Studienarbeit

Lokale Datenaggregation für mehrere Senken in Sensornetzen

von

Bernd-Christian Renner

Matrikel-Nummer: 30539

April 2007

Referent

Prof. Dr. Volker Turau

Institut für Telematik

Technische Universität Hamburg-Harburg, Deutschland

Korreferent

Dr. Kay Römer

Institut für Pervasive Computing

Eidgenössische Technische Hochschule Zürich, Schweiz

Institut für Telematik

Technische Universität Hamburg-Harburg

TUHH

Danksagung

Besonderen Dank möchte ich meinem Betreuer Christoph Weyer für die zahlreichen Besprechungen, Diskussionen, Anregungen und konstruktive Kritik aussprechen. Ein weiteres Dankeschön gilt meinen Eltern, die trotz der unvertrauten Materie die Lektüre der folgenden Seiten nicht gescheut haben, um mich beim Finden und Beseitigen von Fehlern während der Erstellung dieser Arbeit tatkräftig zu unterstützen.

Eidesstattliche Erklärung

Ich, BERND-CHRISTIAN RENNER (Student des Informatik-Ingenieurwesens an der Technischen Universität Hamburg-Harburg, Matrikelnummer 30539), versichere an Eides statt, dass ich die vorliegende Studienarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, den 13. April 2007

Bernd-Christian Renner

Abstract

In der jüngeren Vergangenheit hat sich eine neue Anwendungsklasse in drahtlosen Sensornetzen entwickelt: Mehrere Senken sammeln Daten von Knoten, die sich in einer lokalen Umgebung befinden, ein und verarbeiten sie. Eine spezielle Anwendung dieser Klasse ist das Generieren häufig auftretender verteilter räumlicher Ereignismuster in drahtlosen Sensornetzen. Die vorliegende Studienarbeit entwickelt zwei unterschiedliche, für diese Anwendung optimierte Routing-Verfahren zum Einsammeln der Daten von Knoten in einer lokalen Umgebung. Hierbei handelt es sich um Fluten und Spannbaum-Routing. Die entstehenden Algorithmen werden als TinyOS-Applikationen implementiert, die zur Durchführung verschiedener Simulationen verwendet werden, aber auch für den Einsatz in realen Sensornetzen konzipiert sind. Auf Basis der Simulationsergebnisse erfolgt ein Vergleich beider Verfahren miteinander sowie eine Bewertung ihrer tatsächlichen Eignung für die Applikation. Hierbei lässt sich feststellen, dass Fluten in Netzen mit hoher Senkendichte eine höhere Erfolgsquote verspricht. Spannbaum-Routing hingegen eignet sich bei geringerer Dichte der Senken: Es bietet Erfolgsquoten von 70 bis 80 Prozent und erreicht eine höhere Energieeffizienz. Allerdings ergeben sich bisher ungelöste Probleme beim Einsatz in dichten Sensornetzen: Fluten erzeugt viele Paketkollisionen; dem Spannbaum-Routing stehen nur kleine Bäume zur Verfügung. Beide Verfahren erzielen somit nur niedrige Erfolgsquoten. Aus diesen Erkenntnissen ergeben sich interessante Forschungsrichtungen: Die Entwicklung neuer Nachbarschaftsprotokolle, die auch in Sensornetzen hoher Dichte zuverlässig die Nachbarn eines Knoten identifizieren, verspricht eine erhebliche Verbesserung der Erfolgsquote beim Spannbaum-Routing. Ein weitere, vielversprechende Richtung wäre die Entwicklung eines adaptiven Verfahrens auf Grundlage des Spannbaum-Routings zur Steigerung der Effizienz und Erfolgsquote.

Inhaltsverzeichnis

Liste verwendeter Zeichen

$\mathcal{K}$	Die Menge der Sensorknoten, die zu einem Sensornetz gehören
$\mathbf{k}$	Ein einzelner Sensorknoten aus einem Sensornetz: $\mathbf{k} \in \mathcal{K}$
$\mathcal{S}$	Die Menge der Senken eines Sensornetzes: $\mathcal{S} \subseteq \mathcal{K}$
$\mathcal{N}_{\mathbf{k}}$	1-Hop Nachbarschaft des Knoten $\mathbf{k}$; $\mathbf{k}$ kann mit allen seinen Nachbarn $\tilde{\mathbf{k}} \in \mathcal{N}_{\mathbf{k}}$ direkt kommunizieren
$Q_{\mathbf{k}\leftarrow\tilde{\mathbf{k}}}$	Qualität des Links von einem Knoten $\tilde{\mathbf{k}}$ zum Knoten $\mathbf{k}$. Wertebereich $[0, 1]$ mit dem Maximum 1
$\mathcal{B}$	Menge der Wurzeln verschiedener Spannbäume, zu denen ein Knoten gehört
τ	Eine für alle Knoten gleiche, diskrete Einteilung der Zeit (Epoche)
$\mathcal{H}$	Die Menge derjenigen Knoten, von denen bereits Pakete in einer Epoche empfangen bzw. weitergeleitet wurden (Historie)
$\mathcal{E}$	Die Menge aller Ereignisse, die von den Knoten eines Sensornetzes beobachtet werden
$\mathcal{E}_L$	Linksseitige Ereignisse: Eine Teilmenge $\mathcal{E}_L \subseteq \mathcal{E}$, die bei der Formulierung von Ereignismustern verwendet wird
$\mathcal{E}_R$	Rechtsseitige Ereignisse: Eine Teilmenge $\mathcal{E}_R \subseteq \mathcal{E}$, die bei der Formulierung von Ereignismustern verwendet wird
pos	Position eines Sensorknotens $\mathbf{k}$, z.B. seine x, y-Koordinaten
$daten$	Abkürzung für die auf einem Knoten über mehrere Epochen hinweg zusammengefassten Informationen bzgl. des Auftretens von Ereignissen
$\mathcal{P}$	Ein Puffer zum Zusammenfassen mehrerer Funkpakete, die als ein einzelnes Meta-Paket verschickt werden
χ	Timer: Löst nach Ablauf eines zugewiesenen Zeitintervalls z.B. ein Ereignis aus ($\varnothing$, wenn inaktiv)

Kapitel 1

Einleitung

Drahtlose Sensornetze [RM03] haben in der näheren Vergangenheit große Aufmerksamkeit und einen gesteigerten Stellenwert bei der Datenerhebung und -gewinnung in zahlreichen Szenarien gewonnen. Hierbei handelt es sich um hochintegrierte, batteriebetriebene Kleinstrechner – sog. Sensorknoten oder Knoten –, die im Wesentlichen mit verschiedenen Sensoren – hierunter fallen z.B. Temperatur-, Licht- oder Feuchtigkeitssensoren – und einem Funkmodul ausgestattet sind. Technischer Fortschritt und wachsendes Forschungs- sowie Industrieinteresse haben die Produktion solcher Geräte ansteigen lassen und dadurch verbilligt. Aufgrund dessen werden neue Anwendungsgebiete erschlossen und Szenarien mit einer Vielzahl von Knoten finanzier- und realisierbar.

Es handelt sich noch immer um ein recht junges Gebiet, in dem viele Aspekte nur wenig oder gar nicht erforscht sind. Mit jeder neuen Anwendung werden neue Erkenntnisse gewonnen, wobei diese selbstverständlich auch neue Fragestellungen aufwerfen. Eines dieser neueren Gebiete stellt die Motivation der vorliegenden Arbeit dar. So lassen sich Sensornetze beispielsweise hervorragend zur Beobachtung von Umwelteinflüssen und anderen physikalischen Ereignissen einsetzen. Dabei ist es oft sehr aufwändig und umständlich, die Flut an gewonnenen Daten aus dem Netz herauszutransportieren und dann offline zu analysieren. In vielen Fällen bietet es sich daher an, die Berechnung bzw. Verarbeitung der Daten innerhalb des Netzes durchzuführen. Dies ist insbesondere dann der Fall, wenn ein Knoten hierzu nur Daten aus einer lokalen Umgebung verwendet. So wird einerseits der Aufwand des Datenversands reduziert und andererseits ermöglicht, die Resultate dieser Berechnungen unmittelbar im Netz zu verwenden. Hierbei kann es sich zum Beispiel um das Ableiten statistischer Aussagen und Modelle handeln. Ein solcher Ansatz wird derzeit an der ETH Zürich verfolgt und in [Röm06a] dokumentiert.

Im Rahmen einer solchen Anwendung ist es erforderlich, die auf den Knoten gewonnenen Daten an spezielle Knoten weiterzuleiten, die dann die Berechnung der statistischen Modelle übernehmen. Hierbei werden die Daten in der Regel nicht direkt an den Emp-

fänger versandt, sondern gelangen über mehrere Hops hinweg dorthin. Begründet ist dies einerseits in der beschränkten Funkreichweite von Sensorknoten und andererseits im Energieverbrauch, der mit der Erhöhung der Reichweite ansteigt. Der Versand in Form von Weiterleitung über mehrere Knoten hinweg erfordert auf der einen Seite den Einsatz von Routing-Algorithmen und bringt damit eine erhöhte Komplexität mit sich. Auf der anderen Seite lässt sich durch geeignete Algorithmen und Datenaggregation Energie sparen, indem die Anzahl tatsächlich verschickter Datenpakete im Vergleich zu einfachem Routing, das diese Aspekte ignoriert, verringert wird. Genau dies ist der Ansatzpunkt der vorliegenden Arbeit. Die Applikation in [Röm06a] bietet ausgezeichnete Möglichkeiten zur Datenaggregation, weswegen ein Routing-Verfahren eingesetzt werden sollte, das diesen Umstand ausnutzt und damit ein ressourcenschonendes Einsammeln der Daten ermöglicht. Es existieren bereits viele Algorithmen zum Routing – auch solche, die Datenaggregation unterstützen. Diese sind jedoch auf spezifische Problemstellungen restriktiert oder berücksichtigen die durch die vorliegende Anwendung gestellten Anforderungen nicht oder nicht ausreichend. Daher ist eine weitere, grundlegende Untersuchung von Routing-Verfahren im Kontext der betrachteten Anwendung erforderlich. Dabei sollen verschiedene Algorithmen implementiert, getestet und ihre Leistungsfähigkeit unter den speziellen Bedingungen und Anforderungen der Anwendung bewertet und verglichen werden.

Bei der in dieser Arbeit angestrebten Analyse von Routing-Algorithmen sollen zwei vom Prinzip her gegensätzliche Ansätze implementiert, auf die Bedürfnisse der Rahmenanwendung angepasst und schließlich verglichen werden. Bei den Vergleichskriterien handelt es sich einerseits um den durch die Algorithmen produzierten Datenverkehr. Andererseits stellt die Anzahl der erhaltenen Daten an den Auswertungsknoten – den sog. Senken – ein wesentliches Indiz zur Beurteilung der Leistungsfähigkeit der betrachteten Verfahren dar. Bei diesen handelt es sich um das vergleichsweise naive Fluten einerseits und das Spannbaum-Routing andererseits. Vor der Implementierung dieser Verfahren sollen weitere, aktuelle Ansätze einbezogen und diskutiert werden. Als Resultat der Arbeit lässt sich eine Aussage darüber erwarten, welches der prinzipiell gegensätzlichen Verfahren unter welchen Umständen die beste Performance bietet und wie Optimierungen und Anpassungen aussehen können, um die Leistungsfähigkeit zu steigern.

Der Rest der Arbeit ist wie folgt aufgebaut: Kapitel 2 beschreibt die angesprochene Anwendung und das in dieser Arbeit behandelte Einsammeln der Daten in Form spezieller Routing-Mechanismen. Allgemeine und hierzu verwandte Arbeiten werden in Kapitel 3 vorgestellt. Kapitel 4 liefert das Design der Algorithmen zum Einsammeln der Daten, Kapitel 5 deren konkrete Implementierung. Die Analyse sowie Simulationsergebnisse stellen den Inhalt von Kapitel 6 dar. Kapitel 7 liefert abschließend eine Zusammenfassung.

Kapitel 2

Problemstellung

In diesem Kapitel sollen zwei Aspekte fokussiert werden: Zum einen wird die in [Röm06a] ausführlich dokumentierte Applikation vorgestellt und zum anderen der Kern der vorliegenden Arbeit hiervon abgeleitet: Das Einsammeln der Daten von Knoten durch speziell zur Verarbeitung vorgesehene Knoten.

2.1 Die Applikation

Die Rahmenanwendung, die der vorliegenden Arbeit als Motivation dient, befasst sich mit dem „Generieren häufig auftretender verteilter räumlicher Ereignismuster in drahtlosen Sensornetzen“ – oder im englischen Originaltitel: *Distributed Mining of Spatio-Temporal Event Patterns in Wireless Sensor Networks*. Hiermit stehen gleich verschiedene Aspekte im Vordergrund, die im Anschluss an ein Beispiel zunächst kurz erläutert werden sollen.

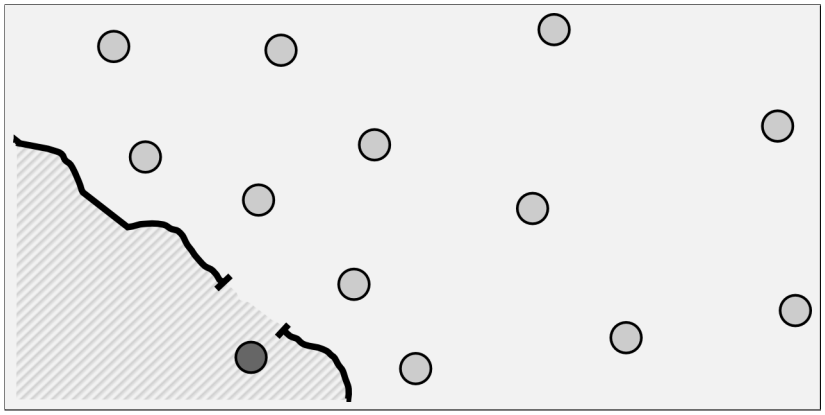


Abbildung 2.1: Brutstätte in der Höhle (schraffiert) und Umgebung. Am Eingang (unterbrochener Umrandungsteil) befindet sich ein Sensor (dunkel) zur Beobachtung des Ereignisses „Vogel verlässt Nest“, die anderen Sensoren (hell) beobachten die Ereignisse „Erschütterung liegt vor“.

Eine einfache Anwendung könnte sich z.B. mit dem Verhalten einer Seevogelart befassen, die ihre Nester in Höhlen anlegt. Ziel der Untersuchung sei es nun, zu erforschen, ob ein Zusammenhang zwischen dem Verlassen der Höhle und Erschütterungen vor dem Eingang besteht. Formuliert werden kann dies durch ein sog. *Ereignismuster*: „Wenn eine Erschütterung vorliegt, verlässt der Vogel die Höhle“. Dieses Muster besteht aus den Ereignissen „Vogel verlässt Höhle“ und „Erschütterung liegt vor“. Um eine derartige Untersuchung durchzuführen, werden Erschütterungssensoren vor der Höhle positioniert, um die Erschütterungen zu registrieren. Zudem wird am Eingang der Höhle ein Sensor befestigt, der feststellt, wann der Vogel die Höhle verlässt. Alle Sensoren gehören zu einzelnen Sensorknoten, die per Funk kommunizieren können. Ein entsprechender Aufbau ist in Abbildung 2.1 dargestellt.

Die Funktion der Anwendung ist nun folgende: Der Sensor, der das Ereignis „Vogel verlässt Höhle“ beobachtet, sammelt die Messdaten der umliegenden Knoten ein. Dies kann periodisch passieren oder nur dann, wenn der Vogel die Höhle überhaupt verlassen hat. Tritt das Ereignis „Vogel verlässt Höhle“ nur selten auf, führt dies zur Verringerung des Funkverkehrs. Zulässig ist dieses Verfahren deshalb, weil für die Auswertung (vgl. [Röm06a]) der linke Teil des Musters nur dann geprüft werden muss, wenn der rechte Teil überhaupt beobachtet wurde.

Im verbleibenden Teil dieses Abschnittes wird nun zunächst das in [Röm06a] beschriebene Verfahren näher erläutert, bevor dann auf den Anknüpfungspunkt zur vorliegenden Arbeit in Abschnitt 2.2 eingegangen wird.

2.1.1 Anforderungen

Die grundlegenden Anforderungen bzgl. der Infrastruktur der Anwendung offenbaren praktisch keine Unterschiede zu den meisten anderen Szenarien drahtloser Sensornetze: Benötigt wird eine Menge $\mathcal{K}$ drahtloser Sensorknoten, die über Funk miteinander kommunizieren. Die einzelnen Knoten sind dabei mit passenden Sensoren zur Beobachtung bestimmter Ereignisse ausgestattet, den Daten des Netzes.

Im betrachteten Verfahren findet die eigentliche Auswertung dieser Daten auf speziellen Knoten im Netz statt. Diese Knoten werden als *Senken* bezeichnet und stellen eine Teilmenge $\mathcal{S} \subseteq \mathcal{K}$ mit erweitertertem Funktionsumfang dar. Eine Senke kann einer anderen damit als „einfacher“ Knoten dienen, der Messdaten liefert. Im Gegensatz zu anderen Definitionen in der Literatur sind diese Senken nicht mit einem PC verbunden, an den die gesammelten Daten übertragen werden. Stattdessen speichern und verarbeiten die Senken die gesammelten Daten autonom. Hierdurch wird die gesamte Anwendung, die aus Daten-

erfassung, -sammlung und -verarbeitung besteht, innerhalb des Netzes ausgeführt. Damit stehen dem Netz diese Resultate unmittelbar zur Verfügung – und zwar bereits im laufenden Betrieb. Hierdurch wird z.B. eine direkte Reaktion auf Ereignisse im Netz mittels sogenannter Aktuatoren (wie in [CMP06] beschrieben) ermöglicht.

Damit eine Senke überhaupt die Daten anderer Knoten zur Analyse und Berechnung verwenden kann, muss sie diese Daten einsammeln. In einem großen Sensornetz mit vielen Senken sind die einzelnen Senken in der Regel nur an Messdaten aus einem kleinen Umkreis interessiert. Nur aus dieser Teilmenge des Datengesamtvolumens des Netzes sollen Ereignismuster und Zusammenhänge abgeleitet werden. Im eingangs verwendeten Beispiel sollte eine am Eingang der Höhle positionierte Senke nur Daten von Knoten innerhalb einer vorher (manuell) festgelegten Entfernung einsammeln und zur Verarbeitung verwenden, da Erschütterungen in hoher Entfernung sicherlich keinen Einfluss auf das Verhalten des Vogels in der Höhle haben. Eine lokale Begrenzung ist auch dann notwendig, wenn viele dicht beeinander liegende Höhlen gleichzeitig überwacht werden sollen. Die Sensorknoten der einzelnen Überwachungsgebiete würden dann ein einziges Netz formen, in dem die Senken jedoch nur an den Informationen von Knoten in ihrer lokalen Umgebung Interesse hätten. Dabei können einige Knoten mehreren Senken als Datenlieferanten dienen. Durch die Festlegung eines maximalen Einzugsgebietes wird bestimmt, welche Knoten dies sind bzw. welche Knoten welchen Senken als Datenlieferanten dienen können.

2.1.2 Messdaten und Ereignisse

Zur Gewinnung der gewünschten statistischen Aussagen und Modellierung werden die Daten zunächst klassifiziert. Anstatt die absoluten Messwerte der einzelnen Sensoren zu betrachten, werden einzelne Ereignisse definiert: $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$. Dabei handelt es sich beispielsweise um das Überschreiten bestimmter Schwellwerte: Eine Erschütterung liegt in obigem Seevogelexperiment z.B. nur dann vor, wenn der entsprechende Sensor eines Knotens einen festgelegten Minimalwert überschreitet. Hieran schließt sich unmittelbar die Fragestellung an, in welchen Abständen es sinnvoll ist, ein Ereignis auszulösen. So ist man beispielsweise lediglich daran interessiert, ob innerhalb einer gewissen Zeiteinheit, z.B. einer Minute, eine Erschütterung beobachtet wurde oder nicht. Dabei ist die jeweilige Diskretisierungseinheit, eine sog. *Epoche*, so zu wählen, dass für die Analyse des betrachteten Ereignismusters keine Beeinflussung zu erwarten ist – sie ist also abhängig vom jeweiligen Experiment. Eine formale Beschreibung der Ereignisse in einer Epoche kann wie folgt beschrieben werden:

Gegeben sei eine Ereignismenge $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ sowie zeitlich diskrete Epochen $\tau_1, \tau_2, \ldots$. Jeder Epoche τ_i wird ein Vektor $v^i \in \{0,1\}^{|\mathcal{E}|}$ zugeordnet mit

$$v_j^i = \begin{cases} 1 & \text{falls das Ereignis } e_j \text{ in der Epoche } \tau_i \text{ aufgetreten ist} \\ 0 & \text{sonst} \end{cases} \tag{2.1}$$

Für jeden Sensorknoten eines zusammenhängenden Netzes werden die Ereignisse separat erfasst. Dabei soll davon ausgegangen werden, dass die Menge der Knoten mit Ausnahme der zusätzlichen Fähigkeiten der Senken homogen ist bzgl. der Fähigkeiten und erfassbaren Ereignisse. Ein solcher Vektor v^i existiert dann für jeden Knoten $\mathrm{k} \in \mathcal{K}$ und für jede Epoche τ_i.

Die Verwendung der Ereignisse in den Ereignismustern erfolgt als *linksseitige* und *rechtsseitige Ereignisse*. Das linksseitige wird hierbei gewöhnlich durch ein „wenn“ eingeleitet und steht im vorderen (linken) Teil der Beziehung, die ein Ereignismuster ausdrückt. Ein Ereignismuster hat demnach für gewöhnlich die Gestalt „Wenn L, dann R“. Diese Formulierung erinnert formal an eine mathematische Implikation.

2.1.3 Partitionierungen

Gemäß Aufgabenstellung sammelt eine Senke Daten von anderen Knoten ein, um statistische Aussagen über die zu betrachtenden Ereignismuster treffen bzw. die dahinterliegenden mathematischen Berechnungen durchführen zu können. Dazu wird eine Menge $\mathcal{E}_L \subseteq \mathcal{E}$ linksseitiger sowie eine Menge $\mathcal{E}_R \subseteq \mathcal{E}$ rechtsseitiger Ereignisse definiert. Unter Verwendung statistischer Analyseverfahren kann nun die Korrelation zwischen den einzelnen Ereignissen in $\mathcal{E}_L$ und denen in $\mathcal{E}_R$ berechnet werden.

Beobachtet die Senke also ein bestimmtes Ereignis in einer beliebigen Epoche, verwendet sie die Ereignisvektoren v aus einer festgelegten Umgebung zur Bewertung der zu observierenden Zusammenhänge. Zur Minimierung des Aufwandes in Form von Rechenzeit und Komplexität werden hierbei weitere Vereinfachungen angenommen. Die Menge aller Knoten wird – von der Senke aus betrachtet – in verschiedene Partitionen bzgl. ihrer Distanz unterteilt. Ein Beispiel hierfür ist die Verwendung der beiden Partitionen *nah* und *fern* mit den entsprechenden Definitionen *nah*$\triangleq (0m, 10m]$, *fern*$\triangleq (10m, 20m]$. Nur die Ereignisvektoren der Knoten mit Distanz zur Senke aus diesen beiden Partitionen werden zur Berechnung der Korrelationen berücksichtigt. Die Einteilung in diese zwei Partitionen ermöglicht es, die Korrelationen in Abhängigkeit der Entfernung bzw. der Distanz-Partitionen zu bestimmen.

Bezüglich der Zeit wird analog verfahren. Tritt ein Ereignis an einer Senke auf, verwendet die Senke nicht nur die aktuellen Ereignisvektoren entfernter Knoten, sondern diejenigen der festgelegten Zeitpartitionen. Dies könnten beispielsweise *jetzt*, *kürzlich* und *alt* sein. Eine Zeitpartition umfasst dabei mehrere Epochen. Wiederum werden hierfür Intervalle definiert, z.B. *jetzt*$\triangleq \{\tau_i\}$, *kürzlich*$\triangleq \{\tau_{i-2}, \tau_{i-1}\}$, *alt*$\triangleq \{\tau_{i-7}, \ldots, \tau_{i-3}\}$. Der Index i bezeichnet dabei die Epoche, in der das Ereignis von der Senke beobachtet wurde.

Die dritte Einteilung dieser Art betrifft die Häufigkeit des Auftretens von Ereignissen. Hier heißen die Partitionen z.B. *keine* und *einige* und sind trivialerweise definiert als *keine*$\triangleq \{0\}$, *einige*$\triangleq \{1, \ldots, \infty\}$. Die Zahlen in den Mengen geben hier die absolute Auftrittshäufigkeit an.

Der Sinn der Partitionierungen ist es nun, die Ereignismuster im Kontext bestimmter Randbedingungen zu evaluieren. Bei den Randbedingungen handelt es sich um die verschiedenen Partitionierungen. Das einfache Muster „Wenn eine Erschütterung vorliegt, verlässt der Vogel die Höhle“ kann somit differenzierter betrachtet werden. Es könnte z.B. der Einfluss der Entfernung analysiert werden, indem man das Muster einmal nur unter Berücksichtigung der Distanzpartition *nah* auswertet und einmal unter der Partition *fern*.

Die hier vorgestellten Partitionierungen sind lediglich beispielhaft und können fast beliebig gewählt werden, was allerdings in vielerlei Hinsicht zu erhöhter Komplexität führt. So werden die Rechnungen aufwändiger, die Aussagen feiner aber ggf. auch unübersichtlicher oder verlieren gar die statistische Relevanz, wenn durch die feine Granularität nicht genügend Messdaten pro Partition vorhanden sind.

