggregationsalgorithmus. Über die Auswahlantwort-Nachrichten wird

geprüft, ob alle Auswahl-Nachrichten empfangen wurden. Ist dies nicht der Fall, werden die entsprechenden Knoten als nicht erreichbar angesehen und müssen manuell geprüft oder in einem weiteren Versuch reprogrammiert werden.

Infrastrukturaufbau: Alle Knoten, die eine Auswahlantwort-Nachricht zur Senke weitergeleitet haben, werden zu Weiterleitungsknoten (gekennzeichnet als grau markierte Kreise in Abbildung 5.3(a)). In Abbildung 5.3 sind die selektierten Knoten als schwarz ausgefüllte Kreise dargestellt.



(a) Senke, Selektierte Knoten (schwarz) und (Teil-)Baum (grau)



(b) Senke, Selektierte Knoten (schwarz) und Redundanter Baum (grau)

Abbildung 5.3: Infrastrukturaufbau

Alle Weiterleitungsknoten senden nun eine *Weiterleiten*-Nachricht per Broadcast, damit ihre direkten Nachbarn ebenfalls Weiterleitungsknoten werden (siehe Abbildung 5.3(b)). Dies ist notwendig, da sonst Schwankungen in den Kommunikationsverbindungen dazu führen können, dass zwischen Senke und selektierten Knoten keine Kommunikation mehr möglich ist. Redundanz muss die fehlende Zuverlässigkeit ausgleichen.

Dieses Vorgehen wird im Weiteren als *Redundanter Baum* bezeichnet, das heißt alle Knoten, die eine Auswahlantwort-Nachricht zur Senke weitergeleitet haben, und alle Knoten, die eine Weiterleiten-Nachricht bekommen haben, bleiben wach und leiten Nachrichten weiter. Zum Vergleich werden zwei andere Ansätze mit unterschiedlichen Auswahlkriterien für Weiterleitungsknoten evaluiert:

1. *Alle*: Alle Knoten bleiben wach.
2. *Baum*: Alle Knoten, die eine Auswahlantwort-Nachricht zur Senke weitergeleitet haben, bleiben wach.

Zusammenfassung: Die erste Phase besteht aus folgenden Einzelschritten:

1. Die Senke flutet die Aufwachzeit, das heißt die Startzeit des Programmierens.
2. Die Senke sendet Präsenzanfrage-Nachrichten und baut dabei Bäume auf.
3. Alle Knoten senden ihre ID und die erforderlichen Informationen zur Senke.
4. Der Anwender oder ein Algorithmus wählt die zu programmierenden Knoten aus.
5. Die Senke sendet die IDs der selektierten Knoten.
6. Die selektierten Knoten senden eine Auswahlantwort-Nachricht zur Senke zurück (vgl. Abbildung 5.3(a)). In Abhängigkeit von der Strategie (bei 'Baum' und 'Redundanter Baum') werden alle Knoten, die eine Bestätigung weitergeleitet haben, zu Weiterleitungsknoten.
7. Redundanter Baum: Weiterleitungsknoten broadcasten eine Nachricht, ihre Nachbarn werden ebenfalls Weiterleitungsknoten (vgl. Abbildung 5.3(b)).

Die Leistungsfähigkeit des Verfahrens wurde auch für den Fall evaluiert, dass fehlende Programmteile in der N-Hop-Nachbarschaft angefordert werden, statt nur bei direkten Nachbarn, sowie für den Fall, dass nicht selektierte Knoten einen Teil des Programms vorhalten können. In beiden Fällen müssten die selektierten Knoten nach dem siebten Schritt noch eine Nachricht in ihre N-Hop- Nachbarschaft fluten, damit diese Knoten ebenfalls wach bleiben. Keines der beiden Vorgehen hat sich jedoch bewährt, so dass dieser Schritt entfällt.

Der letzte Schritt dieser Phase wird in wenigen Hundert Millisekunden absolviert. Daher kann mit einer kurzen Verzögerung (z. B. eine Sekunde) – zur Vermeidung von Kollisionen mit weiteren Auswahlantwort- und Weiterleiten-Nachrichten – die nächste Phase (Verteilen des Programms) gestartet werden, sobald alle Auswahlantwort-Nachrichten an der Senke angekommen sind. Sind nicht alle erwarteten Auswahlantwort-Nachrichten eingegangen, kann der Anwender entscheiden, ob weiter gewartet oder ohne die fehlenden Knoten fortgefahren werden soll, die er gegebenenfalls manuell, z. B. auf ihren Batterieladezustand, prüft.

5.3.4 Verteilen des Programms

Nach dem Aufbau der Infrastruktur verbreitet die Senke das Programm im Netz. Fluten ist ein verbreitetes Mittel, um den Einfluss von Nachrichtenverlusten abzuschwächen. Nachteile redundanter Übertragung bestehen im Vergleich zur optimalen Lösung in einer höheren Anzahl an Nachrichten und einem erhöhten Zeitaufwand, um eine Nachricht im Netz zu verbreiten. Besonders in dichten Netzen ist der Zeitaufwand ein Problem, wenn viele Nachrichten nacheinander geflutet werden sollen. Eine Nachricht darf frühestens dann gesendet werden, wenn die entstandene Flutwelle der vorherigen Nachricht 4 Hops

entfernt ist, damit es nicht zu systembedingten Kollisionen kommt. Je dichter und größer das Netz ist, desto länger dauert dies. Im Rahmen dieser Arbeit wurden verschiedene Weiterleitungswarscheinlichkeiten und Senderaten evaluiert (siehe Abschnitt 5.4.2).

Die Senke flutet nun das gesamte Programm mit einer bestimmten Rate. Eine Programm-Nachricht enthält die in Abbildung 5.4 dargestellten Byte-Felder.

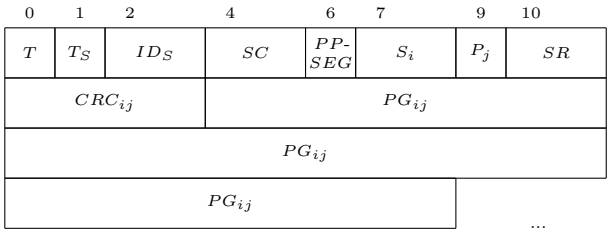


Abbildung 5.4: Programm-Nachricht als Byte-Array

- T: Nachrichtentyp (Typ „Programmierprotokoll“)
- T_S: Subtyp (Subtyp „Programm“)
- ID_S: ID der Senke
- SC: Gesamtsegmentanzahl
- $PPSEG$: Anzahl an Paketen pro Segment
- S_i: Segmentnummer
- P_j: Paketnummer innerhalb des Segments
- SR: Anzahl an verbleibenden, noch zu sendenden Segmenten (wichtig, wenn Programmteile wiederholt gesendet werden)
- CRC: Checksumme über Programmteil ij
- PG_{ij}: Programmteil ij

Eine 4 Byte lange Checksumme über den Programmteil (z.B. mittels XOR-Operator) ermöglicht es den Empfängerknoten zu überprüfen, ob das Programmteil korrekt ist. Ist dies der Fall, schreiben die Knoten den Programmteil in ihren Speicher. Schlägt der Schreibvorgang mehrmals fehl, senden die Knoten eine Nachricht an die Senke mit der Information, dass das Reprogrammieren fehlschlug.

Alle selektierten Knoten speichern aber nicht nur den in der Nachricht enthaltenen Teil des Programms, sondern auch den eigenen Empfangszustand, um zu wissen, ob sie alle Programmteile erhalten haben. Dafür verwenden sie einen Bitvektor (*Zustandsvektor*), wobei eine 0 für *vorhanden* und eine 1 für *fehlend* steht. Damit der Zustandsvektor zuzüglich 4 Byte Headerinformationen (T = „Programmierprotokoll“ = 1 Byte, T_S = „Programmantwort“ = 1 Byte und ID_S = ID der Senke = 2 Byte) beim Nachfordern von fehlenden Teilen in einem Radiopaket übermittelt werden kann, muss die Länge des Vektors auf die maximale Länge eines Radiopakets (MTU) abzüglich 4 Bytes beschränkt

werden. Beispiel: Der Vektor darf für eine MTU von 104 Bytes 100 Bytes = 800 Bits lang sein. Er kann also Informationen für 800 Teile enthalten.

Besteht das Programm aus mehr Teilen als der Vektor fassen kann, müssen mehrere Programmteile in einem *Segment* zusammengefasst und durch dasselbe Bit repräsentiert werden.

Die Anzahl $PPSEG$ (packets per segement) der Nachrichten pro Segment ergibt sich aus der Größe des Programms. Für Programme, welche aus bis zu 1600 Teilen bestehen, ist im Beispiel eine Anzahl von $PPSEG = 2$ Teilen pro Segment ausreichend. Tabelle 5.1 zeigt zusätzlich zur Anzahl der Nachrichten pro Segment auch die sich ergebende Länge des Zustandsvektors. Bei bis zu 2400 Teilen müssen entsprechend 3 Teile in einem Segment zusammengefasst werden. Das Beschränken des Zustandsvektors hat weiterhin den Vorteil, dass der Speicherverbrauch limitiert werden kann.

#Programmteile	PPSEG	Länge des Zustandsvektors
600	1	75 Bytes = 600 Bits
800	1	100 Bytes = 800 Bits
1200	2	75 Bytes = 600 Bits
1600	2	100 Bytes = 800 Bits
1800	3	75 Bytes = 600 Bits
2400	3	100 Bytes = 800 Bits

Tabelle 5.1: Nachrichten pro Segment in Abhängigkeit von der Größe des Programms

Temporär muss sich ein Knoten auch für wenige ($|SEG_{temp}|$) Segmente merken können, welche Teile des Segments er bereits erhalten hat. Da in der folgenden Phase auch Nachbarn Segmente senden, muss ein Knoten sich dies für mindestens so viele Segmente merken können, wie er benachbarte Sender hat. Ein Sender sendet immer alle Teile eines Segments hintereinander, ohne zwischendurch andere Nachrichten zu senden. Der Empfänger benötigt deshalb $|SEG_{temp}| \cdot (\lceil log_2(|SEG_{temp}|) \rceil + PPSEG)$ bits. Eine 0 bedeutet auch hier vorhanden und 1 fehlend. Da die Anzahl benachbarter Sender klein ist, ist dies unproblematisch.

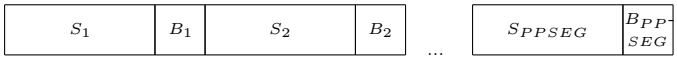


Abbildung 5.5: Temporärer Puffer zum Speichern von Zuständen einzelner Segmente

Abbildung 5.5 zeigt den Puffer, welcher dem Speichern von Empfangszuständen von Nachrichten innerhalb eines Segments dient. S_i ist $\lceil log_2(|SEG_{temp}|) \rceil$ Byte groß und be-

zeichnet die Nummer des Segments, dessen Empfangszustand im nachfolgenden Bitvektor B_i der Länge $PPSEG$ Bits gespeichert ist.

Damit unvollständige Segmente nicht den Puffer füllen und blockieren, wird der Puffer als Ringpuffer verwendet: Wenn eine neuere Nachricht mit dem Nachrichtenindex 0 empfangen wird, wird ein älterer Eintrag wieder überschrieben, auch wenn das überschriebene Segment unvollständig ist.

Falls nicht selektierte Knoten einen Teil des Programms vorhalten können, merken sie sich so viele zufällig ausgewählte Segmente wie möglich. Erhalten sie nicht alle Nachrichten aus einem Segment, überschreiben sie dieses wieder mit einem anderen Segment. Im Folgenden wird der Anteil des Programms, den ein nicht selektierter Knoten im Speicher vorhalten kann, mit p_{store} bezeichnet.

Aus der Gesamtanzahl der Segmente und der Anzahl der Nachrichten pro Segment berechnen die Knoten, aus wie vielen Nachrichten das Programm besteht. So können sie aus der laufenden Nummer jeder Nachricht, den Zeitpunkt bestimmen, wann die letzte Nachricht eintreffen wird, da die Senke mit gleich bleibender, bekannter Rate sendet. Mit jedem Nachrichtenempfang kann die Berechnung des Zeitpunktes verfeinert werden. So können alle Knoten nahezu gleichzeitig in die nächste Phase (Nachliefern von fehlenden Teilen) übergehen. Auch wenn Knoten, die sich näher an der Senke befinden, Nachrichten früher empfangen, als solche, die weiter entfernt sind, bekommen doch benachbarte Knoten eine Nachricht zu einem nahezu gleichen Zeitpunkt, so dass sie die Phase auch zu einem annähernd gleichen Zeitpunkt wechseln.

5.3.5 Übertragen von fehlenden Teilen

Simulationen und Experimente haben gezeigt (vgl. Abschnitt 5.4.2), dass beim Fluten von vielen Nachrichten nacheinander einige Nachrichten von wenigen Knoten nicht empfangen werden und selten Nachrichten bei allen Knoten fehlen. Die fehlenden Nachrichten müssen erneut gesendet werden, damit das neue Programm vervollständigt wird. Anstatt nun immer bei der Senke die fehlenden Programmteile nachzufordern, scheint es vielversprechend, Programmteile mit benachbarten Knoten auszutauschen. Natürlich hilft es nicht, in der Nachbarschaft fehlende Teile anzufordern, wenn diese im gesamten Netz fehlen. Hier kann nur die Senke die Teile nachliefern und es ist erforderlich, dass zentral angefragt wird, wenn die fehlenden Teile lokal nicht mehr geliefert werden können. Es wird daher ein zweistufiges Verfahren vorgeschlagen:

1. Stufe: Knoten fordern fehlende Teile von ihren direkten Nachbarn an.[1]
2. Stufe: Alle Anfragen werden mit Hilfe des Aggregationsalgorithmus entlang eines Baumes zur Senke gesendet.

[1] Bemerkung: Evaluiert wurde auch ein Anfragen in der N-Hop-Nachbarschaft ($N = 2, 3, 4, 5$) mittels N-Hop-Fluten, das heißt, dass Nachrichten $N - 1$ mal weitergeleitet werden. Dies kann die Leistungsfähigkeit aber kaum steigern (siehe Abschnitt 5.4.3) und hat den Nachteil, dass weitere Knoten während des Programmierprozesses wach bleiben müssen.

Im Weiteren wird das zweistufige Verfahren als *Lokales Anfragen* (oder auch: $1 - Hop$) bezeichnet, in Abgrenzung zum Vergleichsverfahren *Zentrales Anfragen* (oder auch: *Senke*), bei dem nur die zweite Stufe angewendet wird.

Um zu verstehen, inwieweit ein Anfragen von fehlenden Programmteilen bei anderen Knoten als der Senke Sinn macht, wird zunächst analysiert, wie viele Teile von M beliebigen Knoten geliefert werden können. Sei dazu angenommen, dass jeder der M Knoten $k\,\%$ des Programms erhalten hat.

Sei x die Anzahl der Programmteile. Folgendes Problem wird nun formuliert: Wie groß ist die Vereinigung von M zufälligen Teilmengen einer x-elementigen Menge, wobei jede Teilmenge die Größe k hat?

$$X = \{A \subseteq \{1, 2, ..., k\} \mid |A| = M\} \tag{5.2}$$

Seien X_1, X_2, ..., X_M unabhängige gleichverteilte Zufallsvariablen auf einem Wahrscheinlichkeitsraum (Ω, P) mit Werten in X.

Gesucht wird zunächst die Wahrscheinlichkeit p_l dafür, dass in der Menge genau l verschiedene Elemente sind:

$$p_l = P\left(|\bigcup_{i=1}^{M} X_i| = l\right). \tag{5.3}$$

Da jede Teilmenge genau k Elemente (ohne Wiederholungen) enthält, gilt:

$$p_0 = p_1 = ... = p_{k-1} = 0. \tag{5.4}$$

Die Wahrscheinlichkeit, dass alle M Knoten die gleichen k Elemente auswählen, beträgt $\frac{1}{\binom{x}{k}^M}$. Es gibt $\binom{x}{k}$ Möglichkeiten k Elemente aus x Elementen ohne Zurücklegen auszuwählen. Für $l = k$ gilt deshalb:

$$
\begin{aligned}
p_k &= \sum_{\substack{A \subseteq X:, \\ |A|=k}} P\left(X_1 = A\right)^M && (5.5)\\
&= \binom{x}{k} \frac{1}{\binom{x}{k}^M} && (5.6)\\
&= \frac{1}{\binom{x}{k}^{(M-1)}} && (5.7)\\
&= \binom{x}{k}^{(1-M)}. && (5.8)
\end{aligned}
$$

Es gibt $\binom{l}{k}$ Möglichkeiten k aus l Elementen auszuwählen. Die Wahrscheinlichkeit p_l, dass genau l verschiedene Teile in der Vereinigung enthalten sind, ist $p_{\leqslant l}$ abzüglich aller Wahrscheinlichkeiten, dass weniger als l Teile enthalten sind. Für $l > k$ gilt somit:

$$
\begin{aligned}
p_l &= p_{\leqslant l} - p_{l-1} - ... - p_k && (5.9)\\
&= p_{\leqslant l} - \sum_{j=k}^{l-1} p_j && (5.10)\\
&= \binom{l}{k} \left(\frac{\binom{l}{k}}{\binom{x}{k}} \right)^m - \sum_{j=k}^{l-1} p_j, && (5.11)
\end{aligned}
$$

Mit dieser rekursiven Formel lässt sich der Erwartungswert numerisch berechnen.

Abbildung 5.6 zeigt, wieviel Prozent des Gesamtprogramms von M (= 2 bis 20) Knoten, die über einen zufällig verteilten Anteil der Größe k (= 9 % bis 99 %) verfügen, geliefert werden können. Auf der y-Achse ist M (die Anzahl der Knoten), auf der x-Achse ist k (der Anteil, über den ein einzelner Knoten verfügt) aufgetragen. Die z-Achse beschreibt den Anteil, der von der Vereinigung der M Knoten abgedeckt wird. Beispielsweise liefern nur 3 Knoten schon 99,06 % des gesamten Programms, wenn jeder Knoten nur 80 % bekommen hat. Dies zeigt, dass es voraussichtlich ausreicht, mit wenigen Nachbarn fehlende Programmteile auszutauschen.

Gegeben sei eine für ein Sensornetz sinnvolle Dichte von 8 Nachbarn, von denen mindestens 2 selektiert sind. Dann reicht $N = 1$ mit großer Wahrscheinlichkeit aus, wenn innerhalb der N-Hop-Nachbarschaft fehlende Teile angefordert werden. Ist die Dichte geringer, der Anteil der selektierten Knoten noch kleiner oder der auf einem einzelnen Knoten vorhandene Anteil des Programms kleiner, könnten größere Werte für N sinnvoll sein. Deshalb wurden Untersuchungen mit $N = 1, 2, ...5$ durchgeführt.

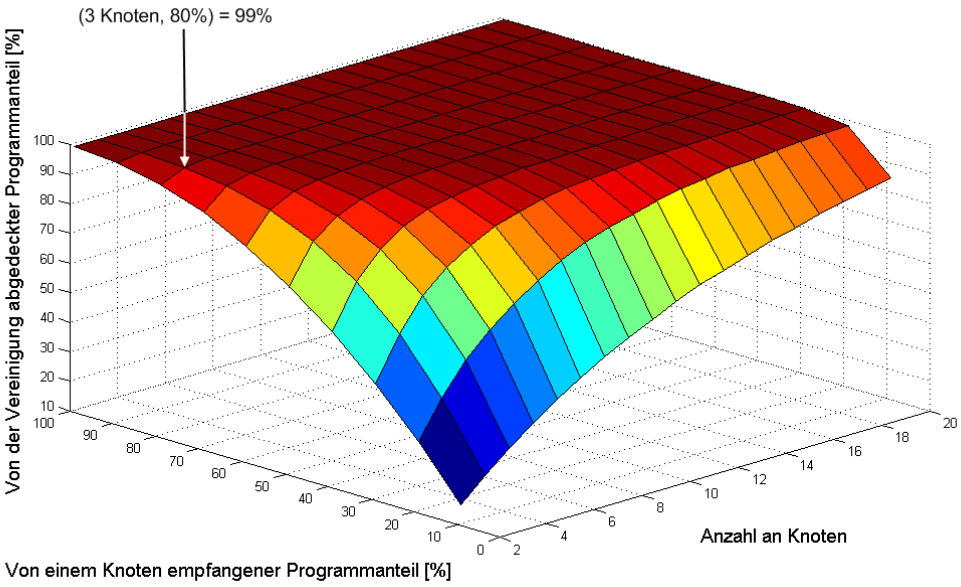


Abbildung 5.6: Mittlere Abdeckung aller Programmteile durch die Vereinigung von $M = 2$ bis 20 Knoten

Umgekehrt lässt sich vermuten, dass bei Betrachtung des gesamten Netzes (M = Anzahl von Knoten im Netz) jedes Teil des Programms mit großer Wahrscheinlichkeit bei mindestens einem Knoten fehlt. Fragen diese die Senke nach den fehlenden Teilen, muss die Senke nahezu jede Nachricht noch einmal fluten.

Damit Teile an der Senke angefordert werden können, ist ein funktionierender Baum unabdingbar. Um zu verhindern, dass ein ausgefallener Vaterknoten für seine Kindknoten zur Unerreichbarkeit der Senke führt, muss die Senke präventiv regelmäßig einen neuen, aktuellen Baum aufbauen, z. B. alle 5 bis 10 Sekunden.

Wie bereits im vorigen Abschnitt beschrieben, halten alle selektierten Knoten den eigenen Zustand von Segmenten in einem Bitvektor vor. Der Vektor hat bei allen Knoten die gleiche Länge und Reihenfolge. Bei Anfragen an die Senke werden die Vektoren verschiedener (potentiell sogar aller) Knoten zusammengefasst weitergesendet. Da eine 1 bedeutet, dass das Segment fehlt, wird zum Zusammenfassen der bitweise OR-Operator verwendet.

Zusätzlich zu ihrem eigenen Zustand müssen selektierte Knoten auch den Zustand ihrer Nachbarn speichern, um einerseits lokale Anfragen bedienen und andererseits Zustände mehrerer Knoten aggregiert zur Senke senden zu können. Hierfür reicht ein akkumulierter Zustand aus, da Programmnachrichten nicht an bestimmte Knoten adressiert sind, sondern per Broadcast oder Fluten gesendet werden. Auch hier wird ein Bitvektor

verwendet, bei dem eine 1 bedeutet, dass ein Segment bei mindestens einem Knoten fehlt und eine 0, dass es nicht fehlt. Bleibt anzumerken, dass auch evaluiert wurde, inwiefern die Leistungsfähigkeit gesteigert werden kann, wenn die Knoten mit Hilfe von B ($= 3$ bzw. 8) Bits pro Segment zählen, wie vielen Nachbarn (bis maximal $2^B - 1$) ein Segment fehlt, und daraufhin häufig gesuchte Segmente zuerst gesendet werden. Die gesteigerte Leistungsfähigkeit legimitiert jedoch nicht den dafür zusätzlich notwendigen Bedarf an Speicher und Code (siehe Abschnitt 5.4.3). Ohne das Zählen wird, wie für den eigenen Zustand, genau 1 Bit pro Segment benötigt.

Der Algorithmus des lokalen Verfahrens besteht aus zwei alternierenden Phasen: Die erste Phase (*Zustand senden*) beinhaltet, dass jeder selektierte Knoten seinen Zustandsvektor in seine N-Hop-Nachbarschaft sendet. In der zweiten Phase (*Segmente senden*) senden Knoten Segmente (ebenfalls in ihre N-Hop-Nachbarschaft), sofern dies notwendig ist. Abbildung 5.7 beschreibt den lokalen Programmablauf dieser Phase. Zunächst prüft jeder Knoten, ob er Segmente erhalten hat, die seinen Nachbarn fehlen. Von diesen sendet er bis zu Z zufällig ausgewählte mit der Wahrscheinlichkeit $p_1 = \frac{100\%}{|sel.neighbors|}$, damit nicht alle Knoten dieselben Segmente senden. Mit Hilfe von Simulationen wurde ermittelt, dass $Z = 15$ ein geeigneter Wert ist. Ein Segment, welches ein Knoten sendet, markiert er als „vorhanden“ in seinem Zustandsvektor für Nachbarknoten. Knoten, die Segmente empfangen, markieren diese in beiden Zustandsvektoren (im eigenen und in dem der Nachbarn) als „vorhanden“, damit sie selbst diese nicht noch einmal senden.

Anschließend überprüft jeder selektierte Knoten, ob ihm Segmente fehlen, die auch allen seinen Nachbarn fehlen. Ist dies der Fall und hat er in letzter Zeit auch keine Segmente (weder von der Senke noch von Nachbarn) bekommen, fordert er die restlichen Segmente mit der Wahrscheinlichkeit $p_2 = \frac{100\%}{|sel.neighbors|+1}$ bei der Senke an. Je weniger selektierte Nachbarn ein Knoten hat, desto größer ist die Wahrscheinlichkeit, dass er an der Senke anfragt.

Die beiden Phasen haben eine feste Länge, wobei die zweite Phase $Z \cdot PPSEG$ mal so lang ist wie die erste, damit nach einem Senden des Zustandsvektors Z Segmente gesendet werden können. Die Nachrichten von der Senke werden unabhängig von den beiden Phasen gesendet. Die Längen der Phasen sind bei allen Knoten gleich. Dadurch sind die Knoten in der Lage, zu zählen, wie viele selektierte Nachbarn sie haben, ohne dass die Uhren synchronisiert sein müssen. Sei L die Länge der ersten Phase, in der jeder selektierte Knoten genau eine Nachricht sendet. Der Wert von L muss von der Dichte abhängen und kann wie folgt abgeschätzt werden:

Sollen Kollisionen vermieden werden, muss jeder Knoten in seiner 2-Hop-Nachbarschaft einen Slot zur Verfügung haben, in dem allein er senden darf. Ein Knoten, der mindestens drei Hops entfernt ist, beeinträchtigt den erfolgreichen Empfang von Broadcast-Nachrichten nicht. Sind die Knoten etwa gleichverteilt und hat ein Knoten m direkte Nachbarn, befinden sich in seiner 2-Hop-Nachbarschaft etwa $4m$ Knoten, da sich die

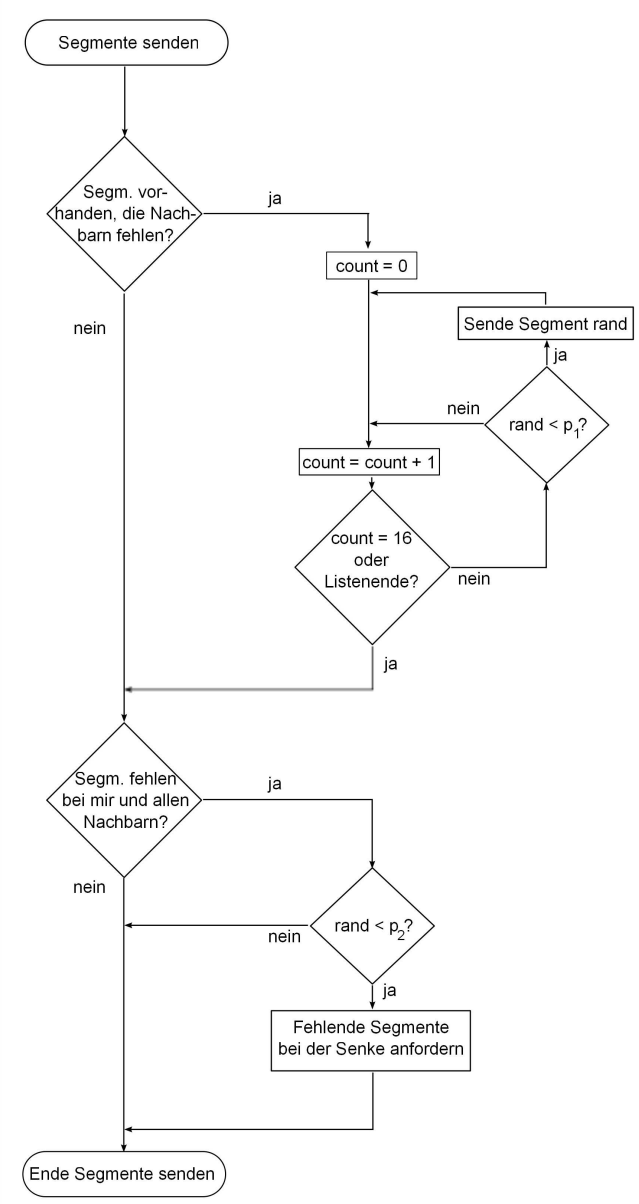


Abbildung 5.7: Segmente senden

Fläche der 2-Hop-Reichweite über $\pi(2r)^2 = 4\pi r^2$ annähern lässt. Dies entspricht dem Vierfachen der Fläche, die mit einem Hop abgedeckt werden kann (πr^2).

Es werden somit $4m$ Slots benötigt, wenn alle Knoten im Netz genau eine Nachricht kollisionsfrei senden sollen. L muss deshalb für den Fall, dass Segmente nur bei direkten

Nachbarn angefragt werden, $4mx$ sein, wobei x die Dauer einer einzelnen Nachrichtenübertragung ist.

Die Fläche der N-Hop-Nachbarschaft eines Knotens beträgt $\pi(Nr)^2 = N^2\pi r^2$ und ist damit N^2 mal so groß wie die Fläche der 1-Hop-Nachbarschaft. Für den Fall $N > 1$, das heißt, dass Nachrichten innerhalb der N-Hop-Nachbarschaft geflutet werden, gilt deshalb

$$L \quad = \quad 4m \cdot x \cdot N^2 m, \tag{5.12}$$

da jede Nachricht der $4m$ Knoten noch N^2m Mal wiederholt wird.

Für eine Dichte von 8, $N = 1$ und $Z = 15$ ergibt sich eine erforderliche Intervalllänge von 224 ms, wenn $x = 7$ ms angenommen wird. Hieraus ergibt sich eine Gesamtlänge für beide Phasen von etwa 3,5 s. Simulationen und Experimente haben jedoch gezeigt, dass für eine Dichte von 8 sogar 1 s ausreicht. Für eine Dichte von 50 reichen 3 s. Dies liegt daran, dass nicht jeder Knoten in der zweiten Phase immer Z Segmente sendet.

Die beiden Phasen werden maximal R-mal wiederholt, damit jeder Knoten irgendwann wieder aufhört, fehlende Segmente anzufordern, insbesondere für den Fall, dass er keine Nachrichten mehr empfängt. Auch beim zentralen Ansatz muss es eine obere Schranke für das Anfragen bei der Senke geben. R sollte vergleichsweise groß sein, z. B. 100, da es nur im Ausnahmefall zum Tragen kommt. Sind danach auf einem Knoten immer noch nicht alle Segmente vorhanden, ist das Verfahren bei diesem Knoten gescheitert. Sind alle Segmente bei einem Knoten und seinen Nachbarn vorhanden oder nach Ablauf der Wiederholungen, geht der Knoten in die nächste Phase über und sendet seinen Zustand. Die Senke geht in die nächste Phase über, wenn sie von allen Knoten eine Zustandsnachricht bekommen hat oder nach R Wiederholungen.

Temporär unerreichbare oder ausgefallene Knoten beeinflussen weder die Programmverteilung noch das Anfordern von fehlenden Teilen, da keine Operation auf Knoten wartet. Die Senke wird einen Ausfall allerdings erst nach Ablauf der nächsten Phase bemerken.

5.3.6 Abschluss und Neustart

Nach der vorherigen Phase, in der fehlende Programmteile nachgeliefert wurden, kann ein Knoten entweder über das gesamte Programm oder nur über einen Teil des Programms verfügen oder er ist zwischendurch ausgefallen. Als Rückmeldung über seinen Programmierstatus liefert er dementsprechend entweder eine Nachricht vom Typ *Erfolgreich* oder *Fehlgeschlagen* oder gar nichts. Die Nachricht muss seine ID enthalten, damit ihm der Nachrichteninhalt zugeordnet werden kann.

Beide Pakettypen (Erfolgreich und Fehlgeschlagen) können separat aggregiert werden, in dem jeder Knoten seine ID an die Nachricht anhängt, so dass eine Liste von IDs

entsteht. Wird ein zusätzliches Bit für den Status spendiert, können die beiden Typen auch kombiniert werden.

Schließlich müssen die Geräte die neue Anwendung starten. Hier gibt es verschiedene Möglichkeiten, zu welchem Zeitpunkt sie dies tun. Ein Ansatz wäre, die Anwendung direkt zu starten, sobald das letzte Programmteil empfangen und die Erfolgreich-Nachricht gesendet wurde.

Problematisch ist dabei vor allem, dass die neue Anwendung einem Duty-Cycle folgen kann, so dass die Kommunikation zwischen der Senke und den übrigen Knoten, die noch nicht das gesamte Programm bekommen haben, unterbrochen sein kann. Deshalb sollte die Senke eine *Neustart*-Nachricht fluten, die den nahezu gleichzeitigen Start der neuen Anwendung auf allen Geräten ermöglicht. Die Neustart-Nachricht beinhaltet die Zeitspanne, nach der der Neustart erfolgen soll.

5.4 Evaluation

Für die Bewertung der jeweiligen Verfahren wurden sowohl Experimente in der FRONTS-Testumgebung als auch Simulationen in Shawn durchgeführt. Bei den Simulationen wurden zunächst in einer quadratischen Fläche gleich bleibender Größe zwischen 225 und 900 Knoten an einem Gitter ausgerichtet, so dass – durch eine gleich bleibende Kommunikationsreichweite – unterschiedliche Dichten entstanden. Abstände von 30 bis 10 m führten zu Dichten zwischen 8 und 50. Anschließend wurde für eine Dichte von 8 die Simulationsfläche von $285x285m^2$ auf $1000x1000m^2$ sowie $1400x1400m^2$ vergrößert.

Um die Robustheit der Verfahren gegenüber Nachrichtenverlust zu überprüfen, wurde das stochastische Kommunikationsmodell eingesetzt. Die maximale Wahrscheinlichkeit p_{max}, mit der eine Nachricht bei einem Knoten eintrifft, der weniger als $d = 30$ m vom Sender entfernt ist, wurde auf 90 % und 70 % gesetzt, so dass sowohl weniger harte als auch sehr harte Bedingungen simuliert wurden. Die Diagramme der folgenden Abschnitte zeigen die Medianwerte und Extreme von 100 Simulationsläufen, teilweise zusätzlich die Quartile.

Um der Tatsache gerecht zu werden, dass die Menge der zu programmierenden Knoten sowohl im Netz verteilt sein als auch eine kompakte Gruppe bilden kann, wurden in dieser Arbeit zwei Arten von Anordnungen der selektierten Knoten innerhalb des Gesamtnetzes betrachtet:

- *Verteilte Struktur:* eine zufällig verteilte Auswahl von 5 % bis 100 % der Knoten sowie
- *Eck-Struktur:* eine Menge von Knoten, die sich in der gegenüberliegenden Ecke bezüglich der Position der Senke befinden.

5.4.1 Ermitteln der Knoten

Das Warten darauf, dass alle Knoten wach sind, hängt von der Anwendung ab und wurde deshalb in der Evaluation nicht betrachtet.

Für die Evaluation der Algorithmen zum Empfangen von Informationen aus dem Netz sendete die Senke eine Anfrage-Nachricht, die alle Empfänger mittels eines Baum-Routings mit ihrer ID beantworteten. Jeder Knoten fügte dabei seine ID zu allen bereits empfangenen IDs der Kindknoten hinzu und sendete eine aggregierte Nachricht oder erzeugte eine neue, falls er noch keine Nachrichten von Kindknoten bekommen hatte. Bekam die Senke innerhalb von d_{rx} ms keine Antwort und hatte noch nicht alle IDs erhalten und die letzte Anfrage war mindestens d_{tx} ms her, so wiederholte sie die Anfrage. Untersucht wurden die drei in Abschnitt 5.3.3 beschriebenen Vorgehen (fortlaufendes, gestaffeltes und zufälliges Senden). d_{rx} und d_{tx} hängen von der Tiefe des Netzes und vom Verfahren ab. Sie werden im folgenden Abschnitt bestimmt.

Ermitteln der Knoten – Evaluation mittels Simulationen

Die Dichte war hier mit 24 recht groß, da eine höhere Dichte zu mehr Wettbewerb beim Medienzugriff führt. Die Nachrichtenverlustrate betrug 10 % bzw. 30 %. Aufzeichnet wurde alle 100 ms, wieviel Prozent der IDs bereits an der Senke angekommen und wie viele Nachrichten gesendet worden waren.

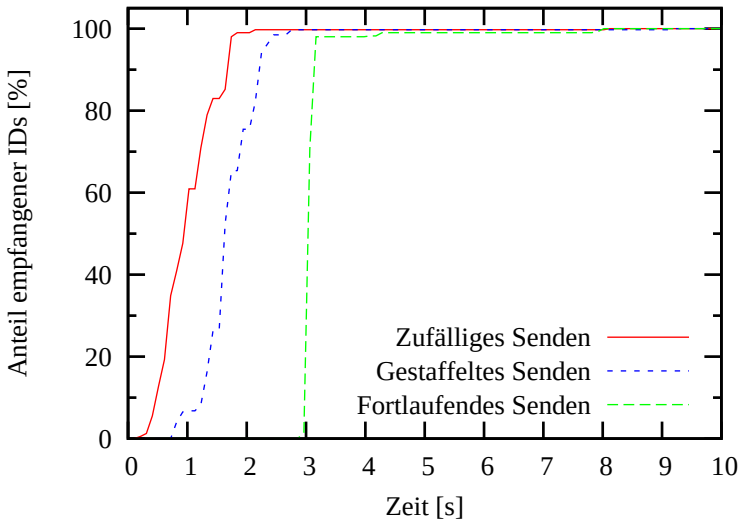
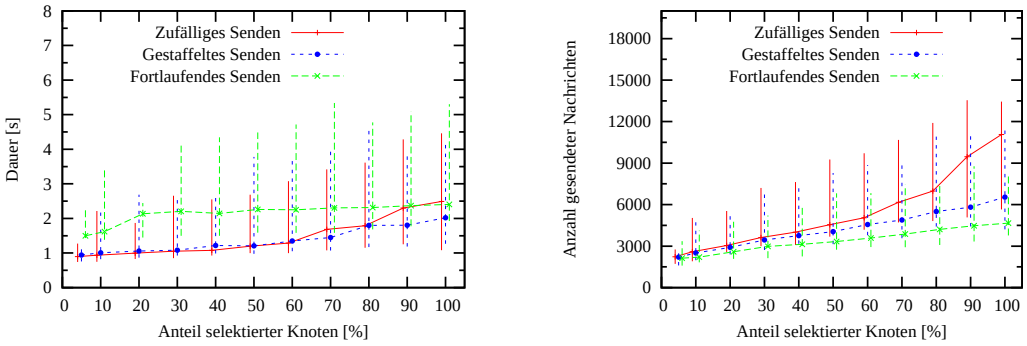


Abbildung 5.8: Zeitlicher Verlauf des Einsammelns von IDs (100 % selektierte Knoten)

Der Anteil der bereits eingetroffenen IDs ist in Abbildung 5.8 als Verlauf zu sehen. Da hier alle Knoten selektiert waren, kommen beim zufälligen Senden die ersten IDs am schnellsten nach dem Fluten der Anfrage-Nachricht an. Hier dauert es nur 200 ms, während es beim gestaffelten Senden 800 ms und beim fortlaufenden Senden 2,9 s dauert. Aus diesen Ergebnissen lässt sich ableiten, dass $d_{tx} = 3200\,\text{ms}$ für das fortlaufende, $d_{tx} = 1200\,\text{ms}$ für das gestaffelte und $d_{tx} = 900\,\text{ms}$ für das zufällige Senden geeignete Werte sind. d_{rx} wurde mittels Simulationen ermittelt und bei allen Verfahren auf 400 ms gesetzt.



(a) Dauer

(b) Anzahl gesendeter Nachrichten

Abbildung 5.9: Einsammeln von IDs im Simulator

Abbildung 5.9 zeigt die Median- und Extremwerte der Dauer, bis alle IDs bei der Senke eingetroffen waren, sowie der Anzahl gesendeter Nachrichten für verschiedene Anteile selektierter Knoten. Alle IDs konnten binnen weniger Sekunden zur Senke gesendet werden. Beim zufälligen Senden steigen die Dauer und die Anzahl gesendeter Nachrichten allerdings am stärksten für einen wachsenden Anteil an selektierten Knoten, da es auch zu mehr Kollisionen kommt, wenn mehr Knoten zu einem zufälligen Zeitpunkt senden.

Beim fortlaufenden Senden werden erwartungsgemäß am wenigsten Nachrichten gesendet, und das Verfahren braucht nur etwa 1 Sekunde länger als das gestaffelte Senden. Allerdings basiert das fortlaufende Senden darauf, dass die Baumtiefe a priori bekannt ist oder abgeschätzt werden kann. Ist dies nicht der Fall, muss die Tiefe (beispielsweise über eine Maximumsbildung entlang des Baumes) erst ermittelt und anschließend wieder ins Netz geflutet werden, was ein großer Nachteil des Verfahrens ist.

Bei allen drei Vorgehensweisen dauert es über mehrere Simulationsläufe betrachtet sehr unterschiedlich lange, bis die letzten IDs bei der Senke eingetroffen sind, da dieselbe ID trotz mehrmaliger Anfrage fehlen kann. Dies führt zu den vergleichsweise großen Maximalwerten. Abhilfe würde hier schaffen, nach den letzten IDs konkret zu fragen, also eine Anfrage-Nachricht zu fluten, die die gesuchten IDs enthält. Dies ist aber nur möglich, falls die gesuchten IDs bereits bekannt sind.

Bei den Simulationen mit einer Nachrichtenverlustrate von 30 % brauchten alle drei Verfahren bei 5 % selektierten Knoten im Mittel genauso lange wie bei 10 % Nachrichtenverlust. Bei 100 % selektierten Knoten dauerte es allerdings etwa 1,5 s länger. Das Maximum lag bei allen drei Verfahren bei 10 s.

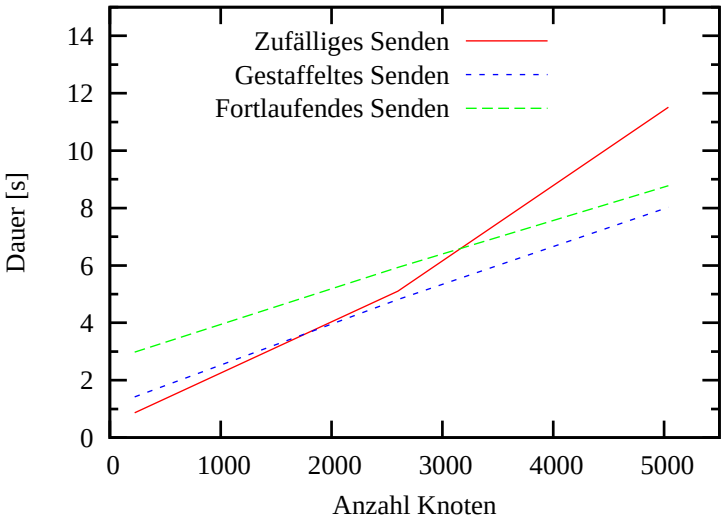


Abbildung 5.10: Dauer des Einsammelns von IDs

Abbildung 5.10 zeigt Ergebnisse aus Simulationen mit Simulationsflächen unterschiedlicher Größe: $285x285m^2$, $1000x1000m^2$ und $1400x1400m^2$. In diesen waren 225, 2600 bzw. 5040 Knoten mit einem Abstand von 20 m verteilt und die Netze wiesen einen Durchmesser von 15, 50 und 70 Hops auf. Für alle drei Strategien ist nur der schwierigste Fall (100 % selektierte Knoten) dargestellt. Während auf der Abszisse die Anzahl der Knoten aufgetragen ist, ist auf der Ordinate die Dauer des Einsammelns dargestellt. Je mehr IDs eingesammelt werden sollen, desto länger dauert es. Beim zufälligen Senden verläuft die Kurve am steilsten, aber immernoch linear, so dass die IDs von 5000 Knoten in weniger als 12 s eingesammelt werden.

Ermitteln der Knoten – Evaluation in einer Testumgebung

Diese Phase wurde auch in der FRONTS-Testumgebung evaluiert. Auch hier fungierte ein Knoten als Senke, die Anfrage-Nachrichten versendete und die IDs von selektierten Knoten sammelte. Die Knoten wurden ebenfalls basierend auf einer gleichverteilten Zufallsvariablen selektiert und das Experiment wurde 50 Mal durchgeführt. Aufgezeichnet wurden die Dauer, bis alle geforderten IDs an der Senke eingetroffen waren, sowie die Anzahl der bis zu diesem Zeitpunkt gesendeten Nachrichten.