2.2 Einsammeln der Daten

Nach der Einführung in die zugrunde liegende Anwendung und deren Einsatzgebiet folgt nun die hiermit zusammenhängende Darstellung des Themas dieser Arbeit. Während [Röm06a] das Generieren von Ereignismustern beschreibt, beschäftigt sich diese Studienarbeit mit dem Einsammeln der Daten durch die Senken. Die genaue Beschreibung der untersuchten Verfahren zur Datensammlung erfolgt in Kapitel 4. An dieser Stelle werden hingegen allgemeine Beobachtungen und Notwendigkeiten in diesem Zusammenhang beschrieben.

2.2.1 Funkverkehr in Sensornetzen

Damit eine Senke Berechnungen zu vorgegebenen Ereignismustern durchführen kann, muss sie zunächst die auf anderen Knoten gewonnenen Daten per Funk einsammeln. Bei

diesen einzusammelnden Daten handelt es sich um die Vektoren aus Gleichung 2.1. Bei der Realisierung von Verfahren zum Einsammeln sind jedoch die im Folgenden genannten Eigenschaften zu beachten.

Die Knoten eines Sensornetzes unterliegen begrenzten Ressourcen. Hierbei handelt es sich in erster Linie um die limitierte Energie, die von Batterien zur Verfügung gestellt wird. Um eine hohe Laufzeit der einzelnen Knoten zu gewährleisten, sollte so wenig Funkverkehr wie nötig stattfinden, da das Funkmodul den wohl größten Verbraucher eines Sensorknotens darstellt. Daher empfiehlt es sich überdies, die Funkstrecken so kurz wie möglich zu wählen, weil längere Funkstrecken eine höhere Sendeleistung erfordern.

Ein weiteres Problem in Sensornetzen stellen Paketkollisionen dar, die durch das Empfangen überlagerter Pakete von verschiedenen Sendern beim Empfänger auftreten. Zur Umgehung dieser Problematik sollte erstens vermieden werden, dass Knoten, die sich im Einzugsbereich eines gemeinsamen Empfängers befinden, gleichzeitig senden. Zweitens bietet es sich an, die Funkreichweiten der Knoten zu begrenzen.

Die Beschränkung der Funkreichweite verspricht also, die o.g. Probleme zu mindern. Hierdurch ergeben sich jedoch neue Unwägbarkeiten, die gelöst werden müssen. Damit eine Senke auch Daten von Knoten einsammeln kann, die außerhalb ihrer Funkreichweite liegen, müssen die Daten über zwischen der Senke und diesen Knoten liegende Knoten weitergeleitet werden. Dieses Verfahren wird im Allgemeinen als *Multi-Hop Routing* bezeichnet. Jeder Knoten muss damit über die Fähigkeit verfügen, Daten anderen Knoten in geeigneter Weise in Richtung einer oder mehrerer Senken weiterzuleiten. Zwar existieren bereits zahlreiche Routing-Verfahren für Multi-Hop Sensornetze, doch bieten sich diese nicht unmittelbar zum Einsatz in der vorliegenden Anwendung an. Der wesentliche Grund hierfür sind die im nächsten Unterabschnitt behandelten Aggregationsmöglichkeiten bzgl. der versandten Daten. Ein Vergleich mit aktuellen, ähnlichen Verfahren findet sich in Kapitel 3.

Neben der Notwendigkeit des Einsatzes von Multi-Hop Routing muss sich das für das Einsammeln der Daten genutzte Verfahren mit der Problematik der Paketverluste befassen. Neben den Kollisionen können Pakete bei der Übertragung mittels Funk korrumpiert werden, so dass der Empfänger diese nicht verwerten kann. Diesem Problem kann mit zwei Ansätzen begegnet werden. Zum einen können kaputte Pakete einfach verworfen werden, zum anderen können Sie erneut vom Sender angefordert werden. Der zweite Ansatz führt zu erhöhtem Funkverkehr, was einerseits die Batterien der Knoten stärker belastet und andererseits die Wahrscheinlichkeit von Kollisionen erhöhen kann, wenn viele Knoten mit überlappendem Funkbereich ihre Daten mehrfach übertragen (müssen). Daher ist abzuwägen, wie oft und in welcher Weise erneute Übertragungen wünschenswert und nutzbrin-

gend sind. Auf der anderen Seite kann es sich nämlich als problematisch erweisen, die Daten korrumpierter Pakete einfach zu verwerfen, wenn hierdurch die von einer Senke erhaltene gegenüber der möglichen Datenmenge klein wird.

2.2.2 Datenaggregation

Verschickte Datenpakete besitzen neben den eigentlichen Nutzdaten der Applikation zusätzliche Informationen. Hierbei handelt es sich zum Beispiel um Adressierungsinformationen und Prüfsummen zur Fehlererkennung und -korrektur. Die tatsächliche Menge an übertragenen Daten ist damit größer als die zu sendenden Nutzdaten. Insbesondere in Sensornetzen fällt dies ins Gewicht, da Datenpakete üblicherweise nur wenige Bytes groß sind. Folglich kann durch Datenaggregation das Funkdaten-Gesamtvolumen verringert werden, weil diese zusätzlichen Informationen dann nur einmal für mehrere Datensätze mitgeschickt werden müssen. In der vorliegenden Applikation bietet sich eine zweischichtige Aggregation an.

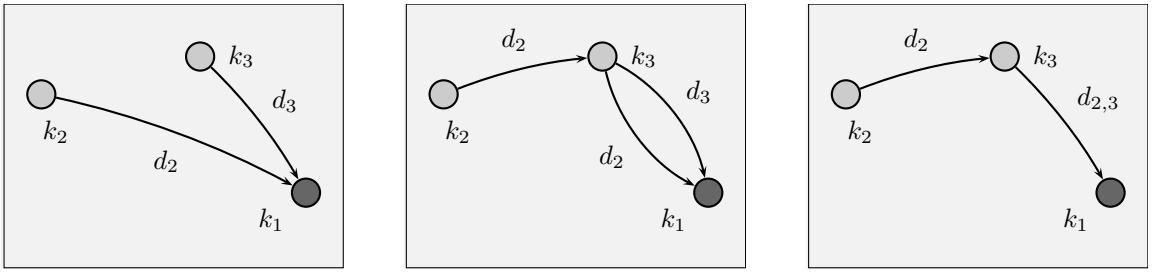


Abbildung 2.2: Direkte Kommunikation (links), einfaches Multi-Hop Routing (Mitte) und Multi-Hop Routing mit Aggregation (rechts). Die Kanten zeigen den Versand von (ggf. aggregierten) Daten d an.

Der erste Teil der Aggregation entsteht aus der Tatsache, dass eine Senke nicht an punktuellen Daten interessiert ist, sondern an Informationen über mehrere Epochen hinweg. Ein Knoten kann damit die Ereignisvektoren mehrerer Epochen in einem einzigen Datenpaket zur Senke schicken. Die einzelnen Vektoren verschiedener Epochen dürfen dabei jedoch nicht zusammengefasst werden, weil die Epochen bei der Partitionierung der zeitlichen Verläufe von Interesse sind.

Der zweite Teil der Aggregation basiert auf dem eingesetzten Multi-Hop Routing. Leitet ein Knoten Daten von einem anderen Knoten in Richtung derselben Senke weiter, muss er seine Daten nicht separat verschicken, sondern kann auf das Eintreffen der weiterzuleitenden Daten warten. Diese fasst er dann geeignet mit den eigenen Daten zusammen und verschickt das Aggregat (siehe Abb. 2.2). Im einfachsten Fall werden die empfange-

nen und eigenen Daten als ein einziges Datenpaket weitergeleitet. Bei der Verwendung eines Spannbaumes kann jedoch auch eine kompaktere Aggregations erreicht werden: Ein Knoten addiert die empfangenen und lokalen Ereignisvektoren (pro Epoche und Distanz-Partition) und schickt die Zähler an seinen Vater im Baum weiter.

2.2.3 Aufgabenstellung

Die in diesem Kapitel zusammengetragenen Informationen und Rahmenbedingungen ergeben sich nun zur konkreten Aufgabenstellung, verschiedene Ansätze zum Einsammeln der Daten auszuwählen, zu implementieren und auf ihre Eignung und Leistungsfähigkeit hin zu evaluieren. Die Implementierung gliedert sich dabei in den allgemeinen, algorithmischen Lösungsansatz bzgl. der einzelnen Verfahren und die konkrete Programmierung für ein Sensornetz mittels einer geeigneten Programmiersprache. Im Rahmen der Evaluierung werden die Verfahren anhand von Simulationsergebnissen verglichen sowie Stärken und Schwächen herausgearbeitet und Verbesserungsideen entwickelt. Insbesondere ergeben sich die folgenden Aufgabenstellungen, die einer eingehenden Untersuchung bedürfen:

Es sollten verschiedene grundlegende Verfahren betrachtet werden, die ein Einsammeln der Daten ermöglichen. Hierzu gehören insbesondere das vergleichsweise naive Fluten sowie das in der Literatur als energieeffizient beschriebene Routing mittels Spannbäumen. Während das Fluten auf den ersten Blick wenige Aggregationsmöglichkeiten anzubieten scheint und hierdurch einen höheren Funkverkehr nach sich ziehen könnte, bietet es andererseits eine gewisse Redundanz, weil Pakete über verschiedene Pfade zu einer Senke gelangen können. Das Routing mittels Spannbaum hingegen verspricht gute Aggregationsmöglichkeiten und geringen Funkverkehr beim Einsammeln der Daten, erfordert jedoch zusätzlichen Aufwand zum Generieren der Bäume. Eine Analyse auf diesem Gebiet erscheint hiermit sinnvoll.

In diesem Kontext spielen auch die maximalen Distanzen eine Rolle, aus der die Senken ihre Daten beziehen dürfen sowie die hierfür zur Verfügung stehende maximale Hop-Anzahl. Ein weiteres Kriterium wird durch die Dichte des Netzes und die Anzahl der Senken definiert. Zu vergleichen sind ebenfalls der Aufwand bei der Umsetzung der Verfahren hinsichtlich der Programmierung sowie die Speicherkomplexität. Schließlich besitzen Sensorknoten vergleichsweise kleine Hauptspeicher, die sich derzeit im Bereich weniger Kilobyte bewegen.

Ein weiterer interessanter Aspekt ist die Frage, ob die Daten aktiv von den Senken eingesammelt oder periodisch von den Knoten in Richtung der Senken geschickt werden sollen. Schließlich muss eine Senke nur dann Informationen einsammeln, wenn tatsächlich

ein rechtsseitiges Ereignis beobachtet wurde. Passiert dies nur selten, lässt sich durch das gezielte Abfragen von Informationen unnötiger Funkverkehr von vornherein unterbinden. Auf der anderen Seite verursacht die aktive Anfrage selbst Datenverkehr, so dass durch entsprechende Versuche eine Erkenntnis abgeleitet werden kann, wann sich welcher Ansatz lohnt.

Neben der reinen Analyse sollen die entwickelten Verfahren letztlich auch in der Praxis einsetzbar sein. Neben schlankem und ressourcensparendem Design bzgl. der Länge des Codes, des Funktionsumfanges und der Wartbarkeit bedingt dies auch eine gewisse Modularität, um einzelne Komponenten einfach anpassen zu können. Daher ist es wünschenswert, für die Implementierung zu Simulationszwecken eine Plattform zu wählen, die es erlaubt, denselben oder leicht modifizierten Code auch in realen Sensornetzen zum Einsatz zu bringen.

Kapitel 3

Grundlegende und verwandte Arbeiten

Das Einsammeln und Verteilen von Daten ist ein zentraler Aspekt in drahtlosen Sensornetzen. Seit einigen Jahren beschäftigt sich die Forschung mit diesem relativ jungen Gebiet, so dass bereits einige Experimente und Erfahrungen vorliegen. Als Einstieg in diese Thematik werden in diesem Kapitel zunächst einfache Anwendungen drahtloser Sensornetze zur generellen Orientierung über deren Einsatzgebiete vorgestellt. Hierauf folgt ein Überblick zu Arbeiten bzgl. der Unwägbarkeiten beim Übertragen von Daten per Funk. Schließlich werden einige spezielle Verfahren eingehend betrachtet, die eine hohe Relevanz und thematische Nähe zur konkreten Aufgabenstellung der vorliegenden Arbeit besitzen.

3.1 Allgemeine Anwendungen drahtloser Sensornetze

Eine Vielzahl von Experimenten und Forschungen in drahtlosen Sensornetzen widmet sich dem Messen von Umwelteinflüssen oder Beobachten von Naturphänomenen. Die Eignung drahtloser Sensornetze hierzu liegt begründet in der geringen Beeinflussung der Messungen durch ihre kompakte Größe und den Umstand, dass die Knoten – einmal platziert – unbeaufsichtigt betrieben werden können und prinzipiell wartungsfrei sind. Um ein Experiment mit drahtlosen Sensornetzen durchzuführen, werden mit passenden Sensoren ausgestattete Knoten an (für das jeweilige Experiment) interessanten Lokalitäten positioniert. Die einzelnen Knoten dienen dann primär der Gewinnung von Messdaten, sekundär leiten sie Daten anderer Knoten weiter.

Um bereits während des Experiments – und ohne physikalischen Zugriff auf jeden einzelnen Knoten – mit den erhobenen Daten arbeiten zu können, müssen diese per Funk ausgelesen werden. Dies geschieht entweder, indem die einzelnen Knoten ihre Daten z.B. periodisch an eine zentrale Stelle schicken, oder alternativ durch das gezielte Abfragen

der Informationen von anderen Knoten. Gemäß Kapitel 2 handelt es sich bei den abfragenden Knoten um die sog. Senken. Je nach Experiment verarbeitet eine Senke die Daten autonom oder leitet diese, z.B. per serieller Schnittstelle, an einen PC weiter. Die in diesem Abschnitt angesprochenen Experimente nutzen die Senken lediglich zum Übertragen der Daten an einen PC, der diese dann speichert oder bereits während des Experiments verarbeitet.

Beispiele für Anwendungen drahtloser Sensornetze gibt es viele. Eine kleine Auswahl hieraus stellen die folgenden Experimente dar: In [MPS+02] werden Sensorknoten eingesetzt, um das Brutverhalten von Vögeln zu observieren. Die in [HHSH06] vorgestellte Anwendung realisiert die Analyse und Beobachtung der Wetterverhältnisse zur Erkennung und Bekämpfung von Wald- und Flächenbränden.

Diese Anwendungen widmen sich ausschließlich der Beobachtung von Naturereignissen bzw. dem Verhalten von Tieren. Die Daten werden erhoben und an Senken weitergeleitet. Die Auswertung findet weitgehend manuell am PC statt. Die Zielsetzung unterscheidet sich damit von dem in Kapitel 2 vorgestellten Verfahren.

3.2 Kommunikation in drahtlosen Sensornetzen

Zum zuverlässigen Einsatz drahtloser Sensornetze ist ein grundlegendes Verständnis ihrer Funktionsweise erforderlich. Von besonderem Interesse gestaltet sich hierbei die Analyse der Kommunikation per Funk. Im Gegensatz zu drahtgebundenen treten in drahtlosen Netzen u.a. die in Kapitel 2.2.1 genannten Phänomene auf, die beispielsweise in [TWW06, TRV+05], [WTC03] und [SPMC04] näher untersucht werden.

Ein wesentlicher Aspekt der drahtlosen Kommunikation ist die Funkreichweite, die über die Sendeleistung gesteuert wird. Wie in Kapitel 2 erläutert, bietet sich zur Reduzierung der Sendeleistung und damit einhergehenden Einschränkung der Funkreichweite das Multi-Hop Routing an, um trotzdem über größere Entfernungen hinweg kommunizieren oder Daten versenden zu können.

Eine mögliche Implementierung dieser Art von Routing kann über die Verwendung von Spannbäumen realisiert werden. Diesen Ansatz verfolgt [TRV+05]. Zum Aufbau der Bäume stellen alle Knoten zunächst fest, mit welchen anderen Knoten eine Kommunikation in beide Richtungen möglich ist, die auch als *bidirektionaler Link* bezeichnet wird. Zu diesem Zweck kommt hier das in [OTL04] beschriebene Verfahren zum Einsatz, das auch in der vorliegenden Arbeit Verwendung findet und in Kapitel 4 kurz erläutert wird. Die Übermittlung der Daten von jedem Knoten zur Senke erfolgt periodisch. Dabei leitet ein Vaterknoten die Daten seiner Söhne einfach den Baum hinauf, bis die Daten schließlich an

der Senke ankommen. Eine Aggregation der Daten zur Einsparung von Funkverkehr findet nicht statt.

Bei der Auswertung in [TWW06, TRV+05] werden ähnliche Probleme identifiziert, wie sie auch in [WTC03] sowie [SPMC04] beschrieben sind. Durch Paketverluste und -kollisionen treten Störungen im Betrieb des Netzes auf. Stabile, bidirektionale Links sind zeitlichen Schwankungen unterlegen, so dass der Funkverkehr über die durch die Bäume festgelegten Pfade im Laufe des Experiments fehlschlägt. Es zeigt sich unter anderem, dass die bidirektionalen Links der Knoten zu verschiedenen Zeitpunkten des Experiments unterschiedlich sind.

Eine weitere Erkenntnis besteht darin, dass die Qualität einer Funkverbindung zwischen zwei Knoten nicht als einfache Funktion der Entfernung beider Knoten voneinander dargestellt werden kann. So kommt es vor, dass in bestimmten Distanzen der Funkverkehr plötzlich spürbar beeinträchtigt wird. Ferner wurde in [TRV+05] festgestellt, dass die Qualität der Funkverbindung oft nicht symmetrisch ist. Viele Links sind nur *unidirektional*.

Paketkollisionen und -fehler stellen zwei weitere Problem dar, die empfangene Daten unbrauchbar machen. Dabei lässt sich feststellen, dass eine Vergrößerung des Funkverkehrs und der Knotendichte zu einer erhöhten Kollisionswahrscheinlichkeit führen. Dies ist nachvollziehbar, da es sich bei den Kollisionen um Überlagerungen von Funksignalen handelt, die zur Störung beim Empfang führen. Einfache Paketfehler können verschiedene Ursachen haben, z.B. schwache Funksignale oder Reflektionen.

Je nach Applikation erweist es sich aufgrund dieser Schwierigkeiten bereits als notwendig, sich mit der Zuverlässigkeit des Funkverkehrs zu beschäftigen. So können fehlerhafte Pakete durch den Empfänger erneut vom Sender angefordert werden. Durch geeignete Verfahren kann außerdem versucht werden, zur Kommunikation in Multi-Hop Netzen die jeweils besten Routen auszuwählen. Dies erhöht jedoch die Anwendungskomplexität und ggf. auch den Funkverkehr. Dies wiederum wirkt sich negativ auf den Energieverbrauch und die damit verbundene Lebenszeit der einzelnen Knoten aus. Es bedarf folglich auch einer Abwägung, wie wichtig der Empfang bestimmter Daten ist oder wie hoch die Rate korrekt empfangener im Verhältnis zur möglichen Datenmenge sein muss, um das Netz ordnungsgemäß arbeiten zu lassen. Diese Punkte spielen auch beim Design der Verfahren in Kapitel 4 eine wichtige Rolle und werden in Kapitel 6 bei der Auswertung aufgegriffen.

3.3 Datenaggregation

Im letzten Abschnitt dieses Kapitels soll nun speziell auf diejenigen Anwendungen eingegangen werden, die in konkretem Zusammenhang mit dem in dieser Arbeit behandelten

Thema stehen. Hierbei handelt es sich um Verfahren, die eine lokale Datenaggregation umsetzen: Einzelne Knoten teilen innerhalb räumlicher oder logischer Grenzen ihre Daten mit anderen Knoten. Dies kann z.B. durch einfaches Versenden der eigenen Daten an andere Knoten geschehen – entweder auf Anfrage oder automatisch. Die grundlegende Anwendung ähnelt damit dem gewünschten Mechanismus zum Einsammeln von Daten in [Röm06a], trotzdem existieren Unterschiede oder Unzulänglichkeiten, die eine einfache Adaption der Verfahren für das hier gewünschte Routing verhindern.

3.3.1 Intelligentes Routing

Das in [TGS06] beschriebene Verfahren widmet sich der Optimierung von Sensordaten-Abfragen einzelner Knoten eines Netzes. Hierbei werden zwei Optimierungen verwendet: Einerseits werden die Abfragen derart aggregiert, dass die Antworten der Knoten minimiert werden (hinsichtlich Größe und Anzahl). Andererseits wird das Routing der Daten zu den Senken mittels Spannbäumen so gewählt, dass der Funkverkehr reduziert wird. Die dazu verwendeten Spannbäume werden durch Fluten des Netzes generiert, wobei jeder Knoten seinen Vater so wählt, dass eine minimale Anzahl von Links verwendet wird, um eine Anfrage an eine Teilmenge der Knoten des Netzes zu beantworten.

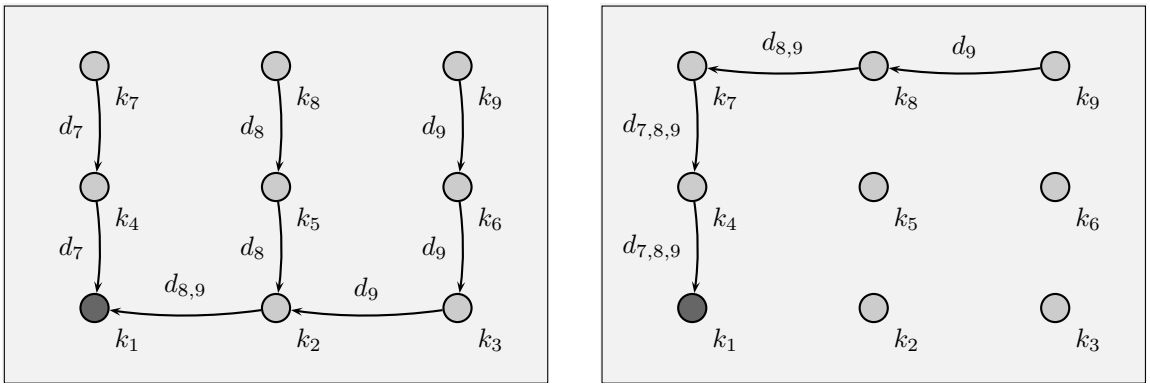


Abbildung 3.1: Multi-Hop Spannbaum-Routing ohne Baum-Optimierung (links) und mit Baum-Optimierung (rechts). Die Kanten zeigen den Versand ggf. aggregierter Datenpakete d an.

Ein Beispiel für eine derartige Optimierung kann Abbildung 3.1 entnommen werden. Die Senke $\mathbf{k_1}$ fordert Daten von den Knoten $\mathbf{k_7}$, $\mathbf{k_8}$ und $\mathbf{k_9}$ an, wobei jeder Knoten ausschließlich mit den jeweils nächstliegenden Knoten in horizontaler bzw. vertikaler Richtung kommunizieren kann. Die Pfeile zeigen an, wie die Daten (unter Berücksichtigung möglicher Aggregationen) verschickt werden. Das Routing im linken Fall stellt eine mögliche, aber nicht optimale Lösung dar (8 Datenpakete), die Daten zur anfragenden Senke

zu transportieren. Der rechte Fall hingegen ist im Sinne der Aufgabenstellung aus [TGS06] optimal (4 Datenpakete).

Die zweite Art der Optimierung betrifft die Ausführungsreihenfolge der Anfragen, die an dieser Stelle nicht näher erläutert werden soll, da sie für die vorliegende Arbeit nicht relevant ist. Während der vorgeschlagene Spannbaum-Ansatz zum optimierten Routing durchaus von Interesse für die Lösung des Routing-Problems aus Abschnitt 2.2.2 ist, findet die Aggregation der Anfragen hier keinerlei Verwendung, da sie die umliegenden Knoten lediglich informieren, dass eine Senke Daten benötigt. Weitere Informationen beinhaltet die Anfrage jedoch nicht.

3.3.2 Programmierabstraktionen für Nachbarschaften

Eine eigene Klasse zum Datenaustausch in drahtlosen Sensornetzen stellen sog. Programmierabstraktionen für Nachbarschaften dar. Unter Nachbarschaften ist in diesem Kontext eine Teilmenge der Knoten eines Netzes zu verstehen, die bestimmte Kriterien erfüllen oder gemeinsam haben. Ein einfaches Beispiel für ein solches Kriterium ist die maximale Distanz zu einem festgelegten Punkt oder Knoten.

Derart definierte Nachbarschaften werden verwendet, um Daten oder Zustände zwischen den zugehörigen Knoten auszutauschen bzw. zu teilen. Weil dieses Verhalten in vielen Applikationen für drathlose Sensornetze eingesetzt wird, existieren verschiedene Ansätze zur Umsetzung solcher Programmierabstraktionen. Sie sollen die Anwendungsentwicklung in drahtlosen Sensornetzen vereinfachen.

Ein Vertreter dieser Klasse ist „Hood“ [WSBC04]. Eine Programmierabstraktion übernimmt hier die Bildung der Nachbarschaften und den Austausch der Daten im laufenden Betrieb. Der Entwickler legt nur fest, welche Kriterien für eine Nachbarschaft gelten und welche Variablen geteilt werden sollen. Zudem bietet „Hood“ verschiedene Schnittstellen an, um den Datenaustausch auch manuell anzustoßen sowie eingehende Daten zu filtern. Eine Besonderheit von „Hood“ ist der Umstand, dass eine Nachbarschaft nicht als Gruppe von Knoten zu verstehen ist, sondern jedem Knoten des Netzes andere Knoten als Nachbarn zugeordnet werden.

Der eigentliche Versand der Daten bleibt vor der Applikation verborgen. „Hood“ realisiert diesen durch einfache Broadcasts und ohne Verwendung von Routing-Mechanismen, d.h. ohne Weiterleitung von Daten über andere Knoten. Die Knoten einer Nachbarschaft müssen damit innerhalb einer gemeinsamen Funkreichweite sein. Dies jedoch widerspricht der Problemstellung in Abschnitt 2.2, so dass „Hood“ keine Lösung für die dort geschilderte Problematik darstellen kann.