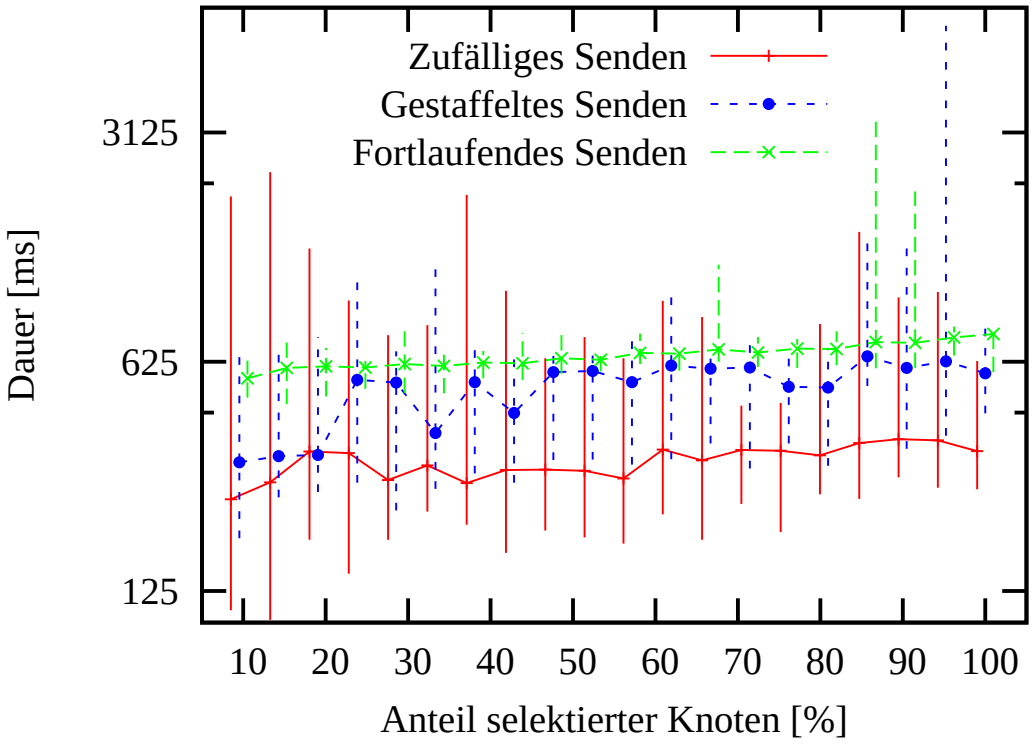


Abbildung 5.11: Dauer des Einsammelns von IDs in der Testumgebung

Während Abbildung 5.11 die Zeitspannen in Millisekunden zeigt, ist in Abbildung 5.12 die Anzahl gesendeter Nachrichten eingetragen. Gerade in einem so kleinen Szenario führt das zufällige Senden im Mittel zu den schnellsten Ergebnissen. Allerdings sind die Maximalwerte hier deutlich höher als bei den beiden anderen Verfahren, da mehr Nachrichten gesendet werden, so dass es auch häufiger zu Kollisionen kommt.

Die Ergebniswerte des gestaffelten Sendens liegen zwischen den Werten der beiden anderen Verfahren. Allerdings ist die Varianz hier vergleichsweise groß, so dass auch die Medianwerte für verschiedene Anteile selektierter Knoten recht unterschiedlich ausfallen. Die Ursache hierfür liegt darin, dass die Leistungsfähigkeit des Verfahrens in so einem kleinen Szenario von der Baumtiefe abhängt. Jede Gruppe von Knoten, die durch die hop-basierte Distanz zur Senke bestimmt wird, hat hier einen gemeinsamen Sendeslot. Insgesamt gibt es aber nur vier Slots, so dass Knoten der Distanz 1, 5 und 9 beispielsweise gleichzeitig senden. Ist der entstandene Baum weniger als vier Hops tief, dauert es im besten Fall nur vier Slotlängen, bis alle IDs an der Senke eingetroffen sind. Ist der Baum tiefer, muss es zwangsweise mindestens doppelt so lange dauern.

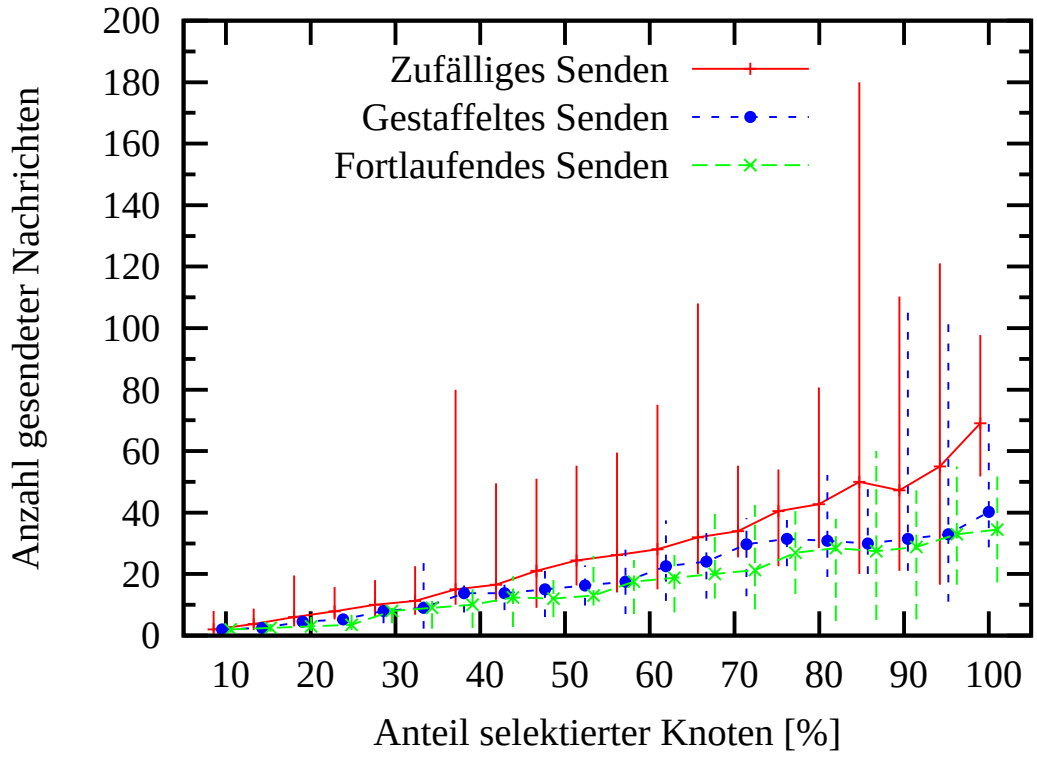


Abbildung 5.12: Anzahl gesendeter Nachrichten in der Testumgebung

Extrem hohe Werte können so auch beim fortlaufenden Senden entstehen, dadurch dass der Baum tiefer ist als die maximal angenommene Tiefe von 6. Würde man eine maximale Tiefe von 4 annehmen, würde das Verfahren exakt dem gestaffelten Senden entsprechen.

Aufgrund der vielen Kollisionen beim zufälligen Vorgehen sollte ein geordnetes Senden eingesetzt werden. Können alle einzusammelnden Informationen in einer Nachricht aggregiert und der Netzdurchmesser abgeschätzt werden, benötigt das fortlaufende Verfahren die wenigsten Nachrichten und liefert alle IDs sehr zuverlässig in einer bestimmten Zeit. In extrem großen Netzen und in dem Fall, dass der Netzwerkdurchmesser nicht bekannt ist, ist das gestaffelte Senden am geeignetsten. Es realisiert einen guten Kompromiss zwischen Nachrichtenaufkommen und Dauer.

5.4.2 Verteilen des Programms

Um eine Grundvorstellung davon zu bekommen, wie viele Nachrichten bei wie vielen Knoten ankommen, wenn sehr viele Nachrichten nacheinander geflutet werden, wurden zunächst Experimente in der Testumgebung durchgeführt. Ein Knoten flutete hierbei

1000 Nachrichten mit unterschiedlichen Raten. Auch die Weiterleitungswahrscheinlichkeit P wurde in den Experimenten variiert; sie war innerhalb eines Experiments jedoch bei allen Knoten immer gleich. Alle Knoten zeichneten auf, welche Nachrichten sie empfingen, so dass die Auslieferungsrate der 1000 Nachrichten ermittelt werden konnte.

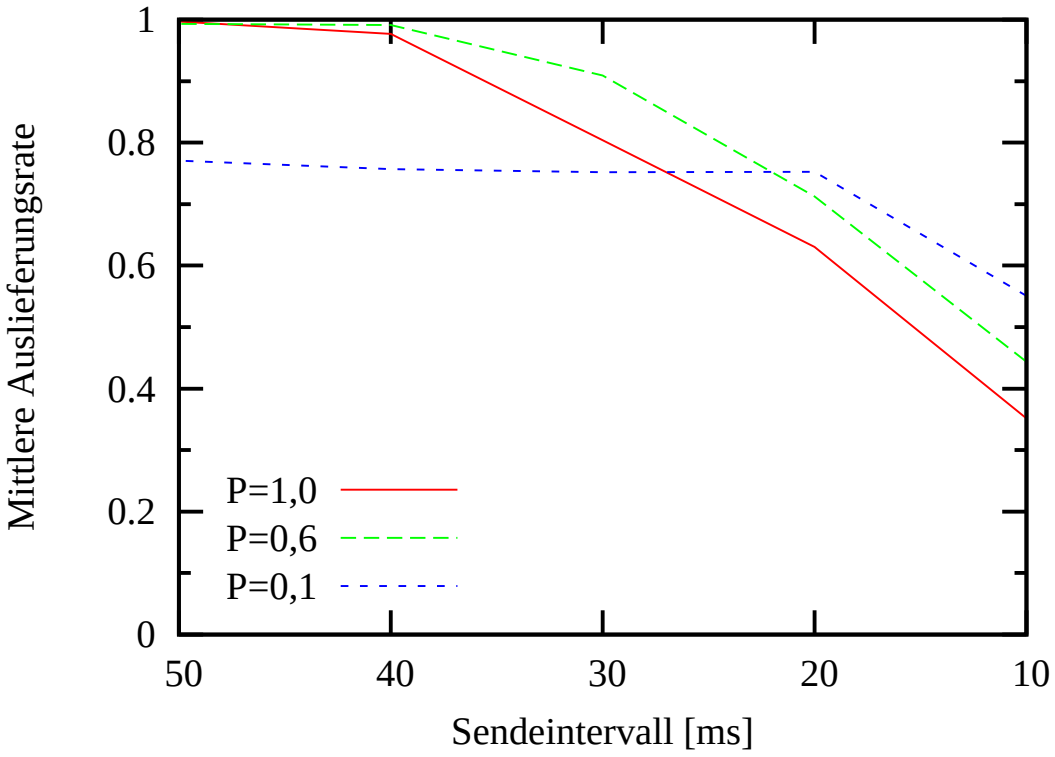


Abbildung 5.13: Auslieferungsrate beim Fluten von 1000 Nachrichten in der Testumgebung

Abbildung 5.13 zeigt die über 50 Durchläufe gemittelten Ergebnisse für die Weiterleitungswahrscheinlichkeiten 10 %, 60 %, und 100 %. Zunächst fällt auf, dass die Gesamtauslieferungsrate bei einer Weiterleitungswahrscheinlichkeit von 10 % kleiner als 80 % und somit nicht akzeptabel ist. Es ist aber auch zu erkennen, dass für eine Weiterleitungswahrscheinlichkeit von 60 % keine höhere Senderate als bei 100 % verwendet werden kann, ohne dass die Auslieferungsrate merklich sinkt. Das gleiche gilt für Raten zwischen den beiden Werten, so dass eine geringere Wahrscheinlichkeit als 100 % in diesem Szenario nicht sinnvoll ist.

In der Testumgebung dauerte es bei einer Weiterleitungswahrscheinlichkeit von 100 % bzw. 10 % nur 33 ms bzw. 12 ms, bis keine Nachricht mehr im Netz gesendet wurde. Da die Testumgebung jedoch ein recht kleines Szenario darstellt, die Latenz einer gefluteten

Nachricht aber von der Dichte und von der Netzgröße abhängt, wurde der Test auch in Shawn durchgeführt. Hierbei wurde das am Beginn dieses Abschnitts beschriebene Szenario mit 225, 400 bzw. 900 Knoten verwendet. Die Senke befand sich in einer Ecke des Quadrats. Als Kommunikationsmodell wurde das stochastische Modell mit maximalen Paketankunftsraten von 98 % und 70 % verwendet. Die Verlustrate p_l betrug dementsprechend 2 % und 30 %.

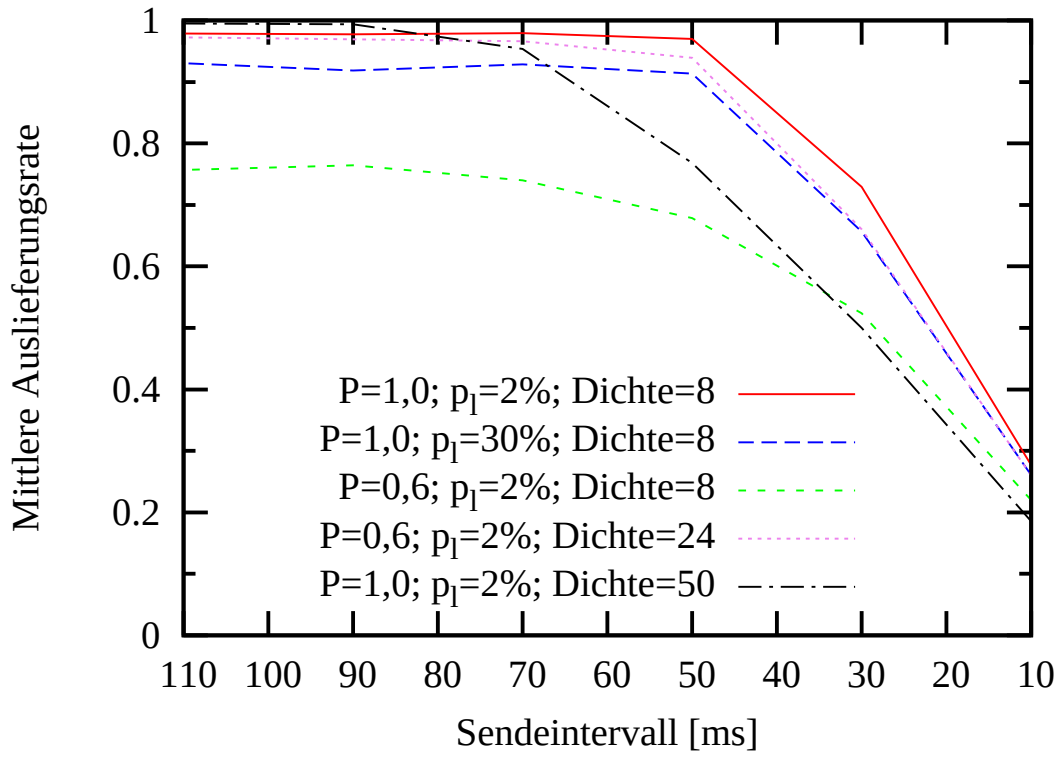


Abbildung 5.14: Auslieferungsrate beim Fluten von 1000 Nachrichten in Shawn

Abbildung 5.14 zeigt die Auslieferungsrate über verschiedene Senderaten der 1000 Nachrichten. Zu erkennen ist, dass bei einer Dichte von 8 Nachbarn, einer Weiterleitungswahrscheinlichkeit von 100 % und einer Nachrichtenverlustrate von 2 % die Auslieferungsrate bereits ab 50 ms pro Nachricht bei 98 % liegt und kaum noch steigt.

Leiten die Knoten nun nicht jede Nachricht weiter, sondern nur 6 von 10 Nachrichten, wird im Maximum nur noch eine Auslieferungsrate von 75 % erreicht. Dies ist also absolut ineffizient. Eine Nachrichtenverlustrate von 30 % beeinflusst das Ergebnis dagegen kaum. Hier werden immer noch 93 % ausgeliefert.

Nur mit einer höheren Dichte (von 24) werden bei einer Weiterleitungswahrscheinlichkeit von 60 % immerhin 97 % ausgeliefert. Trotzdem ergibt sich hier kein Vorteil, da eine solch

hohe Auslieferungsrate ebenfalls erst ab 50 ms pro Nachricht erreicht wird. Es kann also nicht schneller gesendet werden als bei 100 % Weiterleitungswahrscheinlichkeit.

Selbst bei einer extrem hohen Dichte von 50 Nachbarn wird eine vergleichbare Auslieferungsrate trotz 100 % Weiterleitungswahrscheinlichkeit immerhin bei 70 ms pro Nachricht erreicht. Dieser Unterschied ist in Relation zum Unterschied in der Dichte gering.

Zusammenfassend lässt sich feststellen, dass niedrigere Weiterleitungswahrscheinlichkeiten als 100% keine höhere Senderate erlauben, aber bei geringen Dichten sowie höheren Nachrichtenverlustraten zu Problemen führen. Die Simulationsergebnisse bestätigen die Ergebnisse aus der Testumgebung und im Weiteren wurde eine Weiterleitungswahrscheinlichkeit von 100 % verwendet. Große Datenmengen wurden mit 70 ms pro Nachricht verteilt.

#Knoten, denen ein Paket fehlt	Sim. Verlustrate 50% 399 Knoten	Sim. Verlustrate 30% 399 Knoten	Sim. Verlustrate 10% 399 Knoten	Testumgebung 2-hop 21 Knoten
0	18,7	75,1	94,4	99,8
1	34,3	20,9	5,6	0,13
2	23,9	3,5	0	0,02
3	13,9	0,2	0	0
4	4,2	0,2	0	0
5	2,1	0	0	0
6	0,4	0	0	0
7	0,1	0	0	0
Alle: 399 bzw. 21	2,4	0,1	0	0,04

Tabelle 5.2: Histogramm: Anteil fehlender Nachrichten über die Anzahl von Knoten (1. Spalte)

Als nächstes wurde untersucht, wie häufig jede einzelne Nachricht fehlte. Die letzte Spalte von Tabelle 5.2 zeigt, wie viele der 1000 Nachrichten bei wie vielen Knoten in der Testumgebung nach dem Fluten fehlten. Etwa 99,8 % der Nachrichten kamen bei allen Knoten an, während 0,13 % bei genau einem Knoten fehlten, 0,02 % bei zwei und 0,04 % bei allen Knoten.

Zusätzlich zu den Ergebnissen aus den Experimenten zeigt Tabelle 5.2 Simulationsergebnisse, die in einem größeren Netz mit 400 Knoten (1 Gateway, 399 potentielle Empfänger) entstanden sind. Die maximale Paketankunftsrate wurde hierbei auf 90 %, 70 % und 50 % gesetzt. Der Tabelle ist zu entnehmen, dass – unabhängig von der Paketverlustrate – die meisten Nachrichten bei nur einem Knoten fehlten. Einige fehlten auch bei zwei

Knoten und bei sehr hohen Verlustraten bei mehr als zwei Knoten. Aber selbst wenn jede zweite Übertragung fehlschlug, fehlte keine Nachricht bei mehr als sieben Knoten, falls sie überhaupt von einem Knoten empfangen wurde. Weiterhin ist zu erkennen, dass sehr wenige Nachrichten bei allen Knoten fehlten, was beim Design des Programmierverfahrens beachtet wurde.

5.4.3 Verteilen und Nachliefern von Segmenten

Für die Evaluation wurde ein Beispielprogramm von etwa 100 KByte angenommen, das in 1200 Nachrichten aufgeteilt werden musste. Da höchstens 800 Zustände von Segmenten in einer Nachricht enthalten sein konnten, mussten jeweils 2 Nachrichten in ein Segment gruppiert werden. So entstanden 600 Segmente. Verglichen wurde das vorgestellte zweistufige Verfahren (jeweils gekennzeichnet mit *lokal* oder *N-Hop*) mit dem zentralen Ansatz (*zentral* oder *Senke*), bei dem nach dem Verteilen der Segmente fehlende Teile ausschließlich an der Senke angefordert werden.

Gemessen wurde sowohl, wie lange es dauerte, bis alle selektierten Knoten alle Segmente erhalten hatten, als auch die Anzahl der gesendeten Nachrichten. Weiterhin wurde gezählt, wie viele Knoten wach waren. Einerseits wurden die bereits beschriebenen Eingabeparameter variiert:

- Simulationsfläche: $285x285m^2$, $1000x1000m^2$ und $1400x1400m^2$,
- Dichte ($d = 8, 24, 50$),
- Nachrichtenverlust ($p_l = 10\,\%$, $30\,\%$),
- Verteilte Struktur, Anteil selektierter Knoten ($p_{sel} = 5\,\%$ bis $100\,\%$),
- Eck-Struktur ($p_{sel} \approx 6\,\%$).

Andererseits gibt es Parameter, die ausschließlich in dieser Phase relevant sind. Sie wurden wie folgt gesetzt:

- lokales Anfragen: Die Anzahl der Bits pro Segment, um zu zählen, wie vielen Nachbarn das Segment fehlt ($B = 1, 2, 8$),
- lokales Anfragen: Verschiedene Werte für N beim Anfragen in der N-Hop-Nachbarschaft ($N = 1, 2, 3, 4, 5$),
- lokales Anfragen: Nicht selektierte Knoten können einen Anteil p_{store} der Segmente im Speicher vorhalten ($p_{store} = 0\,\%$, $1\,\%$, $3\,\%$, $5\,\%$, $10\,\%$),
- beide Strategien: Auswahl der Weiterleitungsknoten (Alle, Baum, Redundanter Baum)
- beide Strategien: Eck-Struktur ($p_{sel} \approx 6\,\%$).

Verteilen und Nachliefern (Simulation) – Einfluss der Anzahl der Bits

Um die Wirkung der Anzahl der Bits B, die ausschließlich während der Phase des Nachforderns zum Tragen kommt, vom Einfluss von Nachrichtenverlusten während des Verteilens zu extrahieren, wurde hier nur die Phase des Nachforderns simuliert. Es wurde dabei davon ausgegangen, dass jedem selektierten Knoten 10 % zufällig ausgewählte Segmente fehlten und allen selektierten Knoten dieselben 1 % fehlten. Da B vor allem entscheidend ist, wenn Knoten viele selektierte Nachbarn haben, die sie potentiell zählen müssen, waren bei diesen Simulationen 100 % der Knoten selektiert bei einer Dichte von 24.

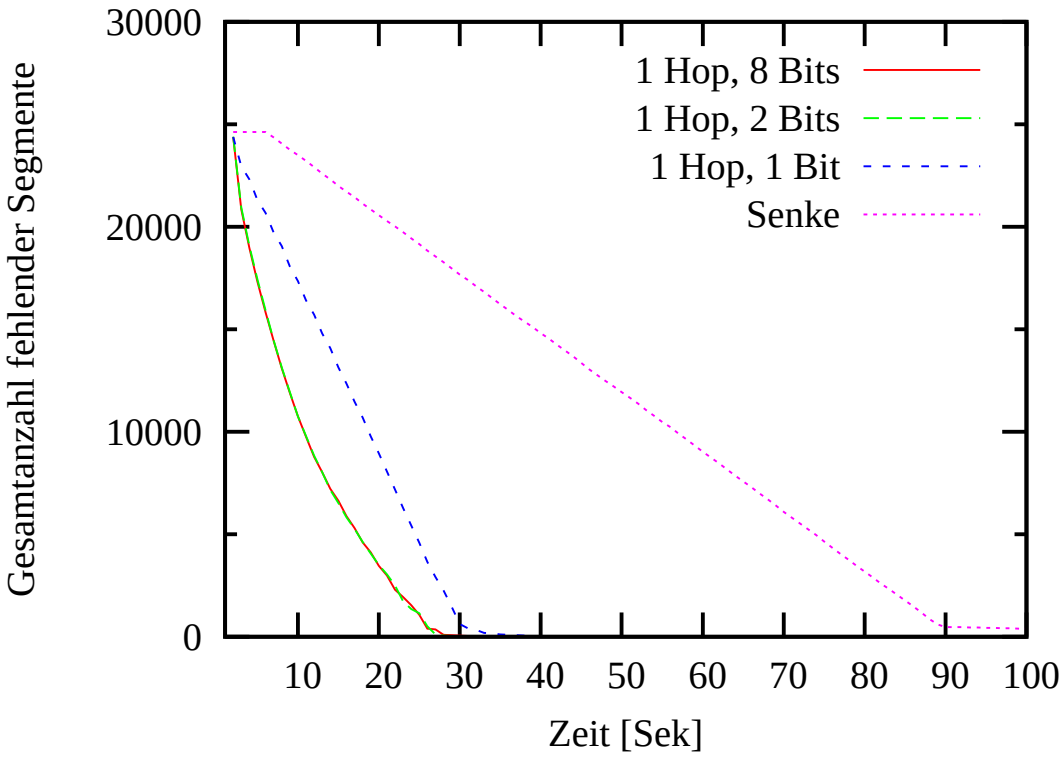


Abbildung 5.15: Einfluss der Anzahl der Bits beim lokalen Anfragen

Abbildung 5.15 zeigt den zeitlichen Verlauf der Anzahl fehlender Segmente, aufsummiert über alle selektierten Knoten für beide Ansätze (lokal und zentral) sowie beim lokalen Verfahren für verschiedene Werte für die Anzahl der Bits B. In diesem Szenario braucht das zentrale Anfragen dreimal so lange wie das lokale Vorgehen. Beim lokalen Anfragen lässt sich feststellen, dass die Kurven für $B = 2$ und $B = 8$ nahezu übereinander liegen, was darauf schließen lässt, dass ein Segment bei höchstens zwei Nachbarn fehlt. Für $B = 1$ sinkt die Zahl der fehlenden Segmente linear, da hier nicht mehr die Segmente

zuerst gesendet werden, die bei den meisten Knoten fehlen, sondern zufällig ausgewählte fehlende Segmente. Die längere Dauer von 6 s für $B = 1$ ist vernachlässigbar, wenn man bedenkt, dass das Verteilen der Daten mehr als 60 s dauert und hier im Vergleich zu $B = 2\,600$ Bits (= 75 Bytes) an Speicherplatz gespart werden.

Hieraus lässt sich ableiten, dass ein Bit genügt, um eine gute Leistungsfähigkeit des lokalen Anfragens zu erzielen. In allen weiteren Simulationen wurde deshalb $B = 1$ gesetzt. Außerdem wurde beides simuliert: das Verteilen der Segmente sowie das Nachliefern.

Verteilen und Nachliefern (Simulation) – Einfluss von N

Da die Knoten nur bei der verteilten Struktur potentiell wenig selektierte Nachbarn haben (bei der Eck-Struktur sind alle selektierten Knoten nah beieinander), wurde anhand dieser Struktur untersucht, ob es lohnt, Anfragen nach fehlenden Segmenten und Segmentnachrichten weiterzuleiten, anstatt nur mit den direkten Nachbarn zu kommunizieren. Die Dichte wurde entsprechend auf einen geringen Wert von 8 gesetzt. Der Anteil der selektierten Knoten im verteilten Szenario wurde auf Werte zwischen 5 % und 100 % gesetzt.

Während Abbildung 5.16 die Zeit in Sekunden enthält, die benötigt wurde, bis alle selektierten Knoten das gesamte Programm erhalten hatten, ist in Abbildung 5.17 die Anzahl gesendeter Nachrichten über unterschiedliche Anteile selektierter Knoten für die verschiedenen Werte von N aufgetragen.

Zu erkennen ist, dass $N = 2$ bis zu einem Anteil von 60 % an selektierten Knoten alle Segmente am schnellsten ausliefert. Die recht niedrige Dichte von 8 führt zu diesem Ergebnis, da hier Knoten für $N = 1$ wenig selektierte Nachbarn haben, so dass viele fehlende Segmente von der Senke angefordert werden müssen. Je größer die Dichte desto besser sind auch die Ergebnisse für $N = 1$, auch wenn der Anteil selektierter Knoten gering ist. Der Abbildung ist aber auch zu entnehmen, dass größere Werte für N keine weitere Effizienzsteigerung erzielen.

Für $N = 1$ sinkt die Dauer mit steigendem Anteil selektierter Knoten linear, während alle anderen Kurven leicht ansteigen. Sind mehr Knoten als 63 % selektiert, führt der Wert 1 für N bereits bei dieser geringen Dichte zu den besten Ergebnissen. Im ungünstigsten Fall (für 20 % selektierte Knoten) dauert es bei $N = 1$ verglichen mit $N = 2$ im Mittel auch nur 24 s länger (147 s statt 123 s) bis alle selektierten Knoten alle Segmente haben. Da der Prozess für $N = 2$ aber im schlechtesten Fall (100 % selektierte Knoten) auch 21 s länger dauert als für $N = 1$, statt 360000 sogar 620000 Nachrichten versendet werden müssen und $N > 1$ außerdem erfordert, dass zusätzliche Weiterleitungsknoten während des Programmiervorgangs wach sind, ist nur $N = 1$ sinnvoll, so dass in allen weiteren Untersuchungen $N = 1$ gesetzt wurde.

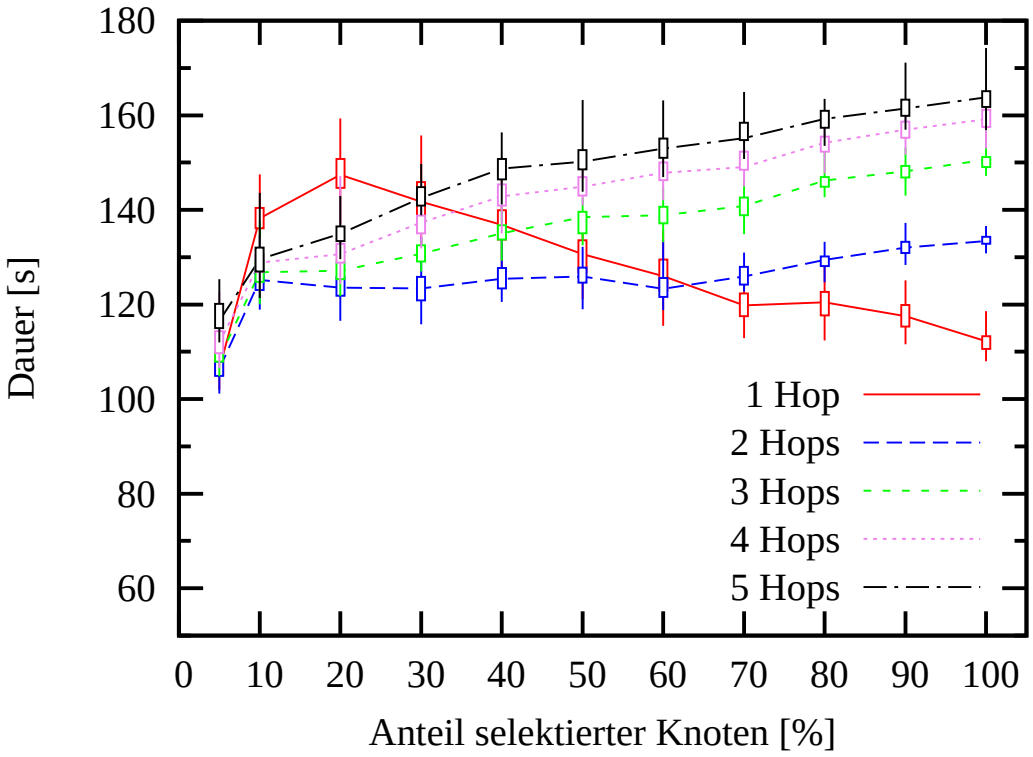


Abbildung 5.16: Dauer

Verteilen und Nachliefern (Simulation) – Einfluss nicht selektierter Knoten

Vorstellbar ist, dass nicht selektierte Knoten einige Segmente in ihrem Speicher vorhalten und selektierten Nachbarn liefern können. Sind sie jedoch nicht aufgrund ihrer Rolle als Weiterleitungsknoten wach, müssen sie zusätzlich wach bleiben, so dass der Energieaufwand steigt. Da dieses Vorgehen nur beim $N - Hop$ Anfragen zum Tragen kommt, wurde es nur für diesen Ansatz, aber für verschiedene N evaluiert.

Abbildung 5.18 zeigt, wie lange das Ausliefern von Segmenten dauert, falls nicht selektierte Knoten $p_{store} = 0\,\%$ bis $10\,\%$ der Segmente speichern können. Der Anteil der selektierten Knoten beträgt hier $5\,\%$ bei einer Dichte von 8, da diese Fähigkeit besonders bei geringen Anteilen an selektierten Knoten positiv wirkt. Zunächst bestätigt sich auch hier, dass es sich nicht lohnt, $N > 1$ zu setzen, denn alle Kurven sind monoton steigend.

Die Ergebnisse zeigen aber auch, dass die Fähigkeit von nicht selektierten Knoten, Segmente zu speichern, die Dauer im Vergleich zu $p_{store} = 0\,\%$ im Mittel nur wenig verkürzt. Für $N = 1$ wird sie nur um $1{,}8\,\%$ bis $6{,}6\,\%$ (2 s für $p_{store} = 0\,\%$ bis 7 s

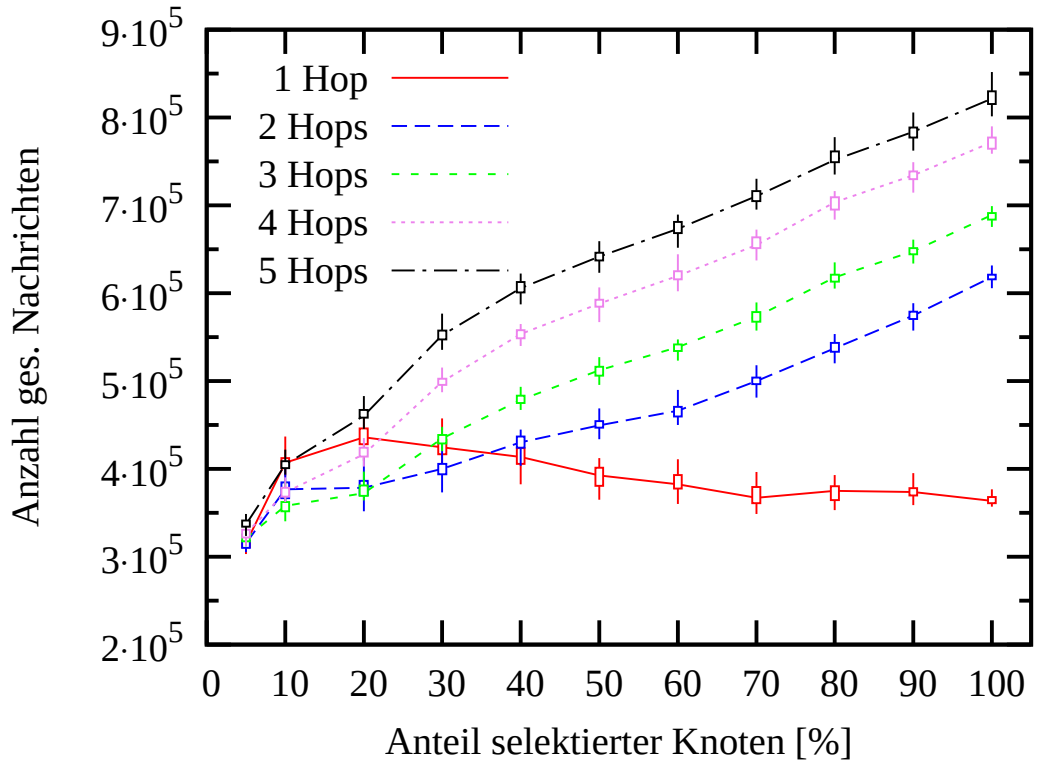


Abbildung 5.17: Anzahl gesendeter Nachrichten

für $p_{store} = 10\,\%$) verkürzt. Hinzukommt, dass diese Knoten zusätzlich während des Reprogrammierens wach bleiben müssen und somit Energie verbrauchen.

Aus diesen Ergebnissen lässt sich ableiten, dass es nur dann sinnvoll ist, nicht selektierte Knoten an diesen Phasen zu beteiligen, wenn diese in der Lage sind, einen erheblichen Teil des Programms (in der Größenordnung von ca. 50 %) im Speicher zu halten. Anderenfalls ist der Zeitgewinn vernachlässigbar und legitimiert den zusätzlichen Energieaufwand nicht. Außerdem ist festzuhalten, dass bereits 5 % des Programms (was im untersuchten Fall 5 KByte entspricht) eine Datenmenge darstellt, die üblicherweise nicht vorgehalten werden kann.

Verteilen und Nachliefern (Simulation) – Einfluss der Auswahl der Weiterleitungsknoten

Flutet die Senke Nachrichten, ist die Anzahl und Anordnung der beteiligten Knoten, die die Nachricht weiterleiten, von entscheidender Bedeutung für die erfolgreiche Auslieferung

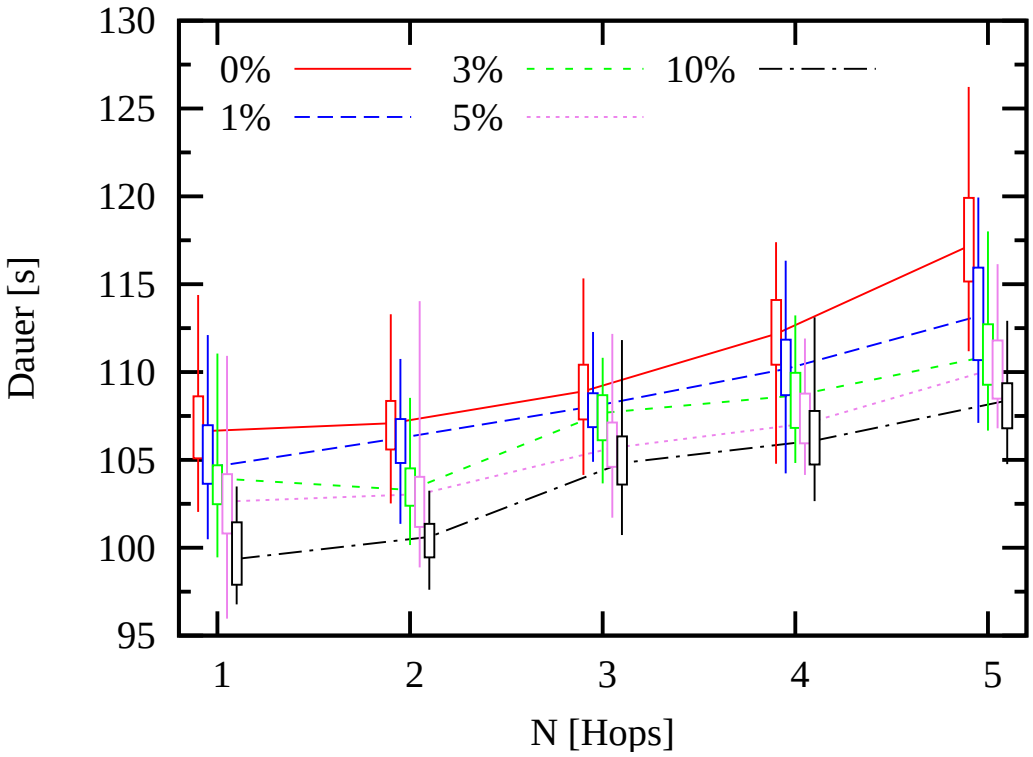


Abbildung 5.18: Einfluss nicht selektierter Knoten

und für die Anzahl an gesendeten Nachrichten. Aus diesem Grund wurden die drei Strategien 'Alle', 'Baum' und 'Redundanter Baum' miteinander verglichen.

Abbildung 5.19 stellt für die drei Strategien zunächst dar, wieviel Prozent der Knoten für die Dauer des Reprogrammierens wach sein müssen, also beteiligt sind. Für die recht geringe Dichte von 8 steigt der Anteil der beteiligten Knoten für die 'Baum'-Strategie logarithmisch von 20 % auf 100 %. Beim 'Redundanten Baum' sind 50 % bis 100 % der Knoten beteiligt, bei 'Alle' logischerweise immer 100 %. Je dichter das Netz ist, desto vorteilhafter wirkt sich die 'Baum'-Strategie im Vergleich mit den beiden anderen Strategien auf den Gesamtenergieverbrauch des Netzes aus, da der Anteil an Knoten, der den Baum bildet, mit wachsender Dichte sinkt, nicht aber der Anteil der Knoten, der den 'Redundanten Baum' bildet.

Abbildung 5.20(a) bzw. Abbildung 5.20(c) zeigen die Zeit, die beim zentralen bzw. lokalen Verfahren vergangen war, bis alle selektierten Knoten das gesamte Programm erhalten hatten, Abbildung 5.20(b) und Abbildung 5.20(d) die Anzahl versendeter Nachrichten.

Zunächst ist zu erkennen, dass die Kurven der drei Verfahren mit steigender Zahl selektierter Knoten konvergieren. Dies ist nicht überraschend, da die Mengen der Weiter-

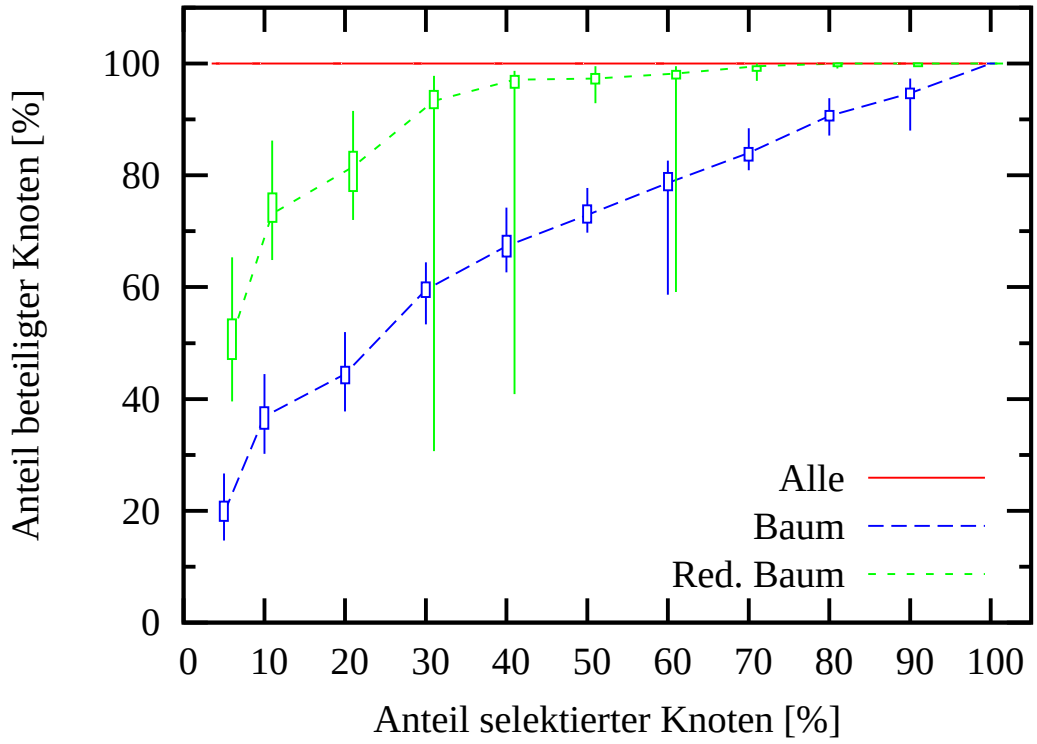
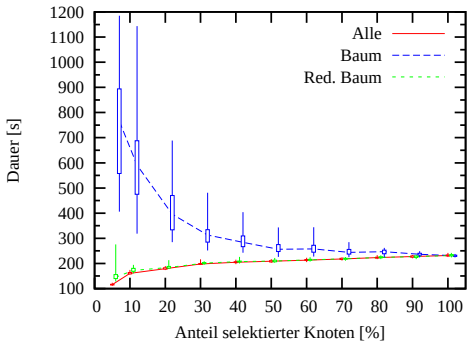


Abbildung 5.19: Verteilte Struktur, prozentualer Anteil beteiligter Knoten

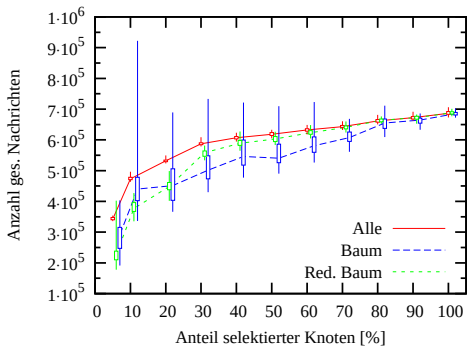
leitungsknoten mit steigendem Anteil selektierter Knoten immer ähnlicher werden, bis sie schließlich für einen Anteil von 100 % selektierter Knoten gleich sind.

Vor allem wird aber auch deutlich, dass die 'Baum'-Strategie bereits bei 10 % Nachrichtenverlusten im Mittel extrem viel Zeit und teilweise auch viele Nachrichten benötigt, wenn weniger als 60 % der Knoten selektiert sind. Beispielsweise dauert es beim lokalen Verfahren und einem Anteil von 5 % selektierter Knoten fast 10 min statt 106 s ('Alle') und 128 s ('Redundanter Baum'). Werden fehlende Segmente ausschließlich an der Senke angefordert (zentrales Verfahren), müssen sogar fast 13 min aufgewendet werden.

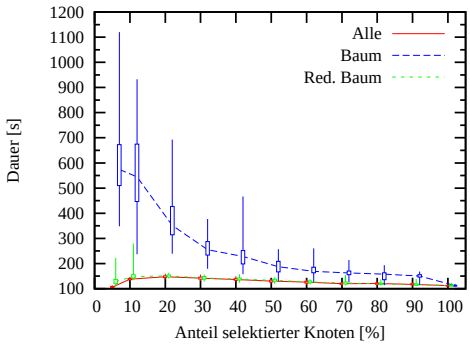
Für geringere Nachrichtenverlustraten als 10 % ist der Unterschied zwischen den drei Strategien weniger ausgeprägt. Für noch höhere Senderaten sowie schwankende Ankunftsraten steigt die Dauer bei 'Baum' jedoch dramatisch. Fällt ein Knoten des Baumes aus, zerfällt das Netz sogar in zwei Partitionen und einige Knoten können gar nicht mehr mit der Senke kommunizieren. Einen 'Redundanten Baum' zu nutzen, scheint deshalb am sinnvollsten. Das Vorgehen ist nahezu genauso schnell wie 'Alle', aber verringert die Anzahl beteiligter Knoten um bis zur Hälfte und die Anzahl gesendeter Nachrichten um bis zu einem Drittel. Aus diesem Grund wurde es in den weiteren Simulationen eingesetzt.