Ein weiteres Verfahren zum Datenaustausch innerhalb von Nachbarschaften bieten „Abstract Regions“ [WM04]. Die prinzipielle Funktionsweise ähnelt derjenigen von „Hood“, das verwendete Routing ist jedoch nicht auf einen Hop beschränkt. Als Routing-Methode kann zum Beispiel ein Spannbaum zum Einsatz kommen, dessen Größe durch einen maximalen Abstand zur Wurzel begrenzt wird. Das Abstandsmaß wird in diesem Zusammenhang als eine Distanz in Metern oder als maximale Hop-Zahl angegeben – auch eine Kombination ist möglich.

Trotzdem eignet sich das Konzept der „Abstract Regions“ nicht für den unmittelbaren Einsatz der Datensammlung wie in Kapitel 2 gefordert. Die verfügbaren Aggregationsmöglichkeiten können durch die „Abstract Regions“ nicht direkt umgesetzt werden und bedürften einer grundlegenden Anpassung der Methode, was einen erheblichen Aufwand darstellt. Außerdem besitzen die „Abstract Regions“ einen hohen Grad an Komplexität, weil jeder Knoten diverse Nachbarschaften mit verschiedenen Kriterien besitzen kann. Im Rahmen des nur kleinen Anteils genutzter Funktionalität erweist sich dies damit als wenig effizient hinsichtlich der resultierenden Programmgröße.

„Logical Neighborhoods“ [MP06, CMP06] stellen eine etwas andere Möglichkeit dar, Sensorknoten zu Nachbarschaften zusammenzufassen. Zusätzlich zur Lokalität von Knoten entscheiden hier zur Laufzeit veränderliche Kriterien, ob ein Knoten zu einer bestimmten Nachbarschaft gehört oder nicht. Dies wird u.a. durch Schwellwerte bewirkt: Ein Knoten gehört zu einer Nachbarschaft, wenn ein bestimmter Sensor Messdaten liefert, die über oder unter einem definierten Schwellwert liegen.

Dieses Konzept stellt sich für das in Kapitel 2 beschriebene Einsammeln als unbrauchbar dar, weil hierdurch in jeder Epoche überprüft werden müsste, welche Knoten zur Nachbarschaft einer Senke gehören. Damit können die Möglichkeiten der (zeitlichen) Aggregation nicht voll ausgeschöpft werden.

Zu den „Logical Neighborhoods“ existiert ein dediziertes Routing-Verfahren [CMP07], um den speziellen Ansprüchen dieser Programmierabstraktion gerecht zu werden. Hierbei wird insbesondere dem Umstand Rechnung getragen, dass Daten im Netz nicht nur an eine zentrale, sondern an viele verschiedene Stellen bzw. Senken transportiert werden, welche die Daten schließlich verarbeiten. Durch geschickte Wahl der Routen durch das Netz können damit Pakete gespart werden. Die Problemstellung ergibt sich als die Optimierung eines linearen Programms (ILP): Grob gesprochen ist die Anzahl der zum Routing ausgewählen Kanten in einem zusammenhängenden Netz bzw. Graphen zu minimieren, wobei ein Pfad von jeder Senke zu denjenigen Knoten existieren muss, von denen sie Daten einsammeln will. Eine derartige Baumoptimierung mit mehreren Senken ist in Abbildung 3.2 dargestellt. Die beiden Senken $\mathbf{k_1}$ und $\mathbf{k_4}$ sammeln Daten vom Knoten $\mathbf{k_6}$ ein. Die Kanten

stehen für das Versenden von (ggf. aggregierten) Daten. Der linke Fall zeigt eine mögliche, aber nicht optimale Lösung (5 Pakete), der rechte eine optimale Lösung (3 Pakete).

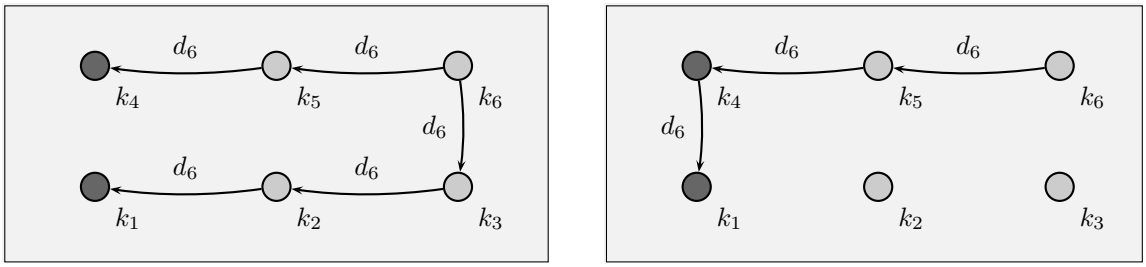


Abbildung 3.2: Multi-Hop Spannbaum-Routing zu zwei Senken ohne Optimierung (links) und mit Optimierung (rechts). Die Kanten zeigen den Versand ggf. aggregierter Datenpakete d an.

Weil die Lösung des o.g. ILP globales Wissen über den Graphen bzw. die Links im Sensornetz benötigt, wird in [CMP07] eine Näherungslösung für den verteilten Fall vorgestellt. Hierbei handelt es sich um ein adaptives Verfahren, das nur lokal verfügbare Informationen benötigt und verwendet. Dieser Ansatz stellt sich – auch aufgrund der im Papier vorgestellten Ergebnisse aus Experimenten – als prinzipiell vielversprechender Optimierungsansatz dar, sofern sich viele Pfade von Knoten zu Senken (teilweise) zusammenfassen lassen (vgl. Abbildungen 3.1 und 3.2). Dies ist insbesondere dann der Fall, wenn globales Routing über viele Hops hinweg zu einer größeren Menge Senken stattfindet. Im Verfahren gemäß [Röm06a] ist der Einzugsbereich der Senken allerdings als eher klein gegenüber der Netzgröße einzuschätzen und erstreckt sich nur über wenige Hops. Auch ist die Menge bzw. Dichte der Senken in vielen Fällen nicht sehr hoch. Diese beiden Umstände schränken damit die Optimierungsmöglichkeiten, die der adaptive Ansatz verspricht, ein. Es lässt sich somit erwarten, dass der Nutzen des Verfahrens in vielen Fällen durch die zusätzlichen Verwaltungsinformationen in den Nutzdaten-Paketen mindestens aufgewogen wird. Daher wird von einem Einsatz dieses adaptiven Verfahrens abgesehen.

Obwohl die im letzten Abschnitt vorgestellten Verfahren interessante Ansätze und Aspekte beinhalten, die für die Lösung der Problematik aus Kapitel 2 relevant sind, bietet keines dieser Verfahren eine umfassende Lösung hierfür. Damit erweist es sich als notwendig, ein eigenes Verfahren zu entwickeln. Wegen der verschiedenen Ansätze mit unterschiedlichen Vor- und Nachteilen erscheint es sinnvoll, mehr als eine Herangehensweise zu verfolgen. Dies bietet einerseits weitreichendere Analysemöglichkeiten als die Beschränkung auf nur eine Lösungsmethode und überhaupt erst Vergleichsmöglichkeiten. Andererseits lässt sich hierdurch feststellen, ob immer dasselbe Verfahren eine optimale Lösung darstellt oder eine Abhängigkeit von den Randbedingungen der Anwendung ab-

hängt. Randbedingungen könnten beispielsweise die Anzahl der Senken, die Dichte des Netzes oder die gewünschte Datenmenge in Bezug auf die maximal mögliche sein.

Kapitel 4

Protokoll-Design

Nach der Erläuterung der Problemstellung in Kapitel 2 und dem Vorstellen verwandter Arbeiten in Kapitel 3 folgt nun das Design der Algorithmen, die zum Einsammeln der Daten durch die Senken verwendet werden. Es handelt sich hierbei um zwei verschiedene Ansätze. Das erste Verfahren nutzt vergleichsweise einfaches Fluten, wohingegen das zweite die Daten auf zuvor erstellten Spannbäumen in Richtung der Senken transportiert. Die Lösungen werden in den Abschnitten 4.2 bzw. 4.3 ausführlich beschrieben. Zuvor werden in Abschnitt 4.1 einige Grundlagen diskutiert.

4.1 Grundlagen

Bevor die einzelnen Verfahren ausführlich vorgestellt werden können, bedarf es zunächst der Klärung einiger Voraussetzungen. Zudem sollen spezielle Notationen des verwendeten Pseudocodes kurz erklärt werden.

4.1.1 Voraussetzungen

Die Algorithmen der Abschnitte 4.2 und 4.3 dieses Kapitels beschreiben ausschließlich das für das Einsammeln der Applikationsdaten erforderliche Verhalten. Dabei wird davon ausgegangen, dass alle Knoten eine einheitliche Epochenlänge verwenden und dieselben Ereignisse beobachten. Die verwendeten Partitionen sind ebenfalls einheitlich und auf allen Knoten bekannt. Dies gilt auch für alle verwendeten Konstanten. Der Einfachheit halber wird zudem festgelegt, dass die beobachteten Ereignisse nur gemeinsam eingesammelt werden. Neben diesen Annahmen bzgl. der Applikation aus Abschnitt 2.1 werden die im Folgenden beschriebenen Rahmenbedingungen vorausgesetzt.

Jeder Knoten $\mathbf{k} \in \mathcal{K}$ besitzt eine eindeutige Kennung innerhalb des Netzes. Diese Kennung wird im Folgenden synonym mit dem Knoten $\mathbf{k}$ selbst verwendet. Zudem hat jeder

Knoten Kenntnis über seine Position pos. Die zu versendenden Ereignisvektoren werden durch eine geeignete Datenstruktur $daten$ repräsentiert.

Es existiert eine globale Zeit, die jedem Knoten bekannt ist. Dies kann zum Beispiel bedeuten, dass jeder Knoten eine Uhr besitzt und die einzelnen Uhren stets synchronisiert sind. Zudem verfügen die Knoten über Timer, die eine verzögerte Ausführung einzelner Aktionen ermöglichen. Diese werden im Folgenden durch einfaches Zuweisen ganzer Zahlen $t \geq 0$ gesetzt und durch Zuweisung von $\varnothing$ wieder gelöscht. Die Zahlen können als (irgendeine) feste Zeiteinheit aufgefasst werden, z.B. Millisekunden.

Es existieren geeignete Funktionen zum Senden der Daten via Funk – die versendeten Daten werden als Paket oder Nachricht bezeichnet. Hierzu gehören insbesondere die Versandmethoden *Unicast* und *Broadcast*. Unicast bedeutet, dass die Daten gezielt an einen einzelnen Knoten versendet bzw. adressiert werden. Ein Knoten verarbeitet ein Paket nur dann, wenn es an ihn adressiert ist. Der Versand eines Pakets als Broadcast bedeutet, dass alle Knoten, die das Paket empfangen, dieses auch verarbeiten dürfen. Die Zuverlässigkeit des Funks ist nicht notwendig – es können also Pakete verloren gehen.

Der Verzicht auf zuverlässige Funkübertragung bedeutet lediglich, dass die vorgestellten Verfahren hierauf prinzipiell nicht angewiesen sind. In der rein algorithmischen Beschreibung werden keine besonderen Vorkehrungen gegen Paketverluste und -kollisionen getroffen. Die Methoden verfahren nach dem Prinzip der „größten Mühe“ (engl. best effort). Zuverlässigkeit wird demnach entweder von Seiten der Knoten (in Form des Betriebssystems) oder der Implementierung gewährleistet, falls erforderlich (vgl. Kapitel 5).

4.1.2 Pseudocode

Jeder Codeblock wird durch ein geschweiftes Klammerpaar eingeleitet, dessen Inhalt ein Ereignis beschreibt – die Ereignisdefinition. Nur bei Auftreten dieses Ereignisses wird der folgende Code ausgeführt. Der Übersichtlichkeit halber wird pro Listing nur ein solcher Abschnitt aus Ereignis und zugehörigem Codeblock definiert. Ferner beziehen sich Ereignisse und Code auf jeden einzelnen Knoten – die Algorithmen sind also verteilt.

Pakete sind durch ein spitzes Klammerpaar gekennzeichnet und können mehrere Datenfelder bzw. Variablen enthalten, die durch Kommata voneinander getrennt sind. Ein Paket erhält außerdem einen Namen, der als Subskript an die schließende spitze Klammer angehängt wird, sofern das Paket in der Ereignisdefinition steht. Der Zugriff auf die Variablen eines Pakets erfolgt in der Punktnotation: Die Paketvariable wird an den Paketnamen angehängt und mit einem Punkt von diesem getrennt. Dieselbe Punktnotation wird auch dann

verwendet, wenn auf die einzelnen Komponenten eines Tupels einer Menge zugegriffen wird.

4.2 Fluten

Das hier verwendete Fluten stellte eine Erweiterung des herkömmlichen Flutens dar, in der die Eigenheiten der Applikation aus Abschnitt 2.1 berücksichtigt werden. Zunächst wird allerdings kurz das Grundverfahren erläutert. Jeder Knoten sendet seine Daten periodisch, damit die Senken diese einsammeln können. Die Periodizität entspricht einer global festgelegten und damit für alle Knoten gleichen Anzahl an Epochen. Neben den eigentlichen Ereignisvektoren $daten$ sendet ein Knoten einen Hop-Zähler $hops$, seine eindeutige Kennung als $quelle$ sowie seine Position pos mit. Das so entstehende Paket wird als Broadcast verschickt. Empfängt ein Knoten ein solches Paket, leitet er dieses mit um eins erhöhtem Hop-Zähler wiederum als Broadcast weiter. Dies geschieht aber nur dann, wenn der Hop-Zähler seinen Maximalwert MAXHOPS noch nicht erreicht hat und wenn die Distanz zwischen Empfänger und der Positionsangabe im Paket die zulässige Maximalentfernung MAXDIST nicht überschreitet. Ist der Empfänger selbst eine Senke, verarbeitet er die Daten lokal.

Dieses Verfahren lässt sich nun wie folgt verbessern. Anstelle der einzelnen Pakete werden nur noch sog. *Meta-Pakete* verschickt, die mehrere Einzelpakete enthalten können. Hierdurch kann eine gute Aggregation erreicht werden, wenn man folgende Aspekte berücksichtigt: Weil alle Knoten ihre Daten gleichzeitig versenden würden, sendet ein Knoten seine Daten nicht sofort, sondern fügt diese in einen Puffer $\mathcal{P}$ ein und setzt einen Timer χ (vgl. Algorithmus 4.1). Für das Setzen der Timer existiert ein minimaler (MINTIMER) und ein maximaler Wert (MAXTIMER).

Empfängt ein Knoten ein Meta-Paket, so fügt er die enthaltenen Pakete ebenfalls in diesen Puffer ein (und setzt ggf. den Timer). Für jedes einzelne Paket tut er dies jedoch nur dann, wenn er in der aktuellen Periode noch keine von $quelle$ generierten Daten erhalten hat. Die übrigen Kriterien aus dem vorangegangenen Absatz gelten hierüber hinaus. Zur Realisierung dieser Merkfähigkeit verwendet jeder Knoten eine Historie $\mathcal{H}$, die beim Start jeder Periode geleert wird. Sie verhindert, dass ein Knoten die Daten eines Knoten $quelle$ mehr als einmal pro Periode weiterleitet. Das vollständige Verhalten zeigt Algorithmus 4.2 und ist inspiriert durch [FR05].

Das tatsächliche Senden der Daten im Puffer erfolgt, sobald dieser seine maximale Größe MAXPUFFER erreicht hat oder der Timer abgelaufen ist. Dabei werden alle im Puffer

zwischengespeicherten Datenpakete in einem Meta-Paket versendet, der Puffer geleert und der Timer gelöscht (Algorithmus 4.3).

{ Knoten $\mathbf{k}$ registriert den Beginn einer neuen Periode }
$\mathcal{H} \leftarrow \emptyset$
if $\chi = \varnothing$
  $\chi \leftarrow Zufallszahl \in [\text{MINTIMER}, \text{MAXTIMER}]$
end if
$\mathcal{P} \leftarrow \mathcal{P} \cup \{<\mathbf{k}, pos, 1, daten>\}$

Algorithmus 4.1: Fluten – Beginner einer neuen Periode zum Einsammlen der Daten

4.3 Spannbaum-Routing

Im Gegensatz zum Fluten gestaltet sich das Spannbaum-Routing komplexer und erfordert zudem die Aufteilung in zwei Phasen, die nachfolgend Initialisierungs- und Applikationsphase genannt werden. Der Übersichtlichkeit wegen werden beiden Phasen in zwei getrennten Abschnitten beschrieben.

4.3.1 Nachbarschaften und Link-Qualitäten

Für den im Folgenden betrachteten Aufbau eines Spannbaums werden einige spezielle Kenntnisse jedes einzelnen Knoten über seine lokale Nachbarschaft vorausgesetzt, die genauer erläutert werden sollen. Weil dieser Begriff je nach Kontext unterschiedlich ausgelegt wird, bedarf es einer genaueren Definition: Die *Nachbarschaft* eines Knoten $\mathbf{k}$ ist die Teilmenge $\mathcal{N}_\mathbf{k} \subseteq \mathcal{K}$ aller Knoten eines Netzes, mit denen $\mathbf{k}$ bidirektional und direkt kommunizieren kann. Ein Knoten $\tilde{\mathbf{k}} \in \mathcal{N}_\mathbf{k}$ heißt *Nachbar* von $\mathbf{k}$.

Ein *Link* ist die Bezeichnung einer Funk-Verbindung zwischen zwei Knoten $\mathbf{k}$ und $\tilde{\mathbf{k}}$, die in mindestens einer Richtung besteht. Ein Link kann unidirektional (Verbindung nur in eine Richtung) oder bidirektional (Verbindung in beide Richtungen) sein. Diese Trennung ist jedoch nicht exklusiv: Ein bidirektionaler Link kann auch als zwei einzelne, unidirektionale Links betrachtet werden.

Für die Spannbaumoptimierung in Abschnitt 4.3.2 sind Link-Qualitäten erforderlich. Hierbei handelt es sich um ein Gütemaß unidirektionaler Links, das mit $Q_{\mathbf{k}\leftarrow\tilde{\mathbf{k}}}$ bezeichnet wird. Es beschreibt die Güte des Links von Knoten $\tilde{\mathbf{k}}$ (Sender) nach $\mathbf{k}$ (Empfänger) und

{ Knoten **k** empfängt n aggregierte Pakete $< quelle, pos, hops, daten >_{M_i}$ }

for $i = 1 \ldots n$
  if $M_i.quelle \in \mathcal{H} \mid M_i.quelle = \mathbf{k}$
    return
  else if $M_i.quelle \notin \mathcal{H}$
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{M_i.quelle\}$
  end if
  if $dist(M_i.pos, pos) > \text{MAXDIST}$
    return
  end if
  if $\mathbf{k} \in \mathcal{S}$
    Verarbeite $M_i.daten, M_i.pos$
  end if
  if $M_i.hops < \text{MAXHOPS}$
    if $\chi = \varnothing$
      $\chi \leftarrow Zufallszahl \in [\text{MINTIMER}, \text{MAXTIMER}]$
    end if
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{< M_i.quelle, M_i.pos, M_i.hops + 1, M_i.daten >\}$
  end if
end for

Algorithmus 4.2: Fluten – Reaktion bei Empfang eines Pakets

{ Timer χ läuft ab ($\chi = 0$) oder Puffer ist voll ($|\mathcal{P}| = \text{MAXPUFFER}$) }

Sende alle Pakete aus $\mathcal{P}$ aggregiert als Broadcast
$\mathcal{P} \leftarrow \emptyset$
$\chi \leftarrow \varnothing$

Algorithmus 4.3: Fluten – Aggregierendes Senden der Pakete im Puffer

wird hier durch eine reelle Zahl aus dem Intervall $[0, 1]$ beschrieben, wobei 0 der schlechteste und 1 der beste Wert ist.

In der vorliegenden Lösung identifizieren die einzelnen Knoten ihre Nachbarn durch den Einsatz des in [OTL04] beschriebenen Verfahrens, das auch in [TWW06] zum Einsatz kommt. Die Funktionsweise ist wie folgt: Durch das periodische Versenden sog. Hallo-Nachrichten gewinnt ein Knoten Informationen darüber, welche Knoten sich in seiner Umgebung befinden. Jedem dieser Knoten wird ein lokaler Status zugeordnet, der den zugehörigen Link als unbrauchbar, unidirektional oder bidirektional einstuft. Bei Änderung dieser Statusinformation oder dem Identifizieren eines bisher unbekannten Knotens werden die eindeutige Kennung und der aktuelle Status des Knotens in die Hallo-Nachrichten integriert. Auf diese Weise werden lokale Status ausgetauscht, so dass insbesondere bidirektionale Links festgestellt werden können. Wenn ein Knoten $\mathbf{k}$ ausreichend viele Hallo-Nachrichten von einem anderen Knoten $\tilde{\mathbf{k}}$ empfangen hat, stuft er den zugehörigen Link als unidirektional ein. Wenn $\mathbf{k}$ nun von $\tilde{\mathbf{k}}$ erfährt, dass dieser die Verbindung ebenfalls als (mindestens) unidirektional einstuft, so ändert er den Status auf bidirektional.

Das periodische Versenden der Hallo-Nachrichten kann außerdem dazu benutzt werden, die gewünschten (unidirektionalen) Link-Qualitäten zu erhalten. Ein Knoten $\mathbf{k}$ weiß, dass er von jedem bekannten Knoten $\tilde{\mathbf{k}}$ pro Periode eine Hallo-Nachricht erwartet. Diese Information kann er dazu verwenden, mittels rekursiver Schätzung [KS06, WTC03] die Empfangswahrscheinlichkeit von diesem Knoten zu schätzen:

$$\underset{\mathbf{k}\leftarrow\tilde{\mathbf{k}}}{Q}[n] \;=\; (1-\alpha)\cdot \underset{\mathbf{k}\leftarrow\tilde{\mathbf{k}}}{Q}[n-1] + \alpha \cdot \underset{\mathbf{k}\leftarrow\tilde{\mathbf{k}}}{\tilde{Q}}[n]\,, \qquad 0<\alpha<1,\ \underset{\mathbf{k}\leftarrow\tilde{\mathbf{k}}}{Q}[0]=0 \tag{4.1}$$

$Q_{\mathbf{k}\leftarrow\tilde{\mathbf{k}}}[n]$ ist die von Knoten $\mathbf{k}$ berechnete Schätzung der Empfangswahrscheinlichkeit von Knoten $\tilde{\mathbf{k}}$ nach n Perioden. Der Parameter α gibt den Einfluss des aktuellen Wertes $\tilde{Q}_{\mathbf{k}\leftarrow\tilde{\mathbf{k}}}[n]$ an, der 1 ist, falls in der Periode n eine Hallo-Nachricht von $\tilde{\mathbf{k}}$ empfangen wurde, und 0 sonst. Zwei positive Eigenschaften dieser Methode sind die Adaptivität – also die Reaktion auf Veränderungen – sowie der Umstand, dass bei der Schätzung der Qualität zu einem bestimmten Zeitpunkt auch die Vergangenheit durch die Rekursion in der Berechnungsvorschrift berücksichtigt wird.

4.3.2 Initialisierungsphase

Der erste Schritt auf dem Weg zum fertigen Spannbaum-Routing ist der Aufbau der Spannbäume. Jede Senke generiert einen eigenen Spannbaum, der jedoch nicht das ganze Netz umfasst, sondern durch eine maximale Tiefe MAXHOPS und maximale Distanz MAXDIST

begrenzt wird, weil die Senken gemäß Abschnitt 2.1 nur an den Daten aus einer lokalen Umgebung interessiert sind.

Im einfachsten Fall startet jede Senke den Spannbaum-Aufbau einmalig mit dem Broadcasten einer Spannbaum-Nachricht. Diese enthält die eindeutige Kennung der Senke als $wurzel$ und $vater$, deren Position pos sowie die aktuelle Baumtiefe $tiefe$ in Hops (die Tiefe ist aus der Sicht des Empfängers definiert, so dass die Senke diese initial auf 1 setzt).

Jeder Knoten, der so eine Nachricht empfängt, merkt sich diese Werte als Tupel der Menge $\mathcal{B}$, sofern die Distanz zur Senke nicht größer als MAXDIST ist und ihm der Baum mit der Senke $wurzel$ noch unbekannt ist. Wenn die Tiefe $tiefe$ in der gerade erhaltenen Nachricht noch nicht MAXHOPS erreicht hat, leitet er diese als Broadcast weiter. Dabei ersetzt er das Feld $vater$ durch seine eigene Kennung und erhöht $tiefe$ um eins.

Dieser Algorithmus ist bereits funktionsfähig und erlaubt durch das Mitsenden der Senken-Kennung die Verwaltung mehrerer Bäume pro Knoten. Ein Knoten kann also Teil diverser Spannbäume sein.

Allerdings existiert ein entscheidender Nachteil: Die Analysen und Resultate in [WTC03] und [TWW06] belegen, dass der Empfang eines einzelnen Pakets in drahtlosen Sensornetzen kein zuverlässiges Maß für die Verbindungsqualität zweier Knoten via Funk sein kann. Im hier präsentierten Verfahren sollen die Spannbäume allerdings dauerhaften Bestand haben, also für vergleichsweise viele Datensammlungen eingesetzt werden. Darum ist es sinnvoll, diejenigen Kanten bevorzugt als Baumkanten zu verwenden, die eine gute Link-Qualität (in beide Richtungen) aufweisen. Für das folgende Verfahren ist es ausreichend, wenn ein Knoten $\mathbf{k}$ seine Nachbarn $\tilde{k}_i \in \mathcal{N}_\mathbf{k}$ und die zugehörigen Qualitäten $Q_{\mathbf{k}\leftarrow\tilde{\mathbf{k}}_\mathbf{i}}$ der Links in seine Richtung kennt. Es bedarf also einer Anpassung des o.g. Algorithmus, die im Folgenden erläutert wird.

Die Senken starten den Aufbau der Spannbäume, indem sie die bisherigen Spannbaum-Nachrichten um ein Feld ergänzen, das die Qualität der Verbindung des Empfängers zur Senke beschreibt (siehe Algorithmus 4.4). Diese Erweiterung erfahren auch die Tupel der Menge $\mathcal{B}$. Zudem werden die Nachrichten (in geeigneter Weise) nur noch an die bidirektionalen Kommunikationspartner versendet.

Beim Erhalt einer Spannbaum-Nachricht verwirft ein Knoten diese nicht automatisch, wenn bereits ein Baumeintrag zur zugehörigen Senke bekannt ist, sondern ersetzt den bisherigen Vater dieses Baumes durch den Sender der soeben erhaltenen Nachricht, wenn die Pfadqualität sich hierdurch verbessert und die Tiefe nicht größer ist als vorher. Erhält ein Knoten eine Spannbaum-Nachricht, die zu einem bisher unbekannten Baum (bzw. einer unbekannten Senke) gehört oder ändert er den Vater zu einem bereits bekannten Baum, so informiert er hierüber alle bidirektionalen Kommunikationspartner (mit Ausnahme der

Senke und des Vaters, falls diese sich hierunter befinden). Bedingung ist erneut, dass die maximale Baumtiefe noch nicht erreicht sein darf. Die vollständige Bearbeitung empfangener Spannbaum-Nachrichten liefert Algorithmus 4.5.

Die Tiefen-Bedingung beim Ersetzen des bisherigen Vaters während der Initialisierungsphase ist notwendig, um die vorgegebene, maximale Baumtiefe garantieren zu können. Existiert beispielsweise ein Baum wie in Abbildung 4.1 und erhält Knoten $\mathbf{k_3}$ eine Spannbaum-Nachricht von $\mathbf{k_2}$, so darf er diesen auch dann nicht als neuen Vater im Baum verwenden, wenn die Pfadqualität (zur Senke) sich hierdurch verbessern würde. Dies hätte zur Folge, dass $\mathbf{k_5}$ die maximal erlaubte Tiefe von 3 verletzt.

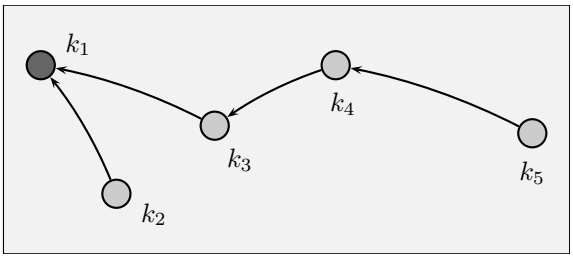


Abbildung 4.1: Ein möglicher Spannbaum der Senke $\mathbf{k_1}$ mit Maximaltiefe 3.

{ Spannbaum-Aufbau wird auf Knoten $\mathbf{k}$ initiiert }
for all $\tilde{\mathbf{k}} \in \mathcal{N}_{\mathbf{k}}$
  Sende $< \mathbf{k}, \mathbf{k}, pos, 1, Q_{\mathbf{k} \leftarrow \tilde{\mathbf{k}}} >$ an $\tilde{\mathbf{k}}$
end for

Algorithmus 4.4: Spannbaum-Initiierung

Werden die Daten der Knoten nicht periodisch von den Senken eingesammelt, sondern nur dann, wenn die Senken diese anfordern, sollten die Pfad-Qualitäten ein bidirektionales Maß darstellen. Algorithmus 4.5 berücksichtigt diesen Umstand. Werden die Daten aber periodisch ohne Anforderung durch die Senken verschickt, so reicht es aus, wenn die Pfad-Qualitäten nur die unidirektionale Qualität in Richtung der Senke eines Baumes ausdrücken. Dies kann erreicht werden, indem anstelle des eigentlichen Wertes von $Q_{\mathbf{k} \leftarrow \mathbf{M.vater}}$ in Algorithmus 4.5 die maximale Qualität verwendet wird.

Abschließend sei darauf hingewiesen, dass eine Senke sowohl beim Fluten als auch beim Aufbau der Spannbäume ggf. nicht von allen Knoten innerhalb der maximalen Distanz MAXDIST Daten einsammeln kann. Durch die beschränkten Funkreichweiten ist es mög-

{ Knoten **k** empfängt eine Nachricht $< wurzel, vater, pos, tiefe, qualität >_M$ }
if $dist(M.pos, pos) >$ MAXDIST
  return
end if
$updated \leftarrow false$
$M.qualität \leftarrow M.qualität \cdot Q_{\mathbf{k} \leftarrow \mathbf{M.vater}}$
$b \leftarrow (wurzel, v, p, tiefe, qualität) \in \mathcal{B} : wurzel = M.wurzel$
if $b = \varnothing$
  $updated \leftarrow true$
  $b \leftarrow (M.wurzel, M.vater, M.pos, M.tiefe, M.qualität)$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{b\}$
else if $b.qualität < M.qualität \ \ \& \ b.tiefe \geq M.tiefe$
  $updated \leftarrow true$
  $\mathcal{B} \leftarrow \mathcal{B} \setminus \{b\}$
  $b \leftarrow (M.wurzel, M.vater, M.pos, M.tiefe, M.qualität)$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{b\}$
end if
if $updated = true \ \ \& \ M.tiefe <$ MAXHOPS
  for all $\tilde{k} \in \mathcal{N}_{\mathbf{k}} \setminus \{M.vater, M.wurzel\}$
    $Q \leftarrow M.qualität \cdot Q_{\mathbf{k} \leftarrow \tilde{\mathbf{k}}}$
    Sende $< M.wurzel, M.vater, M.pos, M.tiefe + 1, Q >$ an $\tilde{\mathbf{k}}$
  end for
end if

Algorithmus 4.5: Empfangen und Weiterleiten von Spannbaum-Nachrichten

lich, dass für einen Knoten trotz ausreichender Nähe kein Pfad mit höchstens MAXHOPS Hops zur Senke existiert.

4.3.3 Applikationsphase

Sind die Spannbäume einmal berechnet, können diese für das Einsammeln der Daten verwendet werden. Im einfachsten Fall senden alle Knoten ihre Daten periodisch an ihre Väter – wie bereits beim Fluten sind die Periodenlängen durch eine globale, allen Knoten bekannte Anzahl an Epochen definiert. Damit ergibt sich eine gute Aggregationsmöglichkeit, indem ein Knoten wartet, bis er die Daten seiner Söhne pro Baum erhalten hat. Erst danach sendet er die Daten an seinen Vater weiter. In jeder Periode wird damit über jede Baumkante nur ein Paket verschickt.

Die in der Initialisierungsphase berechneten Spannbäume sind allerdings derart konstruiert, dass ein Knoten nur den Vater zu einem bestimmten Baum kennt, nicht aber seine Söhne. Daher ist es nicht auf direkte Weise möglich, auf den Erhalt der Daten aller Söhne zu warten. Allerdings kennt jeder Knoten die eigene Tiefe in einem Baum sowie die Maximaltiefe. Setzt man voraus, dass alle Knoten gleichzeitig den Beginn einer neuen Periode registrieren, so kann diese Problematik durch den Einsatz einfacher Timer χ umgangen werden. Hierzu wählt man ein adäquates Intervall INTVL. Bei Beginn einer neuen Periode zur Datensammlung wartet ein Knoten ein Vielfaches dieses Intervalls ab. Dieses Vielfache ergibt sich unmittelbar aus der Differenz der maximalen und der eigenen Tiefe in einem bestimmten Baum. Nach Ablauf eines der Timer sendet der Knoten seine Daten an den Vater des zugehörigen Spannbaums. Bei den versendeten Daten handelt es sich nicht mehr nur um die eigenen Daten, sondern um ein Aggregat D aus den eigenen und denen (ggf. wiederum aggregierten) der Söhne. Die hier beschriebene Sendelogik findet sich in kompakter Darstellung in den Algorithmen 4.6 sowie 4.7.

Algorithmus 4.8 beschreibt das Verhalten eines Knotens beim Empfang von Daten. Ist der empfangene Knoten selbst die Senke des Baumes, zu dem die Daten gehören, verarbeitet er diese. Ansonsten aggregiert er die Daten (und wartet auf den Ablauf des Timers). Empfängt ein Knoten Daten von einem Sohn erst nach Ablauf des entsprechenden Timers, verwirft er diese. Zwar könnte er versuchen, diese Daten zu einem späteren Zeitpunkt nachzusenden, aber dieser Fall kann vernachlässigt werden, wenn man das Intervall INTVL groß genug wählt.

Dieser proaktive, periodische Ansatz kann durch einen zusätzlichen Teil derart verändert werden, dass die Senken die Daten aktiv anfordern können. Das Ereignis „Daten eines Knoten $\mathbf{k}$ werden angefordert“ in Algorithmus 4.6 wird in diesem Fall nicht mehr peri-

{ Daten eines Knoten k werden angefordert }

for all $b \leftarrow (wurzel, v, p, tiefe, q) \in \mathcal{B}$
  $\chi_{b.wurzel} \leftarrow (\text{MAXHOPS} - b.tiefe) \cdot \text{INTVL}$
  $D_{b.wurzel} \leftarrow daten$
end for

Algorithmus 4.6: Vorbereitungen zum Senden der Daten im Spannbaum

{ Der Timer $\chi_{\mathbf{s}}$ des Baumes mit der Senke s läuft ab ($\chi_{\mathbf{s}} = 0$) }

$\chi_s \leftarrow \varnothing$
$b \leftarrow (wurzel, vater, p, t, q) \in \mathcal{B} : wurzel = \mathbf{s}$
Sende $<\mathbf{s}, D_{\mathbf{s}}>$ an $b.vater$

Algorithmus 4.7: Senden der Daten im Spannbaum

{ Empfang eines Datenpakets $<wurzel, daten>_M$ }

if $\chi_{M.wurzel} \neq \varnothing$
  if $\mathbf{k} = M.wurzel$
    Verarbeite $M.daten$
  else
    $D_{M.wurzel} \leftarrow$ Aggregiere$(D_{M.wurzel}, M.daten)$
  end if
end if

Algorithmus 4.8: Empfangen und Verarbeiten von Daten im Spannbaum

odisch ausgelöst, sondern bei Empfang einer Sammel-Nachricht. Das Setzen des Timers muss ebenfalls leicht modifiziert werden: Die Wartezeit muss verdoppelt werden, um dem Umstand Rechnung zu tragen, dass vor dem eigentlichen Einsammeln zunächst alle Knoten im Baum informiert werden müssen.

Der eigentliche Algorithmus zum Anfordern der Daten aller Knoten eines Baumes gestaltet sich recht einfach. Die Senke schickt eine Sammel-Nachricht mit der eigenen Kennung als $wurzel$ und $vater$ per Broadcast (vgl. Algorithmus 4.9). Bei Empfang einer solchen Nachricht prüft der Knoten zunächst, ob er Teil des Baumes der Senke $wurzel$ ist. Handelt es sich bei $vater$ außerdem um seinen Vater in diesem Baum, löst er das Ereignis „Daten eines Knoten k werden angefordert“ für sich selbst aus. Es werden nur Nachrichten vom Vater akzeptiert, um das korrekte Setzen der Timer in Algorithmus 4.6 zu gewährleisten. Zusätzlich leitet er die Nachricht weiter, wenn die zugehörige Baumtiefe nicht maximal ist. Dabei ersetzt er das Feld $vater$ durch seine eigene Kennung. Dieses Verhalten spiegelt Algorithmus 4.10 wider.

```
{ Senke s benötigt Daten }
Sende <s, s> als Broadcast
```

Algorithmus 4.9: Initialisierung der Datensammlung durch eine Senke

```
{ Knoten k empfängt eine Sammel-Nachricht <wurzel, vater>_M }
b ← (wurzel, vater, p, tiefe, q) ∈ B : wurzel = M.wurzel
if b ≠ ∅  & b.vater = M.vater
   Löse Ereignis „Daten eines Knoten k werden angefordert“ aus
   if b.tiefe < MAXHOPS
      Sende <M.wurzel, k> als Broadcast
   end if
end if
```

Algorithmus 4.10: Empfangen und Weiterleiten einer Sammel-Nachricht

4.3.4 Datenaggregation

Nach der genauen Beschreibung des Spannbaum-Routings folgt nun die Betrachtung der in diesem Kontext eingesetzten Aggregierungsfunktion. Für die in Kapitel 2 beschriebene

statistische Auswertung benötigt eine Senke pro Epoche die absoluten Auftrittshäufigkeiten linksseitiger Ereignisse aus den zu bewertenden Ereignismustern.

Es ist ausreichend, pro relevanter Epoche und Distanzpartition je einen Zähler anstelle der Ereignisvektoren zu verschicken. Es ergibt sich ein rekursives Verfahren, das an den Blättern des Baumes startet. Jedes Blatt schickt für jede relevante Epoche die Bitvektoren als (binäre) Zähler der beobachteten Ereignisse als Datenfeld $daten$. (Der Fall, dass ggf. nur eine Teilmenge der beobachteten Ereignisse von der Senke abgefragt werden kann, wird an dieser Stelle der Einfachheit halber nicht betrachtet.) Ein Vater im Baum addiert die einzelnen Zähler, die er von seinen Söhnen erhält, zu seinen lokalen Bitvektoren. Nach Ablauf des Timers sendet er diese Zähler als Feld $daten$ an seinen eigenen Vater. Die Senke erhält letztlich von allen Söhnen im letzten Schritt die entsprechenden Zähler, getrennt nach Distanz und Epoche, und kann diese nun korrekt verarbeiten. Die Aggregation besteht einerseits darin, dass pro Baumkante nur eine Nachricht erforderlich ist. Andererseits erfolgt eine Konvertierung von n Binärwerten in einen Zähler mit Maximalwert n, der folglich mit weniger Bits auskommt, nämlich $\lceil \log_2 n \rceil$.

Es ist anzumerken, dass die Größe der Zähler zur Senke hin wächst. Es sind damit immer nur so viele Bits zu übertragen, wie der aktuelle Zählerstand z benötigt, also $\lceil \log_2 z \rceil$. Tritt ein Ereignis nur selten auf, ist auch die durchschnittliche Anzahl nötiger Bits für die zugehörigen Zähler kleiner als das Maximum.

Es bleibt abschließend zu klären, welche Epochen beim Einsammeln der Daten relevant sind. Beim periodischen Ansatz sind dies einfach alle Epochen der letzten Periode. Beim aktiven Ansatz ergibt sich eine Abhängigkeit von der ältesten Periode in den Zeitpartitionen. Liegt das letzte Einsammeln länger zurück als diese, so müssen die Daten aller Epochen in den Zeitpartitionen eingesammelt werden. Ist dies nicht der Fall, müssen nur die Daten seit dem letzten Einsammeln geschickt werden. Kompliziertere Fälle werden dadurch ausgeschlossen, dass vorausgesetzt wird, dass das Auftreten eines rechtsseitigen Ereignisses einer Ereignsdefinition bei einer Senke immer zum sofortigen Einsammeln der Daten führt.

Kapitel 5

Implementierung

Zur Simulation der Algorithmen aus Kapitel 4 werden diese in nesC [GLvB+03] für TinyOS [HSW+00] implementiert. Neben der Simulation ist damit auch der Einsatz der Programme in einer realen Anwendung im Kontext von [Röm06a] möglich. Eine detailierte Erläuterung zu nesC und TinyOS findet sich in Anhang A. Die von TinyOS unterstützte Modularisierung findet in der hier vorgestellten Implementierung zweifach Anwendung: Zum einen sind die verschiedenen Module so gut wie möglich unabhängig voneinander konzipiert, so dass diese bei Bedarf ohne großen Aufwand gegen andere Implementierungen ausgetauscht werden können. Zum anderen verwenden die Applikationen einheitliche Schnittstellen, die sich erst in der konkreten Implementierung der Module unterscheiden.

Dieses Kapitel beschreibt Besonderheiten bei der Implementierung der Algorithmen aus Kapitel 4 sowie die verwendeten Datenstrukturen. Diese ergeben sich z.B. aus dem Umstand, dass es in TinyOS keine einfache, dynamische Speicherwaltung gibt, weswegen an vielen Stellen Arrays fester Größe verwendet werden müssen. Eine weitere Restriktion liegt in der Beschränkung der maximalen Nutzdatenmenge innerhalb der Funk-Nachrichten. Diese sind standardmäßig auf 29 Byte begrenzt, können aber durch einfache Änderung einer Konstanten vergrößert werden. Hierauf wird allerdings näher in Kapitel 6 eingegangen.

5.1 Schnittstellen und Module

Die Abbildungen 5.1 und 5.2 zeigen die verwendeten Schnittstellen inklusive der jeweiligen Module beim Fluten bzw. Spannbaum-Routing. Die Pfeile zeigen die Abhängigkeiten der einzelnen Schnittstellen voneinander an. Die Darstellung beschränkt sich der Übersichtlichkeit halber auf die selbstgeschriebenen Schnittstellen und Module – diese verwenden ihrerseits Schnittstellen der TinyOS Bibliothek. Diese Abhängigkeiten werden in den

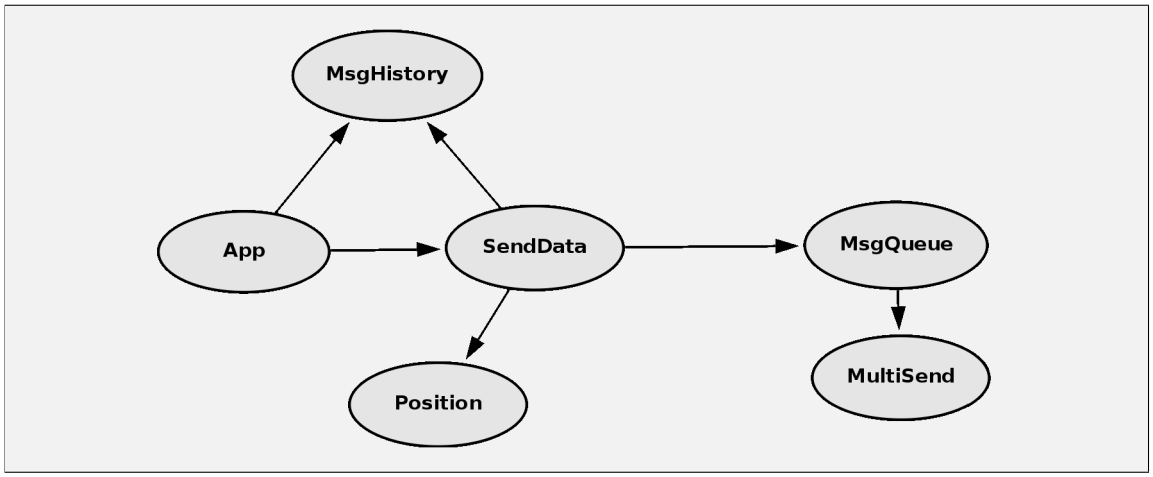


Abbildung 5.1: Eingesetzte Module beim Fluten und ihre Abhängigkeiten

folgenden Beschreibungen der einzelnen Schnittstellen und ihrer Module bei Bedarf genannt und ihr Einsatz kurz erläutert.

5.1.1 Warteschlange und Multiplexer

TinyOS stellt eine Schnittstelle mit einem Kommando zum Senden von Nachrichten per Funk zur Verfügung. Über eine Parametrisierung dieser Schnittstelle können verschiedene Nachrichtentypen realisert werden. Der Vorteil hiervon ist, dass für die verschiedenen Typen unterschiedliche Empfangsereignisse ausgelöst werden, d.h. zu jedem Typ muss eine eigene Funktion implementiert werden, die bei Empfang einer Nachricht dieses Typs von TinyOS aufgerufen werden kann. Damit kann eine Anwendung in verschiedene Module aufgeteilt werden, die jeweils eigene Nachrichtentypen verwenden. Jedes Modul stellt dann eine eigene Funktion zur Behandlung der eigenen Nachrichtentypen bereit.

Weil TinyOS für das Senden von Nachrichten keine Warteschlange zur Verfügung stellt, muss die Anwendung insbesondere den Fall behandeln, dass gerade keine neue Nachricht gesendet werden kann, weil das Betriebssystem noch mit dem Versand einer älteren Nachricht beschäftigt ist. Zur Umgehung dieser Problematik dient eine eigene Warteschlange (MsgQueue), die von allen nachfolgenden Applikationen verwendet wird. Dabei handelt es sich um einen einfachen Ringpuffer in Form eines Arrays mit fester Größe. Dieses Array speichert den Nachrichten-Typ, den Adressaten, die Nachricht im TinyOS Format, deren Größe in Byte und eine Anzahl Wiederholungen. Zum Einfügen in die Warteschlange existiert ein Kommando, dass genau diese Parameter erhält. Sofern der Puffer nicht voll ist, wird der neue Eintrag gespeichert, ansonsten wird er verworfen und mit dem Rückgabewert der Fehler angezeigt.

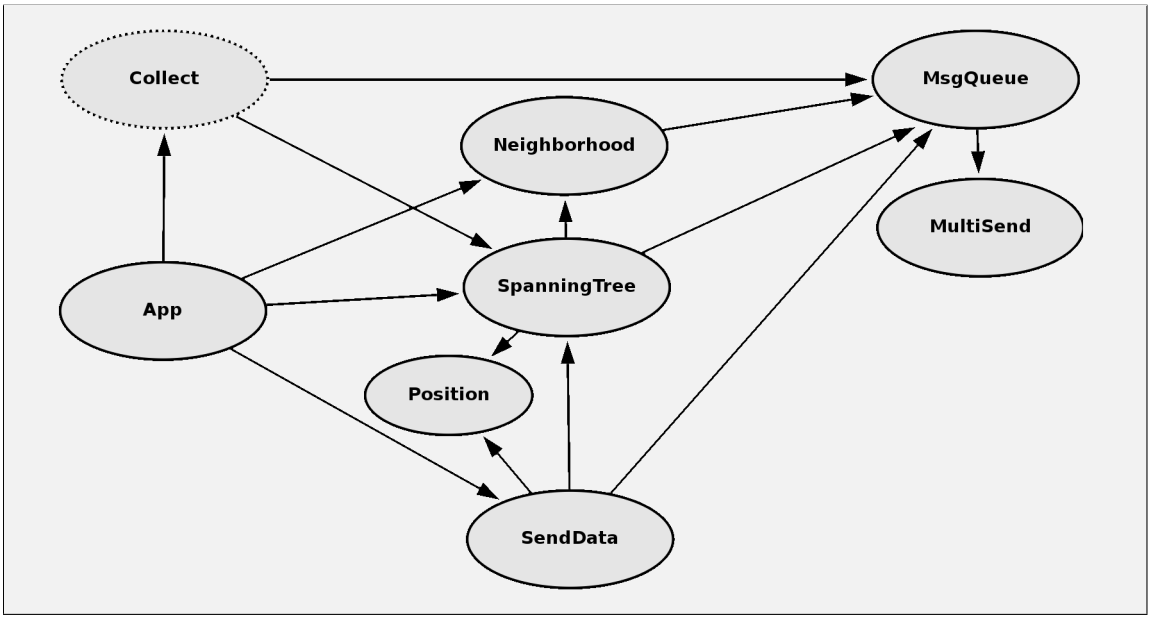


Abbildung 5.2: Eingesetzte Module beim Spannbaum-Routing und ihre Abhängigkeiten

Der interne Ablauf gestaltet sich wie folgt: Solange der Puffer nicht leer ist, prüft ein Timer, ob die älteste Nachricht in der Warteschlange gesendet werden kann. Sobald dies möglich ist, wird die Nachricht gesendet. Der Timer wird nach jedem Ablauf mit einer Zufallszahl aus einem vorgegebenen Intervall initialisiert. Beim tatsächlichen Senden der Nachrichten muss ein Multiplexing bzgl. der Nachrichtentypen durchgeführt werden. Um die Warteschlange auch unabhängig von den verwendeten Nachrichtentypen einsetzen zu können, ist das Multiplexing über eine eigene Schnittstelle (MultiSend) mit zugehörigem Modul realisiert. Dieses übernimmt lediglich die Aufgabe, das jeweils korrekte Sende-Kommando aufzurufen.

Mit der Anzahl der Wiederholungen beim Hinzufügen einer Nachricht zur Warteschlange kann gesteuert werden, wie oft versucht werden darf, diese zu verschicken. Der Multiplexer signalisiert der Warteschlange über das Auslösen eines Ereignisses, wann das Senden beendet ist. Ein zusätzlicher Parameter gibt an, ob die Nachricht vom Empfänger bestätigt wurde. In den folgenden Applikationen wird dieser Mechanismus wie folgt verwendet: Beim Senden der Daten Richtung Senke werden sowohl beim Fluten als auch beim Spannbaum-Routing die von TinyOS bereitgestellten Empfangsbestätigungen[1] verwendet, in allen anderen Fällen zeigt der Indikator an, dass die Nachricht nicht empfangen wurde. Die Anzahl der Wiederholungen dient der Steigerung der Zuverlässigung bei der Funk-Übertragung durch Redundanz. Der Timer dient zur Verminderung von Kollisionen.

[1] Bei einem Broadcast gilt eine Nachricht als bestätigt, wenn der Sender mindestens eine Empfangsbestätigung erhält

5.1.2 Knotenpositionen

Wie bei der Erläuterung der Algorithmen in Kapitel 4 beschrieben, benötigen die Knoten Kenntnis über ihre Position. Hierzu dient die Schnittstelle Position, die ein Kommando zur Abfrage der Position in Form von Koordinaten anbietet. Sobald diese verfügbar sind, wird ein Ereignis ausgelöst. Die Abfrage der Koordinaten wird über Positionssensoren realisiert, welche die Daten von einem TinyViz-Plugin beziehen.