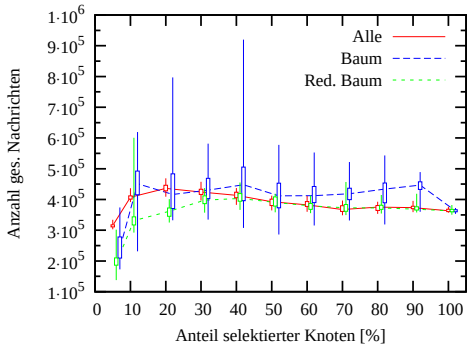
(a) Senke, Dauer



(b) Senke, Anzahl gesendeter Nachrichten



(c) 1 Hop, Dauer



(d) 1 Hop, Anzahl gesendeter Nachrichten

Abbildung 5.20: Verteilte Struktur, Zentrales Anfragen

Verteilen und Nachliefern (Simulation) – Einfluss von Nachrichtenverlusten

Abbildung 5.21 und Abbildung 5.22 stellen den lokalen dem zentralen Ansatz jeweils unter Verwendung des 'Redundanten Baumes' direkt gegenüber. Auch hier sind die Dauer, bis alle Knoten alle Programmteile hatten, sowie die Anzahl gesendeter Nachrichten über verschiedene Anteile selektierter Knoten aufgetragen. Generell lässt sich feststellen, dass das lokale Anfragen zu besseren Ergebnissen führt, als das Anfragen bei der Senke. Je mehr Knoten selektiert sind, desto höher ist der Nutzen des lokalen Ansatzes. Sind beispielsweise 100 % der Knoten selektiert, dauert es hier nur 112 s anstatt 231 s, bis alle Knoten das gesamte Programm erhalten haben, und es werden nur 360.000 anstatt 680.000 Nachrichten dafür versendet. Dieses Vorgehen verringert also nicht nur den Zeitaufwand um bis zu 65 % für das Reprogrammieren, sondern spart auch Energie durch eine geringere Anzahl an Nachrichten (bis zu 47 % weniger).

Man könnte vermuten, dass das lokale Verfahren aufgrund der kürzeren Kommunikationspfade besonders bei höheren Nachrichtenverlustraten vorteilhaft sei, da die Wahrschein-

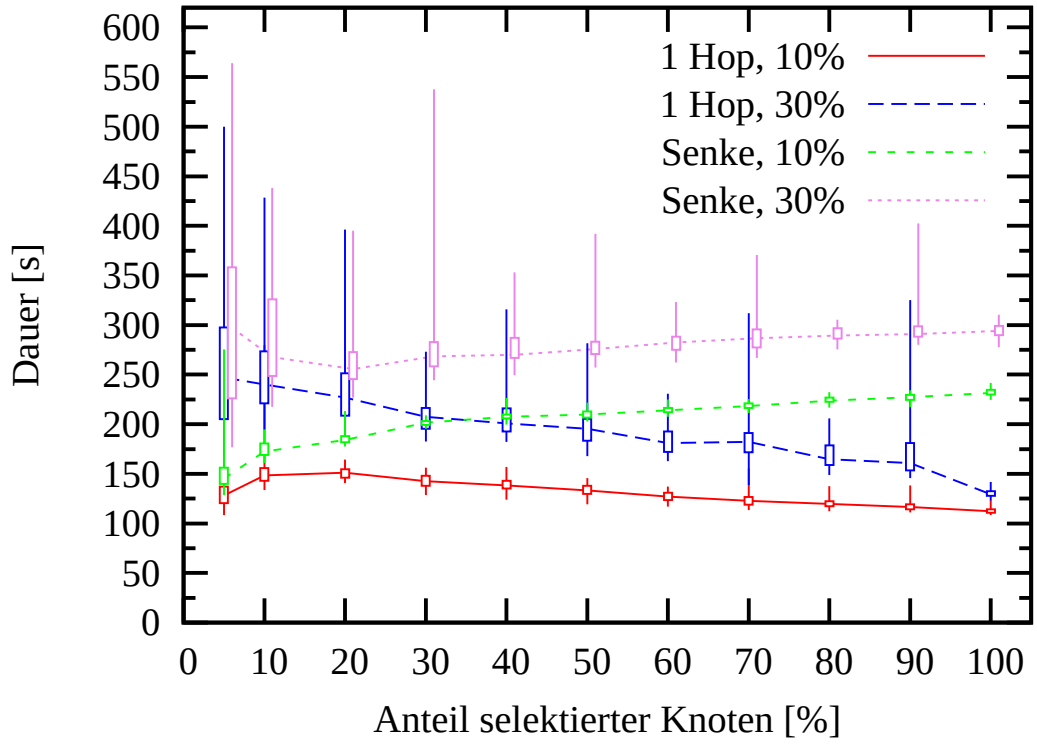


Abbildung 5.21: Dauer

lichkeit für einen erfolgreichen Multihop-Transport mit wachsender Länge des Pfades sinkt. Um diese Vermutung zu überprüfen, wurden Simulationen mit einer Nachrichtenverlustrate von 30 % durchgeführt, deren Ergebnisse ebenfalls in den beiden vorherigen Abbildungen eingezeichnet sind.

Wie erwartet, dauert es bei beiden Verfahren länger, und es werden mehr Nachrichten gesendet, wenn mehr Nachrichten verloren gehen. Beim lokalen Verfahren dauert es bis zu 119 s (92 %) länger. Sind 100 % der Knoten selektiert, verlängert sich der Prozess im Mittel jedoch nur um 18 s (16 %). Beim zentralen Ansatz beträgt die Menge an zusätzlich benötigter Zeit zwischen 152 s und 63 s (105 % bzw. 27 %).

Die prozentuale Leistungseinbuße des zentralen Verfahrens durch die höhere Nachrichtenverlustrate ist also nur geringfügig größer als die des lokalen Verfahrens. Hierbei gilt es zu bedenken, dass lokal immer per Broadcast (also ohne Bestätigung) angefragt und geantwortet wird. Wird an der Senke angefragt, werden die Anfrage-Nachrichten (entlang des Baumes) immer bestätigt und die Nachrichten, die Programmteile enthalten, redundant gesendet, so dass hier in beiden Richtungen eine höhere Zuverlässigkeit in der Nachrichtenübertragung vorhanden ist. Aus diesem Grund ist die Leistungseinbuße des

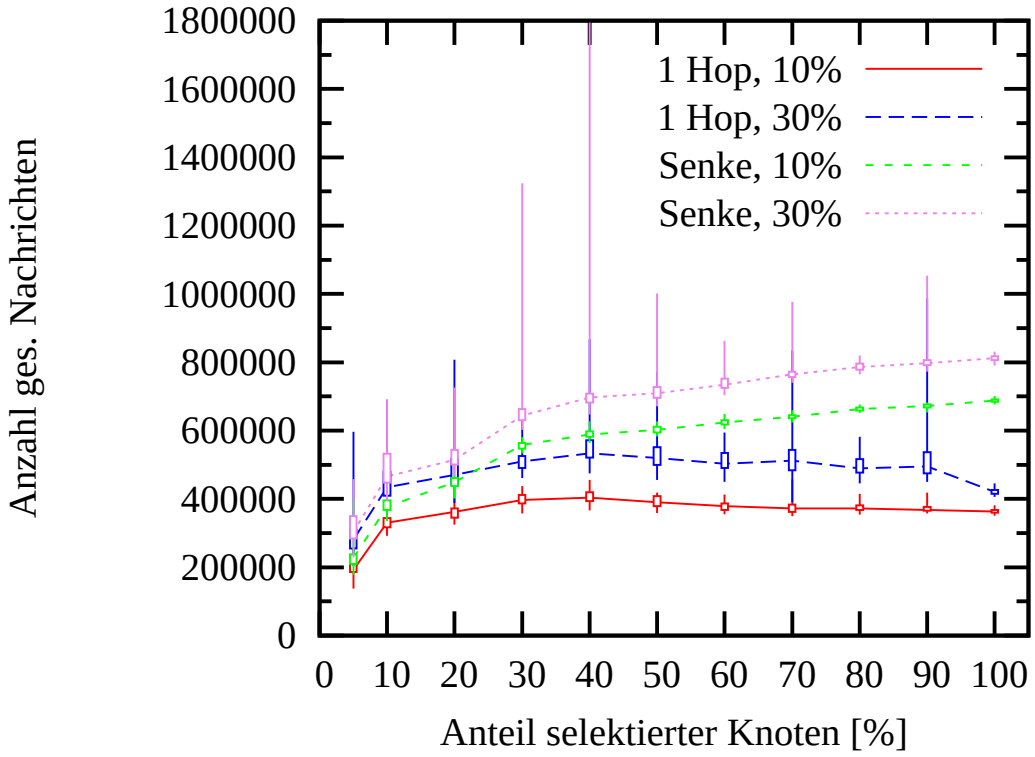


Abbildung 5.22: Anzahl gesendeter Nachrichten

zentralen Anforderns nicht so groß wie zunächst vermutet. Das lokale Verfahren erreicht diese Redundanz nur, wenn viele Knoten selektiert sind und jeder Knoten somit auch viele selektierte Nachbarn hat, die auf seine Anfrage antworten.

Auffällig ist auch, dass die maximalen Werte bei der größeren Nachrichtenverlustrate bei beiden Verfahren stärker schwanken und deutlich weiter von den Medianwerten entfernt sind. Dies ist nicht verwunderlich, denn die höhere Nachrichtenverlustrate verstärkt die Zufallskomponente, die einen erfolgreichen Abschluss des Reprogrammierens verhindert. Insgesamt lässt sich aber feststellen, dass beide Verfahren auch bei 30 % Nachrichtenverlust gut funktionieren.

Verteilen und Nachliefern (Simulation) – Einfluss der Dichte

Simulationen mit höheren Dichten (24 und 50 Nachbarn) führten zu ähnlichen Ergebnissen wie es die vorigen Abschnitte bereits zeigten. Je größer die Anzahl selektierter Knoten, desto länger dauert es beim zentralen Ansatz, bis alle Knoten das gesamte Programm erhalten haben. Das lokale Verfahren profitiert davon, dass Knoten trotz niedriger

Selektionsrate einige selektierte Nachbarn haben. Die Ersparnis bezüglich der Anzahl gesendeter Nachrichten, die durch das lokale Vorgehen erzielt werden kann, steigt mit wachsender Dichte, da das Fluten von Nachrichten durch die Senke eine steigende Anzahl an Nachrichten zur Folge hat.

Verteilen und Nachliefern (Simulation) – Einfluss der Netzgröße

Abbildung 5.23 zeigt wieder Ergebnisse von Simulationen mit den Simulationsflächen der Größe $285x285m^2$, $1000x1000m^2$ und $1400x1400m^2$. Auch hier waren 225, 2600 bzw. 5040 Knoten gleichmäßig über die jeweilige Fläche verteilt und die Netze hatten einen Durchmesser von 15, 50 und 70 Hops.

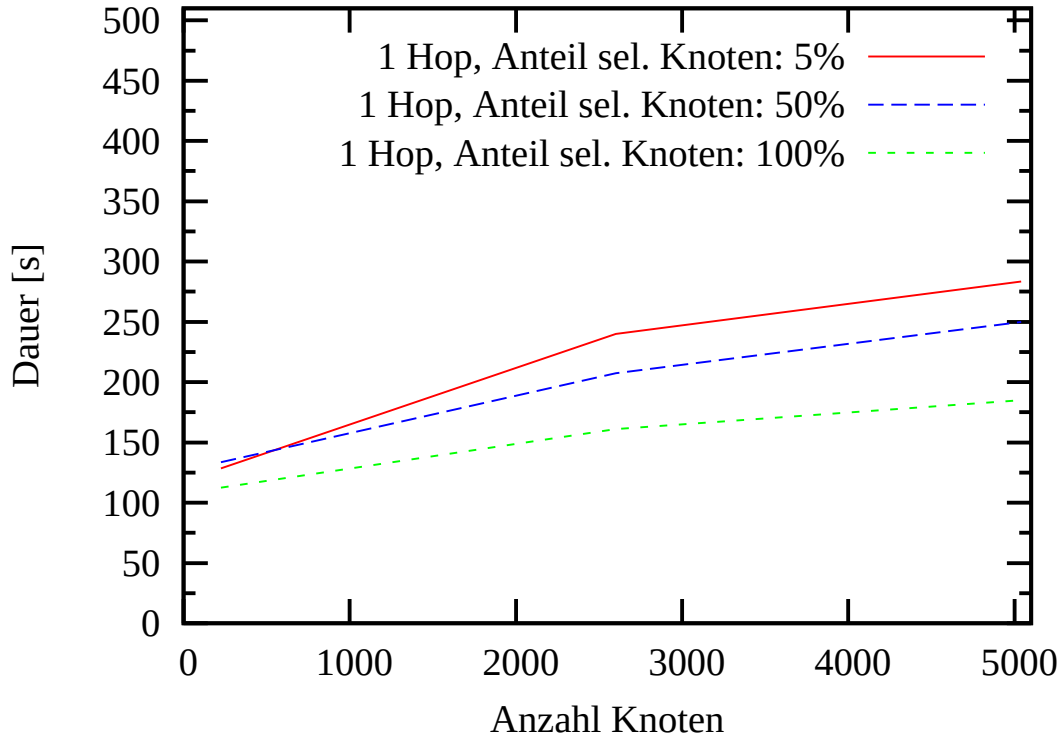


Abbildung 5.23: Dauer

Zur besseren Lesbarkeit sind in der Abbildung nur die Werte für Anteile von 5 %, 50 % und 100 % selektierter Knoten dargestellt. Zu erkennen ist, dass die Dauer bei allen drei Kurven nur langsam linear steigt. Der Algorithmus funktioniert also auch mit vielen Knoten sowie in einem tiefen Netz mit einem Durchmesser von 70 Hops.

Verteilen und Nachliefern (Simulation) – Eck-Struktur

Tabelle 5.3 enthält die minimal und maximal aufgetretenen Ergebniswerte der beiden Verfahren (lokal und zentral) für verschiedene Mengen an Weiterleitungsknoten in der Eck-Struktur. Hier befanden sich die selektierten Knoten räumlich nah beieinander, aber insbesondere weit entfernt von der Senke, was den Nutzen des lokalen Anfragens noch steigerte. Das Verfahren benötigte bei 'Alle' und 'Redundanter Baum' im Vergleich zum zentralen Ansatz 30 % bis 50 % weniger Zeit und versendete nur 60 % der Nachrichten.

$Min - Max$	Alle	Baum	Red. Baum
Dauer 1 Hop	100 – 109	290 – 1060*	100 – 133
Dauer Senke	150 – 166	320 – 1426*	151 – 286
#Nachr. 1 Hop	270.000 – 289.000	120.000 – 252.000*	90.000 – 184.000
#Nachr. Senke	444.000 – 476.000	160.000 – 252.000*	148.000 – 339.000

Tabelle 5.3: Extrema (Minimum - Maximum) bei der Eck-Struktur

Die mit * gekennzeichneten Werte sind zwar die Maximalwerte, aber in etwa 20 % der Fälle hatten sowohl beim lokalen als auch beim zentralen Vorgehen nach 4000 s noch nicht alle Knoten alle Segmente erhalten, so dass das Verfahren als gescheitert angesehen wurde. Die Wirkung der fehlenden Redundanz beim 'Baum'-Verfahren wie es Lee et al. vorschlagen wird also extrem verstärkt.

Gerade bei der Eck-Struktur ist der 'Redundante Baum' als Weiterleitungsknoten am effizientesten. Das lokale Verfahren ist hier in etwa genau so schnell wie 'Alle', benötigt aber nur 60 % der Nachrichten und erlaubt insbesondere, dass 56 % der Knoten während des Programmiervorgangs im Schlafmodus sein können.

Verteilen und Nachliefern – Evaluation in der FRONTS-Testumgebung

Die beiden Verfahren (lokal und zentral) wurden ebenfalls in der FRONTS-Testumgebung erprobt. Hier wurden zwischen 2 und 21 der 22 Knoten selektiert, während ein Knoten als Senke fungierte. Weiterleitungsknoten waren immer alle Knoten, da der 'Redundante Baum' sich in diesem kleinen Szenario, welches etwa 1 Hop breit und 2 bis 3 Hops lang ist, nur in sehr seltenen Fällen von 'Alle' unterscheidet. Gemessen wurden sowohl die Dauer, bis jeder selektierte Knoten das vollständige Programm erhalten hatte, als auch die Anzahl an Nachrichten, die jeder Knoten sendete. 50 Testläufe wurden mit jedem Verfahren durchgeführt, so dass in den Diagrammen die Medianwerte, untere und obere Quartile sowie minimal und maximal aufgetretene Werte abgebildet sind.

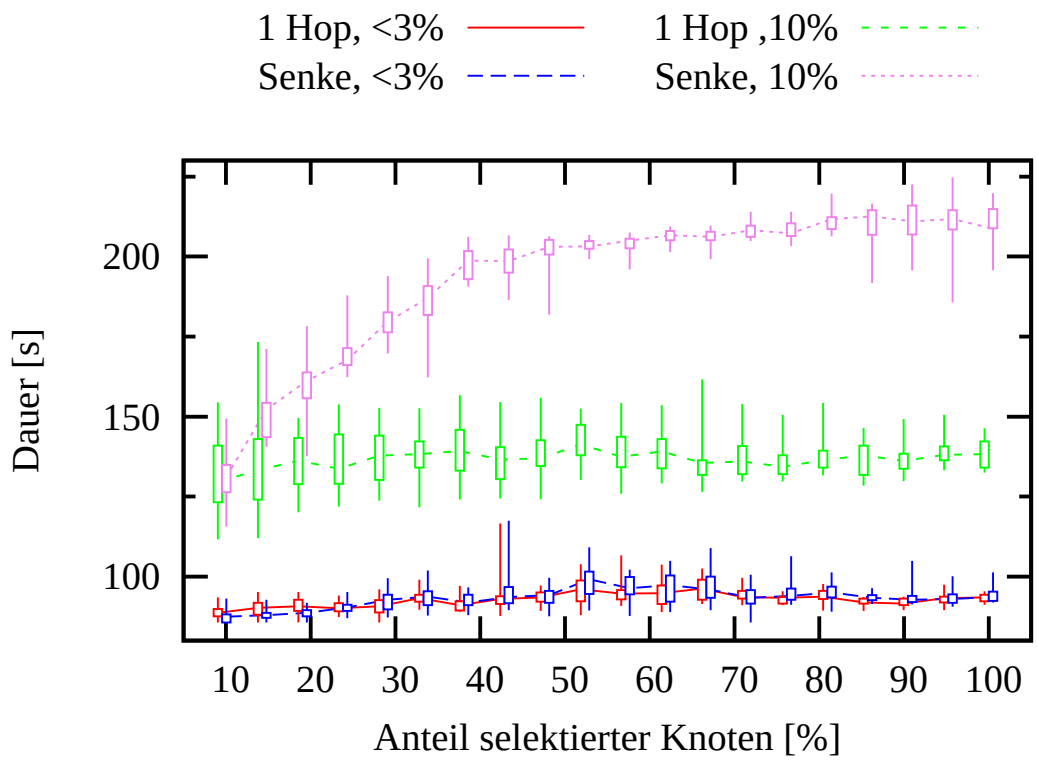


Abbildung 5.24: Verteilte Struktur, Testumgebung, Dauer

Abbildung 5.24 zeigt diese Werte für die Dauer. Die Leistungsfähigkeit beider Verfahren ist hier sehr ähnlich, da aufgrund der geringen Größe des Netzes und der vielen Kommunikationsverbindungen, bei denen mehr als 98 % der Nachrichten ankommen, kaum Segmente nachgefordert werden müssen und die Gesamtdauer maßgeblich durch das Verteilen der Segemente bestimmt wird. Bei beiden Verfahren liegt der Median zwischen 86 s und 95 s.

Gehen mehr Nachrichten verloren, ist das Verhalten prinzipiell ähnlich zu dem in den Simulationen (vgl. 10 % Nachrichtenverlust in Abbildung 5.21). Der lokale Ansatz benötigt ab einem Anteil von 20 % selektierter Knoten deutlich weniger Zeit, um alle Segmente an alle Knoten auszuliefern. Während die Dauer hier mit steigendem Anteil selektierter Knoten kaum steigt (von 130 s auf 138 s), erhöht sich die Dauer beim zentralen Ansatz von 131 s auf 208 s. Im größeren (simulierten) Szenario dauert es, verglichen mit dem kleineren Szenario der Testumgebung, insgesamt länger. Wie jedoch bereits in den Simulationen gezeigt werden konnte, wächst der Vorteil des lokalen Anfragens mit steigendem Anteil selektierter Knoten.

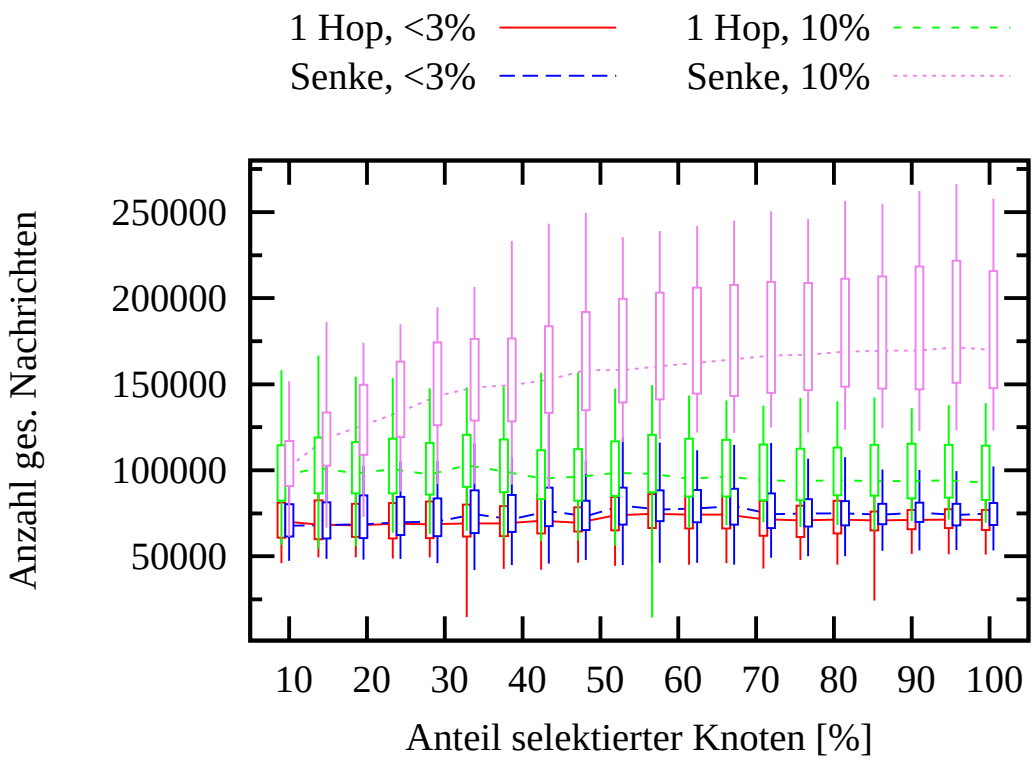


Abbildung 5.25: Verteilte Struktur, Testumgebung, Anzahl gesendeter Nachrichten

Abbildung 5.25 veranschaulicht die Anzahl der gesendeten Nachrichten. Die Kurven verlaufen bei beiden Verfahren wie die Dauer. Während die Kurve des lokalen Anfragens bei einer Nachrichtenverlustrate von 10 % nahezu konstant ist und bei etwa 124.000 Nachrichten liegt, senden die Knoten beim Anfragen an der Senke im besten Fall (9 % selektierte Knoten) 128.000 und im schlechtesten Fall (100 % selektierte Knoten) 203.000 Nachrichten.

Insgesamt lässt sich aus den Ergebnissen ableiten, dass es in verlustbehafteten Netzen immer sinnvoll ist, bei Nachbarknoten fehlende Segmente anzufordern, um den Zeit- und Nachrichtenaufwand gering zu halten.

5.4.4 Abschluss und Neustart

Aufgrund der großen Wichtigkeit, dass selektierte Knoten die Neustart-Nachricht bekommen, muss sie mehrmals geflutet werden. Im simulierten Testszenario mit 225 Knoten reichte bei einer Dichte von 8 und einer Nachrichtenverlustrate von 2 % ein viermaliges Fluten aus, um über 1000 Wiederholungen hinweg alle Knoten zu erreichen. Wurde die

Nachricht nur dreimal geflutet, so empfing in 8 von 1000 Fällen ein Knoten die Nachricht nicht. Bei höheren Dichten und Nachrichtenverlustraten von bis zu 30 % reicht viermaliges Fluten ebenfalls aus.

5.4.5 Zusammenfassung der Ergebnisse

Zunächst wurden drei verschiedene Strategien für einen Aggregationsalgorithmus evaluiert, der Daten von jedem Knoten im Netz zu einer Senke transportiert. Abschnitt 5.4.2 belegt, dass alle drei Varianten im untersuchten Szenario nur wenige Sekunden benötigen. Beim zufälligen Senden werden jedoch die meisten Nachrichten versendet. Das fortlaufende Senden führt zu den besten Ergebnissen, da es die wenigsten Nachrichten versendet. Es setzt allerdings voraus, dass der Netzwerkdurchmesser bekannt ist. Ist dies nicht der Fall, sollte das gestaffelte Senden verwendet werden.

Anschließend wurde untersucht, welche Senderate und welche Weiterleitungswahrscheinlichkeit beim zufallsbasierten Fluten von vielen Nachrichten günstig sind, d. h. eine hohe Auslieferungsrate erreichen. Abschnitt 5.4.1 ist zu entnehmen, dass für Netze bis zu einem Durchmesser von 15 Hops und einer Dichte von 50 maximal alle 70 ms eine Nachricht geflutet werden sollte. Als Weiterleitungswahrscheinlichkeit empfiehlt es sich, 100 % zu verwenden, da die Ankunftsrate andernfalls in dünnen Netzen stark sinkt. Bei noch tieferen oder dichteren Netzen sollte die Senderate voraussichtlich leicht verringert werden.

Es konnte weiterhin festgestellt werden, dass – unabhängig von der Nachrichtenverlustrate – einige Nachrichten von einzelnen Knoten und wenige Nachrichten von allen Knoten nicht empfangen werden. Diese müssen nach dem Verteilen des Programms erneut übertragen werden.

Abschnitt 5.4.3 analysiert die Wirkung verschiedener Parameter beim Anfordern von fehlenden Segmenten. Hinsichtlich des Anforderns von fehlenden Segmenten lassen sich die folgenden Ergebnisse zusammenfassen:

Lokales Anfragen:

- Die Anzahl der Bits pro Segment, um zu zählen, wie viele Nachbarn das Segment suchen: 1 Bit reicht aus; Knoten müssen nicht zählen, wie vielen ihrer Nachbarn das Segment fehlt, sondern müssen nur speichern, ob es überhaupt fehlt.
- Die Knoten fluten ihren Zustand und Segmente in ihre N-Hop-Nachbarschaft: $N = 1$ ist ein geeigneter Wert, der keine unnötigen Nachrichten und keinen zusätzlichen Energieverbrauch von nicht selektierten Knoten zur Folge hat. Nachrichten werden deshalb nur per Broadcast gesendet und nicht von Nachbarn weitergeleitet.
- Nicht selektierte Knoten können einen Teil p_{store} der Segmente im Speicher vorhalten: Dieses Vorgehen ist nur sinnvoll, wenn die Knoten einen sehr großen Teil des Programms vorhalten können. Ansonsten ist die Wahrscheinlichkeit, dass sie ein

gesuchtes Segment zur Verfügung haben, nur gering und lohnt den zusätzlichen Energieaufwand, der durch den Wachmodus entsteht, nicht.

Beide Strategien:

- Auswahl der Weiterleitungsknoten ('Alle', 'Baum', 'Redundanter Baum'): 'Redundanter Baum' ist das effizienteste Vorgehen. Diese Strategie braucht deutlich weniger Nachrichten als 'Alle', aber nicht mehr Zeit. Durch die Redundanz ist es robuster als die einfache 'Baum'-Strategie, welche schon bei wenigen Nachrichtenverlusten immens länger dauert.
- Verteilte Struktur, Anteil selektierter Knoten (p_{sel} = 5 % bis 100 %): Je mehr Nachbarn pro Knoten selektiert sind (das heißt je größer der Anteil selektierter Knoten und je größer die Dichte), desto vorteilhafter ist das lokale Vorgehen.
- Dichte: Eine höhere Dichte führt bei beiden Strategien zu einer längeren Zeitspanne, bis alle Knoten alle Segmente bekommen haben. Das lokale Anfragen bringt jedoch immer eine kürzere Dauer und weniger versendete Nachrichten mit sich als das Anfragen bei der Senke.
- Eine Nachrichtenverlustrate von 30 % führt, verglichen mit 10 % Nachrichtenverlust, zu ähnlichen Leistungseinbußen. Bei beiden Verfahren sind auch die maximal aufgetretenen Werte deutlich größer. Je mehr Knoten selektiert sind, desto geringer ist die Einbuße jedoch beim lokalen Ansatz.
- Eck-Struktur: Aufgrund des langen und unzuverlässigen Multihop-Pfades zur Senke lohnt es gerade in diesem Szenario, erst in der Nachbarschaft fehlende Segmente anzufordern und nur dann bei der Senke anzufragen, wenn lokal nichts mehr geliefert werden kann. Die 'Baum'-Strategie benötigt hier bereits bei 10 % Nachrichtenverlust inakzeptabel lange, bis alle Knoten alle Segmente empfangen haben.

Insgesamt dauert ein Programmiervorgang von 225 Knoten in der untersuchten Netzwerkstruktur mit einer Dichte von 8 und einer Nachrichtenverlustrate von 10 % nur 2 min. Hinzukommt noch die Zeit, die zum Auswählen der zu programmierenden Knoten aufgewendet werden muss. Dabei werden bei einem Programm von 100 KByte etwa 3,5 % Overhead-Nachrichten versendet. Je größer das Programm, desto kleiner der Overhead.

Im Hinblick auf die Größe des Codes ist festzuhalten, dass das Programmierverfahren, welches fehlende Programmteile ausschließlich von der Senke anfordert, 2,9 KByte Code umfasst. Der Code, der für das lokale Austauschen von Programmteilen zusätzlich benötigt wird, ist etwa 1 KByte groß. Hinzukommt ein Baum-Routing, welches auch von der Anwendung verwendet werden kann. Das in dieser Arbeit zum Einsatz gekommene Baum-Routing war 3,8 KByte groß. In der einfachsten Variante (ohne die Verwendung einer Metrik) kann der Code jedoch auf etwa 1,5 KByte reduziert werden.

5.5 Eigenschaften des vorgestellten Verfahrens

Die einzelnen Phasen des vorgestellten Verfahrens laufen nacheinander ab, so dass sie unabhängig voneinander den Anforderungen an Sensornetze genügen müssen. Insgesamt müssen sie aber auch die Anforderungen an Programmierverfahren erfüllen.

Zuverlässigkeit: Durch das Speichern des eigenen Zustandes (empfangen oder fehlend) von jedem einzelnen Programmteil und der in Nachrichten enthaltenen Information über die Gesamtanzahl an Segmenten, ist jeder Knoten in der Lage, zu prüfen, ob er alle Teile bekommen hat. Erst wenn der Knoten das gesamte Programm sowie von der Senke eine Neustart-Nachricht erhalten hat, nimmt er einen Reboot des Mikroprozessors, um die neue Anwendung zu starten.

Feedback über den Erfolg und Misserfolg bei jedem einzelnen Knoten: Nach dem Liefern fehlender Programmsegmente werden alle IDs und Zustände gesammelt und von der Senke angezeigt.

Robustheit gegen unzuverlässige Kommunikation: Mittels Simulationen wurde gezeigt, dass das vorgeschlagene Verfahren zum Anfordern von Programmteilen im Gegensatz zu Lees Vorschlag auch bei hohen Nachrichtenverlustraten von 30 % gut funktioniert. Auch beim Einsammeln von Daten und beim Verbreiten der Neustart-Nachricht können solche Nachrichtenverlustraten toleriert werden.

Umgang mit ausfallenden Knoten: Ausfallende Knoten haben keine Auswirkung auf den lokalen Austausch von Programmnachrichten, da die Algorithmen keine ID-bezogenen Elemente enthalten und die Zahl der selektierten Knoten keine Rolle spielt. Beim Einsammeln von Daten sorgt eine obere Schranke dafür, dass die Senke nicht endlos nach der Antwort eines ausgefallenen Knoten fragt.

Geringer Ressourcenbedarf: Verglichen mit der (unbeschränkten) Größe des Programms, welches mit dem Protokoll verteilt werden kann, hat das Programmierverfahren mit insgesamt 5,4 KByte eine akzeptable Größe. Das Betriebssystem, welches zum Einsatz kam, benötigte etwa 25 bis 30 KByte und umfasste unter anderem die Kommunikation, die Bereitstellung von Timern und die Speicherverwaltung. Für die Anwendung blieben im konkreten Fall 60 KByte. Zur Laufzeit müssen Knoten sich merken, welche Segmente sie selbst bereits erhalten haben sowie temporär, welche Segmente anderen Knoten fehlen. Dazu benötigen sie jeweils maximal die Größe einer Nachricht im Speicher. Reduziert werden kann der Bedarf, indem mehr Nachrichten in einem Segment gruppiert werden, so dass die Gesamtanzahl der Segmente sinkt. Dies hat allerdings den

Nachteil, dass mehr Nachrichten erneut gesendet werden müssen, falls eine Nachricht nicht ankommt. Zudem muss der Knoten für eine geringe Anzahl $|SEG_{temp}|$ an Segmenten speichern, welche Nachrichten er aus diesem Segment bereits bekommen hat. Dies erfordert $|SEG_{temp}| \cdot log(PPSEG)$ Bits, wenn $PPSEG$ die Anzahl der Nachrichten pro Segment ist. Da $PPSEG$ und $|SEG_{temp}|$ klein sind, werden hierfür nur wenige Byte benötigt. Während der anderen Phasen müssen Knoten nur temporär vermerken, wann der Programmierprozess beginnen soll und ob sie programmiert werden sollen.

Sicherheit: Sicherheitsmaßnahmen, die mit dem vorgeschlagenen Programmierverfahren kombiniert werden können, wurden in Abschnitt 5.3.2 beschrieben. Sie liegen außerhalb des Rahmens dieser Arbeit.

Geringe Dauer: Simulationen und Experimente in der FRONTS-Testumgebung haben gezeigt, dass das lokale Anfragen von fehlenden Programmsegmenten bei den Nachbarn die Dauer des Programmierprozesses deutlich verkürzt verglichen mit dem Anfragen bei einer weit entfernten Senke. Das Einsammeln von Daten geht im Vergleich zum Verteilen des Programms aufgrund der geringeren Datenmenge und der möglichen Aggregation relativ schnell.

Geringer Energieverbrauch: In den Untersuchungen konnte nicht nur gezeigt werden, dass ein lokales Austauschen von Programmnachrichten die Dauer verkürzt, sondern auch, dass weniger Nachrichten gesendet werden. Soll nur ein Teil aller Knoten programmiert werden, brauchen meist nicht alle Knoten Nachrichten weiterzuleiten, sondern können für die Dauer des Programmierprozesses im Schlafzustand sein. Dies spart ebenfalls Energie.

Skalierbarkeit: Simulationen mit bis zu 4400 Knoten, einem Netzwerkdurchmesser bis 70 Hops und einer Dichte von bis zu 50 haben gezeigt, dass das Verfahren auch bei großen und dichten Netzen funktioniert, ohne dass der Aufwand unverhältnismäßig steigt.

Geringe Kosten: Da der Programmiervorgang vollends automatisiert abläuft, ist der Aufwand für den Anwender gering.

Feedback über den Fortschritt bei allen Knoten oder insgesamt (optional): Beide Verfahren können während des Verteilens des Programms an der Senke anzeigen, wieviel Prozent bereits geflutet wurde. Das zentrale Anfordern von fehlenden Segmenten bei der Senke bringt es mit sich, dass der Gesamtfortschritt über die Anzahl fehlender Segmente an der Senke ebenfalls bekannt ist. Werden Segmente bei Nachbarn angefordert, entfällt diese Möglichkeit. Ein Knoten bezogener Status müsste bei beiden Verfahren separat, parallel zum Nachliefern von Programmteilen, durchgeführt werden.

5.6 Zusammenfassung

Heterogenität von Sensornetzen führt zu einer Struktur, die beim Reprogrammieren des Netzes beachtet werden muss, da nicht alle Knoten dasselbe Programm erhalten können. In diesem Kapitel wurde deshalb ein effizientes und robustes Verfahren vorgestellt, welches die Anwendung bei einem beliebig wählbaren Teil der Knoten in einem Sensornetz durch eine andere ersetzt.

Programmierverfahren bestehen aus mehreren Einzelschritten. Als erstes muss ermittelt werden, welche Knoten im Netz existieren und welche reprogrammiert werden sollen. In Abhängigkeit von dieser Selektion muss mit Hilfe von allen Knoten im Netz eine Infrastruktur gebildet werden, die robuste Kommunikationspfade zwischen der Senke und den selektierten Knoten herstellt.

Anschließend kann das Programm, welches in viele einzelne Nachrichten aufgeteilt wird, durch die Senke verteilt werden. Da davon auszugehen ist, dass beim Verteilen des Programms Nachrichten verloren gehen, und somit nicht alle Knoten sofort alle Programmteile erhalten, müssen im nächsten Schritt die fehlenden Teile nachgefordert und nachgeliefert werden. Basierend auf der Tatsache, dass die meisten Teile des Programms nur bei sehr wenigen Knoten fehlen, ist es sinnvoll, diese von benachbarten Knoten, anstatt diese bei einer gegebenenfalls weit entfernten Senke anzufordern, woraufhin sie durch das gesamte Netz geflutet werden. Das lokale Vorgehen spart nicht nur Zeit, sondern auch das Senden von Nachrichten und somit Energie. Im besten Fall dauerte es in den untersuchten Strukturen nur ein Fünftel so lange wie beim zentralen Ansatz.

Sowohl zum Ermitteln der im Netz existierenden Knoten als auch zum Informieren des Anwenders über den Erfolg des Reprogrammierens ist es notwendig, dass ein Protokoll Daten von jedem Knoten im Netz zur Senke transportiert. Hierzu wurden verschiedene Ansätze vorgeschlagen, welche unterschiedliche Kompromisse zwischen der benötigten Dauer oder dem Nachrichtenaufkommen bieten, und in Testreihen untersucht. Es konnte gezeigt werden, dass mit Hilfe von Aggregation selbst 900 IDs in wenigen Sekunden eingesammelt werden können.

Kapitel 6

Zusammenfassung und Ausblick

Die Vereinigung von Messtechnik und drahtloser Kommunikation in einem Gerät ließ in den letzten Jahren die drahtlosen Sensornetze entstehen. Eingebettet in die Umwelt erheben die einzelnen Sensorknoten Daten, die sie verarbeiten, aufbereiten und untereinander austauschen. Durch Anreicherung mit Kontextinformationen über ihre Erhebung erhalten die Daten ihre Aussagekraft. Kontextinformationen bezüglich der Positionen können z.B. aus der räumlichen Anordnung der Sensorknoten abgeleitet werden, die durch die Struktur des Einsatzgebietes sowie durch die Aufgabe des Sensornetzes bedingt wird. Neben der räumlichen Anordnung führt Heterogenität zu besonderen Strukturen. Diese entsteht z.B. durch unterschiedliche Hardwareausstattungen von Knoten oder unterschiedliche Rollen oder Aufgaben, wie erheben von Daten, weiterleiten von Nachrichten, speichern oder darstellen von Informationen. Diese Arbeit beschäftigt sich sowohl mit der Erkennung von derartigen Strukturen als auch mit dem Umgang und der Nutzung von vorgegebenen Strukturen.

In Kapitel 3 dieser Arbeit wird ein neues Verfahren vorgestellt, welches dezentral Informationen über symbolische Positionen innerhalb einer Graphstruktur akquiriert, wie sie durch ein Straßen-, Kanal- oder Flursystem gebildet wird. Sensorknoten identifizieren dabei, ob sie sich an einem Knoten oder einer Kante des Graphen befinden. Über einen lokal durchgeführten Abstimmungsprozess werden Fehlentscheidungen, die z. B. durch unzuverlässige Kommunikation und fehlerhafte Distanzschätzungen entstehen können, korrigiert.

Zudem ermitteln die Sensorknoten dezentral die Anordnungen ihrer Nachbarn. Aufbauend auf der Kenntnis der Anordnungen und der symbolischen Positionen wird ein neuer, adaptiv arbeitender Algorithmus präsentiert, mit dem sich im Bereich des Netzes bewegende Objekte verfolgt werden können. Ein einzelner Knoten muss dafür nur die Anwesenheit eines sich bewegenden Objekts registrieren können. Während sich Autoren verwandter Arbeiten meist mit der Frage beschäftigen, wie nur ein Objekt gezielt verfolgt werden kann, schließt das vorgestellte Verfahren trotz unzuverlässiger Kommunikation und ungenauer Sensorik die Erkennung komplexer Bewegungsmuster mehrerer Objekte ein. Andere Autoren gehen von deutlich komplexerer und kostenintensiver Sensorik wie Kameras aus.

Um die Wartungskosten niedrig zu halten, muss das Netz autonom arbeiten, insbesondere können nicht ständig die Batterien der Knoten gewechselt werden. Um dennoch eine lange Laufzeit des Netzes zu ermöglichen, können Prozessor und Funkchip von Sensorknoten in einen Energiesparmodus versetzt werden. Da die Verarbeitung und Aufbereitung von Daten sowie der Austausch von Nachrichten aber nur im aktiven Zustand möglich sind, werden Schlaf- und Wachmodus alternierend angewendet. Allerdings wirkt sich die Verwendung eines solchen Duty-Cycles nachteilig auf die Verzögerung von Nachrichten aus, wenn bei einer Übertragung gewartet werden muss, bis der gewünschte Empfänger empfangsbereit ist.

Mit CUPID (Communication Pattern informed Duty Cycling) wird daher in Kapitel 4 ein flexibles Duty-Cycle-Verfahren präsentiert, welches die Verzögerung von Nachrichten in einer Vorzugsrichtung trotz großer (hop-basierter) Entfernung zwischen Quell- und Zielknoten und gleich bleibenden, geringen Duty-Cycles (weniger als 1 %) niedrig hält. Es realisiert eine Art Welle, bei der die Knoten der Reihe nach in den aktiven und anschließend wieder in den inaktiven Modus wechseln, so dass Nachrichten mit nur kurzen Verzögerungen weitergeleitet werden können. Durch die Aufteilung von Knoten in Gruppen, von denen jeweils nur eine zur Zeit Nachrichten sendet, wird zudem die Kollisionswahrscheinlichkeit im Vergleich zum wahlfreien Medienzugriff verringert. Die grundlegende Idee der Welle wurde bereits von anderen Autoren vorgeschlagen. Diese gehen jedoch von anderen Voraussetzungen aus. Zudem wird in der vorliegenden Arbeit zum ersten Mal gezeigt, inwiefern sich Parameter wie die maximale Ende-zu-Ende-Verzögerung, der Duty-Cycle und die Anzahl zu sendender Nachrichten pro Wachphase gegenseitig bedingen und welche Kommunikationsmuster und Netzstrukturen unterstützt werden. In Simulationen und Experimenten mit bis zu 160 Knoten konnte gezeigt werden, dass sich das Verfahren besonders für einen Einsatz in Sensornetzen mit großem Durchmesser eignet.

Kapitel 5 thematisiert das Reprogrammieren von großen, heterogenen Sensornetzen. Da ein manuelles Reprogrammieren lange dauert und mit hohen Kosten einhergeht, ist das automatisierte Reprogrammieren von Knoten über die Funkschnittstelle von essenzieller Bedeutung. Nur so wird auch eine Aktualisierung der Software zur Verfeinerung von Messverfahren oder zur Effizienzsteigerung beim Routing oder beim Duty-Cycling möglich, was beispielsweise bei Langzeitanwendungen von Relevanz ist. In Kapitel 5 werden die notwendigen Schritte herausgearbeitet sowie entsprechende Algorithmen vorgestellt und evaluiert. Unter anderem werden ein Aggregationsalgorithmus sowie ein Verfahren zum zuverlässigen Verteilen von großen Datenmengen diskutiert. Ein robuste Infrastruktur, die aus einem Teil des Netzes besteht, dient der Kommunikation zwischen dem Knoten, der ein neues Programm verteilt, und den Knoten, die das Programm empfangen sollen.