5.1.3 Historie

Die Historie (MsgHistory) wird nur für das Fluten benötigt, um diejenigen Knoten festzuhalten, deren Daten in der aktuellen Periode bereits weitergeleitet wurden. Da in realen Sensornetzen keine exakt synchronen Uhren vorausgesetzt werden können, setzt die Implementierung nicht voraus, dass alle Knoten die Historie gleichzeitig leeren können. Neben der Kennung desjenigen Knotens, dessen Daten weitergeleitet wurden, wird darum auch die Anfangsepoche der weitergeleiteten Daten gespeichert.

Zur Optimierung der Ausführungsgeschwindigkeit bei der Simulation existiert jedoch ein Kommando zum Leeren der Historie. Diese ist als Ringpuffer aufgebaut, so dass die Komplexität beim Einfügen und Prüfen des Enthaltenseins einer Kennung-Epochen-Kombination linear ist (zur Länge des Puffers). Um den verwendeten Speicherplatz zu minimieren, wird beim Einfügen ein Eintrag mit selber Knoten-Kennung und älterer Epoche überschrieben. Die Länge des Puffers muss immer so gewählt werden, dass dieselben Daten eines Knotens nie mehr als einmal weitergeleitet werden bzw. nie mehr als einmal von einer Senke verarbeitet werden. Sie lässt sich durch Kenntnis der Knotendichte im Netz, der maximal zulässigen Anzahl Hops und der maximalen Funkreichweite abschätzen (ohne Formel). In kleinen Netzen kann vereinfachend die um eins verringerte Anzahl der Knoten im Netz verwendet werden.

5.1.4 Nachbarschaft und Link-Qualitäten

Bei der Nachbarschaft (Neighborhood) handelt es sich um die Implementierung des in [OTL04] spezifizierten Verfahrens. Ein periodischer Timer fügt eine neue Hallo-Nachricht zur Warteschlange hinzu und aktualisiert die Nachbarschaftstabelle. Es existiert die Möglichkeit, den Timer und damit das Nachbarschaftsprotokoll anzuhalten, obwohl dieses dafür nicht vorgesehen ist. Im Gegensatz zum Ansatz in [TRV+05] wird dabei nicht explizit versucht, beim Anhalten konsistente Zustände zu garantieren. Es kann also vorkommen, dass ein Knoten einen anderen als Nachbarn betrachtet, obwohl dies andersherum nicht der

Fall ist. Damit verbundene Probleme werden von der Implementierung des Spannbaum-Algorithmus behandelt.

Von besonderem Interesse sind die verwendeten Hallo-Nachrichten, deren Aufbau Listing 5.1 entnommen werden kann. Durch die Beschränkung der maximalen Nutzdatengröße $MaxDaten$ bei der Funkübertragung in TinyOS ist die Anzahl der maximalen Nachbarn N (entspricht der Konstanten TND_NBRTBL_SIZE in Listing 5.1) beschränkt. Sie lässt sich durch folgende Ungleichung (Einheiten in Byte) ausdrücken:

$$6 + N \cdot 2 \leq MaxDaten \quad \Rightarrow \quad N \leq \left\lfloor \frac{MaxDaten - 6}{2} \right\rfloor \tag{5.1}$$

Das Makro `HELLO_MSG_SIZE` in Listing 5.1 wird verwendet, um beim Einfügen einer Hallo-Nachricht in die Warteschlange einfach deren tatsächliche Länge berechnen zu können. So wird beim Senden nur die verwendete Anzahl Bytes der Nachricht übertragen.

```
typedef struct HelloMsg_s {
   uint16_t   senderId;
   uint8_t    hseq;
   uint8_t    replySize;
   uint8_t    requestSize;
   uint8_t    lostSize;
   uint16_t   nbrLst[TND_NBRTBL_SIZE];
} HelloMsg;
#define HELLO_MSG_SIZE(t)  (sizeof(HelloMsg) - (TND_NBRTBL_SIZE - ((t)
    .replySize + (t).requestSize + (t).lostSize)) * sizeof(uint16_t))
```

Listing 5.1: Hallo-Nachrichten

Die unidirektionalen Link-Qualitäten werden wie folgt berechnet: Empfängt ein Knoten eine Hallo-Nachricht eines Nachbarn, aktualisiert er die zugehörige Schätzung gemäß Formel 4.1. Dabei wird vernachlässigt, dass die Hallo-Nachrichten unterschiedlich groß sein können. Nur beim Empfang einer Hallo-Nachricht kann ein Knoten einwandfrei feststellen, wie viele konsekutive Nachrichten er von einem Knoten nicht gehört hat. Zur Abfrage der einzelnen Qualitäten bietet die Nachbarschaftsschnittstelle ein Kommando. Dieses liefert eine aktuelle Schätzung zurück, in der die Anzahl der mindestens verpassten Hallo-Nachrichten eines Nachbarn eingerechnet wird.

5.1.5 Spannbaum-Aufbau

Der Aufbau der Spannbäume wird über die Schnittstelle SpanningTree bzw. das zugehörige Modul bereitgestellt. Die einzelnen Bäume, die durch einen Knoten verlaufen, werden

in einem Array fester Größe gespeichert. Dessen Größe ist aufgrund der kleinen Hauptspeicher von Sensorknoten Einschränkungen unterlegen.

Zum Aufbau der Spannbäume werden die in Listing 5.2 gezeigten Spannbaum-Nachrichten verwendet. Von besonderem Interesse sind die Felder `nbrs` und `pathProb`. Im ersten dieser beiden Arrays werden die Kennungen der $\tilde{N}$ Nachbarn mit der besten Qualität übertragen. Im zweiten stehen die zugehörigen Pfadwahrscheinlichkeiten (vgl. Abschnitt 4.3.2). Der Wert $\tilde{N}$ (entspricht ST_NBRS_MAX in Listing 5.2) muss wieder so gewählt werden, dass die Größe der Nachricht nicht das zulässige Maximum $MaxDaten$ überschreitet. Es gilt (Einheiten in Byte):

$$10 + \tilde{N} \cdot 3 \leq MaxDaten \quad \Rightarrow \quad \tilde{N} \leq \left\lfloor \frac{MaxDaten - 10}{3} \right\rfloor \tag{5.2}$$

```
typedef struct ExplorerMsg_s {
   uint16_t   rootId;    /** ID of tree's root */
   Pos2D      pos;       /** position of root (2*uint16_t) */
   uint16_t   parentId;  /** ID of sender */
   uint8_t    depth;     /** depth of tree on sender's side */
   uint8_t    lstSize;   /** size of nbrs and prob arrays */
   uint16_t   nbrs[ST_NBRS_MAX];      /** sender's neighbors */
   intProb_t  pathProb[ST_NBRS_MAX];  /** recv prob of path */
} ExplorerMsg;
```

Listing 5.2: Spannbaum-Nachrichten

Eine weitere Besonderheit bei der Implementierung ist, dass eine Spannbaum-Nachricht nur dann bearbeitet wird, wenn der Sender lokal als Nachbar des Empfängers bekannt ist (Prüfung mittels der Nachbarschaft-Schnittstelle) und die eigene Kennung im Array `nbrs` der Nachricht enthalten ist. Dieses Vorgehen behebt einerseits das Problem, dass beim einfachen Anhalten des Nachbarschaftsprotokolls inkonsistente Zustände auftreten können (vgl. Abschnitt 5.1.4). Andererseits ist es Voraussetzung, wenn bidirektionale Pfad-Qualitäten verwendet werden sollen – nur wenn die Knoten sich gegenseitig als Nachbarn betrachten, existieren auch Link-Qualitäten in beide Richtungen.

Zusätzlich zur Implementierung der TinyOS-Module wurde ein TinyViz-Plugin zur Verifikation des Spannbaum-Aufbaus und des Nachbarschaftsprotokoll entwickelt.

5.1.6 Einsammeln der Daten

Für das Senden der Daten (in Richtung der Senken) steht die Schnittstelle SendData zur Verfügung. Sie findet in beiden Ansätzen Verwendung und spezifiziert ein Kommando, das

den Versand der lokalen Daten initiiert, sowie ein Ereignis, das beim Empfang von Daten ausgelöst wird.

Das zugehörige Modul, das beim Fluten eingesetzt wird, verwendet die Schnittstelle Position, um die Koordinaten beim Versand der eigenen Daten mitschicken und beim Empfang von Daten prüfen zu können, ob der Sender innerhalb der maximal zulässigen Distanz liegt (vgl. Abschnitt 4.2). Die Warteschlange wird verwendet, um Daten eines Knoten pro Periode höchstens einmal weiterzuleiten oder zu verarbeiten. Das Ereignis beim Empfang von Daten wird für jedes empfangene (neue) Datenpaket einzeln ausgelöst. Die Implementierung spiegelt ansonsten exakt die Beschreibung in den Algorithmen 4.1–4.3 wieder.

Beim Spannbaum-Routing kommen zwei verschiedene Implementierungen zum Einsatz, die sich jedoch nur in der Berechnung der Timer aus Algorithmus 4.6 unterscheiden. Ansonsten existieren keine Unterschiede. Beim Senden werden die von TinyOS bzw. TOSSIM bereitgestellten Empfangsbestätigungen verwendet und Pakete bis zu einer festgelegten Maximalzahl wiederholt, wenn die jeweilige Bestätigung nicht erhalten wurde. Damit muss ein Knoten sich jedoch auch merken, von welchen (ihm prinzipiell unbekannten) Kindern er Daten erhalten hat, um Duplikate erkennen zu können. Diese können auftreten, wenn eine Empfangsbestätigung verloren geht.

Abweichend von Algorithmen 4.6–4.7 existiert nicht für jeden bekannten Baum ein eigener Timer, sondern ein Zähler. Die einzelnen Zähler werden gemäß Algorithmus 4.6 mit der Wartezeit für den jeweiligen Baum initialisiert und durch einen periodischen Timer um dessen Intervalllänge erniedrigt. Eine aggregierte Nachricht wird dann an den entsprechenden Vater weitergeleitet, wenn der Zähler vor einer Erniedrigung kleiner ist als das Timer-Intervall. Beim Empfang einer aggregierten Nachricht wird das Empfangs-Ereignis nur dann ausgelöst, wenn Ziel (Kennung der Senke) und die lokale Kennung übereinstimmen.

Obwohl die beim Spannbaum-Routing mögliche Aggregierung nicht schwer zu implementieren ist, wurde hierauf verzichtet. Es findet keine echte Datenaggregierung statt. Für die Simulation ist es ausreichend, nur einen Zähler zu verschicken, der die Anzahl aggregierter Datensätze anzeigt. Dieser entspricht der Anzahl Knoten, deren Daten aggregiert wurden. Um trotzdem die unterschiedlichen (und zu den Senken hin wachsenden) Paketgrößen zu simulieren, werden diese für gegebene Anzahl eingesammelter Epochen und Ereignisse berechnet und entsprechend große Pakete verschickt.

5.1.7 Steuerung der Applikationen

Bei den in Abbildung 5.1 sowie 5.2 gezeigten Schnittstellen App handelt es sich eigentlich um die Module, welche die Steuerung der Programme übernehmen. In der vorliegenden Arbeit sind dies nur Treiber für die Simulationen. Sie legen fest, ob ein Knoten eine Senke ist oder nicht und steuern den zeitlichen Ablauf.

Beim Fluten ruft ein Timer in festen Intervallen das Sende-Kommando der Schnittstelle SendData auf. Beim Spannbaum-Routing wird zunächst das Nachbarschaftsprotokoll gestartet und nach einer festen Zeit wieder gestoppt. Danach wird der Spannbaum-Aufbau initiiert, indem das entsprechende Kommando der Schnittstelle SpanningTree aufgerufen wird. Im Anschluss wird beim proaktiven Ansatz auf jedem Knoten in festen Intervallen der Versand der lokalen Daten veranlasst (Aufruf des Kommandos der Schnittstelle SendData). Beim aktiven Ansatz starten ausschließlich die Senken in denselben festen Intervallen das Einsammeln über die Schnittstelle Collect.

Alle drei Applikation implementieren dasjenige Ereignis, das von SendData beim Empfang von Daten ausgelöst wird. Die Applikation zur Steuerung des aktiven Spannbaum-Routing behandelt überdies das Ereignis beim Eintreffen einer Sammel-Nachricht der Schnittstelle Collect. Es wird verwendet, um das Versenden der lokalen Daten zu allen bekannten Senken anzustoßen.

5.2 Vergleich der Applikationen

Die Abbildungen 5.1 und 5.2 lassen bereits vermuten, dass die Implementierung des Flutens weniger komplex ist als die des Spannbaum-Routings. Dies wird von der Anzahl der Codezeilen (exkl. Kommentar- und Leerzeilen) in Tabelle 5.1 bestätigt: Das Fluten kommt mit rund der Hälfte an Codezeilen aus. Die eingesetzte Historie in Verbindung mit dem einfacheren SendData Modul hat zwar eine ähnliche Komplexität wie die SendData Module beider Spannbaum-Verfahren, allerdings benötigen das Nachbarschaftsprotokoll und der Aufbau der Spannbäume 770 Zeilen zusätzlichen Code. Der Aufwand für das aktive, senkeninitiierte Einsammeln der Daten beim Spannbaum-Routing beträgt hingegen nur weitere 128 Zeilen.

Diese Beobachtungen wirken sich auch auf die Größe der zur Illustration für Mica-Knoten kompilierten Programme in Tabelle 5.2 aus. Diese zeigt auch Werte für eine „leere Applikation“, die nur die gemeinsam genutzten Schnittstellen Position, MsgQueue und MultiSend verwendet. Hierdurch wird deutlich, wieviel Speicher die einzelnen Implementierungen zusätzlich zu diesen benötigen.

	Codezeilen		
	Fluten	Spannbaum (proaktiv)	Spannbaum (aktiv)
Warteschlange	201		
Multiplexer	127		
Positionsbestimmung	196		
Historie	76	—	—
Nachbarschaft	—	440	
Spannbaum	—	340	
Datenanforderung	—	—	128
Datenversand	198	295	295
Konfiguration	80	75	76
Hauptprogramm	120	129	130
Verdrahtung	37	43	51
Summe	1.035	1.846	1.983

Tabelle 5.1: Codezeilen der verschiedenen Programme mit Aufspaltung in einzelne Module inkl. Schnittstelle

Die Größe des ausführbaren Codes (ROM) ist unabhängig von der jeweiligen Konfiguration der Knoten: Für das Fluten sind im Vergleich zum Spannbaum-Routing nur ca. $\frac{3}{4}$ der Bytes für den ausführbaren Code (ROM) notwendig. Im Gegensatz zur leeren Applikation sind rund 2.300 zusätzliche Bytes erforderlich. Der Spannbaum-Ansatz hingegen benötigt 6.000 bzw. 6.400 Byte mehr.

Der Vergleich des verwendeten (statischen) Hauptspeichers (RAM) hingegen gestaltet sich schwierig, weil der Speicherbedarf unmittelbar von der jeweiligen Konfiguration der Knoten abhängt. Die Daten in Tabelle 5.2 beziehen sich beim Fluten auf eine Historie mit 50 Einträgen, beim Spannbaum-Ansatz auf eine maximale Anzahl von Bäumen durch einen Knoten auf 10. Bereits kleine Verändungen dieser Parameter verändern den Speicherbedarf deutlich: Ein einzelner Eintrag in der Historie benötigt 4 Byte, ein Eintrag eines Baumes durch einen Knoten 10 Byte plus ca. 20 Byte für die Sendelogik pro Eintrag. Beim Fluten hängt der Speicherbedarf von der Dichte des Netzes und der Anzahl maximal zulässiger Hops ab. Beim Spannbaum-Routing verursachen überdies Anzahl und Dichte der Senken einen erhöhten Speicherbedarf. Trotz dieser Parameter lässt sich erwarten, dass beim Fluten weniger Hauptspeicher benötigt wird, sofern kein großes, dichtes Netz mit nur wenigen Senken vorliegt. Nur in diesem Fall ist der Speicherbedarf der Historie groß gegenüber dem der wenigen benötigten Baumeinträge pro Knoten.

	Größe in Byte			
	Leere App.	Fluten	Spannbaum (proaktiv)	Spannbaum (aktiv)
ROM	10.046	12.306	16.086	16.420
RAM	1.108	1.778	1.906	1.947

Tabelle 5.2: Speicherbedarf der Applikationen im Vergleich zu einer leeren Applikation, kompiliert für Mica-Knoten. Das Netz ist auf 50 Knoten mit maximal 2 Hops ausgelegt. Pro Knoten können 10 Bäume zum Routing verwendet werden.

Kapitel 6

Auswertung

Zum Vergleich der implementierten Algorithmen werden diese mit TOSSIM und TinyViz simuliert. Erläuterungen zu den Simulationsabläufen und -parametern bilden den Einstieg in dieses Kapitel. Darauf folgt die Auswertung der Simulationsergebnisse. Ein kurzer Abschnitt mit Anmerkungen und Einschränkungen schließt das Kapitel ab.

6.1 Simulationsablauf und -parameter

In diesem Abschnitt werden zunächst die zu untersuchenden Parameter und ihre Wertebereiche bestimmt. Hinzu kommt ein kurzer Überblick zur Konfiguration der eingesetzten Module und schließlich eine Erläuterung der Simulationsabläufe.

6.1.1 Wertebereiche der Parameter

Bei den zu untersuchenden Parametern handelt es sich um die Größe und Dichte des Sensornetzes, die Dichte der Senken, die Anzahl der maximalen Hops und die maximalen Einzugsbereiche der Senken. Die in den Simulationen verwendeten Werte finden sich in Tabelle 6.1.

Die einzelnen Simulationen verwenden bei gleicher Anzahl Knoten immer dieselben Knotenpositionen. Diese werden dazu vorab mittels gleichverteilter Zufallszahlen auf eine Fläche von ungefähr 30x30 Metern[1] verteilt und in Positionsdateien gespeichert. Die Zahl der Knoten bestimmt damit gleichzeitig die Dichte des Netzes. Sie ist so gewählt, dass verschiedene Konnektivitäten in den Netzen resultieren (vgl. Abschnitt A.5). Tabelle 6.2 zeigt die durchschnittliche Anzahl potentieller Nachbarn eines Knotens für unterschiedliche Funkreichweiten (vgl. Abschnitt A.5).

[1] TOSSIM arbeitet auf Basis von Fuß und bietet eine virtuelle Simulationsfläche von 100x100 Fuß

Knoten	Knoten pro Senke	Maximale Distanz	Maximale Hops
20, 40, 100	1, 2, 3, 5, 10, 20	15 m, 30 m	2, 3, 4

Tabelle 6.1: Untersuchte Simulationsparameter und ihre Werte

Knoten	**Maximale Funkreichweite**			
	4 m	6 m	8 m	12 m
20	1	2	3	5
40	1	3	6	12
100	5	10	17	34

Tabelle 6.2: Gerundete, durchschnittliche Anzahl potentieller Nachbarn eines Knotens in Abhängigkeit der Knotenanzahl und der maximalen Funkreichweite

Die Dichte der Senken wird über ihren Kehrwert festgelegt: die Anzahl der Knoten pro Senke. In allen Simulationen werden die Senken durch einfaches Abzählen bestimmt, so dass diese bei gleicher Knotenzahl immer dieselbe Position besitzen. Dadurch kann für jede Simulation bestimmt werden, wie viele Knoten in den Einzugsbereichen der Senken liegen.

Die maximalen Einzugsbereiche der Senken decken einmal einen kleinen und einmal einen großen Teil des Netzes ab. Die Bestimmung der Maximalzahl erlaubter Hops leitet sich aus der Anwendung der Ergebnisse aus Abschnitt A.5 auf die gewählten Einzugsbereiche ab.

Eine Periode (zum Einsammeln der Daten) umfasst immer 6 Epochen á 10 Sekunden sowie 5 verschiedene Ereignisse pro Epoche, um die Anzahl nötiger Simulationen nicht weiter zu vergrößern. Eine geringfügige Änderung dieser Parameter erhöht die Paketgrößen nur wenig, so dass von ihnen kein entscheidender Einfluss auf die grundsätzlichen Ergebnisse aus Abschnitt 6.2 zu erwarten ist.

6.1.2 Konfiguration der Module

In diesem Abschnitt erfolgt ein Überblick über die Konfiguration der verwendeten Module. Die Werte der einzelnen Konfigurationen sind in Tabelle 6.3 zusammengefasst. Die folgenden Absätze enthalten kurze Anmerkungen zu einigen Konfigurationswerten.

Beim Nachbarschaftsprotokoll ergibt sich die geringe Zahl maximaler Nachbarn aus den Limitierungen gemäß Formel 5.1. Dasselbe gilt für die maximale Anzahl potentieller Söhne beim Spannbaum-Algorithmus bezüglich Formel 5.2. Mit den Ergebnissen aus [XK04] ist ein zusammenhängendes Netz trotzdem wahrscheinlich. Wegen Speicherplatzlimitierungen auf realen Sensorknoten ist die Anzahl von Spannbäumen, die ein Knoten ver-

Warteschlange		
Größe	Verzögerung (leer)	Verzögerung (sonst)
15	5 – 600 ms	40 – 600 ms
Nachbarschaft		
Max. Nachbarn	Hallo-Intervall	Schätzparameter α
11	2,5 s	0,1
Spannbaum-Algorithmus		
Max. Söhne	Max. Bäume pro Knoten	Wiederholungen
6	10	3

Fluten – Einsammeln der Daten			
Wartezeit	Historie	Max. Aggregation	Wiederholungen
10 – 1200 ms	100	4 Pakete	$0 – \infty$

Spannbaum-Routing – Einsammeln und Anfordern der Daten	
Wiederholungen (Einsammeln)	Wiederholungen (Anforderung)
0 – 2	2

Tabelle 6.3: Konfiguration der Module

walten kann, ebenfalls beschränkt. Die Anzahl der Wiederholungen (vgl. Abschnitt 5.1.1) einer Spannbaum-Nachricht ist auf Basis einer dedizierten Testreihe festgelegt. Gleiches gilt für die Parameter der Warteschlange. Zum Erreichen des angegebenen Aggregationsgrades beim Fluten wird die maximale Paketgröße der TinyOS-Applikation hier passend angehoben.

6.1.3 Simulationsablauf

Die Abläufe der Simulationen des Flutens und des Spannbaum-Routings unterscheiden sich nur wenig. In beiden Fällen wird die Simulationslänge so gewählt, dass 10 Perioden zum Einsammeln von Daten genutzt werden können. Damit ergibt sich für das Fluten eine Simulationszeit von 600 Sekunden (vgl. Abschnitt 6.1.1). Beim Spannbaum-Routing ist vor dem Einsammeln der Daten der Aufbau der Nachbarschaften und Spannbäume nötig. Hierfür werden insgesamt weitere 2 Perioden bzw. 120 Sekunden veranschlagt. Mit der Konfiguration aus Abschnitt 6.1.2 reichen für das Nachbarschaftsprotokoll 90 Sekunden aus. Um sicherzustellen, dass beim Aufbau der Spannbäume keine Hallo-Nachrichten mehr verschickt werden, wird nach dem Anhalten des Nachbarschaftsprotokolls eine Epoche gewartet. Danach startet der Aufbau der Spannbäume. Für jede Kombination der Parameter aus Tabelle 6.1 werden zwei Simulationsläufe unter Einsatz des Flutens durchgeführt. Für das aktive und proaktive Spannbaum-Routing werden jeweils drei Simulationen durchgeführt.

6.2 Ergebnisse

In diesem Abschnitt erfolgt die Analyse der zuvor beschriebenen und implementierten Routing-Algorithmen. Die zur Illustration verwendeten Graphen zeigen die Mediane der jeweiligen Messungen. Aufgrund der wenigen Messergebnisse pro Parameterkombination werden keine Statistiken mit Quartilen und Ausreißern verwendet.

6.2.1 Erfolgsquote

Die Erfolgsquote bezeichnet die Anzahl eingesammelter Daten im Verhältnis zum theoretisch möglichen Maximum, das durch die Knotenpositionen und Einzugsbereiche festgelegt ist. Bei der Analyse der Daten hat sich herausgestellt, dass die veranschlagte Zeit zum Aufbau der Spannbäume in vielen Fällen nicht ausgereicht hat. Dies hat sich durch eine deutlich geringere Menge eingesammelter Daten in der ersten Periode gezeigt. Aus diesem Grund wird diese erste Periode bei der Auswertung der Erfolgsquote beim Spannbaum-Routing nicht verwendet.

Abbildung 6.1 zeigt die Erfolgsquoten der drei simulierten Verfahren bei 40 Knoten. Wie erwartet sind sie beim Fluten in den Abbildungen 6.1(a) und 6.1(b) unabhängig von der Dichte der Senken. Die leichten Schwankungen lassen sich durch die unterschiedlichen Positionen der Senken bei Variation der Senkendichte und die geringe Menge an Messwerten erklären. Mit zunehmender Hopzahl steigt auch die Erfolgsquote an, jedoch nicht proportional. Dies lässt sich durch folgende Überlegungen erklären: Bei den gegebenen Funkreichweiten können die meisten Knoten eine Senke, in deren Einzugsbereich sie liegen, mit 2 Hops erreichen. Die Erhöhung der Hopzahl trägt im Wesentlichen durch höhere Redundanz – ein Knoten kann eine Senke so über mehrere Pfade erreichen – zur Steigerung der Erfolgsquote bei. Sie vergrößert die Menge derjenigen Knoten, die eine Senke mit weniger Hops nicht erreichen können, nur geringfügig.

Die Vergrößerung des Einzugsgebiets in Abbildung 6.1(b) bewirkt eine deutliche Reduzierung der Erfolgsquote. Mit den Überlegungen des vorangegangen Absatzes ist diese Beobachtung nachvollziehbar: Die meisten der zu den Einzugsgebieten der Senken hinzugewonnenen Knoten können die Senken mit 2 Hops nicht oder nur schlecht erreichen. Dieselben Einsichten begründen auch, warum sich die Erhöhung der Hopzahl beim größeren Einzugsgebiet relativ gesehen stärker auswirkt. Der Zuwachs ist allerdings trotzdem gering, weil die Pfadwahrscheinlichkeiten bei steigender Hopzahl abnehmen.

Beim proaktiven Spannbaum-Routing in den Abbildungen 6.1(c) und 6.1(d) zeigt sich ein anderes Bild: Bei geringer Senkendichte liegt die Erfolgsquote bei 3 und 4 Hops ungefähr doppelt so hoch wie beim Fluten. Bei nur 2 Hops lässt sich noch ungefähr das Resultat

des Flutens mit 4 Hops erreichen. Mit zunehmender Senkendichte verringert sich die Erfolgsquote, bis sie schließlich bei ca. 50 Prozent Senkendichte auf die erzielte Quote beim Fluten abgesunken ist. Bei höheren Dichten erweist sich das Fluten als erfolgreicher. Hierfür gibt es verschiedene Ursachen: Die Spannbäume werden mit steigender Senkendichte kleiner (vgl. Abbildung 6.2(a)), was sich durch mehr Paketkollisionen und die Limitierung der maximal pro Knoten verwaltbaren Bäume begründen lasst. Außerdem verschickt jeder Knoten beim Spannbaum-Routing ein Datenpaket pro bekannter Senke, so dass bei höherer Senkendichte der Funkverkehr ansteigt. Damit steigt auch beim Einsammeln der Daten die Wahrscheinlichkeit von Paketkollisionen. In Bezug auf die Erhöhung der Einzugsgebiete in Abbildung 6.1(d) lässt sich wiederum eine Reduzierung der Erfolgsquoten beobachten. Auch hier greifen die vorangegangenen Überlegungen bzgl. der Erhöhung der Einzugsgebiete beim Fluten.

Für das aktive Spannbaum-Routing in den Abbildungen 6.1(e) sowie 6.1(f) gelten dieselben Beobachtungen und Erläuterungen wie im proaktiven Fall. Die Erfolgsquoten sind allerdings zum Teil um einige Prozentpunkte geringer. Die Erklärung hierfür liegt im aktiven Anfordern der Daten: Paketverluste führen dazu, dass nicht alle Knoten über das Einsammeln der Daten informiert werden.

Die Erhöhung der maximal zulässigen Spannbaumtiefen führt zu einer Steigerung der Erfolgsquoten beim Einsammeln der Daten analog zum Fluten. Es lässt sich erkennen, dass der Einfluss der höheren erlaubten Spannbaumtiefen in Netzen mit geringer Senkendichte einen stärkeren Einfluss hat als in Netzen mit höherer Senkendichte. Bei einer größeren Hopzahl steigt die Wahrscheinlichkeit, dass ein Knoten während des Spannbaumaufbaus den Vater in einem bekannten Spannbaum durch einen anderen, besseren ersetzt. Dies führt in Kombination mit vielen Senken, die einen Spannbaum aufbauen, zu erhöhtem Funkverkehr und Kollisionen. Die vergleichsweise starken Schwankungen und Unterschiede zwischen den einzelnen Darstellungen in den Abbildungen 6.1(c)–6.1(f) – hervorgerufen durch die geringe statistische Grundlage – verhindern jedoch präzisere Analysen.

Die Ergebnisse und Resultate dieses Abschnittes gelten im Wesentlichen auch für die Simulationen mit 20 bzw. 100 Knoten. Allerdings gibt es einige Besonderheiten. Bei 100 Knoten liegen die Erfolgsquoten beim Fluten immer über denen des Spannbaum-Routings. Aufgrund der hohen Netzdichte ergeben sich viele asymmetrische Einträge in den Nachbarschaftstabellen, d.h. es existieren nur wenige bidirektionale Nachbarn (vgl. Tabelle 6.2). Dadurch sind die vom Verfahren generierten Spannbäume sehr klein (vgl. Abbildung 6.2(c)) und damit auch die Erfolgsquote. Beim Fluten liegt sie im Mittel zwischen 10 und 20 Prozent, beim Spannbaum-Routing und einer mehr als zwanzigprozentigen Senkendichte sogar unter 10 Prozent. Die Resultate für 20 Knoten weisen starke Streuungen

auf, insbesondere bei geringen Senkendichten. Bei einer Senkendichte von 5 Prozent existiert in einem Netz mit 20 Knoten nur eine Senke. Alle Werte beziehen sich dann auf diese eine Senke und beruhen nicht auf der Mittelung mehrerer Senken. Die Ergebnisse dieser Simulationen sind daher nur schwer auswertbar.

6.2.2 Spannbaumgrößen

Für das Einsammeln der Daten mittels Spannbaum-Routing ist die Größe der Bäume ein entscheidender Faktor, so dass dieser näher untersucht werden soll. Die Abbildungen 6.2(a) sowie 6.2(c) zeigen die Spannbaumgrößen im Verhältnis zu den theoretischen Maxima in einem Netz mit 40 bzw. 100 Knoten bei 15 m Einzugsbereich der Senken. Die Analyse der Baumgrößen beruht auf nur 6 Werten pro Experiment, so dass sich nur näherungsweise, qualitative Schlüsse ziehen lassen.

Der Kurvenverlauf in Abbildung 6.2(a) zeigt Ähnlichkeiten zu Abbildung 6.1(c): Bei zunehmender Senkendichte werden die Spannbäume bzw. Erfolgsquoten kleiner. Dies lässt zunächst auf mehr Paketkollisionen schließen, ist allein jedoch nicht die gesuchte Erklärung. Durch die Limitierung der maximal pro Knoten verwaltbaren Spannbäume kann die theoretische Maximalgröße bei hoher Senkendichte nicht erreicht werden. Die möglichen Baumgrößen in Abhängigkeit der Senkendichte und Größe der Einzugsbereiche zeigen die Abbildungen 6.3(a) und 6.3(b) für 40 bzw. 100 Knoten. Dieses Wissen kann man bei der Betrachtung der relativen Baumgrößen anwenden. Das Ergebnis für 15 m Einzugsbereich und 40 sowie 100 Knoten zeigen die Abbildungen 6.2(b) bzw. 6.2(d). Sie deuten an, dass unter Berücksichtigung der technischen Limitierungen die Senkendichte keinen entscheidenden, negativen Einfluss auf die Spannbaumgröße hat. Bei 4 Hops erreichen die Bäume in einem Netz mit 40 Knoten, 15 m Einzugsbereich und einer hundertprozentigen Senkendichte unter Anwendung der o.g. Betrachtungen eine relative Baumgröße von 80 Prozent gegenüber 40 Prozent in Abbildung 6.2(b). Bei der Analyse eines Netzes mit 100 Knoten in Abbildung 6.2(d) wirkt sich die Erkenntnis aus Abbildung 6.3(b) auch bei geringeren Dichten aus. Ein entscheidendes Resultat aus diesen Analysen besteht darin, dass beim Aufbau eines Sensornetzes zum Einsammeln von Daten mittels Spannbaum-Routing unbedingt darauf zu achten ist, ob die Senken überhaupt alle Daten einsammeln können. Insbesondere ist zu berücksichtigen, dass ein Knoten nur einer beschränkten Anzahl Senken zum Einsammeln seiner Daten zur Verfügung steht.

Die in den vorangegangenen Absätzen gewonnenen Erkenntnisse erklären jedoch nicht, warum die relative Baumgröße bei 100 Knoten unter 20 Prozent liegt (vgl. Abbildung 6.2(c)), denn auch unter Betrachtung der Ergebnisse aus Abbildung 6.3(b) steigen die relativen

Baumgrößen nicht über rund 40 Prozent (vgl. Abbildung 6.2(d)). Paketkollisionen scheiden als zentrale Ursache aus, weil dann in Netzen mit geringer Senkendichte große Spannbäume existieren müssten. Es zeigt sich, dass bereits die Ausgangssituation zum Aufbau der Bäume schlecht ist. Tabelle 6.2 deutet in Kombination mit der geringen Größe der Nachbarschaftstabellen (vgl. Abschnitt 6.1.2) bereits an, dass nur wenige bidirektionale Links zustandekommen.

Dies liegt am verwendeten Nachbarschaftsprotokoll, das bidirektionale Links identifiziert. Ist eine Tabelle kleiner als die Anzahl der potentiellen Nachbarn eines Knoten, findet nur eine Teilmenge dieser Knoten in der Tabelle Platz. Bei dichten Netzen tritt dieser Fall häufig auf und führt dazu, dass ein Knoten einen Nachbarn nicht als solchen erkennen kann, weil er nicht in dessen Tabelle eingetragen ist. Eine Analyse der Protokolldateien belegt, dass in den Netzen mit 100 Knoten im Mittel nur rund 100 bidirektionale Links existieren, d.h. jeder Knoten im Schnitt nur 2 Nachbarn hat. Für den Spannbaumaufbau bedeutet dies, dass ein Knoten bei Erhalt einer Spannbaum-Nachricht diese im Mittel nur an einen Nachbarn weiterleiten kann. In einem Netz mit 40 Knoten hingegen liegt die Anzahl von Nachbarn pro Knoten durchschnittlich in einer Größenordnung von 4 bis 5. Daraus folgt die Erkenntnis, dass zum erfolgreichen Einsatz des Spannbaum-Routings in dichten Netzen eine Lösung für diese Nachbarschaftsproblematik gefunden werden muss.

Die Beobachtungen in den vorangegangenen Absätzen legen nahe, die Erfolgsquote aus Abschnitt 6.2.1 nicht nur in Bezug auf das theoretische Maximum zu betrachten, sondern außerdem einen Vergleich in Bezug auf die tatsächlich erreichten Spannbaumgrößen anzustellen. Eine solche Auswertung zeigen die Abbildungen 6.4(a) und 6.4(b) für das proaktive bzw. für das aktive Spannbaum-Routing exemplarisch anhand von 40 Knoten und einem Einzugsbereich von 15 m. Im Vergleich zu den Ergebnissen aus Abbildung 6.1 zeigt sich eine deutlich geringere Abhängigkeit von der Senkendichte. Dies belegt, dass mit dem in dieser Arbeit entwickelten Spannbaum-Routing prinzipiell eine hohe Erfolgsquote erzielt werden kann, sofern die Bäume eine hohe relative Größe erreichen.

6.2.3 Datenverkehr

Neben der Erfolgsquote in Abschnitt 6.2.1 ist auch das Funkdatenvolumen des Netzes eine wichtige Charakteristik der verschiedenen Verfahren, da dieses einen Rückschluss auf die verbrauchte Energie der Knoten zulässt. Abbildung 6.5 zeigt den Aufwand pro eingesammeltem Datum – einmal gemessen in versendeten Paketen und einmal in Byte.

Beim Fluten hängt die Anzahl Pakete und das hiermit verbundende Datenvolumen von der Dichte des Netzes ab. Abbildung 6.5(a) zeigt dies anschaulich. Diese Beobachtung

lässt sich wie folgt begründen: Beim Fluten verändern sich das Datengesamtvolumen und die Anzahl verschickter Pakete nicht in Abhängigkeit der Senkendichte, weil das Verfahren unabhängig hiervon funktioniert. Bei hoher Senkendichte steigt jedoch die Menge eingesammelter Daten. Folglich erhöht sich die Erfolgsquote gemäß Abschnitt 6.2.1. Die Kurve in Abbildung 6.5(a) belegt dies: Bei einer Verdopplung der Senkendichte erfolgt eine Halbierung der Paketmenge pro eingesammeltem Datum. Die Menge verschickter Bytes in Abbildung 6.5(b) besitzt denselben Verlauf. Es lässt sich ablesen, dass pro Paket ungefähr 25 Byte verschickt werden. Dies entspricht einer mittleren Aggregation von rund 1,5 Daten pro Paket. Eine Untersuchung der Ursache für diesen geringen Aggregationsgrad konnte aus zeitliche Gründen nicht durchgeführt werden.

Die Anzahl verschickter Pakete pro eingesammeltem Datum ist beim Spannbaum-Routing fast konstant (vgl. Abbildungen 6.5(c) und 6.5(e)). Die leichte Zunahme bei steigender Senkendichte lässt sich durch mehr Paketkollisionen erklären, die beim Einsammeln eine höhere Zahl von Wiederholungen verursachen. Da sowohl die Erfolgsquote als auch die Baumgrößen mit zunehmender Senkendichte ungefähr in gleichem Maße zunehmen, erscheint diese Beobachtung plausibel. Die größere Anzahl verschickter Pakete beim aktiven Ansatz liegt am zusätzlichen Versand der Sammel-Nachrichten (vgl. Tabelle 6.3). Während die Hopzahl beim proaktiven Ansatz einen kleinen Einfluss auf das Verhalten in Abbildung 6.5(c) hat, ist sie beim aktiven Anfordern in Abbildung 6.5(e) stärker ausgeprägt. Diese Verstärkung ist ebenfalls in der Verwendung der Sammel-Nachrichten begründet. Der Umstand, dass beim aktiven Anfordern bei 2 und 3 Hops keine Veränderung der relativen Paketzahl auftritt, ließ sich nicht klären, weil die Auswertungen bei einem Einzugsgebiet von 30 Metern dieses Verhalten nicht bestätigen. Die Menge verschickter Bytes in den Abbildungen 6.5(d) unf 6.5(f) ist ungefähr proportional zur Paketzahl. Die hier präsentierten Ergebnisse lassen sich auch auf die Experimente mit 20 und 100 Knoten übertragen.

Die bisherigen Auswertungen berücksichtigen noch nicht den Aufwand zur Bestimmung der Nachbarschaften und zum Aufbau der Spannbäume. Die hierzu verwendeten Verfahren wurden auch nicht auf Effizienz in Form von geringem Funkverkehr optimiert. Eine nähere Untersuchung dieser Thematik ist daher nicht Bestandteil dieser Arbeit.

6.2.4 Fazit und Konsequenzen

Es hat sich gezeigt, dass keines der eingesetzten Verfahren in seiner jetzigen Form eine optimale Lösung für das Einsammeln der Daten zum Generieren von Ereignismustern darstellt. In Netzen mit geringer Knoten- und Senkendichte bietet das angewendete Spann-

baumverfahren eine höhere Erfolgsquote. Bei hoher Netzdichte hingegen ist es wegen der ungelösten Nachbarschaftsproblematik und daraus resultierenden kleinen Spannbäumen kaum einsetzbar. Allerdings liefert hier auch das Fluten schlechte Werte. In Netzen geringer Knoten- aber hoher Senkendichte zeigt sich das Fluten als bessere Lösung. Der Spannbaum-Ansatz erweist sich hier insbesondere wegen der Limitierung der maximal auf einem Knoten verwaltbaren Spannbäume als die schlechtere Wahl.

Bezüglich der Funk- und in eingeschränkter Weise auch der Energieeffizienz lässt sich feststellen, dass sich das Spannbaum-Verfahren bis zu einer Senkendichte von rund 20 Prozent effizienter darstellt. Im Bereich von 20–50 Prozent Senkendichte ist das Fluten offenbar effektiver, weil pro eingesammeltem Datum weniger Bytes verschickt werden, allerdings hat das Spannbaum-Verfahren in diesem Bereich eine höhere Erfolgsquote. Ab einer Senkendichte von über 50 Prozent ist das Fluten in beiden Belangen effektiver bzw. erfolgreicher. Diese Aussagen berücksichtigen nicht den Aufwand, der zum Erstellen der Spannbäume und Nachbarschaften notwendig ist.

Bei den Spannbaumansätzen lassen sich zwei weitere wichtige Aussagen festhalten: Zum einen liefert der aktive Ansatz erwartungsgemäß eine etwas schlechtere Erfolgsquote. Zum anderen verursacht er einen höheren Funkverkehr. Der Ansatz verspricht damit nur dann eine bessere Energieeffizienz, wenn nur selten Daten von den Senken eingesammelt werden müssen. Zu genaueren Aussagen diesbezüglich sind weitere Untersuchungen notwendig, insbesondere solche, die ein asynchrones Einsammeln der Daten durch die Senken realisieren. In diesem Fall lässt sich eine etwas höhere Erfolgsquote erwarten, weil weniger Kollisionen zu erwarten sind.

Die Tatsache, dass keines der Verfahren in der Lage ist, alle Daten aus der Nachbarschaft einzusammeln, birgt für die Applikation aus Kapitel 2 folgende Konsequenzen: Beim Generieren und Evaluieren von Ereignismustern muss die z.T. geringe statistische Grundlage bei der Berechnung berücksichtigt werden. Zum anderen kann keine klare Aussage bezüglich der Häufigkeitspartitionen getroffen werden, weil hierzu alle Daten eingesammelt werden müssten.

6.3 Anmerkungen und Einschränkungen

TinyViz hat sich als einfach zu handhabende Plattform zur Simulation von Sensornetz-Applikationen, die für reale Netze und TinyOS konzipiert sind, präsentiert. Die Ergebnisse sind in weiten Teilen nachvollziehbar und geben gute Anhaltspunkte zum Verhalten der Applikationen in einem realen Netz. Eine einzelne Simulation benötigt jedoch bereits bei 100 Knoten und 600 s Simulationszeit auf einem Pentium D mit 3,4 Ghz und 2 Gb

Knoten	20	40	60	80	100
Zeit	9:08 Min.	19:32 Min.	29:47 Min.	40:53 Min.	52:02 Min.

Tabelle 6.4: Benötigte reale Zeit in Abhängigkeit der Knotenanzahl für einen 600 s TinyViz-Simulationsdurchlauf mit Fluten

	Fluten	Spannbaum (proaktiv)	Spannbaum (aktiv)
TOSSIM	18:51 Min.	18:54 Min.	19:19 Min.
TinyViz	19:32 Min.	19:49 Min.	21:20 Min.

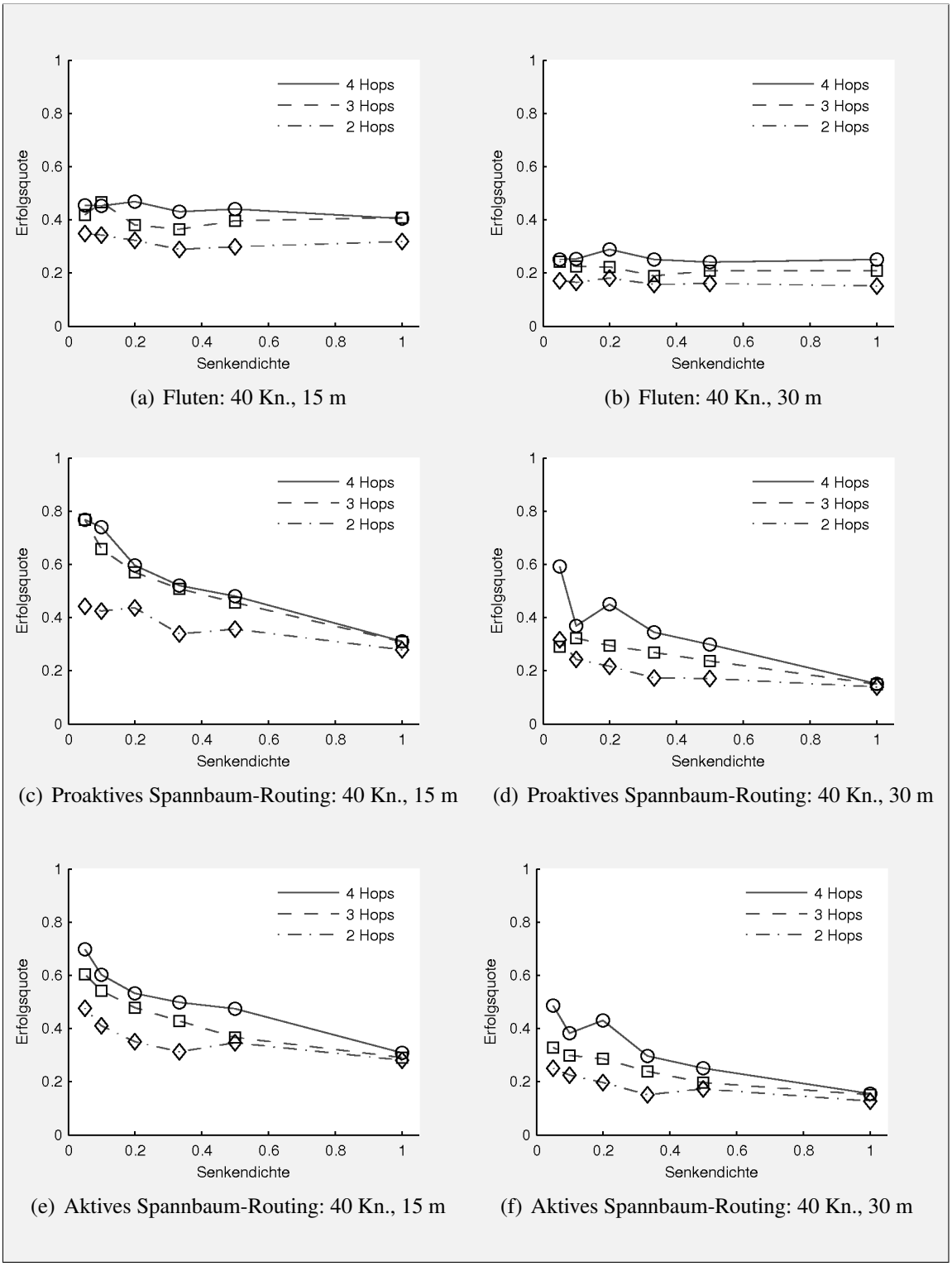
Tabelle 6.5: Vergleich der Ausführungszeiten zwischen TOSSIM und TinyViz bei 40 Knoten und 600 s Simulationszeit

RAM rund 50 Minuten. Die Werte aus Tabelle 6.4 legen dabei einen linearen Zusammenhang zwischen benötigter Zeit und und simulierten Knoten nahe. Die Vermutung, dass das in Java implementierte TinyViz eine deutliche Verlangsamung der TOSSIM-Simulationen verursacht, hat sich in Bezug auf die Messungen in Tabelle 6.5 nicht bestätigt.

Die langen Simulationszeiten erwiesen sich in der vorliegenden Arbeit als problematisch, weil durch die Vielzahl von Parameter-Kombinationen nur wenige Simulationen pro Kombination möglich waren. Eine verlässliche statistische Grundlage der Messwerte konnte so nicht erzielt werden. Aus zeitlichen Gründen war es damit außerdem nicht möglich, weitere Simulationen mit variierten Knotenanordnungen, mehr Knoten oder anderen Senkendichten durchzuführen. Die in Abschnitt 6.2 präsentierten Ergebnisse sind damit nur vergleichsweise vage Anhaltspunkte.

Eine umfassende Auswertung der mit PowerTOSSIM generierbaren Energiestatistiken war aus zeitlichen Gründen leider nicht mehr möglich. Eine stichprobenartige Analyse scheiterte an einem internen Fehler des PowerTOSSIM-Skripts, der nicht behoben werden konnte.

Bei der Auswertung der Simulationen hat sich gezeigt, dass viele Empfangsbestätigungen beim Einsammeln der Daten mittels Spannbaum-Routing verloren gegangen sind. Auch die empirisch festgelegte Anzahl an Wiederholungen beim Spannbaumaufbau deutet auf viele Paketverluste hin. Es wäre somit interessant, dieselben Applikationen in einem realen Sensornetz zu testen, um einerseits die Effizienz in realen Netzen zu beobachten und andererseits einen zuverlässigen Eindruck über die Aussagekraft der Simulationsergebnisse zu gewinnen.