In Simulationen wurde gezeigt, dass mehr als 200 Knoten innerhalb von zwei Minuten mit einem Programm von 100 KByte programmiert werden können, selbst wenn 10 % der Nachrichten verloren gehen. Im besten Fall werden in diesem Szenario nur ein Sechstel der Zeit des Verfahrens von Lee et al. benötigt (130 s statt 13 min). Binnen

3,5 min können bereits 5040 Knoten reprogrammiert werden. Andere verwandte Verfahren erlauben es dagegen gar nicht, nur einen Teil des Netzes neu zu programmieren, oder beruhen auf bidirektionalen, konstanten Kommunikationsverbindungen. Experimente in einem Sensornetz, bestehend aus 22 Knoten, belegen die Realisierbarkeit des in dieser Arbeit vorgeschlagenen Verfahrens und die Einhaltung der in Kapitel 2 beschriebenen Ressourcenbeschränkungen. Dazu gehört nicht nur, dass geringe Datenmengen gespeichert und keine komplexen Berechnungen durchgeführt werden, sondern auch, dass möglichst wenig Nachrichten ausgetauscht werden.

Wie die überwiegende Anzahl an Forschungsarbeiten lässt auch dieser Beitrag Raum für zukünftige Arbeiten. Verfeinern ließe sich das Objektverfolgungsverfahren durch eine Berücksichtigung der Anzahl der aktuell beobachteten Objekte beim Übertragen von Reihenfolgen in die Historie. Je weniger Objekte kürzlich passierten, desto wahrscheinlicher ist es, dass die ermittelten Reihenfolgen der Realität entsprechen, was die Qualität der Historie noch weiter erhöhen würde. Raum für zukünftige Arbeiten bietet weiterhin die experimentelle Evaluation, da die zur Objektverfolgung eingesetzten Verfahren bisher nur in Simulationen untersucht wurden. Es wurde nicht nur aufgezeigt, dass die Verfahren ressourcensparend arbeiten, sondern es wurde zudem von sehr pessimistischen Bedingungen bezüglich Messfehler ausgegangen, so dass zu erwarten ist, dass das Verfahren auch außerhalb von Simulatoren funktionieren wird. Werden höhere Nachrichtenverlustraten als 5 % registriert, müssen Nachrichten gegebenenfalls mehrmals gesendet werden.

Im Hinblick auf das Programmierverfahren ist eine Erprobung in einem größeren Sensornetz der nächste Schritt. Es wurde bereits in einem Netz mit 22 Knoten gezeigt, dass es beliebige Teilmengen des Netzes zuverlässig und robust gegen Nachrichtenverlust reprogrammiert und die Anforderungen an Sensornetzanwendungen erfüllt. Da das Verfahren jedoch gerade für große Netze entwickelt wurde, wäre eine derartige Evaluation von Interesse. Simulationen, in denen von sehr schwierigen Bedingungen ausgegangen wurde, lassen vermuten, dass es gerade in großen Netzen effizienter ist als bestehende Verfahren.

Anhang A

Vergleich von Zeichenketten mittels Local Alignment

Der Operator zum Vergleich von Ereignisketten bei der Objektverfolgung wurde vom so genannten *Local Alignment* [102] abgeleitet, welches zum Finden von ähnlichen Strings in der DNA-Analyse verwendet wird. Es wurde wie folgt eingesetzt:

$$\text{Sei } s : \mathbb{N}^n \times \mathbb{N}^m \rightarrow \mathbb{N}^{n \times m}, s_{i,j} = \begin{cases} 0 : x_i \neq y_j \\ 1 : x_i = y_j \wedge |i-j| > 1 \\ 2 : sonst \end{cases}$$

die Gleichheitsmatrix zweier Sequenzen $X = \{x_1, ..., x_n\}$ und $Y = \{y_1, ..., y_m\}$, sei d (mit $d < 0$) das Strafmaß für einen fehlenden Eintrag und sei F die Matrix, mit der das Local Alignment berechnet wird:

$$F(0,0) = 0$$
$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_i) \\ F(i-1, j) + d \\ F(i, j-1) + d \\ 0 \end{cases} .$$

Die Matrix F wird nun zeilenweise gefüllt, ihr maximaler Wert $\max\limits_{0 \leqslant i,j \leqslant m,n} F(i,j)$ zeigt den Ähnlichkeitsgrad der beiden Ketten an, wobei höhere Werte einen höheren Grad an Ähnlichkeit bedeuten.

Sollen beispielsweise die Zeichenketten $X =$ `1 2 4 5` und $Y =$ `1 2 3 4 5` miteinander verglichen werden, wird die Matrix zunächst wie in Abbildung A.1 dargestellt initialisiert. Anschließend wird sie sukzessive nach der oben genannten rekursiven Berechnungsvorschrift von links nach rechts und von oben nach unten ausgefüllt. d wurde auf -1 festgelegt.

Der höchste Wert in der Matrix enthält die Aussage über die Ähnlichkeit der Zeichenketten; je größer er ist, desto besser stimmen sie überein. Bei maximaler Übereinstimmung, das heißt, dass einer der Strings Teilstring des anderen ist, liefert das Local Alignment $2 \cdot L_{min}$ (mit L_{min} = Länge des kürzeren Strings) als Ergebnis. Bei einem fehlenden oder überflüssigen Zeichen wird das Ergebnis um $|d|$ vermindert. Im Minimum ist das

i \ j			0	1	2	3	4
	X \ Y		1	2	3	4	5
		0	0	0	0	0	0
0	1	0					
1	2	0					
2	4	0					
3	5	0					

Abbildung A.1: Initialisierungsmatrix

i \ j			0	1	2	3	4
	X \ Y		1	2	3	4	5
		0	0	0	0	0	0
0	1	0	2	1	0	0	0
1	2	0	1	4	3	2	1
2	4	0	0	3	2	5	4
3	5	0	0	2	1	4	(7)

Abbildung A.2: Ergebnismatrix

Ergebnis Null und die Strings sind disjunkt. Im Beispiel ist der größte Wert in der Matrix 7, und L_{min} hat den Wert 4. Somit ergibt sich eine Ähnlichkeit von $\frac{7}{2 \cdot 4} = \frac{7}{8} = 0{,}875$.

Jede Zelle $F(i, j)$ kann mit konstant vielen Operationen berechnet werden, so dass es $O(m \cdot n)$ Zeit kostet, die gesamte Matrix zu füllen.

Anhang B

Verzeichnisse

Abbildungsverzeichnis

Abkürzungsverzeichnis

AAL	Ambient Assisted Living
BAN	Body Area Network
CDS	Connected Dominating Set
CTS	Clear To Send
CUPID	Communication Pattern informed Duty Cycling
DOI	Degree of irregularity
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPS	Global Positioning System
GSM	Global System for Mobile Communications
IEEE	Institute of Electrical and Electronics Engineers
LAN	Local Area Network
LPL	Low Power Listening
MAC	Media Access Control
MTU	Maximum Transfer Unit
OTSN	Object Tracking Sensor Network
PIR	Passiv-Infrarot
RAM	Random Access Memory
RIM	Radio Irregularity Model
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
RTS	Ready To Send
TDoA	Time Difference of Arrival
ToA	Time of Arrival
UMTS	Universal Mobile Telecommunications System
VANETS	Vehicular Ad-hoc Networks
WSN	Wireless Sensor Network

Literaturverzeichnis

[1] Annamalai, Valliappan ; Gupta, Sandeep K. S. ; Schwiebert, Loren: On tree-based convergecasting in wireless sensor networks. In: *IEEE Wireless Communications and Networking* Bd. 3, 2003

[2] Arora, Anish ; Dutta, Prabal ; Bapat, Sandip ; Kulathumani, Vinod ; Zhang, Hongwei ; Naik, Vinayak ; Mittal, Vineet ; Cao, Hui ; Demirbas, Murat ; Gouda, Mohamed G. ; Choi, Young ri ; Herman, Ted ; Kulkarni, Sandeep S. ; Arumugam, Umamaheswaran ; Nesterenko, Mikhail ; Vora, Adnan ; Miyashita, Mark: A line in the sand: a wireless sensor network for target detection, classification, and tracking. In: *Computer Networks* 46 (2004), Nr. 5, 605-634. `http://dblp.uni-trier.de/db/journals/cn/cn46.html#AroraDBKZNMCDGCHKANVM04`

[3] Arvind, Koshal: Probabilistic Clock Synchronization in Distributed Systems. In: *IEEE Transactions on Parallel and Distributed Systems* 5 (1994), Nr. 5, S. 474–487. – DOI http://dx.doi.org/10.1109/71.282558. – ISSN 1045–9219

[4] Aslam, Javed A. ; Butler, Zack J. ; Constantin, Florin ; Crespi, Valentino ; Cybenko, George ; Rus, Daniela: Tracking a moving object with a binary sensor network. In: *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, ACM, 2003. – ISBN 1–58113–707–9, 150-161

[5] Azim, Tahir ; Mansoor, Qasim ; Levis, Philip: Starburst SSD: an efficient protocol for selective dissemination. In: *ICC'09: Proceedings of the 2009 IEEE international conference on Communications*. Piscataway, NJ, USA : IEEE Press, 2009. – ISBN 978–1–4244–3434–3, S. 74–79

[6] Baer, Dieter (Hrsg.) ; Wermke, Matthias (Hrsg.): *Duden, Das große Fremdwörterbuch: Herkunft und Bedeutung der Fremdwörter*. Mannheim : Dudenverlag, 2000

[7] Bhatti, Sania ; Xu, Jie: Survey of Target Tracking Protocols Using Wireless Sensor Network. In: *Wireless and Mobile Communications, International Conference on* 0 (2009), S. 110–115. – DOI http://doi.ieeecomputersociety.org/10.1109/ICWMC.2009.25. ISBN 978–0–7695–3750–4

[8] Bisnik, Nabhendra ; Abouzeid, Alhussein A.: Delay and capacity in energy efficient sensor networks. In: *PE-WASUN '07: Proceedings of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor,and ubiquitous networks*. New

York, NY, USA : ACM, 2007. – DOI http://doi.acm.org/10.1145/1298197.1298201. – ISBN 978–1–59593–808–4, S. 17–24

[9] Buettner, Michael ; Yee, Gary V. ; Anderson, Eric ; Han, Richard: X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In: *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*. New York, NY, USA : ACM, 2006. – DOI http://doi.acm.org/10.1145/1182807.1182838. – ISBN 1–59593–343–3, S. 307–320

[10] Bulusu, Nirupama ; Bychkovskiy, Vladimir ; Estrin, Deborah ; Heidemann, John: Scalable, Ad Hoc Deployable, RF-Based Localization. In: *Proceedings of the Computing Conference 2002*, 2002

[11] Buschmann, Carsten ; Fekete, Sándor P. ; Fischer, Stefan ; Kröller, Alexander ; Pfisterer, Dennis: Koordinatenfreies Lokationsbewusstsein (Localization without Coordinates). In: *it - Information Technology, Themenheft Sensornetze* 47 (2005), April, Nr. 4

[12] Buschmann, Carsten ; Hellbrück, Horst ; Fischer, Stefan ; Kröller, Alexander ; Fekete, Sándor P.: Radio Propagation-Aware Distance Estimation Based on Neighborhood Comparison. In: *Proceedings of the 4th European conference on Wireless Sensor Networks (EWSN 2007), Delft, The Netherlands*, 2007

[13] Buschmann, Carsten ; Pfisterer, Dennis: iSense: A Modular Hardware and Software Platform for Wireless Sensor Networks / 6. Fachgespräch "Drahtlose Sensornetze"der GI/ITG-Fachgruppe "Kommunikation und Verteilte Systeme". Version: 2007. `http://ds.informatik.rwth-aachen.de/events/fgsn07`. 2007. – Forschungsbericht

[14] Cao, Qing ; Abdelzaher, Tarek ; He, Tian ; Stankovic, John: Towards optimal sleep scheduling in sensor networks for rare-event detection. In: *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. Piscataway, NJ, USA : IEEE Press, 2005. – ISBN 0–7803–9202–7, S. 4

[15] Capkun, Srdjan ; Hamdi, Maher ; Hubaux, Jean-Pierre: GPS-free positioning in mobile ad hoc networks. In: *Proceedings of the 34th Hawaii International Conference on System Sciences*, 2001

[16] Carpi, Federico ; (Editors), Elisabeth S.: *Biomedical Applications of Electroactive Polymer Actuators*. Wiley Press, 2009

[17] Chellappa, Rama ; Qian, Gang ; Zheng, Qinfen: Vehicle detection and tracking using acoustic and video sensors. In: *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, 17-21 May 2004* Bd. 3, IEEE, 2004. – DOI doi:10.1109/ICASSP.2004.1326664. – ISBN 0780384849, S. 793–6. – Source type:Print

[18] coalesenses GmbH: *iSense*. 2010. – http://www.coalesenses.com

[19] Cristian, Flaviu ; Fetzer, Christof: The Timed Asynchronous Distributed System Model. In: *IEEE Transactions on Parallel and Distributed Systems* 10 (1999), Nr. 6, S. 642–657. – DOI http://dx.doi.org/10.1109/71.774912. – ISSN 1045–9219

[20] Crossbow Technology Inc.: *MICAz wireless measurement system*. `http://www.xbow.com`. Version: Juni 2004. – http://www.xbow.com

[21] CST group, FU Berlin: *Website of the Embedded Sensor Board ESB 430/2*. 2010. – http://cst.mi.fu-berlin.de/projects/ScatterWeb/

[22] Dam, Tijs van ; Langendoen, Koen G.: An adaptive energy-efficient MAC protocol for wireless sensor networks. In: *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA : ACM, 2003. – DOI http://doi.acm.org/10.1145/958491.958512. – ISBN 1–58113–707–9, S. 171–180

[23] De, Pradip ; Liu, Yonghe ; Das, Sajal K.: ReMo : An Energy Efficient Reprogramming Protocol for Mobile Sensor Networks. In: *IEEE International Conference on Pervasive Computing (PerCom)*, IEEE Computer Society, 2008, 60-69

[24] De Couto, Douglas S. J. ; Aguayo, Daniel ; Chambers, Benjamin A. ; Morris, Robert: Performance of Multihop Wireless Networks: Shortest Path is Not Enough. In: *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*. Princeton, New Jersey, October 2002

[25] De Couto, Douglas S. J. ; Morris, Robert ; Aguayo, Daniel ; Bicket, John: A High-Throughput Path Metric for Multi-Hop Wireless Routing. In: *MobiCom '03: Proceedings of the 9th ACM International Conference on Mobile Computing and Networking*. San Diego, California, September 2003

[26] Deng, Jing ; Han, Richard ; Mishra, Shivakant: Secure code distribution in dynamically programmable wireless sensor networks. In: *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*. New York, NY, USA : ACM, 2006. – DOI http://doi.acm.org/10.1145/1127777.1127822. – ISBN 1–59593–334–4, S. 292–300

[27] Dieter, Lin L. ; Liao, Lin ; Fox, Dieter ; Hightower, Jeffrey ; Kautz, Henry ; Schulz, Dirk: Voronoi tracking: Location estimation using sparse and noisy sensor data. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.*, 2003

[28] Dieter Fox, Wolfram B. Sebastian Thrun T. Sebastian Thrun ; Dellaert, Frank: Particle filters for mobile robot localization. In: *Sequential Monte Carlo Methods in Practice*. New York : Springer Verlag, 2000. – To appear

[29] Eck, Daniel ; Schilling, Klaus: Entwicklung eines teil-autonomen Scooters zur Unterstützung der Mobilität leistungsgewandelter Personen. In: *Universität Würzburg Tagungsband Ambient Assisted Living des 2.Deutschen AAL-Kongress, VDE Verlag*, 2009

[30] El-Hoiydi, Amre ; Decotignie, Jean-Dominique: WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks. In: *Proceedings of ISCC"04*. Washington, DC, USA : IEEE Computer Society, 2004. – ISBN 0–7803–8623–X, S. 244–251

[31] Elnahrawy, Eiman ; Li, Xiaoyan ; Martin, Richard P.: The Limits of Localization Using Signal Strength: A Comparative Study. In: *The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2004

[32] Elson, Jeremy E.: *Time synchronization in wireless sensor networks*, Diss., 2003

[33] Fekete, Sándor P. ; Kröller, Alexander ; Fischer, Stefan ; Pfisterer, Dennis: Shawn: The fast, highly customizable sensor network simulator. In: *Proceedings of the Fourth International Conference on Networked Sensing Systems (INSS' 07)*, 2007

[34] Fekete, Sándor P. ; Kröller, Alexander ; Pfisterer, Dennis ; Fischer, Stefan ; Buschmann, Carsten: Neighborhood-Based Topology Recognition in Sensor Networks. In: *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004)*, 2004

[35] Ganeriwal, Saurabh ; Kumar, Ram ; Adlakha, Sachin ; Srivastava, Mani: Network-wide Time Synchronization in Sensor Networks / Networked and Embedded Systems Lab (NESL), Electrical Engineering Department, UCLA. 2003. – Forschungsbericht

[36] Ganesan, Deepak ; Krishnamachari, Bhaskar ; Woo, Alec ; Culler, David E. ; Estrin, Deborah ; Wicker, Stephen: Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks / UCLA. Version: 2002. `http://citeseer.ist.psu.edu/ganesan02complex.html`. 2002. – Forschungsbericht

[37] Garcia-Luna-Aceves, Jose J. ; Spohn, Marcelo: Source-Tree Routing in Wireless Networks. In: *ICNP '99: Proceedings of the Seventh Annual International Conference on Network Protocols*. Washington, DC, USA : IEEE Computer Society, 1999, S. 273–282

[38] Gnawali, Omprakash ; Yarvis, Mark ; Heidemann, John ; Govindan, Ramesh: Interaction of retransmission, blacklisting, and routing metrics for reliability in sensor network routing. In: *The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2004, S. 34–43

[39] Gu, Lin ; Stankovic, John A.: Radio-Triggered Wake-Up Capability for Sensor Networks. In: *RTAS '04: Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC, USA : IEEE Computer Society, 2004. – ISBN 0–7695–2148–7, S. 27

[40] Heath, Kyle ; Guibas, Leonidas J.: Multi-person tracking from sparse 3D trajectories in a camera sensor network. In: *2nd ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC-08)*, 2008, S. 1–9

[41] Hellbrück, Horst ; Fischer, Stefan: Towards Analysis and Simulation of Ad-Hoc Networks. In: *Proceedings of the 2002 International Conference on Wireless Networks (ICWN02)*. Las Vegas, Nevada, USA : IEEE Computer Society Press, June 2002, 69–75

[42] Hill, Jason L.: *System architecture for wireless sensor networks*, University of California, Berkeley, Diss., 2003. http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=979516#. – Adviser: David E. Culler

[43] Hui, Jonathan W. ; Culler, David E.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, ACM, 2004. – ISBN 1–58113–879–2, 81-94

[44] IEEE 802.11 Working Group: *IEEE 802.11 Wireless local area networks*. 2010. – http://www.ieee802.org/11/

[45] IEEE 802.15 Working Group: *IEEE 802.15 WPAN Task Group 4 (TG4)*. 2010. – http://www.ieee802.org/15/pub/TG4.html

[46] Jedermann, Reiner: *Der intelligente Container als autonomes Sensorsystem*. DFKI / IRL Forum (Deutsche Forschungszentrum für Künstliche Intelligenz / Innovative Retail Laboratory),

[47] Jeong, Jaein ; Culler, David E.: Incremental Network Programming for Wireless Sensors. In: *IEEE Sensor and Ad Hoc Communications and Networks (SECON)*, 2004, S. 25–33

[48] Jurdak, Raja ; Baldi, Pierre ; Videira Lopes, Cristina: Adaptive Low Power Listening for Wireless Sensor Networks. In: *IEEE Transactions on Mobile Computing* 6 (2007), Nr. 8, S. 988–1004. – DOI http://dx.doi.org/10.1109/TMC.2007.1037. – ISSN 1536–1233

[49] Keshavarzian, Abtin ; Lee, Huang ; Venkatraman, Lakshmi: Wakeup scheduling in wireless sensor networks. In: *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*. New York, NY, USA : ACM, 2006. – DOI 10.1145/1132905.1132941. – ISBN 1595933689, 322–333

[50] Kim, WooYoung ; Mechitov, Kirill ; Choi, Jeung-Yoon ; Ham, Soo K.: On target tracking with binary proximity sensors. In: *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, IEEE, 2005, 301-308

[51] Krishnamachari, Bhaskar ; Estrin, Deborah ; Wicker, Stephen B.: The Impact of Data Aggregation in Wireless Sensor Networks. In: *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1588–6, S. 575–578

[52] Kröller, Alexander ; Fekete, Sándor P. ; Pfisterer, Dennis ; Fischer, Stefan: Deterministic boundary recognition and topology extraction for large sensor networks. In: *Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, 2006, S. 1000–1009

[53] Krontiris, Ioannis ; Dimitriou, Tassos: Authenticated In-Network Programming for Wireless Sensor Networks. In: *Proceedings of the 5th International Conference on AD-HOC Networks & Wireless*, 2006

[54] Krüger, Daniela ; Buschmann, Carsten ; Fischer, Stefan: Discovering topologies in Wireless Sensor Networks / 5. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze. 2006. – Forschungsbericht

[55] Krüger, Daniela ; Buschmann, Carsten ; Fischer, Stefan: *Topology recognition in Wireless Sensor Networks*. The Fourth International Conference on Mobile System, Applications, and Services : Poster, Juni 2006

[56] Krüger, Daniela ; Buschmann, Carsten ; Fischer, Stefan: Object Tracking on Graph Structures. In: *Proceedings of the 3rd IEEE European Conference on Smart Sensing and Context (EuroSSC'08)*, 2008

[57] Krüger, Daniela ; Fischer, Stefan ; Buschmann, Carsten: Solar Powered Sensor Network Design and Experimentation. In: *The Sixth International Symposium on Wireless Communication Systems 2009*, 2009

[58] Krüger, Daniela ; Pfisterer, Dennis ; Buschmann, Carsten: Selective Reprogramming in Sensor Networks. In: *Wireless Sensor Networks - theory and practice (WSN'2011), Under Review*. Paris, France, 2011

[59] Krüger, Daniela ; Pfisterer, Dennis ; Fischer, Stefan: CUPID - Communication pattern informed Duty Cycling in Sensor Networks. In: *The Fifth International Conference on Systems and Networks Communications (ICSNC 2010)*, IEEE Computer Society Conference Publishing Services, August 2010. – Best paper award

[60] Kulkarni, Sandeep S. ; Arumugam, Mahesh: INFUSE: A TDMA based data dissemination protocol for sensor networks / International Journal on Distributed Sensor Networks (IJDSN. 2004. – Forschungsbericht

[61] Kumar, Munish ; Jaffery, Z. A. ; Moinuddin: Greedy Distributed Tree Routing algorithm for wireless ad hoc networks. In: *ACST '08: Proceedings of the Fourth IASTED International Conference on Advances in Computer Science and Technology*. Anaheim, CA, USA : ACTA Press, 2008. – ISBN 978–0–88986–730–7, S. 248–253

[62] Lamport, Leslie: Time, clocks, and the ordering of events in a distributed system. In: *Communications of the ACM* 21 (1978), Nr. 7, S. 558–565. – DOI http://doi.acm.org/10.1145/359545.359563. – ISSN 0001–0782

[63] Lee, Cheng-Ta ; Lin, Frank Yeong-Sung ; Wen, Yean-Fu: An Efficient Object Tracking Algorithm in Wireless Sensor Networks. In: *9th Joint Conference on Information Sciences (JCIS)*, 2006

[64] Lee, In-Sook ; Fu, Zhen ; Yang, WenCheng ; Park1, Myong-Soon: An efficient dynamic clustering algorithm for object tracking. In: *Complex Systems and Applications-Modeling, Control and Simulations* 14 (S2) (2007), S. 1484–1488

[65] Lee, Kangwoo ; Kim, Jae eon ; Thuy, Do D. ; Kim, Daeyoung ; Ahn, Sungjin ; Yang, Jinyoung: Multi-hop Network Re-programming Model for Wireless Sensor Networks. In: *5th IEEE Conference on Sensors*, 2006, S. 884–887

[66] Lemmon, Michael D. ; Ganguly, Joydeep ; Xia, Li: Model-based clock synchronization in networks with drifting clocks. In: *PRDC '00: Proceedings of the 2000 Pacific Rim International Symposium on Dependable Computing*. Washington, DC, USA : IEEE Computer Society, 2000. – ISBN 0–7695–0975–4, S. 177

[67] Leong, Ben ; Liskov, Barbara ; Morris, Robert: Geographic routing without planarization. In: *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*. Berkeley, CA, USA : USENIX Association, 2006, S. 25–25

[68] Levis, Philip ; Lee, Nelson ; Welsh, Matt ; Culler, David E.: TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In: *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003. – http://www.cs.berkeley.edu/~pal/research/tossim.html

[69] Levis, Philip ; Patel, Neil ; Shenker, Scott ; Culler, David E.: Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In: *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI*, 2004, S. 15–28

[70] Li, Qun ; Rus, Daniela: Global clock synchronization in sensor networks. In: *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies* Bd. 1, 2004, 574

[71] Li, Yuan ; Ye, Wei ; Heidemann, John: Energy and Latency Control in Low Duty Cycle MAC Protocols / USC/Information Sciences Institute. 2004 (ISI-TR-595). – Technical Report

[72] Lin, Chih-Yu ; Peng, Wen-Chih ; Tseng, Yu-Chee: Efficient In-Network Moving Object Tracking in Wireless Sensor Networks. In: *IEEE Transactions on Mobile Computing* 5 (2006), Nr. 8, S. 1044–1056. – DOI http://dx.doi.org/10.1109/TMC.2006.115. – ISSN 1536–1233

[73] Liskov, Barbara: Practical uses of synchronized clocks in distributed systems. In: *Distrib. Comput.* 6 (1993), Nr. 4, 211–219. `http://dx.doi.org/10.1007/BF02242709`

[74] Liu, Hai ; Jia, Xiaohua ; Wan, Peng-Jun ; Liu, Xinxin ; Yao, Frances F.: A Distributed and Efficient Flooding Scheme Using 1-Hop Information in Mobile Ad Hoc Networks. In: *IEEE Transactions on Parallel and Distributed Systems* 18 (2007), Nr. 5, 658-671. `http://dblp.uni-trier.de/db/journals/tpds/tpds18.html#LiuJWLY07`

[75] Liu, Ting ; Martonosi, Margaret: Impala: a middleware system for managing autonomic, parallel sensor systems. In: *PPOPP '03: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ACM, 2003. – ISBN 1–58113–588–2, 107-118

[76] Lu, Jiakang ; Whitehouse, Kamin: Flash Flooding: Exploiting the Capture Effect for Rapid Flooding in Wireless Sensor Networks. In: *The 29th IEEE International Conference on Computer Communications (INFOCOM)*, IEEE, 2009, 2491-2499

[77] Mannermaa, Jukka-Pekka ; Kalliomäki, Kalevi ; Manstén, Tapio ; Turunen, Seppo: Timing performance of various GPS receivers. In: *Proceedings of the 1999 Joint Meeting of the European Frequency and Time Forum and the IEEE International Frequency Control Symposium*, 1999

[78] Maroti, Miklos ; Kusy, Branislav ; Simon, Gyula ; Lédeczi, Ákos: Robust Multi-Hop Time Synchronization in Sensor Networks. In: *International Conference on Wireless Networks*, 2004, S. 454–460

[79] Maróti, Miklós ; Völgyesi, Péter ; Dora, Sebestyén ; Kusý, Branislav ; Nádas, András ; Ledeczi, Akos ; Balogh, Gyorgy ; Molnár, Károly: Radio interferometric geolocation. In: *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, 2005

[80] Marrón, Pedro J. ; Lachenmann, Andreas ; Minder, Daniel ; Hähner, Jörg ; Sauter, Robert ; Rothermel, Kurt: TinyCubus: A Flexible and Adaptive Framework for Sensor Networks. In: *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, 2005, S. 278–289

[81] Mascarenas, David L. ; Todd, Michael D. ; Park, Gyuhae ; Farrar, Charles R.: Development of an impedance-based wireless sensor node for structural health monitoring. In: *Smart Materials and Structures* 16 (2007), Nr. 6, 2137-2145. http://stacks.iop.org/0964-1726/16/2137

[82] Mock, Michael ; Frings, Reiner ; Nett, Edgar ; Trikaliotis, Spiro: Continuous Clock Synchronization in Wireless Real-Time Applications. In: *SRDS '00: Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*. Washington, DC, USA : IEEE Computer Society, 2000. – ISBN 0–7695–0543–0, S. 125

[83] Mottola, Luca ; Picco, Gian P. ; Sheikh, Adil A.: FiGaRo: fine-grained software reconfiguration for wireless sensor networks. In: *EWSN'08: Proceedings of the 5th European conference on Wireless sensor networks*. Berlin, Heidelberg : Springer-Verlag, 2008. – ISBN 3–540–77689–3, 978–3–540–77689–5, S. 286–304

[84] Ni, Sze-Yao ; Tseng, Yu-Chee ; Chen, Yuh-Shyan ; Sheu, Jang-Ping: The Broadcast Storm Problem in a Mobile ad hoc Network. In: *MOBICOM '99: Proceedings of the 5th annual international conference on Mobile computing and networking*, 1999, 151-162

[85] Oh, Songhwai ; Sastry, Shankar: Tracking on a graph. In: *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, IEEE, 2005, 195-202

[86] Pahalawatta, Peshala V. ; Depalov, Dejan ; Pappas, Thrasyvoulos N. ; Katsaggelos, Aggelos K.: Detection, Classification, and Collaborative Tracking of Multiple Targets Using Video Sensors. In: *IPSN '03: Proceedings of the 2nd international symposium on Information processing in sensor networks* Bd. 2634, Springer, 2003 (Lecture Notes in Computer Science). – ISBN 3–540–02111–6, 529-544

[87] Pásztor, Bence ; Mottola, Luca ; Mascolo, Cecilia ; Picco, Gian P.: Selective code dissemination in mobile wireless sensor networks. In: *Middleware (Companion)*, ACM, 2008. – ISBN 978–1–60558–369–3, 113-115

[88] Pfisterer, Dennis ; Lipphardt, Martin ; Buschmann, Carsten ; Hellbrueck, Horst ; Fischer, Stefan ; Sauselin, Jan H.: MarathonNet project. In: *InterSense '06: Proceedings of the 1st international conference on Integrated internet ad hoc and sensor networks*. New York, NY, USA : ACM, 2006. – DOI http://doi.acm.org/10.1145/1142680.1142696. – ISBN 1–59593–427–8, S. 12. – Funded by the Klaus Tschira Foundation, http://www.marathonnet.de (2010)

[89] Polastre, Joseph ; Szewczyk, Robert ; Culler, David E.: Telos: enabling ultra-low power wireless research. In: *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005, 364-369

[90] Reijers, Niels ; Langendoen, Koen G.: Efficient code distribution in wireless sensor networks. In: *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. New York, NY, USA : ACM, 2003. – DOI http://doi.acm.org/10.1145/941350.941359. – ISBN 1–58113–764–8, S. 60–67

[91] Ristic, Branko ; Arulampalm, Sanjeev ; Gordon, Neil: *Beyond the Kalman filter : particle filters for tracking applications / Branko Ristic, Sanjeev Arulampalm, Neil Gordon* . Artech House, Boston, Ma. ; London :, 2004. – xiii, 299 p. : S. – ISBN 158053631

[92] Rossi, Michele ; Zanca, Giovanni ; Stabellini, Luca ; Crepaldi, Riccardo ; III, Albert F. H. ; Zorzi, Michele: SYNAPSE: A Network Reprogramming Protocol for Wireless Sensor Networks Using Fountain Codes. In: *Fifth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, IEEE, 2008, 188-196

[93] Rothenpieler, Peter ; Krüger, Daniela ; Pfisterer, Dennis ; Fischer, Stefan ; Dudek, Denise ; Haas, Christian ; Kuntz, Andreas ; Zitterbart, Martina: FleGSens - secure area monitoring using wireless sensor networks. In: *Proceedings of World Academy of Science, Engineering and Technology* (2009)

[94] Sasson, Yoav ; Cavin, David ; Schiper, André: Probabilistic Broadcast for Flooding in Wireless Mobile Ad hoc Networks. In: *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2003)*, 2003

[95] Saukh, Olga ; Marrón, Pedro J. ; Lachenmann, Andreas ; Gauger, Matthias ; Minder, Daniel ; Rothermel, Kurt: Generic Routing Metric and Policies for WSNs. In: *Proceedings of the 3rd European Conference on Wireless Sensor Networks (EWSN)*, 2006

[96] Savarese, Chris ; Rabaey, Jan M. ; Beutel, Jan: Locationing in Distributed Ad-Hoc Wireless Sensor Networks. In: *International Conference on Accustic, Speech, and Signal Processing (ICASSP 2001)* Bd. 4, 2001, S. 2037–2040

[97] Savvides, Andreas ; Han, Chih-Chieh ; Strivastava, Mani B.: Dynamic fine-grained localization in Ad-Hoc networks of sensors. In: *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA : ACM Press, 2001. – DOI http://doi.acm.org/10.1145/381677.381693. – ISBN 1–58113–422–3, S. 166–179

[98] Savvides, Andreas ; Park, Heemin ; Srivastava, Mani B.: The n-Hop Multilateration Primitive for Node Localization Problems. In: *Mobile Networks and Applications* (2003), Nr. 8, S. 443–451

[99] Schiller, Jochen ; Liers, Achim ; Ritter, Hartmut ; Winter, Rolf ; Voigt, Thiemo: ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing. In: *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International*

Conference on System Sciences (HICSS'05) - Track 9. Washington, DC, USA : IEEE Computer Society, 2005, 286.3

[100] Schurgers, Curt ; Tsiatsis, Vlasios ; Ganeriwal, Saurabh ; Srivastava, Mani B.: Topology management for sensor networks: exploiting latency and density. In: *MobiHoc '02: Proceedings of the 3rd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing*, ACM, 2002. – ISBN 1–58113–501–7, 135-145

[101] Schwiebert, Loren ; Gupta, Sandeep K. S. ; Weinmann, Jennifer: Research challenges in wireless networks of biomedical sensors. In: *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001, S. 151–165

[102] Setubal, Joao ; Meidanis, Joao: *Introduction to computational molecular biology*. PWS Publishing Company, 1997

[103] Shrivastava, Nisheeth ; Mudumbai, Raghuraman ; Madhow, Upamanyu ; Suri, Subhash: Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In: *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, ACM, 2006. – ISBN 1–59593–343–3, 251-264

[104] Sichitiu, Mihail L. ; Veerarittiphan, Chanchai: Simple, accurate time synchronization for wireless sensor networks. In: *IEEE Wireless Communications and Networking* 2 (2003), S. 1266–1273

[105] Singh, Jaspreet ; Madhow, Upamanyu ; Kumar, Rajesh ; Suri, Subhash ; Cagley, Richard: Tracking multiple targets using binary proximity sensors. In: *IPSN '07: Proceedings of the 6th international symposium on Information processing in sensor networks*, ACM, 2007, 529-538

[106] Solis, Ignacio ; Obraczka, Katia: In-network aggregation trade-offs for data collection in wireless sensor networks. In: *International Journal on Sensor Networks* 1 (2006), Nr. 3/4, S. 200–212. – DOI http://dx.doi.org/10.1504/IJSNET.2006.012035. – ISSN 1748–1279

[107] Srivastava, Mani ; Muntz, Richard ; Potkonjak, Miodrag: Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving environments. In: *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA : ACM, 2001. – DOI http://doi.acm.org/10.1145/381677.381690. – ISBN 1–58113–422–3, S. 132–138

[108] Stankovic, John A.: Research challenges for wireless sensor networks. In: *SIGBED Rev.* 1 (2004), Nr. 2, S. 9–12. – DOI http://doi.acm.org/10.1145/1121776.1121780

[109] Stathopoulos, Thanos ; Heidemann, John ; Estrin, Deborah: A Remote Code Update Mechanism for Wireless Sensor Networks / University of California, Los

Angeles, Center for Embedded Networked Computing. Version: November 2003. `http://www.isi.edu/~johnh/PAPERS/Stathopoulos03b.html`. 2003 (CENS-TR-30). – Forschungsbericht

[110] Tian, Di ; Georganas, Nicolas D.: A Coverage-Preserving Node Scheduling Scheme for Large Wireless Sensor Networks. In: *Proceedings of the ACM Workshop on Wireless Sensor Networks and Applications*. Atlanta, Oktober 2002

[111] Timm, Ingo J.: *Intelligent Software Agents for Autonomous Logistic Processes*. Colloquium of the SPP 1183 Organic Computing, 2006

[112] Tran, Son D. ; Davis, Larry S.: Object Tracking at Multiple Levels of Spatial Resolutions. In: *ICIAP '07: Proceedings of the 14th International Conference on Image Analysis and Processing*, 2007, S. 149–154

[113] Tsai, Hsiao-Ping ; Yang, De-Nian ; Peng, Wen-Chih ; Chen, Ming-Syan: Exploring Group Moving Pattern for an Energy-Constrained Object Tracking Sensor Network. In: *The 11th Pacific-Asia Knowledge Discovery and Data Mining conference (PAKDD)*, 2007, S. 825–832

[114] Tseng, Vincent S. ; Lu, Eric Hsueh-Chan: Energy-efficient real-time object tracking in multi-level sensor networks by mining and predicting movement patterns. In: *Journal of Systems and Software*. 82 (2009), Nr. 4, S. 697–706. – DOI http://dx.doi.org/10.1016/j.jss.2008.10.011. – ISSN 0164–1212

[115] University of Southern California, Information Sciences Institute (ISI): *Ns-2: Network simulator-2*. 2010. – http://www.isi.edu/nsnam/ns/

[116] Wang, Limin: MNP: multihop network reprogramming service for sensor networks. In: *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA : ACM, 2004. – DOI http://doi.acm.org/10.1145/1031495.1031538. – ISBN 1–58113–879–2, S. 285–286

[117] Wang, Qiang ; Zhu, Yaoyao ; Cheng, Liang: Reprogramming wireless sensor networks: challenges and approaches. In: *IEEE Network* 20 (2006), Nr. 3, S. 48–55

[118] Warneke, Brett ; Last, Matt ; Liebowitz, Brian ; Pister, Kristofer S.: Smart Dust: Communicating with a Cubic-Millimeter Computer. In: *Computer* 34 (2001), S. 44–51. – DOI http://doi.ieeecomputersociety.org/10.1109/2.895117. – ISSN 0018–9162

[119] Woo, Alec ; Tong, Terence ; Culler, David E.: Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In: *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, ACM Press, 2003, S. 14–27

[120] Xie, Jason ; Talpade, Rajesh R. ; Mcauley, Anthony ; Liu, Mingyan: AMRoute: ad hoc multicast routing protocol. In: *Mobile Network Applications* 7 (2002),

December, Nr. 6, S. 429–439

[121] Xu, Ning: A Survey of Sensor Network Applications. In: *IEEE Communications Magazine* 40 (2002)

[122] Xu, Yingqi ; Winter, Julian ; Lee, Wang-Chien: Dual prediction-based reporting for object tracking sensor networks. In: *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, 2004. – DOI 10.1109/MOBIQ.2004.1331722, S. 154–163

[123] Xue, F. ; Kumar, P. R.: The number of neighbours needed for connectivity of wireless networks. In: *IEEE Wireless Networks* 10 (2004), Nr. 2, S. 169–181

[124] Yang, Haiming ; Sikdar, Biplab: A protocol for tracking mobile targets using sensor networks. In: *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, 2003. – DOI 10.1109/SNPA.2003.1203358, S. 71–81

[125] Ye, Wei ; Heidemann, John ; Estrin, Deborah: Medium access control with coordinated adaptive sleeping for wireless sensor networks. In: *IEEE/ACM Transactions on Networking* 12 (2004), Nr. 3, S. 493–506. – DOI http://dx.doi.org/10.1109/TNET.2004.828953. – ISSN 1063–6692

[126] Yu, Yang ; Rittle, Loren J.: Supporting concurrent applications in wireless sensor networks. In: *Conference On Embedded Networked Sensor Systems*, ACM Press, 2006, S. 139–152

[127] Zhang, Hongwei ; Arora, Anish ; Choi, Young ri ; Gouda, Mohamed G.: Reliable bursty convergecast in wireless sensor networks. In: *MobiHoc '05: Proceedings of the 3rd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing*, ACM Press, 2005, S. 266–276

[128] Zhao, Feng ; Guibas, Leonidas: *Wireless Sensor Networks: An Information Processing Approach (The Morgan Kaufmann Series in Networking)*. Elsevier/Morgan-Kaufmann, 2004. – ISBN 1558609148

[129] Zheng, Rong ; Hou, Jennifer C. ; Sha, Lui: Asynchronous Wakeup for Ad Hoc Networks. In: *MobiHoc '03: Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2003

[130] Zheng, Tao ; Radhakrishnan, Sridhar ; Sarangan, Venkatesh: PMAC: An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. In: *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 12*. Washington, DC, USA : IEEE Computer Society, 2005. – DOI http://dx.doi.org/10.1109/IPDPS.2005.344. – ISBN 0–7695–2312–9, S. 237.1

[131] Zhou, Gang ; He, Tian ; Krishnamurthy, Sudha ; Stankovic, John A.: Impact of radio irregularity on wireless sensor networks. In: *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services.* New York, NY, USA : ACM Press, 2004. – DOI http://doi.acm.org/10.1145/990064.990081. – ISBN 1–58113–793–1, S. 125–138

[132] Zhou, Gang ; He, Tian ; Krishnamurthy, Sudha ; Stankovic, John A.: Models and solutions for radio irregularity in wireless sensor networks. In: *ACM Transactions on Sensor Networks* 2 (2006), Nr. 2, 221–262. http://doi.acm.org/10.1145/1149283.1149287

[133] Zuniga, Marco ; Krishnamachari, Bhaskar: Analyzing the Transitional Region in Low Power Wireless Links. In: *First IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON)*, 2004

Anhang C

Persönliche Informationen

Eigene Publikationen

[1] Baumgartner, Tobias ; Chatzigiannakis, Ioannis ; Fekete, Sándor P. ; Fischer, Stefan ; Koninis, Christos ; Kröller, Alexander ; Krüger, Daniela ; Mylonas, Georgios ; Pfisterer, Dennis: Distributed Algorithm Engineering for Networks of Tiny Artifacts. In *Computer Science Review*, Vol.5, No.1, 2011

[2] Buschmann, Carsten ; Krüger, Daniela ; Fischer, Stefan: Lean and Robust Phenomenon Boundary Approximation. In *Proc. of the 12th IEEE International Conference on Emerging Technologies and Factory Automation, Patras, Greece*, 2007.

[3] Dudek, Denise ; Haas, Christian ; Kuntz, Andreas ; Zitterbart, Martina ; Krüger, Daniela ; Rothenpieler, Peter ; Pfisterer, Dennis: A Wireless Sensor Network For Border Surveillance (Demo). In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, Berkeley, CA* (2009)

[4] Krüger, Daniela ; Buschmann, Carsten ; Fischer, Stefan: Topology recognition in Wireless Sensor Networks. In *The Fourth International Conference on Mobile System, Applications, and Services* (Juni 2006).

[5] Krüger, Daniela ; Buschmann, Carsten ; Fischer, Stefan: Discovering topologies in Wireless Sensor Networks. In *5. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze* (2006).

[6] Krüger, Daniela ; Buschmann, Carsten ; Fischer, Stefan: Object Tracking on Graph Structures. In *Proceedings of the 3rd IEEE European Conference on Smart Sensing and Context (EuroSSC'08)* (2008).

[7] Krüger, Daniela ; Fischer, Stefan ; Buschmann, Carsten: Solar Power Harvesting - Modeling and Experiences. In *8. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze* (2009).

[8] Krüger, Daniela ; Fischer, Stefan ; Buschmann, Carsten: Solar Powered Sensor Network Design and Experimentation. In *The Sixth International Symposium on Wireless Communication Systems 2009* (2009).

[9] Krüger, Daniela ; Haas, Christian ; Rothenpieler, Peter ; Dudek, Denise ; Pfisterer, Dennis: Security in border control and area monitoring. In *it - Information Technology, Themenheft Sicherheit in und durch Sensornetze 52* (2010), Dez., Nr. 6.

[10] Krüger, Daniela ; Pfisterer, Dennis ; Buschmann, Carsten: Selective Reprogramming in Sensor Networks. In *Wireless Sensor Networks - theory and practice (WSN'2011)*.

[11] Krüger, Daniela ; Pfisterer, Dennis ; Fischer, Stefan: CUPID - Communication pattern informed Duty Cycling in Sensor Networks. In *Proceedings of the Fifth International Conference on Systems and Networks Communications (ICSNC 2010)* (Aug. 2010) – Best paper award.

[12] Rothenpieler, Peter ; Krüger, Daniela ; Pfisterer, Dennis ; Fischer, Stefan ; Dudek, Denise ; Haas, Christian ; Kuntz, Andreas ; Zitterbart, Martina: FleGSens - secure area monitoring using wireless sensor networks. In *Proceedings of the International Conference on Sensor Networks, Information, and Ubiquitous Computing (ICSNIUC 2009)* (2009), pp. 81–92.

[13] Rothenpieler, Peter ; Krüger, Daniela ; Pfisterer, Dennis ; Fischer, Stefan ; Dudek, Denise ; Haas, Christian ; Kuntz, Andreas ; Zitterbart, Martina: FleGSens - secure area monitoring using wireless sensor networks. In *Proceedings of the 4th Safety and Security Systems in Europe* (2009), pp. 136–139.