(a) Fluten: 40 Kn., 15 m

(b) Fluten: 40 Kn., 30 m

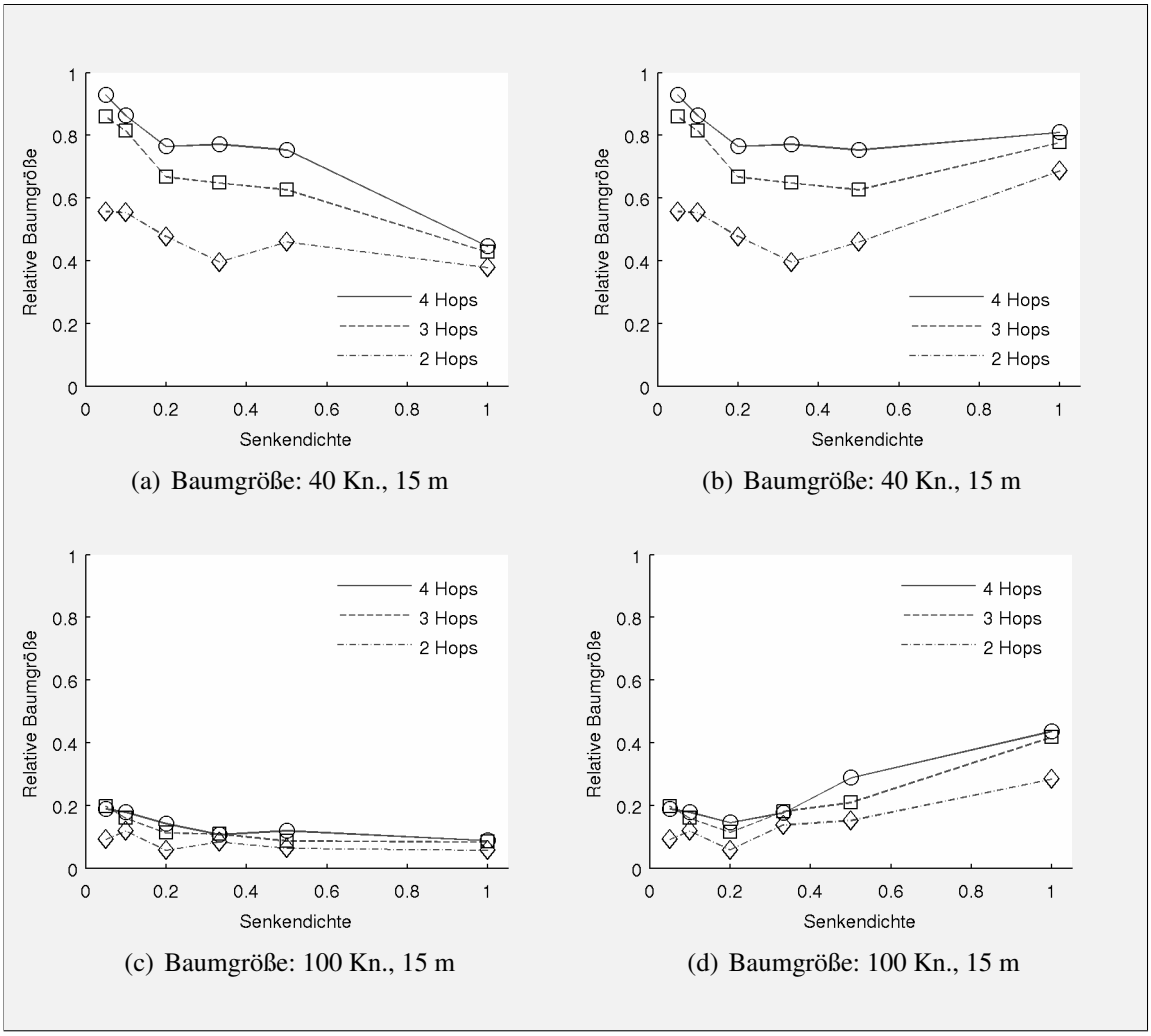
(c) Proaktives Spannbaum-Routing: 40 Kn., 15 m

(d) Proaktives Spannbaum-Routing: 40 Kn., 30 m

(e) Aktives Spannbaum-Routing: 40 Kn., 15 m

(f) Aktives Spannbaum-Routing: 40 Kn., 30 m

Abbildung 6.1: Erfolgsquote beim Einsammeln der Daten bzgl. der Einzugsbereiche



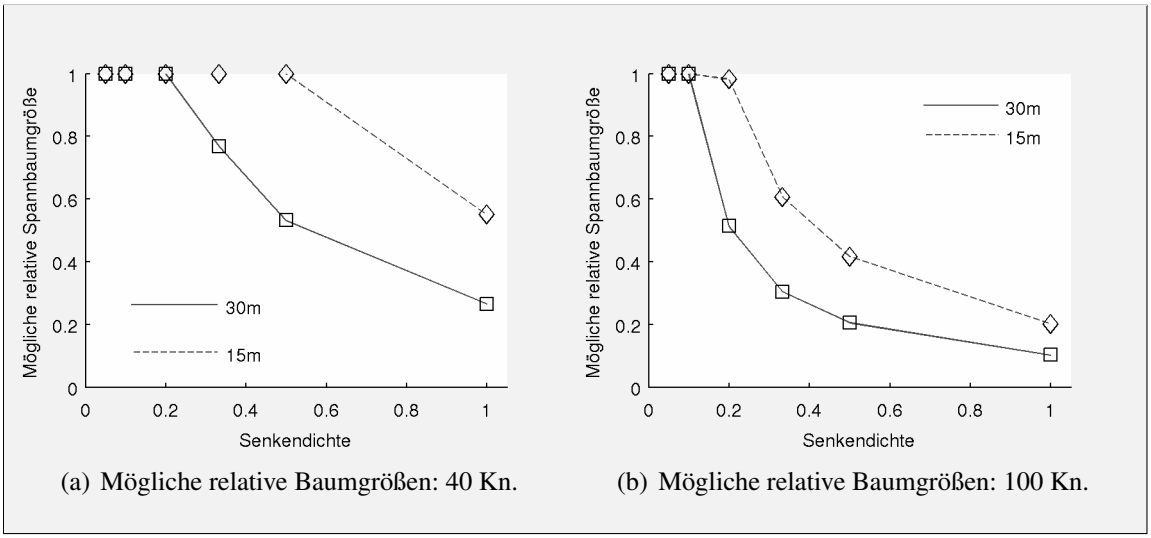
(a) Baumgröße: 40 Kn., 15 m

(b) Baumgröße: 40 Kn., 15 m

(c) Baumgröße: 100 Kn., 15 m

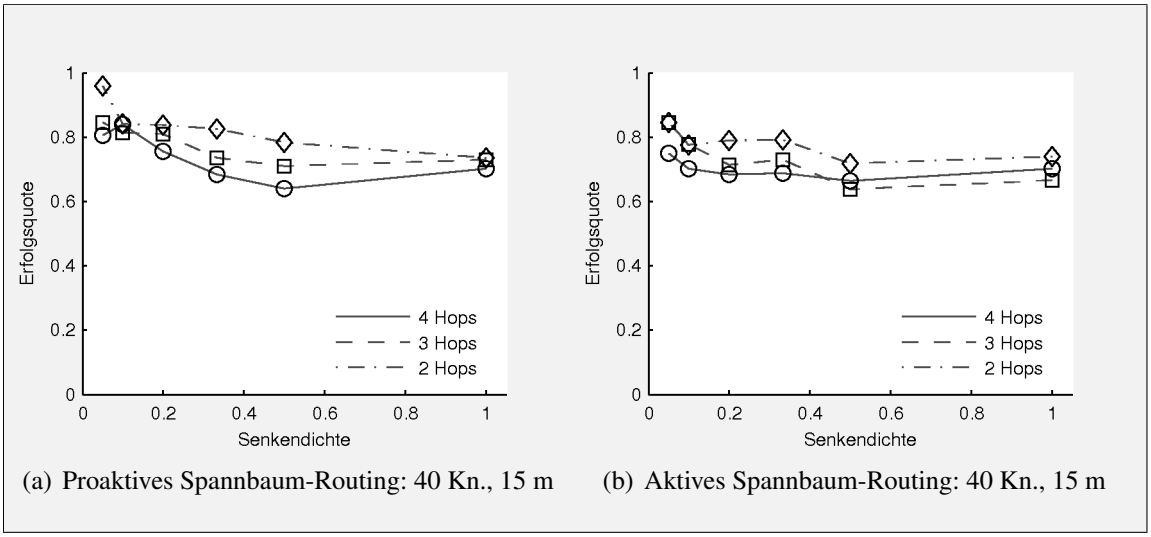
(d) Baumgröße: 100 Kn., 15 m

Abbildung 6.2: Relative Baumgrößen einmal in Bezug auf den Einzugsbereich (links) und einmal unter Berücksichtigung der Beschränkung aus Tabelle 6.3 bzgl. der verwaltbaren Bäume pro Knoten



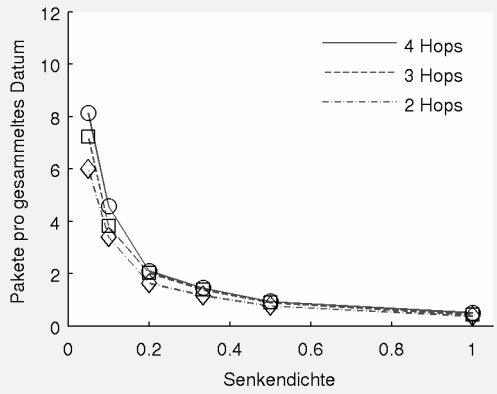
(a) Mögliche relative Baumgrößen: 40 Kn. (b) Mögliche relative Baumgrößen: 100 Kn.

Abbildung 6.3: Mögliche relative Baumgrößen durch Limitierungen gemäß Tabelle 6.3

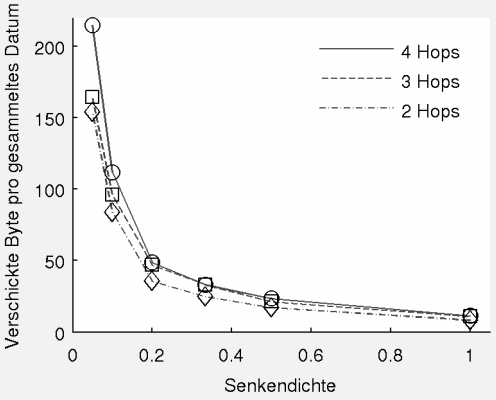


(a) Proaktives Spannbaum-Routing: 40 Kn., 15 m (b) Aktives Spannbaum-Routing: 40 Kn., 15 m

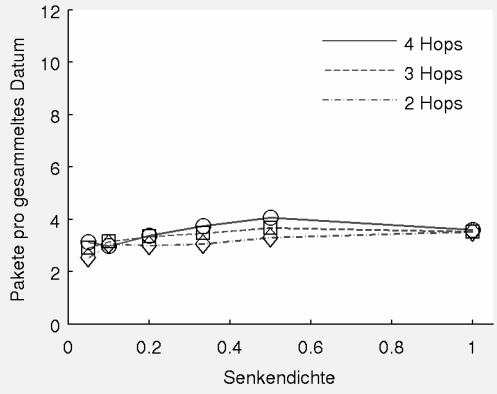
Abbildung 6.4: Erfolgsquote beim Einsammeln der Daten bzgl. der Spannbaumgrößen



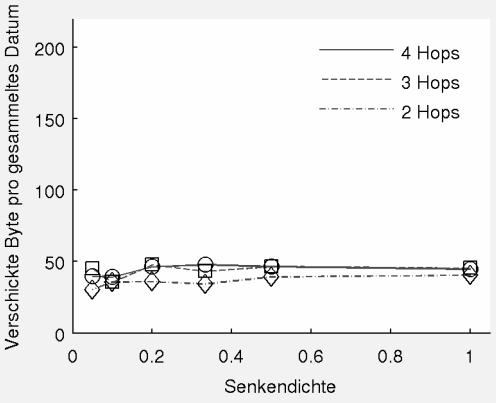
(a) Fluten: 40 Kn., 15 m



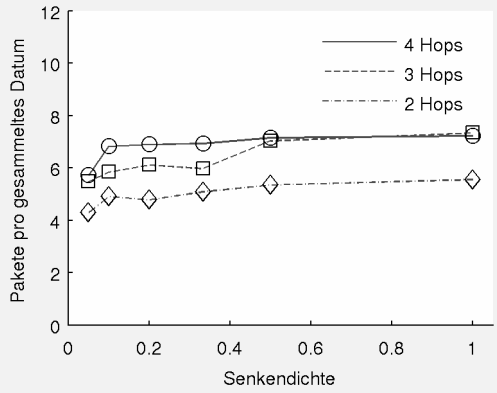
(b) Fluten: 40 Kn., 15 m



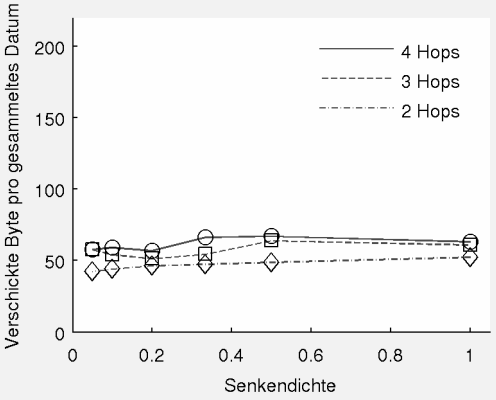
(c) Proaktives Spannbaum-Routing: 40 Kn., 15 m



(d) Proaktives Spannbaum-Routing: 40 Kn., 15 m



(e) Aktives Spannbaum-Routing: 40 Kn., 15 m



(f) Aktives Spannbaum-Routing: 40 Kn., 15 m

Abbildung 6.5: Verschickte Pakete (links) und Byte (rechts) pro eingesammeltem Datum

Kapitel 7

Zusammenfassung

Der Einsatz drahtloser Sensornetze hat sich in den letzten Jahren verändert: Erste Applikationen basierten auf dem Messen von Daten durch Sensorknoten, die diese dann an eine einzige Senke schickten. Von dort gelangten die Daten auf einen PC, der dann die Auswertung übernahm. Viele moderne Anwendungen setzen auf den Datenaustausch zwischen Knoten oder verwenden mehrere Senken, die Daten aus einer lokalen Umgebung einsammeln und autonom verarbeiten.

Der Überblick zu aktuellen Ansätzen zur lokalen Datenaggregation in drahtlosen Sensornetzen hat gezeigt, dass noch nicht für jede denkbare Applikation eine passende Programmierabstraktion existiert, welche die Daten in einer lokalen Umgebung effizient einsammelt. Dies ist zum Beispiel für das Generieren häufig auftretender verteilter räumlicher Ereignismuster in drahtlosen Sensornetzen der Fall. Aus diesem Anlass beschäftigt sich die vorliegende Arbeit mit einer ersten Entwicklung und Analyse von Routing-Verfahren, die genau auf diesen Anwendungsfall zugeschnitten sind.

Beim Design dieser Routing-Algorithmen werden bekannte Verfahren eingesetzt: Fluten und Spannbaum-Routing. Während das erste Verfahren einfach zu implementieren ist, aber vergleichweise wenige Möglichkeiten zur Aggregation bietet, erfordert das zweite einigen Zusatzaufwand wie z.B. die Anwendung von Nachbarschaftsprotokollen und das Aufbauen der Spannbäume. Dafür ergeben sich gute Aggregationsmöglichkeiten. Beide Varianten unterstützen das periodische (proaktive) Einsammeln von Daten durch die Senken, indem die Knoten ihre Daten in Richtung der Senken schicken. Der Spannbaum-Ansatz bietet hierüber hinaus eine gute Möglichkeit, Daten nur bei Bedarf (aktiv) einzusammeln. Der Baum wird in diesem Fall sowohl zum Benachrichtigen der Knoten in der lokalen Umgebung der Senke als auch zum Einsammeln der Daten verwendet.

Ein zentraler Aspekt der vorliegenden Arbeit ist die Anpassung der genannten, grundlegenden Verfahren an die Besonderheiten des Einsammelns lokaler Daten für das Generieren von Ereignismustern. Die entstehenden Algorithmen werden dabei im Detail be-

schrieben. Der zweite Schritt ist die Implementierung dieser Algorithmen als TinyOS-Applikationen. Damit bietet sich einerseits eine einfache Analyse-Möglichkeit in Form von Simulationen mit TOSSIM an, andererseits kann der so entstehende Code unmittelbar in realen Sensornetzen eingesetzt werden.

Zur Analyse der in dieser Arbeit entwickelten Algorithmen werden Simulationen mit den TinyOS-Implementierungen durchgeführt. Als Fehlermodell beim Paketversand wird das von TOSSIM bereitgestellte, empirische Modell verwendet. Bei den untersuchten Parametern handelt es sich um die Dichte des Netzes und der Senken, die Größe ihres Einzugsgebiets und die maximale Hopanzahl, die zum Routing verwendet werden darf.

Die Auswertungen haben dabei gezeigt, dass aufgrund von Paketkollisionen insbesondere in Netzen hoher Dichte nur ein kleiner Teil der verfügbaren Daten von den Senken eingesammelt werden kann. Selbst bei wenigen Senken in Netzen niedriger Dichte werden wegen Paketverlusten nur Erfolgsquoten von 70–80 Prozent erreicht. Mit steigender Hopzahl kristallisiert sich eine bessere Erfolgsquote beim Spannbaum-Routing gegenüber dem Fluten heraus. Dabei ist zu beobachten, dass hierzu mehr Funkverkehr notwendig ist. Einerseits müssen Paketverluste durch Wiederholungen kompensiert werden, weil im Gegensatz zum Fluten keine Redundanz durch mehrere mögliche Pfade existiert. Andererseits benötigt das Aufbauen der Bäume bereits den Versand vieler Funk-Nachrichten. Für das aktive Spannbaum-Routing ist festzuhalten, dass sich Paketverluste bereits beim Anfordern der Daten auswirken, so dass die Erfolgsquote hier geringer ausfällt als beim proaktiven Verfahren. Das Verfahren lohnt sich hinsichtlich der Energieeffizienz erst, wenn die Senken nur selten Daten einsammeln. Schließlich verursacht das Einsammeln weiteren Funkverkehr.

Die geringe Ausbeute beim Fluten ist im Wesentlichen auf Paketkollisionen beim Einsammeln der Daten zurückführbar. Der Spannbaum-Ansatz hingegen leidet hauptsächlich darunter, dass in dichten Netzen nur kleine Spannbäume entstehen. Dies liegt jedoch nur nebensächlich an Paketkollisionen oder -verlusten, denn diese werden durch Wiederholungen der Pakete häufig kompensiert. Das Problem liegt im Wesentlichen in der Bestimmung der Nachbarschaften, die dann zum Aufbau der Bäume verwendet werden. Im proaktiven Ansatz ließe sich diese Problematik durch den Verzicht auf die Bestimmung bidirektionaler Nachbarschaften umgehen. Allerdings bedürfte es dann eingehender Überlegungen zur in dieser Arbeit eingesetzten Spannbaumoptimierung durch Link-Qualitäten.

Die Ergebnisse dieser Arbeit bereiten den Grund für weitere Forschungen auf diesem Gebiet. Zu untersuchen sind beispielsweise Methoden, welche die beschriebene Nachbarschaftsproblematik lösen. Dies ist insbesondere für Sensornetze hoher Dichte von Relevanz. Des Weiteren bedarf es einer Steigerung der Baumgrößen, um die Datenausbeu-

te zu verbessern, sowie der Beantwortung der Frage, wie sich der Funkverkehr beim Spannbaum-Routing reduzieren lässt. Hieran schließt sich die Suche nach effizienten Algorithmen zum Aufbau von Spannbäumen an. Diese Thematik wurde in der vorliegenden Arbeit nicht betrachtet, spielt aber für den Einsatz des Spannbaum-Routings eine entscheidende Rolle. In diesem Kontext wäre ggf. auch die Entwicklung und Untersuchung eines adaptiven Verfahrens von Vorteil. Ein entsprechender Algorithmus würde verhindern, dass ein Spannbaum aufgrund von sich ändernden Nachbarschaften vollständig neu aufgebaut werden müsste. Ein vollständiger Neuaufbau wäre mit vergleichsweise hohem Aufwand verbunden, so dass eine automatische und lokale Reaktion auf derartige Änderungen attraktiv erscheint.

Literaturverzeichnis

[Beu06] BEUTEL, JAN: *Fast-Prototyping Using the BTnode Platform*. In: *Proceedings of the conference on Design, automation and test in Europe (DATE '06)*, Seiten 977–982, März 2006.

[CMP06] CICIRIELLO, P., L. MOTTOLA und G. P. PICCO: *Building Virtual Sensors and Actuators over Logical Neighborhoods*. In: *Proceedings of the international workshop on Middleware for sensor networks (MidSens '06)*, Seiten 19–24, 2006.

[CMP07] CICIRIELLO, P., L. MOTTOLA und G. P. PICCO: *Efficient Routing from Multiple Sources to Multiple Sinks in Wireless Sensor Networks*. In: *Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN '07)*, Januar 2007.

[FR05] FRANK, C. und K. RÖMER: *Algorithms for Generic Role Assignment in Wireless Sensor Networks*. In: *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*, November 2005.

[FR06] FRANK, C. und K. RÖMER: *Solving Generic Role Assignment Exactly*. In: *Workshop on Parallel and Distributed Real-Time Systems (WPDRTS '06) / IEEE International Parallel & Distributed Processing Symposium (IPDPS '06)*, April 2006.

[GLCB03] GAY, D., P. LEVIS, D. CULLER und E. BREWER: *nesC 1.1 Language Reference Manual*, Mai 2003.

[GLvB+03] GAY, D., P. LEVIS, R. VON BEHREN, M. WELSH, E. BREWER und D. CULLER: *The nesC Language: A Holistic Approach to Networked Embedded Systems*. In: *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation (PLDI '03)*, Seiten 1–11, 2003.

[HHSH06] HARTUNG, C., R. HAN, C. SEIELSTAD und S. HOLBROOK: *FireWxNet: A Multi-Tiered Portable Wireless System for Monitoring Weather Conditions in Wildland Fire Environments*. In: *Proceedings of the 4th international conference on Mobile systems, applications and services (MobiSys '06)*, Seiten 28–41, 2006.

[HSW+00] HILL, J., R. SZEWCZYK, A. WOO, S. HOLLAR, D. CULLER und K. PISTER: *System Architecture Directions for Networked Sensors*. In: *Architectural Support for Programming Languages and Operating Systems*, Seiten 93–104, 2000.

[IGE00] INTANAGONWIWAT, C., R. GOVINDAN und D. ESTRIN: *Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks*. In: *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom '00)*, Seiten 56–67, 2000.

[KS06] KIM, K.-H. und K. G. SHIN: *On Accurate Measurement of Link Quality in Multi-hop Wireless Mesh Networks*. In: *Proceedings of the 12th annual international conference on Mobile computing and networking (MobiCom '06)*, Seiten 38–49, 2006.

[Lev06] LEVIS, P.: *TinyOS Programming*, Juni 2006.

[LL03] LEVIS, P. und N. LEE: *TOSSIM: A Simulator for TinyOS Networks*, September 2003.

[LLWC03] LEVIS, P., N. LEE, M. WELSH und D. CULLER: *TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications*. In: *Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys '03)*, Seiten 126–137, 2003.

[Mat] MATTERN, F.: *Allgegenwärtige Informationsverarbeitung – Technologietrends und Auswirkungen des Ubiquitous Computing*. Erscheint 2007.

[Mat05] MATTERN, F.: *Die technische Basis für das Internet der Dinge*. In: FLEISCH, E. und F. MATTERN (Herausgeber): *Das Internet der Dinge – Ubiquitous Computing und RFID in der Praxis*, Seiten 39–66. Springer-Verlag, 2005.

[MD02] MARINA, M. K. und S. R. DAS: *Routing Performance in the Presence of Unidirectional Links in Multihop Wireless Networks*. In: *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '02)*, Seiten 12–23, 2002.

[MP06] MOTTOLA, L. und G. P. PICCO: *Programming Wireless Sensor Networks with Logical Neighborhoods*. In: *Proceedings of the 1st international conference on Integrated internet ad hoc and sensor networks (InterSense '06)*, Seite 8, 2006.

[MPR+05] MARTINEZ, K., P. PADHY, A. RIDDOCH, H. L. R. ONG und J. K. HART: *Glacial Environment Monitoring Using Sensor Networks*. In: *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN '05)*, Stockholm, Sweden, Juni 2005.

[MPS+02] MAINWARING, A., J. POLASTRE, R. SZEWCZYK, D. CULLER und J. ANDERSON: *Wireless Sensor Networks for Habitat Monitoring*. In: *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, Atlanta, Georgia, USA, September 2002.

[OTL04] OGIER, R., F. TEMPLIN und M. LEWIS: *Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)*, 2004.

[PHC04] POLASTRE, J., J. HILL und D. CULLER: *Versatile low power media access for wireless sensor networks*. In: *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys '04)*, Seiten 95–107, 2004.

[RM03] RÖMER, K. und F. MATTERN: *Drahtlose Sensornetze*. Informatik Spektrum, 26(3):191–194, Juni 2003.

[Röm06a] RÖMER, K.: *Distributed Mining of Spatio-Temporal Event Patterns in Sensor Networks*. In: *Euro-American Workshop on Middleware for Sensor Networks (EAWMS '06) / International Conference on Distributed Computing in Sensor Systems (DCOSS '06)*, Seiten 103–116, Juni 2006.

[Röm06b] RÖMER, K.: *Drahtlose Sensornetze*. Visionen 02/06, Seiten 32–36, Februar 2006.

[SHC+04] SHNAYDER, V., M. HEMPSTEAD, B. CHEN, G. W. ALLEN und M. WELSH: *Simulating the Power Consumption of Large-Scale Sensor Network Applications*. In: *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys '04)*, Seiten 188–200, 2004.

[SLR+05] SCHILLER, J., A. LIERS, H. RITTER, R. WINTER und T. VOIGT: *ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing*. In: *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Track 9 (HICSS '05)*, Januar 2005.

[SPMC04] SZEWCZYK, R., J. POLASTRE, A. MAINWARING und D. CULLER: *Lessons from a Sensor Network Expedition*. In: *Proceedings of the 1st European Conference on Wireless Sensor Networks (EWSN '04)*, Januar 2004.

[TGS06] TRIGONI, N., A. GUITTON und A. SKORDYLIS: *Routing and Processing Multiple Aggregate Queries in Sensor Networks*. In: *Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06)*, Seiten 391–392, 2006.

[TRV+05] TURAU, V., C. RENNER, M. VENZKE, S. WASCHIK, C. WEYER und M. WITT: *The Heathland Experiment: Results And Experiences*. In: *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN '05)*, Stockholm, Sweden, Juni 2005.

[TUHa] *SensorNet - Sensor Networks – Project: Heatland Experiment*. http://www.ti5.tu-harburg.de/projects/SensorNet/heathland.html.

[TUHb] *SensorNet - Sensor Networks – Project: Wireless Neighborhood Exploration (WNX)*. http://www.ti5.tu-harburg.de/projects/SensorNet/wnx.html.

[TWW06] TURAU, V., M. WITT und C. WEYER: *Analysis of a Real Multi-hop Sensor Network Deployment: The Heathland Experiment*. In: *Proceedings of the 3rd International Conference on Networked Sensing Systems (INSS '06)*, Chicago, Illinois, USA, Juni 2006.

[WM04] WELSH, M. und G. MAINLAND: *Programming Sensor Networks Using Abstract Regions*. In: *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI '04)*, Seiten 29–42, 2004.

[Woo04] WOO, A.: *A Holistic Approach to Multihop Routing in Sensor Networks*. University of California, Berkeley, 2004.

[WSBC04] WHITEHOUSE, K., C. SHARP, E. BREWER und D. CULLER: *Hood: A Neighborhood Abstraction for Sensor Networks*. In: *Proceedings of the 2nd international conference on Mobile systems, applications and services (MobiSys '04)*, Seiten 99–110, 2004.

[WTC03] WOO, A., T. TONG und D. CULLER: *Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks*. In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, Seiten 14–27, 2003.

[XK04] XUE, F. und P. R. KUMAR: *The Number of Neighbors Needed for Connectivity of Wireless Networks*. Wireless Networks, 10(2):169–181, 2004.

Anhang A

Verwendete Werkzeuge

Zum Testen der verschiedenen Algorithmen und zur Beurteilung ihrer Leistungsfähigkeit im Kontext der vorgestellten Data-Mining Applikation erweist sich ein Simulator als nützlich. Aufgrund der vielen veränderlichen Parameter und der Größe der untersuchten Netze wären Entwicklung und Test in einem realen Netz mit großem zeitlichen Aufwand verbunden. Zur grundsätzlichen Analyse der vorgestellen Verfahren bietet es sich daher an, einen Simulator einzusetzen, der eine grundsätzliche und realitätsnahe Analyse und Evaluation der getesteten Verfahren ermöglicht.

Bei der Wahl eines Simulators ist es somit von großem Interesse, gewisse Anforderungen bestmöglich zu erfüllen: So sollte es zum einen möglich sein, die zu testenden Algorithmen bei Bedarf auf realen Sensorknoten ohne große Änderungen betreiben zu können. Zum Zweiten ist eine realitätsnahe Umsetzung des Verhaltens der Knoten inklusive des Funkverkehrs wünschenswert, weil Paketverluste und -kollisionen großen Einfluss auf die Leistungsfähigkeit eines drahtlosen Sensornetzes haben. Insbesondere letztgenannter Aspekt hatte bereits in [TRV+05] einen erheblichen Einfluss auf die Zuverlässigkeit in Bezug auf den Paketerhalt über mehrere Hops hinweg. Diesen Ansprüchen genügt der Simulator TOSSIM, weil er einerseits TinyOS-Anwendungen ohne Anpassungen ausführen kann und andererseits den Funk auf der physikalischen Ebene realistisch modelliert. Hierbei verwendet der Simulator empirisch gewonnene Bitfehlerwahrscheinlichkeiten [LLWC03]. Zudem ist TinyOS auf den an der ETH Zürich verwendeten BTNodes lauffähig, was eine Verwendung der implementierten Algorithmen in der tatsächlichen Anwendungsumgebung gemäß [Röm06a] ermöglicht.

TOSSIM steht als Abkürzung für „TinyOS Simulator“ und ist integraler Bestandteil des TinyOS-Betriebssystem-Pakets [HSW+00], das den Anspruch vertritt, eine praktische und übersichtliche Bibliothek zur Programmierung von aktuellen Sensorknoten bereitzustellen. TinyOS wurde in nesC geschrieben, einer Spezialisierung und Erweiterung der Programmiersprache C für den Einsatz in der Programmierung von Sensorknoten bzw. ganzen Net-

zen. Ein weiterer interessanter Aspekt am TinyOS-Paket ist der visuelle TOSSIM-Aufsatz TinyViz, mit dem kleinere Netze grafisch betrachtet und analysiert werden können. Diese Ansammlung von Tools soll im Folgenden näher vorgestellt und ihre Verwendung erläutert werden.

A.1 nesC

Die Sprache nesC wurde für die Programmierung sogenannter eingebetteter Geräte (hierunter fallen z.B. Sensornetze) an der University of California, Berkeley, unter Mitarbeit von Intel entwickelt und entworfen. Sie ist an die weit verbreitete Sprache C angelehnt und passt diese gewissermaßen an die Bedürfnisse drahtloser Sensornetze an.

Da eingebettete Geräte Einschränkungen bezüglich Energie und Ressourcen unterliegen, stellen sich an ihre Programmierung besondere Design-Kriterien zum Umgang mit diesen Restriktionen. Ferner zeichnen sich insbesondere Sensorknoten dadurch aus, dass sie ereignisgesteuert arbeiten, also reaktiv funktionieren. Gleichzeitig eignet sich eine Modularisierung bei der Programmierung von drahtlosen Sensorknoten, um portablen Code produzieren zu können, der auf verschiedenen Plattformen läuft. nesC setzt insbesondere die beiden letztgenannten Anforderungen konsequent um.

Bei der Programmierung bietet nesC ein System zur Modularisierung, das einzelne Komponenten kapselt. Dies gilt sowohl für reale Komponenten wie Hardwarebausteine als auch für logische Komponenten wie z.B. die Trennung von Applikation und Routing. Dabei trennt nesC die einzelnen Komponenten in die Schnittstelle (engl. Interface) und die Module auf, bei denen es sich um die verschiedenen Implementierungen einer Schnittstelle handelt. Hierdurch lassen sich spezifische Anpassungen an verschiedene Hardware umsetzen, so dass eine Anwendung auf diversen Plattformen einsetzbar ist. Dieses Konzept wird dadurch ermöglicht, dass für eine Schnittstelle mehrere Module existieren dürfen. Die tatsächliche Auswahl des passenden Moduls erfolgt entweder über das Verdrahten (engl. Wiring, s.u.) oder durch den Compiler. Die Auswahl durch den Compiler ist dabei für die verschiedenen Hardware-Plattformen gedacht, das Verdrahten hingegen für die Nutzung alternativer Implementierungen bzw. Module. Des Weiteren definieren die Schnittstellen Kommandos (engl. Command) und Ereignisse (engl. Event). Dabei entsprechen die Kommandos denjenigen Schnittstellen-Funktionen, die ein zugehöriges Modul bereitstellen muss. Die Ereignisse hingegen werden durch ein Modul der entsprechenden Schnittstelle ausgelöst und müssen von denjenigem Modul aufgefangen (d.h. implementiert) werden, welches die Schnittstelle verwendet.

```
interface Timer {
   command result_t start (char type, uint32_t interval);
   command result_t stop ();
   event result_t fired ();
}
```

Listing A.1: Ein TinyOS Interface

Ein kleines Beispiel soll die Begrifflichkeiten näher ausführen: Listing A.1 legt die Schnittstelle eines einfachen Timers fest. Diese bietet die Kommandos `start` und `stop` zum Starten und Anhalten eines Timers. Ein Modul, das diese Schnittstelle implementiert, muss damit diese definierten Kommandos bereitstellen – hier also die eben genannten Kommandos `start` und `stop`. Gleichzeitig definiert die Schnittstelle ein Ereignis namens `fired`. Verwendet ein Modul die Schnittstelle `Timer`, muss es dieses Ereignis verarbeiten können. Obwohl die Ähnlichkeit zur Sprache C offensichtlich ist, gibt es in Bezug auf Kommandos und Ereignisse kleine Unterschiede zu herkömmlichen C-Funktionen: So werden Kommandos mit dem vorangestellten Schlüsselwort `call` aufgerufen, Ereignisse mit `signal` ausgelöst.

Eine weitere interessante, wenn auch in dieser Arbeit nicht explizit verwendete, Eigenschaft von nesC stellt die einfache Realsierung von Nebenläufigkeit dar. Sie ist unmittelbarer Bestandteil der Sprachspezifikation. Neben eigenen Tasks, in Form von speziell deklarierten Funktionen, existieren wie auch in C Interrupt-Handler zur Behandlung von Hardware-Ereignissen.

Neben weiteren kleinen Spezifika erfolgt die generelle Programmierung mit nesC ansonsten wie jede andere Umsetzung einer Anwendung in der Sprache C. Ein kleines, jedoch vollständiges, Beispiel im Abschnitt A.2 soll ein wenig näher auf die Programmierung mit nesC eingehen. Eine ausführliche Sprach-Referenz sowie eine Vorstellung der Sprache finden sich unter [GLCB03] respektive [GLvB+03].

A.2 TinyOS

Bei TinyOS handelt es sich um ein Open-Source Betriebssystem für drahtlose Sensornetze, entwickelt an der University of California, Berkeley. Das gesamte System wurde in nesC geschrieben und basiert damit auf dessen Funktionalitäten und Anforderungen. Im Folgenden soll ein kleiner Überblick an Hand eines einfachen Beispiels über die Programmierung mit nesC und TinyOS gegeben werden.

Listing A.2 zeigt eine typische TinyOS-Anwendung. Hinter dem Schlüsselwort `module` folgt der Name des Moduls. Im zugehörigen Block werden die vom Modul bereitgestellten

```
module App {
   provides {
      interface StdControl;
   }
   uses {
      interface Timer;
      interface Leds;
   }
}
implementation {
   /* StdControl hat die drei Kommandos init, start und stop. */
   command result_t StdControl.init () {
      call Leds.init();
      return SUCCESS;
   }

   command result_t StdControl.start () {
      return call Timer.start(TIMER_REPEAT, 1000);
   }

   command result_t StdControl.stop () {
      return call Timer.stop();
   }

   /* Die Schnittstelle Timer ruft das Ereignis fired auf */
   event result_t Timer.fired () {
      call Leds.redToggle();
      return SUCCESS;
   }
}
```

Listing A.2: Ein TinyOS Modul

(gekennzeichnet durch `provides`) sowie die verwendeten (gekennzeichnet durch `uses`) Schnittstellen bekanntgegeben. Weil das Modul alle Kommandos von bereitgestellten sowie alle Ereignisse von verwendeten Schnittstellen enthalten muss, folgen in der Implementierung (`implementation`) die Kommandos `init`, `start` und `stop` der Schnittstelle `StdControl` sowie das Ereignis `fired` des in Abschnitt A.1 vorgestellten Timers. Im letzten Schritt der Programmierung wird nun die Verdrahtung, wie in Listing A.3 gezeigt, vorgenommen. Sie legt fest, welche Module den bereitgestellten und verwendeten Schnittstellen zugeordnet werden. Ebenso wie bei der Schnittstelle `Timer` handelt es sich bei `StdControl` um einen Bestandteil des TinyOS Betriebssystems, das folglich auch die zugehörigen Module[1] bereitstellt.

[1] Streng genomen handelt es sich bei `TimerC` und `LedsC` um sogenannte Konfigurationen, die hier aber nicht näher erläutert werden sollen

```
configuration AppC {
}
implementation {
   components Main, App, TimerC, LedsC;

   /* Verdrahtung des Moduls Main */
   Main.StdControl -> App.StdControl;
   Main.StdControl -> TimerC.StdControl;

   /* Verdrahtung des Moduls App */
   App.Timer -> TimerC.Timer[unique("Timer")];
   App.Leds  -> LedsC;
}
```

Listing A.3: Eine TinyOS Verdrahtung

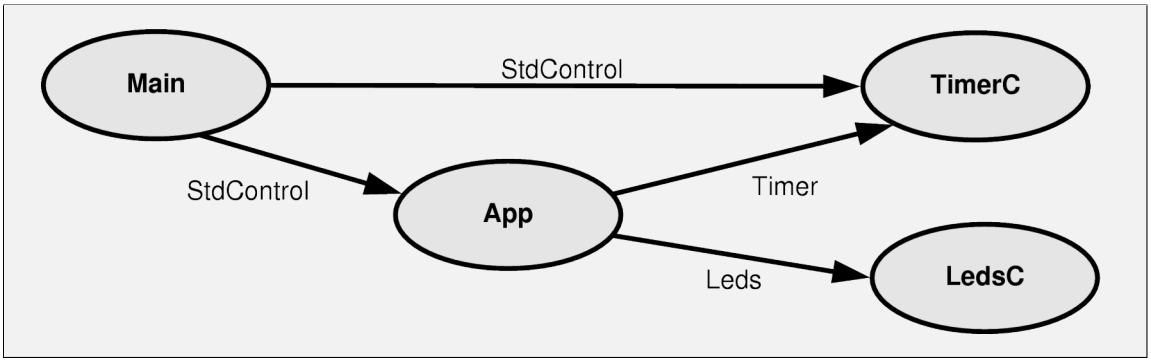


Abbildung A.1: nesC Verdrahtung zu Listing A.3, Module dargestellt durch Kreise, Verdrahtung von Schnittstellen zu Modulen als gerichtete Pfeile

Die Applikation aus den Listings A.2 und A.3 bewirkt das Blinken der roten LED. Das Modul `Main` sorgt dafür, dass `StdControl.start` aller verbundenen Instanzen am Anfang des Programms automatisch aufgerufen wird. Hierdurch wird der Timer gestartet. Er löst periodisch alle 1000 ms aus, was das Umschalten der LED zur Folge hat.

TinyOS bietet den Vorteil, diverse Plattformen zu unterstützen. Hierzu zählen unter anderem Mica-Knoten, BTnodes [Beu06] und auch die ESB-Knoten [SLR+05]. Relevant sind hierbei im Zusammenhang mit der vorliegenden Arbeit insbesondere die beiden letztgenannten Knoten-Typen: BTnodes werden an der ETH Zürich eingesetzt, ESB-Knoten an der TU Hamburg-Harburg. Wichtiger als die Plattform-Unterstützung jedoch ist die Möglichkeit, bestehende Applikationen problemlos simulieren zu können. Hierzu bedarf es in der Regel keiner speziellen Anpassung. Das Programm kann direkt als TOSSIM-Anwendung kompiliert und auf dem PC ausgeführt werden. Im Zusammenspiel mit TinyViz ist sogar eine grafische, interaktive Simulation möglich.

In der vorliegenden Arbeit wird die TinyOS Version 1.1.15 verwendet. Zwar existiert mittlerweile die Version 2.0, jedoch ist diese sehr neu. Dies bedeutet einerseits, dass ein Vergleich mit ähnlichen Arbeiten kaum möglich ist, da diese allesamt in Version 1.1.x implementiert wurden. Andererseits gibt es für TinyOS 2.0 zur Zeit der Anfertigung dieser Arbeit noch keinen visuellen Simulator wie TinyViz. Zudem verspricht Version 1.1.15 im Vergleich zur taufrischen Version 2.0 eine höhere Stabilität und weniger Fehler im Betriebssystem und seinen Bibliotheken.

A.3 TOSSIM

Mit TOSSIM steht dem TinyOS-Paket ein mächtiger Simulator zur Seite [LL03, LLWC03]. Es handelt sich hierbei um einen diskreten Ereignis-Simulator von TinyOS-Code. Wegen der Modularität und der daraus entstehenden Hardwareabstraktion von TinyOS bzw. nesC kann TinyOS-Code ohne Anpassung direkt für eine TOSSIM-Simulation kompiliert werden. Daher eignet sich TOSSIM neben dem Evaluieren und Vergleichen von Algorithmen zudem als geeignetes Werkzeug zum Debuggen und Testen. Trotzdem unterliegt TOSSIM gewissen Einschränkungen im Vergleich zu realen Anwendungen. So liegt beispielsweise der Fokus auf der akkuraten Simulation von TinyOS-Code und nicht auf der vollständigen Abbildung der Realität. Dies äußert sich beispielsweise im Funkverkehr: Dieser wird zwar auf der Bitebene modelliert, eine Anpassung der Sendeleistung ist in TOSSIM aber nicht möglich. Interferenzen haben einen weitaus stärkeren Einfluss auf Paketkollisionen als in realen Sensornetzen.

Mit der Erweiterung PowerTOSSIM [SHC+04] bietet der Simulator eine (wenn auch eingeschränkte) Möglichkeit, Aussagen über den Energieverbrauch der beteiligten Knoten zu treffen. Die Einschränkungen begründen sich durch das verwendete Zustandsmodell, das u.a. Leckströme in Schlafmodi ignoriert und die angelegte Versorgungsspannung vernachlässigt. Zudem kann eine Auswertung nicht zur Laufzeit durchgeführt werden, sondern erst im Nachhinein mittels Analyse der Log-Dateien durch zu PowerTOSSIM gehörenden Python-Skripten.

Trotz dieser Unzulänglichkeiten bietet TOSSIM eine geeignete Simulationsplattform für die vorliegende Arbeit. Dies ist zum einen aus der Tatsache heraus begründet, dass als Randprodukt einsatzfähiger Code entsteht. Gleichzeitig liegt dieser Code in einer wohlgetesteten Form vor – insbesondere seine korrekte Lauffähigkeit unter TinyOS ist gesichert. Andererseits stellt TOSSIM das Funkverhalten in Bezug auf Paketkollisionen und Paketfehlerwahrscheinlichkeiten realitätsnah dar. Zudem werden im Rahmen dieser Arbeit

Algorithmen verglichen, die denselben Restriktionen unterliegen, so dass die Ergebnisse vergleichbar bleiben.

Eine nutzbringende Option von TOSSIM ist die Möglichkeit, fertige Szenarien – Bitfehlerwahrscheinlichkeiten zwischen je zwei Knoten – für die Simulation zu laden. Das hierfür eigens entwickelte Java-Tool „LossyBuilder“ nimmt dabei als Parameter Knotenpositionen entgegen. An Hand von Ergebnissen aus empirischen Messungen in realen Szenarien werden für alle Knotenpaare die Bitfehlerwahrscheinlichkeiten bidirektional bestimmt. Somit lassen sich besonders realitätsnahe Simulationen vergleichsweise einfach durchführen.

A.4 TinyViz

Das Tool TinyViz ist eine grafische Erweiterung von TOSSIM [LL03]. Beide Programme starten prinzipiell separat und kommunizieren dann über Sockets miteinander. Die Kommunikation zwischen den beiden Anwendungen erfolgt allerdings bidirektional, so dass TinyViz mehr ist als ein rein grafischer Aufsatz für TOSSIM.

Das Tool ist Bestandteil des TinyOS-Pakets und wurde vollständig in Java geschrieben. TinyViz an sich stellt lediglich Grundfunktionalitäten wie die Kommunikation mit TOSSIM, das Applikationsfenster, das Zeichnen der Knoten sowie eine Schnittstelle für Plugins bereit. Die tatsächliche Visualisierung übernehmen die Plugins. Dieses System hat den nutzbringenden Vorteil, dass auf einfache sowie vergleichsweise schnelle Art und Weise eigene Anforderungen an die Visualisierung umgesetzt werden können. Dabei werden durch TOSSIM Ereignisse ausgelöst (hierunter fallen z.B. Debug-Ausgaben), die in TinyViz verarbeitet werden können. Der Anzeigebereich des Anwendungsfensters (vgl. Abbildung A.2) unterteilt sich in zwei Hälften: Rechts befindet sich für jedes Plugin ein Reiter, hinter dem sich verschiedene Ausgaben des jeweiligen Moduls befinde, links ist eine Zeichenfläche angeordnet. Diese bietet eine Visualisierung der Sensorknoten und kann von den einzelnen Plugins verwendet werden.

Obwohl für das Sammeln von Mess-Daten im vorliegenden Fall TOSSIM alleine ausreichen würde, wird auf das Zusammenspiel mit TinyViz zurückgegriffen. Dies geschieht einerseits zur visuellen Verifikation während der Entwicklungsphase sowie andererseits auch während der Messungen selbst. Über das TinyViz-Plugin „Location“ existiert nämlich eine einfache Möglichkeit, den Knoten ihre Position bekannt zu machen. Das Plugin simuliert dabei zwei Sensoren, über die die Knoten ihre Position abfragen können (X-, Y-Koordinaten). Diese Möglichkeit ist für die Rahmenapplikation sehr interessant. Überdies kann TinyViz bei Bedarf eine Initialisierungsdatei zur Steuerung der gesamten Simulation

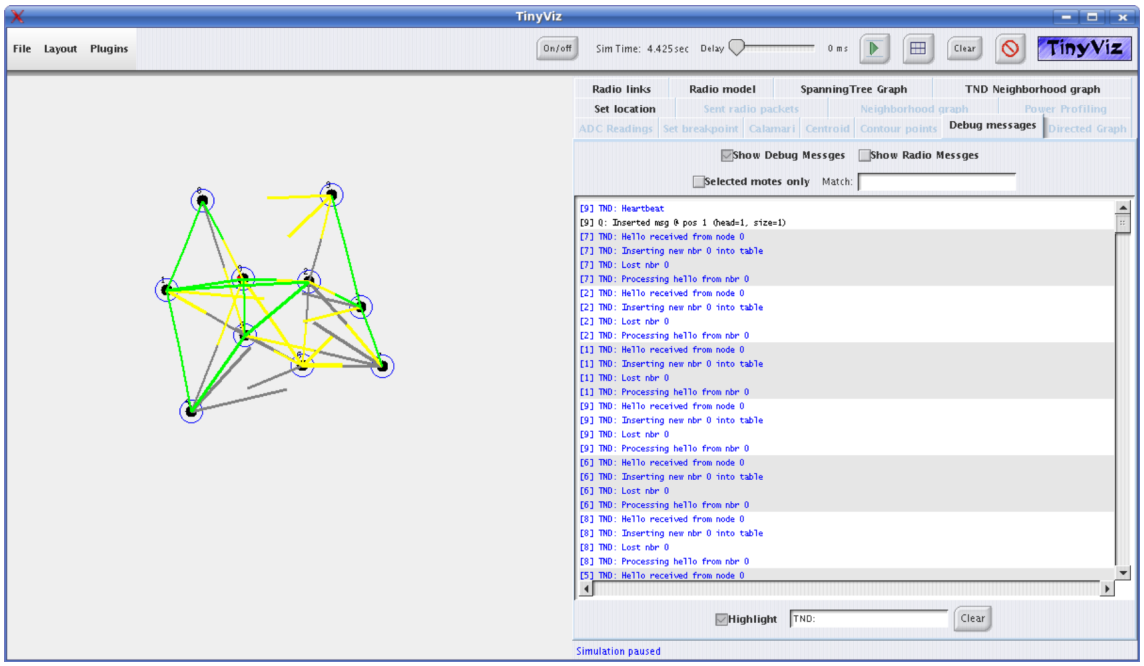


Abbildung A.2: TinyViz Visualisierung mit Nachbarschaftsplugin (linke Fensterhälfte) und Debug-Plugin (rechte Fensterhälfte)

einlesen. Das Laden der Positionen sowie aller anderen Simulationsparameter erfolgt auf diese Weise und vereinfacht das Setup der Testläufe.

A.5 Empirisches Funkmodell

TOSSIM und TinyViz bieten verschiedene Funkmodelle an. In der vorliegenden Arbeit wird ein Modell verwendet, das sich auf empirisch gewonnene Daten stützt [WTC03]. Dieses Modell weist einer Verbindung zwischen je zwei Knoten zwei Bitfehlerwahrscheinlichkeiten zu, eine für jede Kommunikationsrichtung. Jede dieser Wahrscheinlichkeiten wird aus einer Normalverteilung mittels einer Zufallszahl erzeugt. Mittelwert und Standardabweichung hängen dabei von der Distanz der zur Verbindung gehörenden Knoten ab. Durch die zufällige Komponente in der Berechnung ergeben sich asymmetrische Links, wie sich auch in realen Sensornetzen häufig auftreten.

Die Bitfehlerwahrscheinlichkeiten ergeben zusammen mit der Paketlänge und der Distanz zwischen zwei Knoten die entsprechende Paketfehlerwahrscheinlichkeit. Abbildung A.3 stellt die Paketfehlerwahrscheinlichkeiten für verschiedene Knotenentfernungen und Paketgrößen unter Verwendung der Mittelwerte der einzelnen Verteilungen dar. Hieraus lässt sich ablesen, dass in einer Entfernung von maximal 4 Metern die Paketlänge nur einen

kleinen Einfluss auf eine erfolgreiche Übertragung hat. Bei einer Entfernung von 4 bis 6 Metern wird dieser Einfluss jedoch signifikant. Ab ca. 7 bis 8 Metern ergibt sich eine Rate von bereits mindestens 50 Prozent. Für eine zuverlässige Kommunikation ist damit eine maximale Entfernung von ca. 7–8 Metern, besser 4–5 Metern erforderlich.

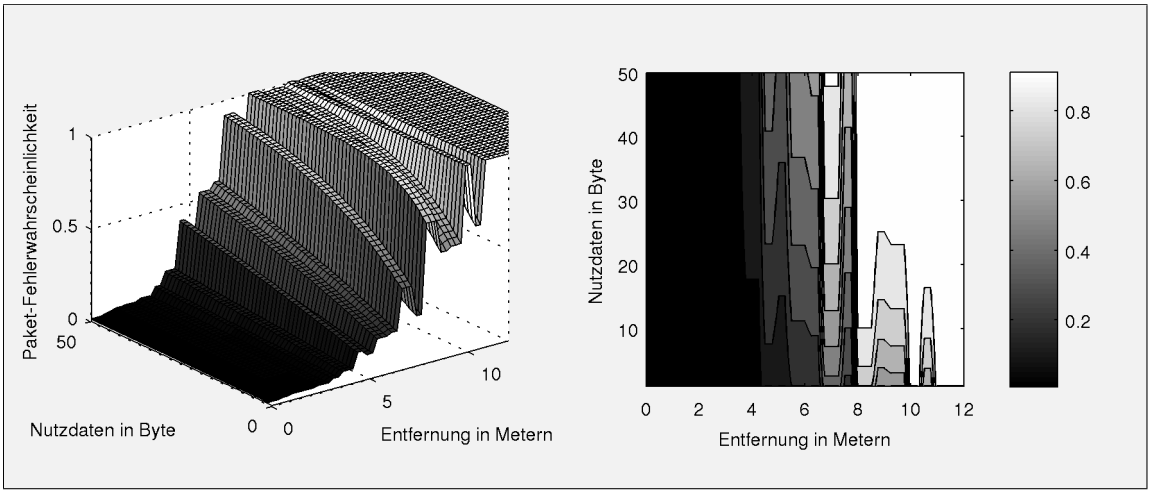


Abbildung A.3: Paketfehlerwahrscheinlichkeit in Abhängigkeit der Nutzdaten in Byte sowie der Entfernung in Metern. Darstellung dreidimensional (links) und als vereinfachte Intensitätskontur.